An Introduction to

Java and Object Oriented

Programming

Abstract

The primary aim of the notes are to provide an introduction to the Java programing language. It is assumed that you know one programming language moderately well. The secondary aim is to look at object oriented programming and the features that support it in Java. The third aim is to look at the functioanlity provided in the language by the extensive application programming interface – api.

The material in these notes is evolving as Java evolves.

Author: Ian D Chivers

Email: ian.chivers@kcl.ac.uk

Version 3.2: 24/11/2000

All comments welcome.

Table of Contents

1
Overview

‘The first thing we do, let’s kill all the language lawyers.’

Henry VI, part II

Aims

The aims of the chapter are to provide a background to the organisation of the course.

1 Overview

1.1 Aims

The primary aim of the course is to provide an introduction to programming using Java. Some of the examples are taken from the Fortran 95 and C++ courses. These examples are available on the college web server enabling you to compare the syntax of the three languages.

The rest of the examples look at what Java has to offer in its own right. In particular there is a coverage of both Java programs and Java applets. Java programs are usually interpreted using the Java interpreter provided with the development kit you use, whilst Java applets run in a web browser. We will look into the differences in more depth later.

1.2 Assumptions

It is assumed that the reader:–

- has a working knowledge of programming with one of Fortran 77, Fortran 90, Fortran 95, Pascal, Modula 2, C or C++;
- knows about the benefits of structured programming;
- knows about the data structuring facilities in one of the above languages;
- has some knowledge of object oriented programming;

1.3 Course Material and Recommended Sources

The course material is not complete. It is useful to make the following distinctions when learning any programming language:–

- introductory complete texts on a programming language
- reference texts on a programming language
- good programming practice in a programming language
- introductions to object oriented programming

The following are some sources I've found useful. The list is not exhaustive and your mileage will vary. To get a reasonable working knowledge of Java I've found that I've had to:–

- read a variety of books;
- used the on-line documentation that comes with each version I've used.
- use the links that Sun provide to their web server. They provide a lot of essential information regarding Java.
- read the definitions of the language provided both in books published by Sun and available on line with the development kit;
- last and most importantly write examples to test out my understanding;

Sun provide on-line turorials and you may like to have a look at what they offer. Home page is:

http://java.sun.com/docs/books/tutorial/index.html

The tutorial is also available for download. You might want to consider this option when working at home. Avoids the cost of the telephone.

URL is

http://java.sun.com/docs/books/tutorial/information/download.html

The size of these may put you off downloading over a telephone line. Around 9-10 Mb in March 2000.

1.4 Java Versions – Bits of History

Java is evolving. There are a number of versions around. The first I used was JDK 1.0.2. The oldest books will use this version. The date of the book is very important. Later chapters in the notes will provide additional information.

1.4.1 1.0.x

The first release was in early 1996. 1.0.2 was the first version I used and came out in 1996 too. 1.0.2 added an improved set of graphics classes. First College course given in 1996–1997.

1.4.2 1.1.x

Early 1997. Added a new event handling mechanism. By now the primitive nature of the AWT (Abstract Windows Toolkit) had became apparent. 1997 saw the introduction of the Java Foundation Classes which supersedes and includes AWT. These new components were called Swing. JavaBeans came into existence. JavaBeans is a component architecture for the Java platform. We will look into this in much greater depth in later chapters.

1.4.3 1.2.x – aka Java 2.

Sun rebadged Java in December 1998 when Java 1.2 became Java 2.

Be very careful when buying books on Java. You need to ensure that the books that you buy address the version we will be using.

1.5 Development platforms

The majority of the examples have been tested using the Sun development kits (JDKs) on both Solaris and Windows environments. These are both free. They may be downloaded and installed on your own machine if you have one. The first JDK used was 1.0.2. I am now using 1.2.2. JDK 1.3 is in the pipeline.

You should also get the documentation that Sun provide. This is installed on the College web server. There are links from the Java pages. If you program at home I would download it and install it.

The following is the sun home page:

- http://www.java.sun.com/

The following has details of the jdk

- http://java.sun.com/products/jdk/1.2/

that is available for download.

I have also used:

- IBM Visual Age for Java. It exists in three editions:–
  - Entry Edition – free
  - Professional – 65 UK pounds;
  - Enterprise – approx 2,000 uk pounds;

    Needless to say I haven't used the Enterprise Edition. The Entry Edition can be downloaded from the web. Visit:

    http://www7.software.ibm.com/vad.nsf

for more details.

- Microsoft Visual J++ 6.0. This comes bundled with Developer Studio Enterprise Edition. This is available at quite a reasonable price. I also use Visual Basic and Visual C++. I also have Visual Fortran from DEC – Now Compaq. This is integrated into the Developer Studio Environment.
- Sun Java Workshop
- Netbeans

Details of the last two can be found at Sun's addresses above.

Other implementations of Java exist and include:–

- Java Workshop – Sun
- JBuilder – Sybase
- PowerJEnterprise Edition – Sybase
- Visual Cafe
- IBM Jikes – this is freely available and provides a dos based working enviornment on the pc.

There are a number of ways of working with Java:

- Use the command prompt under DOS or Unix and compile and run the programs.
- Use a workbench to provide a graphical interface to the above.
- Use a visual development environment that enables you to develop programs using a mouse and drag and drop components.

The quick and dirty method is to use the command prompt. The other two approaches take additional time due to their learning curve. The environments are generally different, so learning one may not shorten the learning curve for the next.

Web access to data is looked at too, and obviously companies like Oracle and IBM have offerings in this area.

1.6 Development kits and Standards

I've tried to adhere to the Sun definition. Work is underway to get a formal standard. Other implementations are available and there are differences.

The lack of a standard and the differences between the various implementations makes learning Java more difficult than other (formally standardised) languages, e.g. Fortran, Pascal, C. C++ has caused problems in this area, but things are improving now with the publication of the C++ standard in November 1997.

1.7 Miscellanea

We will be using a Sun unix system to learn Java. You will have to learn about:–

- X-Windows access to the system using Vista Exceed;
- the file manager;
- terminal window access and the unix operating system;
- the editor
- html;

- netscape and other browsers;

Coverage of these is provided in separate notes and within the body of the notes

1.8 Course Timetable

The following is a rough guideline to the ten week course. It is flexible.

- Overview of program language development and why Java is the way it is;
- Basic introduction to the Java language and its usage;
- Conventional programming language features – arithmetic, expressions, data types, Strings, Arrays, control structures;
- Additional Java syntax and semantics, e.g. exception handling, threads, i/o and streams, graphics, inheritance, class extension, packages, interfaces;
- AWT;
- Swing
- Java Beans – Software Components;
- Java Environments, Workbenches etc;
- Web access to data;

I'll be putting all sources on the web server.

1.9 Coda

Be prepared to devote some time to learning Java. You can't gain an understanding of 50 years of program language development and an object oriented language like Java without some effort on your part. Modifying existing programs is a good place to start, but you have to write your own from scratch to really learn a language. Think about how you learn French, German, etc. Practice makes perfect. Be patient, Rome wasn't built in a day.

1.10 Bibliography

Deitel H.M., Deitel P.J., *Java: How to Program*, Prentice Hall.

1st Edition comments. This is a very well written complete coverage of the Java language. If you had to get just one large text then this is one to consider. 1050 pages in all.

3rd Edition comments**.** I bought this edition in February 2000 and it has been updated considerably. It now covers Java 2 and Swing. The CD has JDK1.2.1 on it as well as Netbeans Developer 2.2.1 and Inprise (nee Borland) JBuilder 3 (University Edition). It also has coverage of JDBC, servlets, remote method invocation, and the attempt to provide an equivalent to the STL in Java. At around £30 it is very good value. Also contains lots of links to Java resources available on the Web.

The following is a list of some of the chapters:

2. Java Applications
3. Java Applets
4. Control structures – 1
5. Control structures 2
6. Methods
7. Arrays

8. Object based programming
9. Object oriented programming
10. Straings and characters
11. Graphics and Java2D
12. Basic GUI components
13. Advanced Gui
14. Exception handling
15. Multihtreading
16. Multimedia, animation, audio, video
17. Files and streams
18. JDBC
19. Servlets
20. RMI
21. Networking
22. Data structures
23. Utilities
24. Collections
25. JavaBeans

Eckstein R., Loy M., Wood D., *Java Swing*, O R'Reilly.

There are a number of books on Swing and this one looks to be one of the better ones with a coverage of most of the features I was interested in. The following are the chapters:

1 Introducing Swing
2 Jump starting a Swing application
3 Swing component basics
4 Labels and Icons
5 Buttons
6 Bounded range components
7 Lists and Combo boxes
8 Swing containers
9 Internal frames
10 Swing dialogs
11 Speciality panes and layout managers
12 Chooser dialogs
13 Borders
14 Menus and toolbars
15 Tables
16 Advanced table examples
17 Trees
18 Undo

19 Text

20 Document models and events

21 Styled documents and Jtextpane

22 Carets, highlighters and keymaps

23 text views

24 editorkits and text actions

25 Programming with accessibility

26 Look and feel

27 Swing utilities

28 Swing under the hood

Nilsson D.R., Jakab P.M., *Developing JavaBeans Using VisualAge for Java*, Wiley.

I have had a look at JavaBeans and decided that a graphical front end would probably be a good idea. This meant looking at the issue from proprietary software. As IBM make their version available for download I chose a book that was specifically about VisualAge for Java. Comes with 1.1.5 on a CD. Release 2 is available for download.

Winder R., Roberts G., *Developing Java Software*, Wiley.

If you don't have a formal background in programming, algorithms and data structures, object oriented programming then I would suggest that you have a look at this book.

Hunt J., *Java and Object Orientation*, Springer.

If you want an introduction to object oriented programming take a look at this one.

JDK 1.2.2 On-line documentation.

This is available on the college web server. Just follow the links. You can also download and install to your own machine. This is the source of technical information on Java.

Arnold K., Gosling J., *The Java Programming language*, Addison Wesley.

Written by two of the Java team. I normally try to get hold of a book by people who have been behind the design of a programming language when trying to learn it. This one is very disappointing.

Gosling, Yellin and the Java Team, *The Java API, Volume 1, Core Packages, Addison Wesley.*

Paper definition of the application programming interface. Also available on-line but obviously more difficult to use. Covers the core of the language. Out of date as soon as I bought it! Not recommended. Persevere and use the on-line material instead.

Gosling, Yellin and the Java Team, *The Java API, Volume 2, Window Toolkit and Applets*, Addison Wesley.

The second api text. Not recommended. Use the on-line material instead.

Flanagan, *Java in a Nutshell*, O'Reilly & Associates.

For people familiar with the Nutshell series this book is what one would expect. It offers an introduction to Java, programming with the Java api, a language reference and api quick reference. If you had to buy one book then this is one to consider.

Cowell, *Essential Visual J++ – **fast***, Springer Verlag.

> One of the new series by Springer. Tries to put a quart into a pint pot and does very well. If you want a quick introduction to what Microsoft J++ can offer then this is one to consider.

What you get depends on your background, what access you have to a printer, and the internet. A lot of material is available from the web. Look at the above books and think about your background and experience in:

- programming and programming languages
- algorithms
- data structures
- object oriented programming
- windows programming

and also what you want to do.

The on-line FAQ is ok. Not as good as the C++ one.

2
Introduction to Programming Languages and Object Oriented Programming

‘We have to go to another language in order to think clearly about the problem.’

Samuel R. Delany, Babel–17

Aims

The primary aim of this chapter is to look at some of the languages used in the sciences. There is a look at the developments from a historical view point; a comparison of their features and a look at future developments.

2 An Introduction to Programming Languages and Object Oriented Programming

The intention of this chapter is to examine, from the viewpoint of languages of use in scientific problem solving, of the background of programming languages and their development. It is essential that you develop an understanding of why there are so many programming languages and their strengths and weaknesses. No one language is suitable for solving all the problems that you will come across. You need to chose the right tool for the job. Think about diy around the home. If the only tool you have is a hammer then everything seems to be seen as a nail.

See the bibliography for a broader coverage.

2.1 Fortran 66, 1966

The original was designed by a team lead by Backus at IBM, in the late 50's. It is therefore quite old. This is the date of the first standard. The language quickly established itself as the language of first choice for numeric programming.

2.2 Pascal, 1975, ANSI & BSI 1982, ISO 1983, Extended Pascal 1991?

Very successful attempt at a *teaching* language. Note that it precedes both C and Fortran 77. Pascal still is the most widely taught programming language in computer science departments, as the introductory programming language. The following summarises the survey done by Dick Reid taken from a number of editions:–

Language	**20th**	**18th**	**15th**	**13th**
Pascal	140	144	151	157
C++	101	100	87	34
Ada	82	82	74	73
C	58	56	51	39
Scheme	50	49	51	50
Modula	32	32	32	35
Java	15	-	-	-
Modula-2	14	15	15	13
Fortran	9	9	9	8
SML	8	7	6	6
Turing	4	4	5	6
Miranda	4	4	4	3
Smalltalk	4	4	4	1
Eiffel	3	3	3	3
Oberon	3	3	2	2
ISETL	2	2	2	2
ML	2	2	2	1
Modula-3	2	2	2	2
ObjPascal	2	2	2	1
Ada95	2	-	-	-
Haskel	2	1	1	1
Beta	1	1	1	1
Oberon-2	1	1	1	-
Orwell	1	1	1	1
Prolog	1	1	1	1

Simula	1	1	1	1
Blue	1	-	-	-

The first edition was May 1990. New editions come out about every six months. He doesn't keep past editions. I've put up the complete survey, which includes the institutions, at:–

`http://www.kcl.ac.uk/kis/support/cc/fortran/dickreid20.txt`

What is interesting is the following:–

`http://www.kcl.ac.uk/kis/support/cc/fortran/sdickreid20.txt`

which is a sorted list by language. Where is your institution?

2.3 Fortran 77, 1978

Modest attempt to update the language. Still largely Fortran 66. Added BLOCK IF, better array subscripting and corresponding do looping. Given the knowledge of the time a very conservative update to the language.

2.4 C, K&R 1978, Standard 1989.

C was developed by Kernighan and Ritchie at Bell Labs. Bell Labs was the research laboratory of the Bell Telephone company in the US. It was originally written for a PDP 11 under UNIX. It is based on the cpl, bcpl, b family of languages and these are typeless languages. It was designed as a systems implementation language and was used to rewrite some 95% of the UNIX operating system. Only 5% or so ended up being written in assembler. The UNIX tools are written in C and are a very good example of what C is best at: the construction of sharp, small tools. There are little or no facilities in the language for the construction of larger code suites. Given the date of the publication of K&R there was the opportunity to have tidied the language up somewhat. Look at the features of some of the other languages covered here and the dates to see what is meant by this statement.

The X library is written in C, is over 15 years old and still leaks memory.

2.5 Modula 2, 1982, Standard 1996?

Wirth's next language after Algol W and Pascal. Attempt to produce a professional programmers Pascal. Very many good features. Let down by the delay in getting standardised. Introduced modules (hence the name), got rid of some of the idiosyncratic syntactic sugar of Pascal, had the idea of separate definition and implementation. Rivals Ada without much of the complexity of Ada for real time applications.

Numeric work in Modula 2 isn't very attractive. Explicit type casts are required in mathematical expressions. The proposed standard alleviates some of the problems in this area.

2.6 Ada, ISO 8652: 1987

Attempt to produce a powerful and expressive language by the American Department of Defence. Given the very large defence spending budget, even a 1% gain from the adoption of a better programming language will be repaid. Gaining ground, from a slow start. See also Ada 95 later. First draft report was 1980.

2.7 C++, 1986, Standard November 1997

Attempt by Stroustrup to produce an object oriented version of C. He had been exposed to Simula early on and realised the benefits of a language like that. Simula is a product of the 1960's. The first version of Simula was available in 1967. Object oriented programming is not new! Simula was widely used for discrete event simulation.

2.8 Oberon 2, Late 1980's, early 1990's.

Very clean and simple OOP language. Partly driven by the visit of Wirth and Gutnecht to Xerox PARC, and Wirth having to take over the operating system course at ETH. Name arises from the Voyager probe taking pictures of Uranus. Oberon is the largest moon. Oberon was simpler than Modula 2. Oberon 2 tidied up a bit. First operational system by the late 1980s. Oberon replaced Modula 2 in 1989 for teaching purposes at ETH. Ported to a Apple MAC, SUN, DEC, IBM RS6000 and MS/DOS by 1991. Free versions available from the ftp server at ETH.

Let down badly by continual development and lack of a standard. It is a research vehicle for Wirth and the CS department at ETH in Zurich. Some of the problems here may be remedied in the near future with the progress being made on the standardisation front.

2.9 Fortran 90, 1991.

Modern language. Limited OO capability. Good information hiding and powerful mathematical capability. The language of first choice for people involved in numeric programming.

2.10 Eiffel, 1988

Date is the publication of *Object Oriented Software Construction*, Meyer, Prentice Hall. Modern object oriented language. Meyer is a very keen exponent of the benefits of object oriented programming. Meyer's book on object oriented software construction is an extremely good introduction to OO programming.

The second edition is even better.

Eiffel is an attempt to produce an industrial strength OOP language. I hope to be able to make available an Eiffel compiler for the alpha in the near future.

Another extremely worthwhile acquisition is *Software Development Using Eiffel: There Can Be Life Other Than C++*, Richard Wiener. He highlights some of the weaknesses of C++. If you are going to attempt a reasonable size application in C++ using OO techniques then you should read this book. He clearly highlights some of the major pitfalls. He provides both academic and commercial courses on C++ and Eiffel.

2.11 Ada, ISO 8652: 1995

Latest standard. Major changes from the 1987 standard are in the areas of:–

- Support for 8 and 16 bit character sets;
- Object oriented programming with run-time polymorphism;
- Extension of access types;
- Efficient data oriented synchronisation;
- library units;
- interfacing to other languages;

There are several so called *Specialised Needs Annexes*. These are:–

- Annex C, *Systems Programming*
- Annex D, *Real-Time Systems*
- Annex E, *Distributed Systems*
- Annex F, *Information Systems*

- Annex G, *Numerics*
- Annex H, *Safety and Security*

2.12 Java

Bill Joy (of Sun fame) had by the late 1980's decided that C++ was too complicated and that an object oriented environment based upon C++ would be of use. At round about the same time James Gosling (mister emacs) was starting to get frustrated with the implementation of an SGML editor in C++. Oak was the outcome of Gosling's frustration.

Sun over the next few years ended up developing Oak for a variety of projects. It wasn't until Sun developed their own web browser, Hotjava that Java as a language hit the streets. And as the saying goes *the rest is history*.

Java is a relatively simple object oriented language. Whilst it has its origins in C++ it has dispensed with most of the dangerous features. It is OO throughout. Everything is a class.

It is interpreted and the intermediate byte code will run on any machine that has a Java virtual machine for it. This is portability at the object code level, unlike portability at the source code level – which is what we expect with most conventional languages.

It has built in garbage collection – no dispose!

It has no pointers – everything is passed by reference!

It is multithreaded, which makes it a delight for many applications.

It has a extensive windows toolkit, the so called AWT that was in the original release of the language and Swing that came in later. It achieves much of what Visual Basic offers but within the framework of a far more powerful language. Development environments are becoming widely available to aid in this task.

Finally it is fun!

Major drawback is the rapid development of the language and the large number of different versions. Further compounded by the different virtual machines available.

2.13 Visual Basic

This language is a development by Microsoft to enable visual user interfaces to be programmed easily. It has subject to continual development and offers one of the easiest ways of developing a windows style program for a pc running Windows 3.x, 95, 98 and NT.

2.14 Language Comparison

The following page has a simple language feature comparison. The emphasis is on highlighting the strengths and weaknesses of languages used mainly in the scientific area.

The following symbols are used:–

Y	supported
y	supported with qualification, e.g. may be achieved using a different mechanism
-	not supported
?	unable to verify adequately at the time of writing

In all cases please see the more detailed coverage that follows.

Simple Program Language Feature Comparison

	Fortran			Pascal			C	C++	Ada		Java
	66	77	90 95	Modula 2 Oberon 2							
				P	M2	O2	C	C++			
Year	66	78	91 96	75 82	82 96?		78 89	86 97	87	95	?
Feature											
Independent compilation	Y	Y	Y	-	-	-	Y	Y	-	-	-
Separate compilation	-	-	Y	-	Y	Y	-	Y	Y	Y	Y
Concrete data types	-	-	Y	Y	Y	Y	Y	Y	Y	Y	[1]
Abstract data types	-	-	Y	-	Y	Y	-	Y	Y	Y	Y
Dynamic arrays	-	-	Y	-	-	-	Y	Y	Y	Y	Y
Modules See below	-	-	Y	-	Y	Y	-	y	y	y	y
Numeric Polymorphism See below	-	-	Y	-	?	Y	-	y	y	y	-
General Polymorphism	-	-	Y	-	?	Y	-	Y	Y	Y	Y
Pointers	-	-	Y	Y	Y	Y	Y	Y	Y	Y	[2]
Procedure variables	-	-	-	Y	Y	Y	Y	Y	?	?	?
Inheritance single/mult	-	-	-	-	-	S	-	M	-	S	S
Dynamic binding	-	-	-	-	-	Y	-	Y	-	Y	Y
Operator overloading	-	-	Y					Y	Y	Y	-
Threads Tasking	-	-	-	-	Y	?	-	-	?	Y	Y
Exception Handling	-	-	-		?	?	-	Y	?	Y	Y

These are some of the major features that we need to look at when comparing programming languages and looking at the development of programming languages.

[1] – Against the spirit of object oriented programming.

[2] – Replaced by references

Dates

Fortran: The dates are the dates of the standards.

Pascal: Preliminary version 1968. Major development and first operational compiler 1970. 1973 publication of first revised report. User Manual and Report 1975.

Modula: Defined experimentally 1975. Lilith research project 1977. First implementation of Modula 2 1979. Technical report March 1980. First distributed compiler March 1981. *Programming in Modula 2* 1982.

Oberon 2: 1988, N. Wirth. *The Programming Language Oberon*, Software Practice and Experience, 1991, Mossenbeck and Wirth, *The Programming Language Oberon 2*. 1991, Reisser , *The Oberon System, User Guide and Programmer's Manual*. 1993, Mossenbeck, *Object Oriented Programming in Oberon 2*.

C: 1969, M Richards: *BCPL A Tool for Compiler Writing and Systems Programming*. 1970, Ken Thompson, B. 1978, K and R. *The C Programming Language,*. 1989 ANSI C Standard.

C++: C with classes, 1979-1983 From C with Classes to C++, 1982-1985 Release 2.0: 1985-1988 Stroustrup, *The C++ Programming Language*, 1st edition, 1986 Stroustrup, *The C++ Programming Language*, 2nd edition, 1991. Standard 1997.

Ada: 1980 original definition. Standard 1987. Latest standard is 1995.

Java: Exact date not well defined. Hopefully the standardisation effort will make life easier!

2.15 Language Features

It is illuminating to look at the languages from the viewpoint of their features and functionality.

2.15.1 Independent Compilation

The ability to break a problem down into parts and work on one part at a time. No checking between compilation units.

2.15.2 Separate Compilation

As above with checking *across* compilation units. Major step forward in the construction of more complex programs. Lint helps out with C. Forcheck is useful for Fortran 66 and 77 programmers.

2.15.3 Concrete Data Types

The ability for the user to define data types that mapped directly onto their problem. A major step forward. User has to know about the implementation however.

2.15.4 Abstract Data Types

The twin concept of data types and procedures that manipulated the data. Hiding the internals from the calling routine.

2.15.5 Dynamic arrays

Arrays that are allocated dynamically at run time.

2.15.6 Numeric and General Polymorphism

Numeric Polymorphism

In the simplest case the ability to have so called mixed mode arithmetic expressions, e.g. mix integers and reals (both of one or more underlying representations) without casting be-

tween one type and another. The explicit type casting required in some languages means that they are not widely used for numeric programming.

At the next level the ability to call in built functions with numeric data of one or more types. This has been in Fortran from a very early stage.

Finally the ability to create one's own functions that accept numeric data of a variety of numeric types.

Languages that support OOP have to offer the last kind of polymorphism.

General Polymorphism

OO programming languages have to offer this functionality.

2.15.7 Modules

The primary purpose of modules is to provide the ability to group related functions and procedures. This is a powerful program decomposition tool. They normally have a well controlled mechanism for making visible what the external, calling routine needs to have access to.

Terminology varies with programming languages, and so does the exact functionality that these different languages support.

Classes and packages are two terms also used.

2.15.8 Pointers and References

Pointers in a programming language extend the range of problems that can be solved considerably. Multi-dimensional structures are easily programmed using pointers, e.g. linked lists, queues, trees, quad-trees, oct-trees etc.

The major problem is that the user is provided with very little help if they are programmed incorrectly. It is assumed that you know what you are doing.

In Java all objects are accessed via an object reference. When you see an object variable in a Java program you actually see a reference to an object. We will look into the concept of references in much greater detail throughout the course.

2.15.9 Procedure Variables

An elegant way of extending the expressive power of a language. Quite old.

2.15.10 Inheritance

The first of the two major step towards OO programming. Allows the user to extend an existing type without having to know what is going on.

2.15.11 Dynamic Binding

The second of the two major features of OO programming. Provided in a limited sense via procedure variables in older languages.

2.15.12 Operator Overloading

Syntactic sugar in many ways and over valued. Very useful to people with numeric problems. Given that C++ has 47 operators it poses problems of readability and comprehensibility with most other areas. The restrictions that C++ has in this area will be looked at later.

2.15.13 Threads/Multitasking

Multitasking and/or threads are needed in a programming language when solving problems in the areas of real-time systems, equipment interfacing, embedded systems and parallel programming.

2.15.14 Exception Handling

Exception handling is needed in a programming language when solving problems in the areas of real-time systems, equipment interfacing, embedded systems, parallel programming and robust systems.

2.16 Some Important Milestones in Program Language Development

We look here at some of the major steps forward in the way we approach problem solving using programming languages. Often people adopted the following methodologies without having features in a programming language that actually supported them.

2.16.1 Structured Programming

Structured programming in its narrowest sense concerns itself with the development of programs using a small but sufficient set of statements and in particular control statements. It has had a great effect on program language design and most languages now support a minimal set of control structures. In a broader sense it subsumes other objectives including simplicity, comprehensibility, verifiability, modifiability and maintenance of programs.

The ideas are very well covered in the Dahl, Dijkstra, Hoare text. This is essential reading.

2.16.2 Stepwise Refinement

The original ideas are very well expressed a a paper by Wirth entitled *Program Development by Stepwise Refinement*, published in 1971. Essential reading.

2.16.3 Data Structuring, Concrete vs Abstract Data Types

With a concrete data structure you have to know how the data structure is organised explicitly. With abstract data types the internals are hidden from you and all you see are a set of procedures that provide the functionality you want.

2.16.4 Information Hiding – Modules

A major step forward in the development of programming languages. The paper by Parnas addresses the idea of information hiding and the module concept is one that a number of languages now offer to help in this area.

2.17 Terminology of Object Oriented Programming

The following provides a link between conventional programming language terminology and that used in object oriented programming.

Class	Extensible abstract data type
Object	Instance of a class
Message	Procedure call, dynamically bound
Method	Procedure of a class.

See Mossenbeck for a good treatment of this. We'll come back to the whole area of object oriented programming after a coverage of the basics of C++.

2.18 Parallel Developments

With the increasing availability of computers with multiple processors at ever decreasing costs the importance of languages that support parallel computation increases. Two languages that offer support in this area are based on Fortran 90 and C++. A brief coverage is given below.

2.18.1 Parallel Fortran – Fortran 95, Fortran 2000, SMP, MPI, HPF

To quote from the HPF Language Specification, version 1.1, November 1994 *The High Performance Fortran Forum (HPFF) was founded as a coalition of industrial and academic groups working to suggest a set of standard extensions to Fortran to provide the necessary information. Its intent was to develop extensions to Fortran that provide support for high performance programming on a wide variety of machines, including massively parallel SIMD and MIMD systems and vector processors. From its very beginning HPFF included most vendors delivering parallel machines, a number of government laboratories and many university research groups. Public input was encouraged to the greatest possible extent.*

A number of suppliers now provide HPF Fortran extensions and these are generally based on Fortran 90, rather than Fortran 77.

Fortran 95 and 2000 offer support for parallelisation.

SMP and MPI are two other developments in this area.

2.18.2 Parallel C++

Similar developments are in the pipeline for C++.

2.19 Object Oriented Programming

Object oriented programming is characterised by two main concepts:–

- inheritance;
- dynamic binding;

The major thing we need to consider is how to extend the functionality of an existing program. To make sense of the benefits of OOP we need to have a good understanding of the strengths and weaknesses of the traditional programming paradigm. We need to look at the way our appreciation of how to use programming languages developed and what we needed from them as the problems we tackled became more complex.

To quote Friedman:–

> Object oriented programming makes good on the promise of structured programming. It implements in a very practical way the principles of program decomposition, data abstraction and information hiding. It ties together and provides a framework for abstraction and structure at all levels; data program and control.
>
> ...
>
> OOP picks up where structured programming methodology leaves off. Dijkstra's concept of structured programming, Wirth's stepwise refinement and Parnas's information hiding all contribute to a software development milieu that promised to become increasingly systematic and scientific. OOP, to a great extent, fulfils that promise. It takes the concepts of data abstraction, modular decomposition, and information hiding and refines them in a cohesive world view, data objects are active entities. Instead of passing data to procedures the user asks objects to perform operations on themselves. A complex problem is viewed as a network of objects that communicate with each other.

The benefits of OOP come with programming in the large. If the problems you have don't warrant it you may never need to devote the time and effort to gain complete mastery of a powerful and complex language like C++.

2.20 Object Oriented Languages

The ideas are not new.

2.20.1 Simula – 1967

The seminal text on OOP is *Simula BEGIN*, Birtwistle, Dahl, Myhrhaug and Nygaard. The book is very well written if a little dated today. I'd recommend it if you are involved in discrete event simulation. This was what the language was used for whilst I worked at Imperial College before I came to King's in 1986. They are the first people to use the concept of a class. Algol 60 based. Stroustrup got many of his ideas from Simula.

2.20.2 Smalltalk – 1978

The text I recommend is *Smalltalk 80, The Language and its Implementation*, Goldberg and Robson. They worked at the Xerox Palo Alto Research Centre (Xerox Parc) Learning Research Group. Ideas are drawn from Simula and the vision of Alan Kay. Steve Jobs was heavily influenced by and the Apple Macintosh owes a big debt to the Xerox Parc people. We take windowing systems for granted these days on many of the systems we work with from the pc with Windows, to Unix workstations with their X windows interfaces and of course the first to bring them to the mass market – the Apple Mac.

Regarded as a pure OO system by most people with everything an object.

Wirth has spent a number of periods at Xerox Parc and that is reflected in Oberon 2.

2.20.3 C++

In Stroustrup's words ...*C++ is a general purpose programming language; its core application domain is systems programming in the broadest sense...*

It is of course also used in a wide range of other application domains, notable graphics programming. It is enormously popular, and there are a very large number of jobs advertised for people with C++ skills. You are unlikely to be out of work if you get to be a good C++ programmer.

C++ supports inheritance through class derivation. Dynamic binding is provided by virtual class functions.

2.20.4 Eiffel

Object Oriented Software Construction is dated 1988. As stated earlier the first four chapters address OOP. The latter chapters look at Eiffel in some depth. The text is a relatively easy read.

Achieves much of the power and expressiveness of C++ without the complexity. As is has its origins in Ada is is also a language that offers far greater support for error protection and safe code.

I'm informed that the Channel Tunnel software uses Eiffel!

I am looking at getting hold of an Eiffel compiler at this time, but am unsure as to what platform it might be available on.

2.20.5 Oberon 2

The language has its origins in a visit that Gutnecht and Wirth made to Xerox PARC in 1985. They decided to design and implement a new operating system. In their words ... *the ultimate goal was to create a system for personal workstations that was not only powerful and convenient for practical use but also describable and explicable in a convincing way...* They had originally decided to use Modula 2 but made the decision to strip out some of the

features of that language and add a very small number of features. The outcome was Oberon. The language was defined in 1986.

The object code size of the outer core of the Oberon system is 200K, and comprises

- a kernel
- a dynamic loader and garbage collector
- a file system
- drivers for disk, diskette, asynchronous and synchronous communication, printer and a bit mapped display;
- local network services;
- support for texts and fonts;
- a window subsystem;
- a text editor;
- the Oberon compiler;

Educational versions of the system are available from the ETH ftp server. I'd recommend 8Mb of memory and an 80486 with 1024*756 display. I've tried at home on a 20 MHz 80386 with 5 Mb of memory – *a bit slow...* Well worth a look at.

Programming in Oberon, Reiser and Wirth, is a good introductory text and combined with *OOP in Oberon 2,* Mossenbeck, you have sufficient information to get started.

There is a lot of documentation that comes with the system and this can be printed.

The system integrates very well with both Windows on the PC and Apple macs. Versions for other platforms are available.

If your are familiar with Pascal or Modula 2 then I'd recommend Oberon very highly to see what OOP has to offer. Very low cost in time, effort and money.

2.20.6 Ada 95

Whilst I do not have access yet to an Ada 95 compiler from what I've read it looks a very good language. The standard is available from a number of ftp servers, and there is also a look at the changes from the original version to 95 available on the web. A text I'd recommend is *Programming in Ada 95*, Barnes. This is well written. Ada 95 is the first language that has been standardised with OOP in mind.

If you want a job in the defence industries, or see yourself working with real time embedded systems then this is certainly a language to consider looking at.

2.20.7 Java

Java is a recent OO language. It is unusual in that it is the product of one company, Sun, rather than the subject to formal language standardisation like the majority of the other languages covered in this chapter. It is hoped that it will pass out of Sun's hands in the near future into the mainstream of language standardisation. Freely available if you have internet access. IBM have taken it on board in a big way and so have Microsoft. They both realise the potential earning capacity of Java and the internet.

Due to long file names and multithreading requirements needs operating systems like Windows 95 and above.

2.21 Other Languages

There are languages that offer limited support for OOP. The two that follow fit into this category. I don't have completely up to date information on what is likely to be in the Modula 2 standard at this time.

2.21.1 Fortran 90 and Fortran 95

Through the functionality provided via user defined data types and modules it offers support for *object based* programming. See Dupee's MSc thesis for a good coverage of what Fortran 90 has to offer here.

2.21.2 Modula 2

The original language was a major advance over Pascal. It corrected many of the deficiencies of Pascal in a straightforward way. From what I've seen of the draft standards it will be a powerful and expressive language.

Standard versions of the language look like being quite expensive and it is unlikely that we would be able to make available a compiler on any platform given the proposed cuts in expenditure by the various government funding bodies.

2.22 The OO Approach

We will look at two approaches here. The work of Meyer and Rumbaugh et al are both well regarded, and we will cover both briefly.

2.22.1 Meyer's Approach

Meyer in his first edition (dated 1988) identified seven steps that lead towards object oriented solutions. These were:–

- object based modular structure – systems are modularised on the basis of the data structures;
- data abstraction – objects should be described as implementations of abstract data types;
- automatic memory management – unused objects should be deallocated by the underlying language systems, without programmer intervention;
- classes – every non simple type is a module, and every high level module is a type;
- inheritance – a class may be defined as an extension of restriction of another;
- polymorphism and dynamic binding – program entities should be permitted to refer to objects of more than one class and operations should be permitted to have different realisations in different classes;
- multiple and repeated inheritance – it should be possible to declare a class as heir to more than one class and more than once to the same class;

The second edition (dated 1997) is a new book, rather than an update of the first edition. Chapters 1 and 2 are an introduction and overview. Chapters 3 through 6 provide coverage of the road to object orientation. Chapters 7 through 18 are the technical core of the book looking at object oriented techniques. There is a coverage of:

- classes
- objects
- memory management

- genericity
- design by contract
- exception handling
- supporting mechanisms
- inheritance
- multiple inheritance
- inheritance techniques
- typing
- global obects and constants

Programming is an engineering activity and has evolved considerably since the first programming languages of the 1950's. The changes between the first and second editions reflect the developments that have taken place over nearly 10 years. This book is essential reading if you are seriously interested in objected oriented programming.

2.22.2 Rumbaugh et al

This book concentrates on OO modelling, rather than using a particular programming language for OO programming. If you have a background in the relational database area then much of the coverage should be quite familiar. They present a methodology for object oriented development – the Object Modelling Technique or OMT. They identify four stages:–

- analysis: build a model of the real world situation;
- system design: make the high level decisions about the overall architecture;
- object design: build a design model (based on the analysis model) with implementation details;
- implementation: translate into an implementation using a particular programming language, database, or hardware implementation;

and three kinds of models to describe the system:–

- the object model: describes the static structure of the objects and their relationships;
- the dynamic model: describes the aspects of the system that change with time;
- the functional model: describes the data value transformations within the system;

and the three models are orthogonal, with cross links.

This book is a must for large scale systems.

2.22.3 Practical Steps

The two major practical steps are:–

- identify the classes; within this discriminate between:–
  - an *is a* relationship, e.g. where one class is a sub-type of another;

    Consider the concept of a point. It has two attributes, an x and y position. So we could write point(x,y). Now consider the concept of a pixel – a point with the

added attribute of colour. Now we could write pixel(point,colour), Thus pixel is-a point.

- a *has a* relationship, e.g. where a class is made up of an object of another class;

  Consider the concepts of engine and car. In this case a car has-a engine.

- define the interfaces to the classes.

Inheritance commits you to much more than becoming a client. As a client you are protected against future changes in the implementation of a class. When you inherit you gain access to the implementation and all that goes with it.

The above have to be done before any code is written. Programming is an iterative process and it is inevitable that you will need to cycle through the design and implementation stages as you write code, i.e. it will be obvious that you will need to go back and redesign and reimplement base classes in the light of experience.

2.23 Simple Example

Consider putting together a graphical drawing system. We are interested in shapes and the concepts of moving and drawing. We can do this in a very straightforward way using an OO approach.

Firstly we have a base abstract class shape with two associated procedures, one to move and one to draw.

- shape(x,y)

  move(shape s)

  draw(shape s)

Secondly we then derive other shapes from them. We provide two derived classes with two associated procedures:–

- square(shape,side)

  move(shape s)

  draw(shape s)

- rectangle(shape,length,breadth)

  move(shape s)

  draw(shape s)

Consider the following pseudo-code segment

```
...
square s(50,50,10);
rectangle r(100,100,10,20);
   move(s)
   draw(s)
   move(r)
   draw(r)

return(0)
}
```


We can now add another shape, e.g. circle, and still have things work *without* recompiling the old code. We just compile the new move and draw procedures and link them in. The method resolution is handled by dynamic binding. We will look into this in greater depth later in the course.

2.24 Other Developments

The chapter rounds off with a look at recent developments that have taken place and that apply to one or more programming languages in some cases.

2.24.1 Development Environments

The traditional working practice for program development involves the following steps:–

- edit
- compile
- link
- run
- debug

in a loop. This method has the advantage of working on most hardware and software platforms. The major drawback is learning several ways of doing exactly the same thing as we move from platform to platform and from one language to another. How many editors do you know how to use?

There have also been developments to provide an integrated environment for program development. These environments started out as workbenches providing simple access to the editor, compiler, linker etc through simple key strokes and the mouse. They have grown very sophisticated.

2.24.2 Graphical Development Tools

There has also been the development of a visual interface to programming. Increasingly people want easy to use software that almost by definition has a wndows based interface. Microsoft Visual Basic provides a good example of this. Products like this typically have:–

- a toolbox of components that can be dragged and dropped onto a screen
- a screen or form that the user will see – this comprises the user interface
- a set of properties for each of the components that can be tailored for your own requirements

You typically use the mouse to select the item you want from the toolbox (menu, form etc) drag and dop onto the form and then alter the various associated setting using the propertiy entries on the right hand side. Skeleton code is often generated for you which you then tailor to your own specific requirements.

For some alternatives to using Visual Basic to put a windows based front end to a program have a look at the following url for more details:–

`http://www.kcl.ac.uk/kis/support/cc/fortran/language.html`

There will be a course later this academic year that looks at this whole area in more depth.

2.24.3 Software Components

As the problems that we attempt to solve become more complex and the interfaces we provide become more sophisticated we need better tools to help with program development.

One major step forward is in reusable software components. This can be seen as an extensions to the object oriented approach to programming.

Consider building an application that required a spelling checker. The idea is to buy one of the shelf and slot it straight in to our program. This is gradually becoming a reality.

Sun and Microsoft both made developments in this area and we will look at each in turn.

2.24.3.1 COM, OLE, ActiveX

First let us define each of the above in turn:–

- COM – Component Object Model
- OLE – Object Linking and Embedding
- ActiveX – Now called Active Technologies

Let is look briefly at each in turn.

ActiveX is a set of technologies that enables software components to interact with one another in a networked environment, regardless of the language in which they were created. ActiveX is build on the Component Object Model.

OLE or OLE Automatation makes it possible for one application to manipulate objects implemented in another application or to expose objects so that they can be manipulated.

A simple example of this would be embedding a spreadsheet of graph from Excel in a Word document. Double clicking on the spreadsheets drops you into Excel. There are some problems with this when working with a document on a number of computer systems with different versions of the applications concerned.

The Component Object Model (COM) is a platform independent, distributed, object orientded, system for creating binary software documents that can interact. COM is the foundation technology for OLE and ActiveX.

2.24.3.2 JavaBeans

JavaBeans is the Java language software component offering. JavaBeans extends the Java language write once run everywhere capability to resuable component development.

There is considerable interest in JavaBeans because of the platform neutrality of the developed code. Sun also provide mechanisms to migrate ActiveX/OLE/COM to JavaBeans.

Sun and Microsoft and battling this one out and there is little love lost between them. Watch this space as they say!

2.25 Coda

There is a lot that John Backus has to be proud of. He achieved a lot with Fortran. As we shall see Fortran is still the language of first choice for the majority of numeric based problem solving. There is a lot of very well written code in Fortran, and we find both commercial and public domain numeric libraries available for most platforms.

Wirth has a lot to be proud of too with the Algol family of languages – Algol, Algol W, Pascal, Modula, Modula 2, Oberon, Oberon 2. The sad thing that is that whilst abandoning the previous language enabled successor languages to be well designed and compact, there was a lot of effort required in moving production code from older languages to their successors. Knowledge of one of the more recent languages (Modula 2 or Oberon) in this family is a worthwhile investment.

C and C++ have a considerable amount of code written in them. C++ represents a very major advance over C, correcting some of the program correctness problems that arise in C from array subscript checking, pointer manipulation, type checking and optimisation prob-

lems with pointer aliasing and array handling. For many applications they offer quite significant advantages over other languages. Graphical based systems are invariably written in C++. Object oriented programming is often done in C++. The maintenance problems C poses are considerable. The steep learning curve that C++ has poses problems for successive generations of would be C++ programmers.

Look at the Computational Science Education paper for an comparison of C, Fortran 77, Fortran 90 and C++. This is on the college web server.

Whither Java? Java seems to be a language that will survive. How widely used it will become depends obviously on the success of the internet and local intranets.

Whither Visual Basic? VB will survive. Many people need to be able to develop easy to use programs and systems. Windows programs offer the possibility of solving this problem.

No one language is appropriate for solving every problem. Many factors come into play in real life. Learning a variety of languages is a good idea. Learning Fortran 90, C++, Java and Visual Basic provides you with a range of very useful skills for work in the academic, scientific, engineering and commercial worlds.

I've included references to a couple of other languages that you might like to look at, and these are Icon, Snobol, Prolog and SQL. Icon and Snobol are very good string processing languages. SQL is **the** database language, and Prolog is a logic based language.

2.26 Bibliography

Adobe Systems Incorporated, *Postscript Language: Tutorial and Cookbook*, Addison Wesley.

Adobe Systems Incorporated, *Postscript Language: Reference Manual,* Addison Wesley.

Adobe System Incorporated, *Postscript Language: Program Design,* Addison Wesley.

The three books provide a comprehensive coverage of the facilities and capabilities of Postscript.

ACM SIG PLAN, *History of Programming Languages Conference – HOPL-II,* ACM Press.

One of the best sources of information on programming language developments, from an historical perspective. The is coverage of Ada, Algol 68, C, C++, CLU, Concurrent Pascal, Formac, Forth, Icon, Lisp, Pascal, Prolog, Smalltalk and Simulation Languages by the people involved in the original design and or implementation. Very highly recommended. This is the second in the HOPL series, and the first was edited by Wexelblat. Details are given later.

Adams, Brainerd, Martin, Smith, Wagener, *Fortran 90 Handbook: Complete ANSI/ISO Reference,* McGraw Hill.

A complete coverage of the language. As with the Metcalf and Reid book some of the authors were on the X3J3 committee. Expensive, but very thorough.

Annals of the History of Computing, *Special Issue: Fortran's 25 Anniversary*, ACM publication.

Very interesting comments, some anecdotal, about the early work on Fortran.

Arnold K., Gosling J., *The Java Programming Language,* Addison Wesley.

Written by the people who designed and implemented the language. A definitive source on the language. A bit expensive at just under thirty uk pounds.

Barnes J., *Programming in Ada 95*, Addison Wesley.

A recent update of his previous Ada book. Comprehensive coverage of Ada 95. Not for the beginner.

Birtwistle G.M., Dahl O. J., Myhrhaug B., Nygaard K., *SIMULA BEGIN*, Chartwell-Bratt Ltd.

A number of chapters in the book will be of interest to programmers unfamiliar with some of the ideas involved in a variety of areas including systems and models, simulation, and co-routines. Also has some sound practical advice on problem solving.

Booch G., *Object Oriented Design with Applications*, Benjamin Cummings, 2nd Ed. 1994.

I've not been able to get hold of a copy of this yet. One is on order at Dillons. Don't buy or bother with the first edition, as there are bound to be major advances in this edition due to his experience between editions. Still not available at this time.

Brinch-Hansen P., *The Programming Language Concurrent Pascal*, IEEE Transactions on Software Engineering, June 1975, 199-207.

Looks at the extensions to Pascal necessary to support concurrent processes.

Cannan S., Otten G., *SQL – The Standard Handbook*, McGraw Hill.

Very thorough coverage of the SQL standard, ISO 9075:1992(E).

Chivers I. D. and Clark M. W., *History and Future of Fortran*, Data Processing, vol. 27 no 1, January/February 1985.

Short article on an early draft of the standard, around version 90.

Chivers I.D., and Sleightholme J., *Introducing Fortran 90*, Springer Verlag.

An introduction to programming using Fortran 90. Aimed at numeric problem solving.

Chivers I.D., and Sleightholme J., *Introducing Fortran 95*, Springer Verlag.

An introduction to programming using Fortran 95. Aimed at numeric problem solving. Updated to reflect the new standard and with coverage of two tecnical updates to the language that will be part of the next standard – F2K!

Computational Science Education Project, *Fortran 90 and Computational Science.*

This paper is a comparison of C, Fortran 77, C++ and Fortran 90 using the following five criteria:– numerical robustness, data parallelisation, data abstraction, object oriented programming and functional programming. A copy is available on the college web server. Essential reading if one is involved in programming with one or more of these languages.

Cowell J., *Essential Java: Fast*, Springer

Compact introduction to Java. Insufficient on its own.

Dahl O. J., Dijkstra E. W., Hoare C. A. R., *Structured Programming*, Academic Press, 1972

The seminal book on structured programming.

Date C. J., *A Guide to the SQL Standard*, Addison Wesley.

Date has written extensively on the whole database field, and this book looks at the SQL language itself. As with many of Dates works quite easy to read. Appendix F provides a useful SQL bibliography.

Deitel H.M., and Deital P.J., *Java – How to Program*, Prentice Hall.

A very good introductory Java text. One of the best currently available. Highly recommended.

Dupee B., *A Study of Object Oriented Methods using Fortran 90*., MSc Thesis.

A look at OO methods in F90, rather than full blown OOP.

Flanagan D., *Java in a Nutshell*, O'Reilly and Associates.

One of the Nutshell series. If you want one book on Java then this is the one I'd recommend. Four parts are introduction to Java, programming with the Java api, Java language reference and api quick reference.

Friedman L. W. , *Comparative Programming Languages*, Prentice Hall.

I've got this on order.

Geissman L. B., *Separate Compilation in Modula 2 and the structure of the Modula 2 Compiler on the Personal Computer Lilith*, Dissertation 7286, ETH Zurich

Jacobi C., *Code Generation and the Lilith Architecture*, Dissertation 7195, ETH Zurich

Fascinating background reading concerning Modula 2 and the Lilith architecture.

Goldberg A., and Robson D., *Smalltalk 80: The language and its implementation*, Addison Wesley.

Written by some of the Xerox PARC people who have been involved with the development of Smalltalk. Provides a good introduction (if that is possible with the written word) of the capabilities of Smalltalk.

Goos and Hartmanis (Eds), *The Programming Language Ada - Reference Manual*, Springer Verlag.

The definition of the language.

Gosling J., Yellin F., The Java Team, *The Java API, Volumes I and II*, Addison Wesley.

Volume I looks at the core packages and Volume II looks at the Window Toolkit and Applets. I find the pricing a bit much at just under 40 uk pounds a book.

Griswold R. E., Poage J. F., Polonsky I. P., *The Snobol4 Programming Language*, Prentice-Hall.

The original book on the language. Also provides some short historical material on the language.

Griswold R. E., Griswold M. T., *The Icon Programming Language*, Prentice-Hall.

The definition of the language with a lot of good examples. Also contains information on how to obtain public domain versions of the language for a variety of machines and operating systems.

Hoare C.A.R., *Hints on Programming Language Design*, SIGACT/SIGPLAN Symposium on Principles of Programming Languages, October 1973.

The first sentence of the introduction sums it up beautifully: *I would like in this paper to present a philosophy of the design and evaluation of programming lan-*

guages which I have adopted and developed over a number of years, namely that the primary purpose of a programming language is to help the programmer in the practice of his art.

Jenson K., Wirth N., *Pascal: User Manual and Report*, Springer Verlag.

The original definition of the Pascal language. Understandably dated when one looks at more recent expositions on programming in Pascal.

Kemeny J.G., Kurtz T.E., *Basic Programming*, Wiley.

The original book on Basic by its designers.

Kernighan B. W., Ritchie D. M., *The C Programming Language*, Prentice Hall: Englewood Cliffs, New Jersey.

The original work on the C language, and thus essential for serious work with C.

Kowalski R., *Logic Programming in the Fifth Generation*, The Knowledge Engineering Review, The BCS Specialist Group on Expert Systems.

A short paper providing a good background to Prolog and logic programming, with an extensive bibliography.

Knuth D. E., *The TeXbook*, Addison Wesley.

Knuth writes with an tremendous enthusiasm and perhaps this is understandable as he did design TeX. Has to be read from cover to cover for a full understanding of the capability of TeX.

Lemay L., Perkins, *Teach Yourself Java in 21 Days*, Sams net.

Most gentle of the books I've found. Includes a CD with the Sun JDK.

Lyons J., *Chomsky*, Fontana/Collins, 1982.

A good introduction to the work of Chomsky, with the added benefit that Chomsky himself read and commented on it for Lyons. Very readable.

Malpas J., *Prolog: A Relational Language and its Applications*, Prentice-Hall.

A good introduction to Prolog for people with some programming background. Good bibliography. Looks at a variety of versions of Prolog.

Marcus C., *Prolog Programming: Applications for Database Systems, Expert Systems and Natural Language Systems*, Addison Wesley.

Coverage of the use of Prolog in the above areas. As with the previous book aimed mainly at programmers, and hence not suitable as an introduction to Prolog as only two chapters are devoted to introducing Prolog.

Metcalf M. and Reid J., *Fortran 90 Explained*, Oxford Science Publications, OUP.

A clear compact coverage of the main features of Fortran 8x. Reid was secretary of the X3J3 committee.

Meyer B., *Object Oriented Software Construction*, Prentice Hall.

I'm just got the second edition. The first edition is dated 1988. Whilst obviously Eiffel based well worth a read. Also looks at other languages. This comparision is not as good as it could be.

Mossenbeck H., *Object-Oriented Programming in Oberon-2*, Springer–Verlag.

A very good and simple introduction to OOP. Uses Oberon-2 as the implementation language. Highly recommended.

Papert S., *Mindstorms - Children, Computers and Powerful Ideas*, Harvester Press

Very personal vision of the uses of computers by children. It challenges many conventional ideas in this area.

Parnas D. L., *On the Criteria to be Used in Decomposing Systems into Modules*, Communications of the ACM, 15 (12), 1972.

One of the earliest papers to look at information hiding.

Sammett J., *Programming Languages: History and Fundamentals*, Prentice Hall.

Possibly the most comprehensive introduction to the history of program language development – ends unfortunately before the 1980's.

Reiser M., Wirth N., *Programming in Oberon – Steps Beyond Pascal and Modula*, Addison Wesley.

Good introduction to Oberon. Revealing history of the developments behind Oberon.

Reiser M., *The Oberon System: User Guide and Programmer's Manual*, Addison Wesley.

How to use the Oberon system, rather than the language.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorenson W.,*Object Oriented Modelling and Design*, Prentice Hall.

I like this book a lot. Having been involved in the relational area for over 10 years the book struck an immediate chord.

Seed G., *An Introduction to OOP in C++*, Springer.

Comprehensive introduction to C++ and OOP. Examples drawn from the computer graphics area.

Young S. J., *An Introduction to Ada, 2nd Edition*, Ellis Horwood.

A readable introduction to Ada. Greater clarity than the first edition. Dated in terms of the recent developments with Ada 95.

Wexelblat, *History of Programming Languages, HOPL I*, ACM Monograph Series, Academic Press.

Very thorough coverage of the development of programming languages up to June 1978. Sessions on Fortran, Algol, Lisp, Cobol, APT, Jovial, GPSS, Simula, JOSS, Basic, PL/I, Snobol and APL, with speakers involved in the original languages. Very highly recommended.

Wiener R., *Software Development using Eiffel: There Can Be Life Other than C++*, Prentice Hall.

Well written, and the case studies include an ecological simulation, a game of strategies and investments and simulated annealing. The chapter on object oriented analysis and design is highly recommended.

Wiener R., *Software Development Using Eiffel: There Can be Life After C++*, Prentice Hall.

Very well written. I'd really recommend getting hold of this book if you are going to seriously program in C++ using oo techniques. He teaches both academic and commercial Eiffel and C++ courses. He knows his stuff!

Winder R., Roberts G., *Developing Java software*, Wiley.

Waiting to get a copy.

Wirth N., *An Assessment of the Programming Language Pascal*, IEEE Transactions on Software Engineering, June 1975, 192-198.

Wirth N., Program Development by Stepwise Refinement, Communications of the ACM, April 1971.

Clear and simple exposition of the ideas of stepwise refinement.

Wirth N., *History and Goals of Modula 2*, Byte, August 1984, 145-152.

Straight from the horse's mouth!

Wirth N., *On the Design of Programming Languages*, Proc. IFIP Congress 74, 386-393, North Holland, Amsterdam.

Wirth N., *The Programming Language Pascal*, Acta Informatica 1, 35-63, 1971.

Wirth N., *Modula: a language for modular multi- programming*, Software Practice and Experience, 7, 3-35, 1977.

Wirth N., *Programming in Modula 2*, Springer Verlag.

The original definition of the language. Essential reading for anyone considering programming in Modula 2 on a long term basis.

Wirth N. *Type Extensions*, ACM Trans. on Prog. Languages and Systems, 10, 2 (April 1988), 2004-214

Wirth N. *From Modula 2 to Oberon*, Software – Practice and Experience, 18,7 (July 1988), 661-670

Wirth N., Gutknecht J., *Project Oberon: The Design of an Operating System and Compiler*, Addison Wesley.

Fascinating background to the development of Oberon. Highly recommended for anyone involved in large scale program development, not only in the areas of programming languages and operating systems, but more generally.

2.27 Problems

What programming languages are available on the system you work with?

3
Introduction to Programming in Java

‘Though this be madness, yet there is method in’t’
Shakespeare.

‘Plenty of practice’ he went on repeating, all the time that Alice was getting him on his feet again. ‘plenty of practice.’
The White Knight, Through the Looking Glass and What Alice Found There, Lewis Carroll.

Aims

The aims of the chapter are:–

- to look at a simple hello world example;
- to look at java programs;
- to look at java applets;
- to look at a simple string example;
- to look at a simple numeric example;
- to introduce some of the formal syntactical rules of Java;
- to look at the Java keywords;
- to provide definitions of some technical terms used throughout the rest of the notes;
- to look at the Java memory model;
- to look at the Java character set;

3 An Introduction to Java

In this chapter we will look at three simple program examples. The first looks at a simple hello world example, the second a simple text i/o and the third looks at simple numeric i/o.

We will also look at variants of these programs that occur with Java depending on whether we want to write a Java program or a Java applet that runs within a web browser. We will also look at the flavours of doing this that exist within Java due to the evolution of the language.

We will also look at some of the syntax of Java programs.

3.1 Program Development

Most people will have the following model of programming:–

- write the program using an editor;
- compile the source using a compiler;
- linking the output from the compiler to produce an executable;
- running the executable;

and as most of us know this is an iterative process. We rarely get it right first time.

Java is different. There is in effect no link stage. The compiler generates byte codes that are then interpreted directly using a java byte code interpreter. This interpreter can be a web browser that supports java or an interpreter that runs on whatever platform that you are using.

This means that if one develops java programs as applets that run on a web browser that the java byte codes are effectively platform independent. The applets can run on any platform that has a java virtual machine built into the web browser.

This is probably the major reason for the take up of Java. With the increasing importance of the web people require a way to develop portable applications. You do not have to reimplement for each platform that you want to support. Note also that java compiled code is small. Another big plus for web based computing.

3.2 Java Programs

Java programs have access to the underlying computer system.

3.3 Java Applets

Java applets do not have access to the native machine on which the web browser runs.

3.4 Hello World – Java Program

The following is a complete Java program.

```
class c0301
{
   public static void main(String[] args)
   {
      System.out.println(“ Hello world”);
   }
}
```

Java programs and applets comprise one or more classes. In these notes we will use the convention of c for chapter, dd, for chapter number and xx for example within that chapter. So this is the first example in chapter 03.

Within a class we have one or more procedures and data. In Java terminology these are methods and fields.

The { character signifies the start of the class.

The `public static void main(String[] args)` statement signifies that this is a Java program. We will look at turning the above into an applet in the next example.

The c0301 class contains one method – main. This method returns a value of void. There is the concept of functions and procedures in other languages with functions like sine returning a value, but within the C/C++/Java family of languages we just have functions and when they behave as a procedure or subroutine in another language they are void, and return a void data type.

This method is static. We will look in more depth a full coverage of static throughout the course. For the time being all that is necessary to know is that if we are writing Java programs that the main method has to be static.

This method has the public attribute. This means that the method is visible whereever the class is used.

The main method takes one argument or parameter, `args` and this is of type `String[]`. This enables the program to access data typed at the command line. We will look at arrays of type String in much greater depth throughout the course.

We then have the { character signifying the start of the main method. There is one statement in this method and that is `System.out.println(" Hello world")`. Let us look at this statement in more depth. With object oriented languages there are the twin concepts of objects and methods. What we are doing is invoking the `println` method on the `System` class's `out` object.

With Java there is the concept of the core language and additional functionality is provided by a number of built in classes. Much work has been done with Java to provide a lot of functionality with these additional in built classes. It is essential to develop a good working knowledge of what has already been provide. There is little point reinventing the wheel. This is achieved in C and C++ using the #include statement.

Text or strings are delimited in Java with " marks.

; is the statement seperator in Java.

The } character signifies the end of the program.

There are a number of issues that are important whilst preparing java programs:–

- the file name of the java program or applet must be the same as the class name
- if using more than one platform then stick to an operating system that supports long file names. This means that using Windows 3.x is not recommended.
- java is case sensitive – I go against the normal style in java texts and stick to lower case. I also use short names.
- compile you java program using

  javac c0301.java
- run your programs using the java interpreter

  java c0301.class

3.5 Hello World – Java Applet

The following is a complete Java applet.

```
import java.awt.*;

public class c0301a extends java.applet.Applet
{
   public void paint(Graphics g)
   {
      g.drawString(“ Hello world”,10,20);
   }
}
```

The first line identifies which of the java classes are to be made available to the applet – the syntax is in fact the same for a Java program.

There are several core classes in the Java application program interface – api. These are:–

- java.lang
- java.io
- java.util
- java.net
- java.awt
- java.applet

and we will look at these in more depth where appropriate. A good understanding of Java therefore requires a knowledge of the contents of each class.

When we import a class we are able to use methods within that class with a shortened name form. So

```
import java.awt.*;
```

makes available the abstract windowing toolkit. This package is made of the following main components:–

- graphics – for controlling fonts, colours etc;
- components – for controlling graphical user interfaces (gui, and pronounced gooey) using buttons, menus, lists etc;
- layout managers – for the control of components within their container objects;
- image – for manipulating images;

We will look into this whole area in more detail throughout the course.

The following identifies this class as an applet, rather then a program:–

```
public class c0301a extends java.applet.Applet
```

The first thing to note is that our class `c0301a` is an extension of one of the inbuilt Java classes – `Applet.` This is a simple example of inheritance. This means we can create our own applets by using the build in class `java.applet.Applet`. This class has a number of methods:–

- init()
- start()
- stop()
- destroy()

but in this example we have not bothered overriding them, rather we have relied on default behaviour.

Within our applet we have one method:–

```
public void paint(Graphics g)
```

and the method is public and void. The paint method takes one parameter and this is an object g of type Graphics, which is one of the data types provided.

Within this method we have one statement:–

```
g.drawString(" Hello world",10,20);
```

and this applies the drawString method to the object g. The arguments to drawString are:–

- " Hello world" – the string we want to appear on the screen
- 10 – the x position in pixels
- 20 – the y position in pixels

The } character signifies the end of the applet.

The outcome of all of this is that we now have an applet that we can run within a web browser, e.g. netscape. This is not the end of the story however. We also need an html file that can be loaded into the browser. HyperText Markup Language (HTML) is a markup language and is based on SGML, which is an ISO standard. Within HTML terms a document comprises two parts, the text and the markup. Markup languages are not new and TeX and Runoff are two widely used markup languages. HTML is widely used and is the basis for the World Wide Web (WWW) distributed information system. The web was originally developed at the European Particle Physics Laboratory (CERN) in Geneva to enable high-energy physics researchers to work collaboratively on online documentation; it soon became apparent that the system could be useful to a broader section of the population, and the software became publicly available in 1991.

Here is the html file associated with this simple applet.

```
<html>
<head>
<title Hello World ></title>
</head>
<body>
<applet code=c0301a.class width=300 height=100></applet>
</body>
</html>
```

This simple html file uses a number of tags

- <html> – an instruction to the web browser that this is an html document;
- <head> – the heading
- <title> – the title of the document which appears somewhere within the browser. Also appears in a booklist.
- <body> – the main body of the document
- <applet> – used to signify a java applet

More infomation can be found on html on the College web server. There are links to some very good coverages of html. You need to have a good working knowledge of html if you are interested in Java applets. They key tag in the above is the applet tag. Let us look at this in more depth:–

```
code=c0301a.class width=300 height=100
```

The actual applet to run is called c0301a.class. When we compile our java programs we end up (if there are no errors) with a file with a .class extension. So this is the compiled java applet. The width and height are the size of the inital sizes of the browser window in pixels. These two values must be specified.

So when we create an applet as opposed to a program we need to also create an html file that can be used by the web browser.

The applet examples will be put up on the college web server as the course progresses. These can then be ran using Netscape on a pc or a mac. I use Netscape 3 under both Windows 95 and NT. You can run them yourselves on the Sun using the appletviewer that comes with the development kit.

So the sequence is now:–

- create the java applet – use the unix editor vi
- compile the java applet – use the Sun Java compiler javac
- create the associated html file – vi
- use the appletviewer to run the html file – appletviewer. Alternatively use a web browser.

I have put up the applets on the college web server so that you can try them out using a web browser.

3.6 Hello World: JApplet

The next variation uses JApplet, rather than Applet. JApplet provides access to the graphical components from javax.swing.

```
import java.awt.*;

public class c0301b extends javax.swing.JApplet
{
   public void paint(Graphics g)
   {
      g.drawString(“ Hello world”,10,20);
   }
}
```

Note that we are still using the graphics components from java.awt.

3.7 Hello World: JApplet alternate syntax

This shows an alternate syntax for this program.

```
import java.awt.*;
import javax.swing.JApplet;

public class c0301c extends JApplet
{
   public void paint(Graphics g)
   {
      g.drawString(“ Hello world”,10,20);
   }
}
```

3.8 Hello World: JComponent

This next example shows how to do it using one of the components from JComponents. Note that this is a program, not an applet.

```
import javax.swing.JOptionPane;

public class c0301d
{
   public static void main(String args[])
   {
      JOptionPane.showMessageDialog(null," Hello world “);
      System.exit(0);
   }
}
```

We have used the simplest form of the showMessageDialog method.

We have seen four ways of achieving much the same end result. The reasons for this are two fold. Firstly there is the concept of a program and an applet. Secondly Java is evolving and there are now several ways to achieve (more or less) the same end result. Different books will show you different ways of writing Java applets and programs.

3.9 Program for line i/o

This program example reads a line of input from the user and echos it. This example introduces a number of very important concepts about java.

```
import java.io.*;

class c0302
{
   public static void main(String[] args)
   {
      try
      {
         InputStream i=System.in;
         DataInputStream in=new DataInputStream(i);
         String Line;
         System.out.println(“ Type in a line of text ”);
         Line=in.readLine();
         System.out.println(Line);
      }
      catch(IOException e)
      {  System.out.println(“ Exceptions raised: ” + e); }
   }
}
```

In this example we are interested in doing i/o. In the previous program and applet we only did output. We first import from java.io to make available in shortened name form the various facilities we need.

The next thing is the name of the class – c0302. Remember java is case sensitive and the file name must match the class name.

We then have the standard statement for a java program:–

```
public static void main(String[] args)
```

where main is made public, static and returns a void type. We have the standard arguments to main which are the argumnets to the program when it is executed returned as a String. Let us now look at each statement in turn.

```
try
```

Mechanisms for handling execution errors in a programming language are to be found in most programming langauges. Java supports exception handling. This facility is also to be found in C++, Modula 2 and Ada. It is a significant improvement over the facilities provided in older languages.

If you are doing i/o you have to include the code doing the i/o in an exception handler. So the try statment indicates the start of an exception handling block.

```
InputStream i=System.in;
```

You should be familiar from your programming background with the the concept of a variable being of a particular data type and having an initial value. So the above means that the variable i is of type InputStream and has an initial value of System.in.

```
DataInputStream in=new DataInputStream(i);
```

This statement is similar to the above but adds one very important extension. Java is an object oriented language and objects have to be created. The in object is of type DataInputStream and is created by the New DataInputStream(i) statement. Objects are created using new.

```
String Line;
```

Line is a variable of type String. In fact String is an object in Java terminology, and is not a fundamental type like integer or float.

```
System.out.println(“ Type in a line of text ”);
```

This statement causes the text string *Type in a line of text* to appear on the screen.

```
Line=in.readLine();
```

This statement reads in a line of text from the keyboard.

```
System.out.println(Line);
```

This statement echos the line back to the screen.

```
catch(IOException e)
```

This is the second part of the execution handling syntax. Within the above block if an error occurs control passes to this statement and the following statement is executed.

```
{  System.out.println(“ Exceptions raised: ” + e); }
```

In the event of an error the above message appears on the screen.

If we compare the above to similar programs in other more conventional programming languages the additional syntax looks very off putting. Text i/o in Basic, C, C++, Fortran, Fortran 90, Pascal or Modula 2 is much simpler. However there are very considerable benefits to the above syntax. Please perservere. The additional complexity is worth it in the longer term.

3.10 Program for numeric i/o

The following is a complete program example of numeric i/o.

```
import java.io.*;
import java.lang.Float;
```

```
class c030301
{
public static void main(String[] args)
   {
      try
      {
         InputStream i=System.in;
         DataInputStream in=new DataInputStream(i);
         String Line;
         float f=1.0f;
         Float F=new Float(f);
         System.out.println(" Type in a number ");
         Line=in.readLine();
         F=Float.valueOf(Line);
         f=F.floatValue();
         System.out.println(f);
      }
      catch(IOException e)
      {  System.out.println(" Exceptions raised: " + e); }
   }
}
```

Let us look at statement in turn.

```
class c030301
```

This is the name of our class. Remember that the file name used at the operating system side must be the same.

```
public static void main(String[] args)
```

This signifies that this is a Java program. This is the standard Java program statement.

```
try
```

We are doing i/o so we must use try and catch to trap errors.

```
InputStream i=System.in;
```

I is a variable or object of type InputStream and it has an inital value of System.in.

```
DataInputStream in=new DataInputStream(i);
```

in is an object of type DataInputStream and it is created using new DataInputStream(i).

```
String Line;
```

Line is of type String.

```
float f=1.0f;
```

f is a numeric variable of type float. It is given an initial value of 1.0 – note the f extension on the initial value. All numeric variables are of type double by default in Java.

```
Float F=new Float(f);
```

F is an object of type Float. There is the concept in Java of fundamental data types and objects. For each fundamental data type there is a corresponding object. so float and Float are two very different concepts. This object is given an inital value by the new Float(f) statement.

```
System.out.println(" Type in a number ");
```

This prints a text prompt on the screen.

```
Line=in.readLine();
```

This reads in the text that the user types into Line. All interaction is done using String objects.

```
F=Float.valueOf(Line);
```

This statement extracts the numeric value from the text typed in. Note that line is of type String and hence is an object and we are extracting an object of type Float from this text string.

```
f=F.floatValue();
```

This converts from a Float object to the build in float data type.

```
System.out.println(f);
```

This statement echos the numeric value back to the user.

```
catch(IOException e)
```

In the event of an i/o error control passes to this statement.

```
{ System.out.println(“ Exceptions raised: ” + e); }
```

In the event of an error the above message appears on the screen.

As you have all programmed using other languages the above appears very long winded. Something we take for granted – numeric i/o – has become seemingly unreasonably complex.

This is the syntax of Java. This is what you have to adapt to if you want to become proficient in Java. Learning a second natural language is difficult at first.

3.11 Some Java Rules and Terminology

Case is significant in Java. Long names using mixed case will almost invariably end up causing compilation errors due to typing mistakes.

Class names and file names must match.

Programs in Java always have the same initial statement

```
public static void main(String[] args)
```

Applets in Java always have the same syntax

```
public class yourname extends java.applet.Applet
```

The file at the operating system level has to be called yourname.

Java keywords are given below.

3.12 Good Programming Guidelines

Every language has its own style. It is advisable to adopt a style that one is comfortable with that draws on ones experience of other languages, and also is similar to the notational style used by Java texts. It is inevitable that one will end up working with algorithms and programs already written in Java, and thus one has to be familiar with the conventional Java style of writing programs.

I prefer to match the { and } at the same indentation level. I find the style adopted in some of the Java books difficult to work with.

I also prefer lowercase.

3.13 Java Character Set

A new standard is becoming increasingly popular where multiple language support is required. This is called Unicode. This is a sixteen bit character set. C++ offers support for 16 bit characters but does not demand suppport of Unicode.

Characters in Java are Unicode based. Given that the web is international you can see why Java uses the Unicode standard. It enables everyone in the world to take advantage of Java using their own character sets. There are two major character sets in computing. These are ASCII (a 7 bit character standard) and ISO-Latin-1 (an 8 bit character standard, commonly called Latin-1).

Look at the differences between

ASCII

DOS character set

Windows character set

DEC character set

Apple character set

ISO-Latin-1

in the additional notes I have provided.

When running a Java applet in a web browser you may not see the characters as they were originally developed as your browser may not be able to support rendering of that character set.

We will look into the whole area of characters, character sets and strings in much greater depth in a later chapter.

3.14 Summary

Don't be put off by the syntax of Java. It doesn't take long to get on top of that syntax. You wouldn't expect natural languages to have identical syntax and semantics, so why expect it from programming languages.

3.15 Bibliography

I've broken down this into three areas. It is essential to get hold of additional sources in the first two areas, and very useful in the third.

3.15.1 Java

As with most programming langauges it is useful to make the following distinctions:–

- language tutorial and examples
- language reference
- language algorithms

The aims of the notes are to provide a brief coverage of the first and second areas. There are little or no sources at this time of algorithms in Java.

Look at the web pages for more details of on-line sources. There are a lot of these. There are a lot of good complete example programs.

3.15.2 HTML

Use the pointers on the college web pages. Most of these are free – you only have to pay for the printing.

3.15.3 Character sets

Have a look in the documentation that comes with the system that you work with. There are also sources on the web – it just takes time to find them and get them printed.

3.16 Problems

1. Type in the examples in this chapter. You will invariably make typing mistakes. Look at the error messages that the compiler give you. The compiler gives error messages from their view point. Make an attempt to understand what this means.

2. What happens with the first example with characters outside of the ASCII character set?

Does the screen representation of these characters match the printed representation? Why do you think that is?

3. With the example what happens to so called *white space*, i.e spaces, tabs, and carriage returns?

4. With the numeric example experiment with the number format, i.e. use integers, reals, exponential format. What happens?

5. Write a program that will read in your name and address and print them out in reverse order.

4
Arithmetic, Expressions and the primitive data types in Java

Taking Three as the subject to reason about —
  A convenient number to state —
We add Seven, and Ten, and then multiply out
  By One Thousand diminished by Eight.
The result we proceed to divide, as you see,
  By Nine Hundred and Ninety and Two:
Then subtract Seventeen, and the answer must be
  Exactly and perfectly true.

Lewis Carroll, The Hunting of the Snark

Round numbers are always false.

Samuel Johnson.

Aims

The aims of this chapter are to introduce:–

- the numeric data types available in Java;
- the integer numeric model used in Java;
- the floating point numeric model used within Java – the IEEE 754-1985 standard
- the rules for the evaluation of arithmetic expressions;
- the rules that apply in type conversion;
- constants – static final;
- char data type;
- boolean data type;
- to introduce briefly all of the operators in Java;

4 Arithmetic and Expressions in Java

This chapter looks at the fundamental numeric data types in Java and the rules for expression evaluation. There are a large number of operators and a quick look at them all is necessary. However for most applications you will only require a good knowledge of a small subset of them.

4.1 Basic numeric types

Java supports the two numeric data types we are familiar with from other programming languages, i.e. integer and real.

4.2 Integer Numeric Type

The standard requires four types of integers, *byte, short , int* and *long*, and these correspond to:–

- byte – 8 bit signed two's complement integer; -128 through +127
- short – 16 bit signed two's complement integer; -32768 through +32767
- int – 32 bit signed two's complement integer
- long – 64 bit signed two's complement integer

They obey the laws of arithmetic modulo 2^n, where n is the number of bits in the implementation. Integer arithmetic never overflows or underflows – it wraps. Add 1 to a byte integer of value 127 and it becomes -128. This is not an error.

Try running the following program on the system you use. What do you think will happen?

```
class c0401
{
   public static void main(String[] args)
   {
      byte i_b = (byte)1;
      short i_s = (short)1;
      int i= 1;
      long l_l = 1;
      System.out.print(i_b);System.out.print("  ");
      System.out.print(i_s);System.out.print("  ");
      System.out.print(i);System.out.print("  ");
      System.out.print(i_l);System.out.println();
      for (int count=1;count<33;++count)
      {
         i_b = (byte)(i_b * 2)
         i_s = (short)(i_s * 2)
         i = i * 2
         i_l = i_l * 2
         System.out.print(i_b);System.out.print("  ");
         System.out.print(i_s);System.out.print("  ");
         System.out.print(i);System.out.print("  ");
         System.out.print(i_l);System.out.println();
      }
   }
}
```

Let us look at this program in some detail.

```
class c0401
```

Class name.

```
public static void main(String[] args)
```

Standard program header.

```
byte i_b = 1;
```

Variable declaration and initialisation.

```
short i_s = 1;
```

Variable declaration and initialisation.

```
int i= 1;
```

Variable declaration and initialisation.

```
long l_l = 1;
```

Variable declaration and initialisation.

```
System.out.print(i_b);System.out.print("  ");
```

Standard mechanism to print to the screen. The print method can only take one argument. There are several print methods – one for each built in type. This makes i/o seem a little verbose compared to more conventional programming languages.

```
System.out.print(i_s);System.out.print("  ");
```

Print short.

```
System.out.print(i);System.out.print("  ");
```

Print int.

```
System.out.print(i_l);System.out.println();
```

Print long.

```
for (int count=1;count<33;++count)
```

Java and C++ syntax of a for loop. We are allowed to declare and initialise variables with a for statement. In this case we introduce a new variable count and initialise to 1. The ; is a statment terminator. count < 33 means repeat the loop whilst count is less than 33. ++count means increment count by 1 each time round the loop.

```
i_b = (byte)(i_b * 2)
```

In this example we are forced to cast the expression i_b*2 from its default type of int back to byte. This is because an integer literal (a number) has a default type of int. Hence the 2 forces the expression i_b*2 to be promoted to be of type int. This is the Java syntax for casting.

```
i_s = (short)(i_s * 2)
```

In this example we are forced to cast the expression i_s*2 from int to short.

```
i = i * 2
```

Familiar arithmetic expression and assignment.

```
i_l = i_l * 2
```

Familiar arithmetic expression and assignment. In this case 2 is promoted to type long by Java automatically.

```
System.out.print(i_b);System.out.print("  ");
System.out.print(i_s);System.out.print("  ");
System.out.print(i);System.out.print("  ");
System.out.print(i_l);System.out.println();
```

Print out the values.

The for loop will terminate when count reaches 32.

4.3 Real Numeric Type

Java uses the IEEE 754-1985 standard as a basis for floating point data types and arithemtic. Two types are required to be supported and these are *float* and *double*. Within this standard there are the concepts of:–

- overflow to infinity
- underflow to zero
- NAN or Not a Number, for invalid expressions

For floating point calculations with F meaning finite number we have the following:–

x	y	x/y	x%y
F	0.0		NaN
F		0.0	x
0.0	0.0	NaN	NaN
	F		NaN
		NaN	NaN

Floating point division and remainder can produce both NaN and infinities without raising an exception.

Some of the major differences between IEEE 754-1985 and the Java variant are:–

- nonstop arithmetic – Java will not signal the IEEE conditions of invalid operation, division by zero overflow, underflow or inexact;
- extended formats – these are optional;
- rounding – Java rounds towards the nearest, which is the IEEE default, but it does not offer user selectable rounding;

Java is in a state of flux in this area, and this is covered in more depth in chapter 25.

The following program provides an example of the use of each real data type supported in Java.

```
class c0402
{
   public static void main(String[] args)
   {
      float f = 1.1f;
      double f_d = 1.1;
      System.out.println(f);
      System.out.println(f_d);
   }
}
```

Let us look at each statement in turn.

```
class c0402
```

Every program has to be a class.

```
public static void main(String[] args)
```

Standard program statement.

```
float f = 1.1f;
```

There are two kinds of real numbers in Java. One is float. The variable f is of type float and has an initial value of 1.1. Note that this real literal has f appended. Real constants are double by default. You have to use a cast or append with an f. We could have also written:–

```
float f = (float)1.1;
```

To convert 1.1 from double to float.

```
double f_d = 1.1;
```

This is the second kind of real number in Java. So f_d is a variable of type double and has an initail value of 1.1

```
System.out.println(f);
```

Print out the float variable.

```
System.out.println(f_d);
```

Print out the double variable.

4.4 IEEE 754-1985

The standard defines two basic real types – single and double, and two extended types – single extended and double extended. Java only supports the basic types. This ensures 24 bit precision with single and 53 bit precision within double. This means 6/7 digits and 10**38 for single and 15/16 digits and 10**308 for double.

The idea behind the standard is that you can run your programs on any IEEE compliant system and get the same numeric results.

4.5 Numeric Type Conversion

With expressions of mixed type the variable of the strongest type will cause promotion of all others. The sequence is byte -> short -> int -> long -> float -> double. Explicit casts can always be used.

4.6 Whither complex?

This is not provided in the language.

4.7 Constants or Parameters

A constant or parameter is defined in Java using the static and final attributes. Consider the following example:–

```
static final double pi=3.14159265358
```

A complete example is given later.

4.8 Operators and Expression Evaluation

Java has a large number of operators, A working knowledge of the more commonly used ones is essential for successful use of Java. You should also be aware of the rest.

4.8.1 Expression Evaluation

Expressions are evaluated left to right in Java when involving operators of a similar precedence. Brackets can be used to alter the order of evaluation. The following example illustrates this.

```
class c0403
{
  public static void main(String[] args)
  {
```

```
    int i=2+3*4;
    System.out.println(i);
  }
}
```

The program prints out 14.

4.8.2 Operators, Precedence and Associativity.

The following table summarises the rules concerning precedence and associativity. All operators associate left to right except for those in the third and eighteenth position in the precedence hierarchy, i.e. the unary and assignment operators.

Operator Summary

.	member selection	object.member
[]	subscripting	[expr]
()	function call	expr (expr_list)
++	post increment	expr ++
—	post decrement	expr —
++	pre increment	++ expr
—	pre decrement	— expr
~	complement	~ expr
!	not	! expr
-	unary minus	- expr
+	unary plus	+ expr
new	new	new object
()	cast	(type) expr
*	multiply	expr * expr
/	divide	expr / expt
%	modulo or remainder	expr % expr
+	plus	expr + expr
-	minus	expr - expr
<<	shift left	expr << expr
>>	shift right, sign extend	expr >> expr
>>>	shift right, zero fill	expr >>> expr
<	less than	expr < expr
<=	less than or equal	expr <= expr
>	greater than	expr > expr
>=	greater than or equal	expr >= expr
instanceof	instanceof	object instanceof object
==	equal	expr == expr
!=	not equal	expr != expr
&	bitwise AND	expr & expr
^	bitwise exclusive OR	expr ^ expr
|	bitwise inclusive OR	expr | expr

&&	logical AND	expr && expr
||	logical OR	expr || expr
?:	conditional expression	expr ? expr : expr
=	conventional assignment	lvalue = expr
*=	multiply and assign	lvalue *= expr
/=	divide and assign	lvalue /= expr
%=	modulo and assign	expr %= expr
+=	add and assign	expr += expr
-=	subtract and assign	expr -= expr
<<=	shift left and assign	expr <<= expr
>>=	shift right and assign	expr >>= expr
>>>=	shift right and assign	expr >>>= expr
&=	AND and assign	expr &= expr
|=	inclusive OR and assign	expr |= expr
^=	exclusive OR and assign	expr ^= expr

There will be more complete examples of each of the following in later chapters.

4.8.2.1 . [member selection] object.member

This operator allows us to select a member of a class.

4.8.2.2 [] [subscripting] pointer [expr]

The normal array subscripting operator.

4.8.2.3 () [function call] expr (expr_list)

The function call operator.

4.8.2.4 ++ [post increment] expr ++

Increment after use.

4.8.2.5 — [post decrement] expr —

Decrement after use.

4.8.2.6 ++ [pre increment] ++ expr

Increment before use.

4.8.2.7 — [pre decrement] — expr

Decrement before use.

4.8.2.8 ~ [complement] ~ expr

One’s complement operator. The operand must be of integral type. Integral promotions are performed.

4.8.2.9 ! [not] ! expr

Logical negation operator.

4.8.2.10 - [unary minus] - expr

As stated.

4.8.2.11 + [unary plus] + expr

As stated.

4.8.2.12 new [create] new type

The new operator attempts to create an object of the type to which it is applied. This type must be an object type, and functions cannot be allocated in this way, though pointers to functions can.

4.8.2.13 () [cast] (type) expr

An explicit type conversion.

4.8.2.14 * [multiply] expr * expr

Conventional arithmetic multiplication.

4.8.2.15 / [divide] expr / expt

Conventional arithmetic division.

4.8.2.16 % [modulo or remainder] expr % expr

Remainder.

4.8.2.17 + [plus] expr + expr

Conventional arithmetic addition.

4.8.2.18 - [minus] expr - expr

Conventional arithmetic subtraction.

4.8.2.19 << [shift left] expr << expr

Shift left. The operands must be of integral type and integral promotions are performed.

4.8.2.20 >> [shift right] expr >> expr

Shift right. Sign extend. The operands must be of integral type and integral promotions are performed.

4.8.2.21 >>> [sihift right] expr >>> expr

Shift right. Zero fill.

4.8.2.22 < [less than] expr < expr

Conventional relational operator.

4.8.2.23 <= [less than or equal] expr <= expr

Conventional relational operator.

4.8.2.24 > [greater than] expr > expr

Conventional relational operator.

4.8.2.25 >= [greater than or equal] expr >= expr

Conventional relational operator.

4.8.2.26 == [equal] expr == expr

Conventional relational operator.

4.8.2.27 != [not equal] expr != expr

Conventional relational operator.

4.8.2.28 & [bitwise AND] expr & expr

The usual arithemic conversions are performed: the result is the bitwise and function of the operands. The operator applies only to integral operands. If both bits are set the result is 1, otherwise 0.

4.8.2.29 ^ [bitwise exclusive OR] expr ^ expr

The usual arithemic conversions are performed: the result is the bitwise exclusive or function of the operands. The operator applies only to integral operands. If either but not both bits are set the result is 1, otherwise 0.

4.8.2.30 | [bitwise inclusive OR] expr | expr

The usual arithemic conversions are performed: the result is the bitwise inclusive or function of the operands. The operator applies only to integral operands. If either bit is set the result is set, otherwise 0.

4.8.2.31 && [logical AND] expr && expr

The operands must be boolean.

Left to right evaluation is guaranteed, and the second operand is not evaluated if the first is false.

4.8.2.32 || [logical inclusive OR] expr || expr

The operands must be boolean. The result is true if either of its operands is true and false otherwise.

Left to right evaluation is guaranteed, and the second operand is not evaluated if the first is true.

4.8.2.33 ?: [conditional expression] expr ? expr : expr

The first expression is converted to bool. It is evaluated and if it is true the result of the conditional expression is the value of the second expression, otherwise that of the third.

All side effects of the first expression except for destruction of temporaries happen before the second or third expression is evaluated.

4.8.2.34 = [conventional assignment] expr = expr

Conventional assignment.

4.8.2.35 *= [multiply and assign] expr *= expr

Multiply and assign, e.g. a=a*expression

4.8.2.36 /= [divide and assign] expr /= expr

Divide and assign, e.g. a=a/expression

4.8.2.37 %= [modulo and assign] expr %= expr

Modulo and assign, e.g. a=a%expression

4.8.2.38 += [add and assign] expr += expr

Add and assign, e.g. a=a+expression

4.8.2.39 -= [subtract and assign] expr -= expr

Subtract and assign, e.g. a=a-expression

4.8.2.40 <<= [shift left and assign] expr <<= expr

Shift left and assign

4.8.2.41 >>= [shift right and assign] expr >>= expr

Shift right and assign

4.8.2.42 &= [AND and assign] expr &= expr

AND and assign

4.8.2.43 |= [inclusive OR and assign] expr |= expr

inclusive OR and assign

4.8.2.44 ^= [exclusive OR and assign] expr ^= expr

exclusive OR and assign

4.9 Expression Examples

It is not appropriate here to cover each and everyone of the above in great depth. We will introduce examples throughout the notes as we progress.

Note that there is no exponentiation operator.

The following programs and program extracts cover some of the above.

```
class c0404
{
   public static void main(String[] args)
   {
      int i=0;
      int j=0;
      System.out.println(++i);
      System.out.println(j++);
   }
}
```

The above example highlights the use of the pre and post increment and decrement operators. Type the above in and run it.

```
class c0405
{
   public static void main(String[] args)
   {
      int i=9;
      int j=2;
      int k=-2;
      System.out.println(i/j);
      System.out.println(i/k);
      System.out.println(i%j);
      System.out.println(i%k);
   }
}
```

Type this example in and run it. Java defines / and % to obey the following:–

```
(x/y)*y + x%y = x
```

Consider the following example:–

```
class c0406
{
   public static void main(String[] args)
   {
      int i=0;
      int j=0;
      int k=0;
      int l=0;
      i=i+1;
      j+=1;
      ++k;
      l++;
```

```
    System.out.println(i);
    System.out.println(j);
    System.out.println(k);
    System.out.println(l);
  }
}
```

What do you think will be the output of this program? Do we need four ways of achieving the same thing in a programming language?

This example is taken from the Fortran 90 text. It is a direct translation.

```
class c0407
{
  static final float light_year = (float)(9.46*10E12) ;

  public static void main(String[] args)

  {
    float light_minute;
    float distance;
    float elapse ;
    int minute ;
    int second ;
    light_minute = light_year/(float)(365.25 * 24.0 * 60.0);
    distance = (float)(150.0 * 10E6) ;
    elapse = distance / light_minute ;
    minute = (int)elapse ;
    second = (int)(( elapse - minute ) * (float)60) ;
    System.out.print(“ Light takes ”);
    System.out.print(minute);
    System.out.print(“ minutes”);
    System.out.print(“ ”);
    System.out.print(second);
    System.out.println(“ seconds”);
  }
}
```

Let us look at each line in turn.

```
class c0407
```

Standard class definiton.

```
static final float light_year = (float)(9.46*10E12) ;
```

This is the way that you define light_year to be a constant or parameter. The two additional keywords are static and final.

The initial value looks a little strange. Remember that real numbers are double by default in Java. We thus have to cast the expression (9.46*10E12) to a float. Note that we have to use brackets around the numeric expression as we are interested in casting the result of the whole expression.

```
public static void main(String[] args)
```

Standard start of program.

```
float light_minute;
```

```
float distance;
float elapse ;
int minute ;
int second ;
```

Standard variable declarations.

```
light_minute = light_year / (float)( 365.25 * 24.0 * 60.0 );
```

Assign value to light_minute. Part of the expression looks a little odd at first. We are doing the complete expression as double and then casting the result to float.

```
     distance = (float)(150.0 * 10E6) ;
```

Calculate distance – again cast from double to float.

```
     elapse = distance / light_minute ;
```

Simple assignment.

```
     minute = (int)elapse ;
```

Assignment with cast.

```
     second = (int)(( elapse - minute ) * (float)60) ;
```

Assignment with cast. Notice the additional brackets to force the cast of the whole expression.

```
     System.out.print(“ Light takes ”);
     System.out.print(minute);
     System.out.print(“ minutes”);
     System.out.print(“ ”);
     System.out.print(second);
     System.out.println(“ seconds”);
```

Print out the answer.

The example looks a little unpleasant do to the requirment to cast from the various default types. Let us now look at a slight variant using double throughout.

```
class c0408
{
  static final double light_year = 9.46*10E12 ;

  public static void main(String[] args)

  {
     double light_minute ,  distance , elapse ;
     int minute , second ;
     light_minute = light_year / ( 365.25 * 24.0 * 60.0 ) ;
     distance = 150.0 * 10E6 ;
     elapse = distance / light_minute ;
     minute = (int)elapse ;
     second = (int)(( elapse - minute ) * 60) ;
     System.out.print(“ Light takes ”);
     System.out.print(minute);
     System.out.print(“ minutes”);
     System.out.print(“ ”);
     System.out.print(second);
     System.out.println(“ seconds”);
```

```
  }
}
```

Better – now look at the last variant.

```
class c0409
{
  static final float light_year = 9.46f*10E12f ;

  public static void main(String[] args)

  {
    float light_minute;
    float distance;
    float elapse ;
    int minute ;
    int second ;
    light_minute = light_year/(365.25f * 24.0f * 60.0f)  ;
    distance = 150.0f * 10E6f ;
    elapse = distance / light_minute ;
    minute = (int)elapse ;
    second = (int)(( elapse - minute ) * 60f) ;
    System.out.print(“ Light takes ”);
    System.out.print(minute);
    System.out.print(“ minutes”);
    System.out.print(“ ”);
    System.out.print(second);
    System.out.println(“ seconds”);
  }
}
```

So the recommendation would seem to be to do all real arithmetic as double, rather than have to cast all the time or append all variables with f. If your program is heavy on numeric computation would you be using Java anyway?

4.10 Char

Characters are Unicode based in Java. Given the large number of natural languages in the world and the requirements of the publishing world several companies got togthor and formed Unicode Inc, a non-profit making consortium to draw up a standard for international character sets. ISO was tackling the same probelm and the outcome was the Unicode Standard.

As Unicode is 16 bit based it can handle 65,536 distinct characters, which is enough for most if not all languages in use today plus a number of older or arcane languages, e.g. Egyptian hieroglyphs.

Note that not all computer systems can necessarily handle the display of the full Unicode character set.

The bibliography contains some pointers to sources of information regarding Unicode. I’ve also printed out the following:–

- ASCII character set – 7 bit
- ISO Latin 1 – 8 bit

- DOS character set – code pages 437 and 850
- Windows character set
- Apple character set
- Unicode

Now you have some idea why the interchange of computer information from one system to another can be fraught with problems.

4.11 Boolean

This is the last primitive data type supported in Java. Values are eithor true or false.

4.12 Example Programs

Here are a small number of complete Java programs illustrating some of the material in this chapter.

4.12.1 Example Program – Simple character and boolean output

```
class c0410
{
  public static void main(String[] args)
  {
    boolean ok=true;
    char c='a';
    System.out.println(c);
    System.out.println(ok);
  }
}
```

4.12.2 Example Program – Unicode character output

```
class c0411
{
  public static void main(String[] args)
  {
    char c='\u0b87';
    System.out.println(c);
  }
}
```

4.12.3 Example Program – Bitwise operators &, ^ and |

```
class c0412
{
  public static void main(String[] args)
  {
    int i=10;
    int j=11;
    int k,l,m;
    k= i & j;
    System.out.println(k);
    l= i ^ j;
    System.out.println(l);
    m= i | j;
    System.out.println(m);
```

```
   }
}
```

4.13 Summary

There are the following primitive data types in Java:–

- boolean – true or false
- char – 16 bit Unicode character
- byte – 8 bit signed integer
- short – 16 bit signed integer
- int – 32 bit signed integer
- long – 64 bit signed integer
- float – 32 bit real: IEEE 754-1985
- double – 64 bit real: IEEE 754-1985

Integer constants in expressions are of type int. Casting is required when doing integer arithmetic using the other integer types.

Real constants are of type double. Casting is required when doing real arithmetic using float types.

Java draws heavily on C and C++ for its operators and syntax.. If you have some background in these languages then this obviously helps. However there are no pointers in Java, or explicit delete mechanisms to free memory. Java has a garbage collector.

Strings and arrays are not fundamental data types, they are objects and as such are covered in separate chapters.

Constants are provided using the static final attributes.

There is a lot of material in this chapter. Don't panic. As the course progresses more and more of it will stick.

4.14 Package java.lang

This package provides classes that are fundamental to Java. They do not need to be imported. They are automatically imported. They have been covered here as I couldn't think of where else they should go!

4.14.1 Interface Summary

4.14.1.1 Cloneable

A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.

4.14.1.2 Comparable

This interface imposes a total ordering on the objects of each class that implements it.

4.14.1.3 Runnable

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.

4.14.2 Class Summary

4.14.2.1 Boolean

The Boolean class wraps a value of the primitive type boolean in an object.

4.14.2.2 Byte

The Byte class is the standard wrapper for byte values.

4.14.2.3 Character

The Character class wraps a value of the primitive type char in an object.

4.14.2.4 Character.Subset

Instances of this class represent particular subsets of the Unicode character set.

4.14.2.5 Character.UnicodeBlock

A family of character subsets representing the character blocks defined by the Unicode 2.0 specification.

4.14.2.6 Class

Instances of the class Class represent classes and interfaces in a running Java application.

4.14.2.7 ClassLoader

The class ClassLoader is an abstract class.

4.14.2.8 Compiler

The Compiler class is provided to support Java-to-native-code compilers and related services.

4.14.2.9 Double

The Double class wraps a value of the primitive type double in an object.

4.14.2.10 Float

The Float class wraps a value of primitive type float in an object.

4.14.2.11 InheritableThreadLocal

This class extends ThreadLocal to provide inheritance of values from parent Thread to child Thread: when a child thread is created, the child receives initial values for all InheritableThreadLocals for which the parent has values.

4.14.2.12 Integer

The Integer class wraps a value of the primitive type int in an object.

4.14.2.13 Long

The Long class wraps a value of the primitive type long in an object.

4.14.2.14 Math

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Fields

- static double E The double value that is closer than any other to e, the base of the natural logarithms.
- static double PI The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.

Methods

- static double abs(double a) Returns the absolute value of a double value.
- static float abs(float a) Returns the absolute value of a float value.
- static int abs(int a) Returns the absolute value of an int value.
- static long abs(long a) Returns the absolute value of a long value.

- static double acos(double a) Returns the arc cosine of an angle, in the range of 0.0 through pi.
- static double asin(double a) Returns the arc sine of an angle, in the range of -pi/2 through pi/2.
- static double atan(double a) Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.
- static double atan2(double a, double b) Converts rectangular coordinates (b, a) to polar (r, theta).
- static double ceil(double a) Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.
- static double cos(double a) Returns the trigonometric cosine of an angle.
- static double exp(double a) Returns the exponential number e (i.e., 2.718...) raised to the power of a double value.
- static double floor(double a) Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.
- static double IEEEremainder(double f1, double f2) Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
- static double log(double a) Returns the natural logarithm (base e) of a double value.
- static double max(double a, double b) Returns the greater of two double values.
- static float max(float a, float b) Returns the greater of two float values.
- static int max(int a, int b) Returns the greater of two int values.
- static long max(long a, long b) Returns the greater of two long values.
- static double min(double a, double b) Returns the smaller of two double values.
- static float min(float a, float b) Returns the smaller of two float values.
- static int min(int a, int b) Returns the smaller of two int values.
- static long min(long a, long b) Returns the smaller of two long values.
- static double pow(double a, double b) Returns of value of the first argument raised to the power of the second argument.
- static double random() Returns a random number greater than or equal to 0.0 and less than 1.0.
- static double rint(double a) returns the closest integer to the argument.
- static long round(double a) Returns the closest long to the argument.
- static int round(float a) Returns the closest int to the argument.
- static double sin(double a) Returns the trigonometric sine of an angle.
- static double sqrt(double a) Returns the square root of a double value.
- static double tan(double a) Returns the trigonometric tangent of an angle.

- static double toDegrees(double angrad) Converts an angle measured in radians to the equivalent angle measured in degrees.
- static double toRadians(double angdeg) Converts an angle measured in degrees to the equivalent angle measured in radians.

4.14.2.15 Number

The abstract class Number is the superclass of classes Byte, Double, Float, Integer, Long, and Short.

4.14.2.16 Object

Class Object is the root of the class hierarchy.

4.14.2.17 Package

Package objects contain version information about the implementation and specification of a Java package.

4.14.2.18 Process

The Runtime.exec methods create a native process and return an instance of a subclass of Process that can be used to control the process and obtain information about it.

4.14.2.19 Runtime

Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running.

4.14.2.20 RuntimePermission

This class is for runtime permissions.

4.14.2.21 SecurityManager

The security manager is a class that allows applications to implement a security policy.

4.14.2.22 Short

The Short class is the standard wrapper for short values.

4.14.2.23 String

The String class represents character strings.

4.14.2.24 StringBuffer

A string buffer implements a mutable sequence of characters.

4.14.2.25 System

The System class contains several useful class fields and methods.

4.14.2.26 Thread

A thread is a thread of execution in a program.

4.14.2.27 ThreadGroup

A thread group represents a set of threads.

4.14.2.28 ThreadLocal

This class provides ThreadLocal variables.

4.14.2.29 Throwable

The Throwable class is the superclass of all errors and exceptions in the Java language.

4.14.2.30 Void

The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the primitive Java type void.

You must look at the on-line documentation for more details of what are in these classes.

4.15 Bibliography

IEEE 754-1985

The standard is the definitive statement. Other sources include Suns Numerical Computation Guide. Similar publications will exist for other platforms.

What Every Computer Scientist Should know About Floating Point Arithmetic

This paper first appeared in the March 1991 issue of Computing Surveys, ACM Inc. It can also be found (after some rumaging around) on Sun systems. I recommend the paper very highly.

4.16 Problems

1. Try typing in and running the examples given in this chapter. Remember that you need to gain familiarity with the Java rules. You need to make mistakes and see what goes wrong.

2. Write a program that will read in your name and address and porint them out. Now modify the program to read in your age also. Print out your name, age and address.

3. One of the easiest ways to write a program is to modify an existing one. The example given earlier, dealing with the time taken for light to travel from the sun to the earth could form the basis of several other programs.

- many communications satellites follow a geosynchronous orbit, some 35,870 Km above the earth's surface. What is the time lag in using such a satellite for a telephone conversation?
- the moon is about 384,400 Km from the earth (on average). what implications does this have for control experiments on the moon? what is the time lag?
- the following table gives the distance in MKm from the sun to the planets in the solar system:

mercury	57.9	venus	108.2
earth	149.6	mars	227.9
jupiter	778.3	saturn	1427.0
uranus	2869.6	neptune	4496.6
pluto	5900.0		

Use this information to find the greatest and least time taken to send a message from the earth to the other planets. Assume that all orbits are in the same plane and circular (if it was good enough for copernicus, its good enough for this example). For all practical purposes the speed of light in a vacuum is a constant, and therefore a good candidate for a const statement. Use it.

4. Write a program to calculate the period of a pendulum. Use the following formula:–

```
t = 2 * pi * sqrt(length/9.81)
```

The length is in metres and the time is in seconds. Use a length of 10.0 metres. Use single precision throughout.

PI is built into Java. Where would you expect it to be? So is square root.

5. Repeat the above, using double precision throughout.

6. Base conversion. The following is a complete program that looks at base conversion. What output do you expect?


```
class c0422
{
  public static void main(String[] args)
  {
    float x1,x2,x3,x4,x5;
    x1=(float)1.0;
    x2=(float)0.1;
    x3=(float)0.01;
    x4=(float)0.001;
    x5=(float)0.0001;
    System.out.println(x1);
    System.out.println(x2);
    System.out.println(x3);
    System.out.println(x4);
    System.out.println(x5);
  }
}
```

7. Rewrite the above program in a language that you already know. Do the results agree.

8. Modify the base conversion program to use double precision. What do expect the ansers to be?

9. Expression equivalence. In mathematics the following expressions are equivalent.

$x^2 - y^2 = (x-y)*(x+y) = x*x - y*y$

Write a Java program to evaluate these three expressions with x=1.002 and y=1.001 There is no exponentiation operator in Java, you have to use the pow function.

Do the three expressions give the same results?

Why do you think that is?

10. Rewrite the above in a another programming language. Do the results agree?

What about Excel?

11. If you have a pc with Java installed repeat the above examples on that system.

5
Strings

‘Don’t Panic’

Douglas Adams, The Hitch Hiler’s Guide to the Galaxy

Aims

The aims are:–

to introduce the String object;

to look at some of the implications of Strings being objects rather than a primitive data type;

to look at the functions provided within Java for the manipulation of Strings;

5 Strings

This chapter looks at strings in Java. Where characters are a built in primitive type strings are fully blown objects. This means that there are some very important differences between the way we think of strings in older programming languages and Java.

5.1 The basics

There are two kinds of string objects in Java. They are objects of one of the following two classes:–

- java.lang.String
- java.lang.StringBuffer

The first is read only. The second can be modified. Let us look at the String class and associated methods first.

5.2 java.lang.String

Whilst Strings are objects they can be created in three ways:–

- by enclosing the text in “ marks.
- by using + and += on two strings to create a new string
- explcitly using new – as with objects

The following program illustrates all three.

```
class c0501
{
  public static void main(String[] args)
  {
    int i1,i2,i3,i4;
    String s1=" This is a string";
    String s2=" and this is another";
    String s3;
    String s4=new String();
    i1=s1.length();
    i2=s2.length();
    i4=s4.length();
    System.out.println(i1);
    System.out.println(i2);
    System.out.println(i4);
    s3=s1+s2;
    i3=s3.length();
    System.out.println(i3);
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
  }
}
```

Let us look at each line in turn.

```
String s1=" This is a string";
String s2=" and this is another";
```

It is important to distinguish between declaration and creation. With our knowledge of convential programming languages we tend not to think very much about this. With object oriented programming we have to think about it all the time. These two statements declare s1 and s2 to be of type String and they also create s1 and s2 as String objects and give them initial values. There is no explicit new here.

```
String s3;
```

s3 is declared to be an object of type String.

```
String s4=new String();
```

s4 is declared to be an object of type String and we have an explicit use of new here to create the object s4.

```
s3=s1+s2;
```

s3 is created (again no explicit new) and is given an initial value of s1 concatenated with s2. We then print out each string.

```
i1=s1.length();
i2=s2.length();
i4=s4.length();
```

These statements highlights the way in which object oriented programming differs from conventional programming.

We have a binding of an object with an action, i.e. a String object with a length determination.

5.2.1 String Methods

Here are some of the more common String methods. They are organised into three categories.

Constructors	public String()
	public String(String value)
	public String(char[] value)
	public String(StringBuffer buffer)
Class Methods	public static String copyValueOf(char[] data)
	public static String valueOf(Object data)
	public static String valueOf(boolean b)
	public static String valueOf(char c)
	public static String valueOf(int i)
	public static String valueOf(long l)
	public static String valueOf(float f)
	public static String valueOf(double d)
Public Instance Methods	public char charAt(int index)
	public int compareTo(String str)
	public boolean equals(Object o)
	public boolean equalsIgnoreCase(String str)
	public int indexOf(int ch)
	public int indexOf(String str)
	public int lastindexOf(int ch)

public int lastIndexOf(String str)
public int length()
public String replace(char old,char new)
public String substring(int i)
public String toLowerCase()
public String toString();
public String toUpperCase()
public String trim()

The first set of methods are constructors that enable us to create objects of String types. The second set of methods are class methods. A static method doesn't act on specific instances of a class. Remember that with object oriented programming we have to bring objects into existance - so called instantiation. Finally we have instance methods – these actually act or work with String objects.

Note that we have several functions with the same name, but different signatures. This is called overloading. It enables us to provide one name for what we want carried out, and is a good example of abstraction. The fact that there are several functions with the same name doesn't matter.

Note also that we have to have methods that convert between the base types and objects and vice versa.

5.2.1.1 String Example 1 – replace

```
class c0503
{
  public static void main(String[] args)
  {
    String s1="ababcadaeafa";
    System.out.println(s1);
    System.out.println(s1.replace('a','e'));
    System.out.println(s1);
  }
}
```

The original String s1 is untouched.

5.2.1.2 String Example 2 - valueOf

```
import java.lang.*;

class c0504
{
  public static void main(String[] args)
  {
    String s0,s1,s2,s3,s4,s5;
    char c='1';
    int i=1;
    long l=1;
    float f=1;
    double d=1;
    s1=String.valueOf(c);
    s2=String.valueOf(i);
```

```
    s3=String.valueOf(l);
    s4=String.valueOf(f);
    s5=String.valueOf(d);
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
    System.out.println(s5);
    s0=s1+s2+s3+s4+s5;
    System.out.println(s0);
  }
}
```

The first line is very important. This makes available all of the methods within java.lang to the following program. We can then use shortened forms of the method names within the following program. The String class is within java.lang.

This example highlights the differences between the base types and String objects. In each case we use the value 1. It has a different meaning in each of the contexts.

5.2.1.3 String Example 3 – as above but no import statement

```
class c0505
{
  public static void main(String[] args)
  {
    String s0,s1,s2,s3,s4,s5;
    char c='1';
    int i=1;
    long l=1;
    float f=1;
    double d=1;
    s1=java.lang.String.valueOf(c);
    s2=java.lang.String.valueOf(i);
    s3=java.lang.String.valueOf(l);
    s4=java.lang.String.valueOf(f);
    s5=java.lang.String.valueOf(d);
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
    System.out.println(s5);
    s0=s1+s2+s3+s4+s5;
    System.out.println(s0);
  }
}
```

Note that in this example we have not used the import statement and have had to fully state the method name, i.e. java.lang.String.valueof().

5.3 java.lang.StringBuffer

This class also represents a string of characters, but now the contents may be modified. This means that:–

- it can grow in length
- characters can be replaced
- characters can be appended
- characters can be inserted

5.3.1 StringBuffer Methods

Here are some of the more commonly used StringBufer methods.

Constructors	StringBuffer()
	StringBuffer(int length)
	StringBuffer(String str)
Instance Methods	public synchronized StringBuffer append(Object obj)
	public synchronized StringBuffer append(String str)
	public synchronized StringBuffer append(char[] str)
	public StringBuffer append(boolean b)
	public synchronized StringBuffer append(char c)
	public StringBuffer append(int i)
	public StringBuffer append(long l)
	public StringBuffer append(float f)
	public StringBuffer append(double d)
	public synchronized StringBuffer insert(int ,Object obj) throws StringIndexOutOfBoundsException
	public synchronized StringBuffer insert(int ,String str) throws StringIndexOutOfBoundsException
	public synchronized StringBuffer insert(int ,char[] str) throws StringIndexOutOfBoundsException
	public StringBuffer insert(int ,boolean b) throws StringIndexOutOfBoundsException
	public synchronized StringBuffer insert(int ,char c) throws StringIndexOutOfBoundsException
	public StringBuffer insert(int ,int i) throws StringIndexOutOfBoundsException
	public StringBuffer insert(int ,long l) throws StringIndexOutOfBoundsException
	public StringBuffer insert(int ,float f) throws StringIndexOutOfBoundsException
	public StringBuffer insert(int ,double d) throws StringIndexOutOfBoundsException
	public int length()
	synchronized void set CharAt(int index,char ch) throws exception StringIndexOutOfBoundsException
	synchronized void setLength(innt newlength) throws exception StringIndexOutOfBoundsException

String toString()

Note again that we have many functions with the same name and different signatures.

There are two new concepts raised here. The first is the concept of synchronised and the second of throwing an exception. Let us look at each in turn.

Something that you may never have thought much about is multiprocessing. With most computer systems there will be several processes running. Type

ps -ef

the next time you are logged onto a unix system to see what proceeses are running.

If you have ever used a pc or a mac you will probably have logged into a computer system whilst at the same time using netscape or doing some printing. Again there are several processes running.

Java supports the concepts of threads. This means that you as the writer of a Java program can create a program that has several threads running. This is useful when retrieving images from another site, whilst at the same time still interacting with the user.

It will therefore be necessary to make certain methods run in a way where they have sole access to an object. You don't want several threads modifying the same StringBuffer object for example.

So we declare a method synchronized to force locking of any objects that they have access to – in reality objects that they change.

The second concept is that of error handling. Most of you will be familiar with programs you have written that terminate when you type in unexpected values. Situations like this are handled in Java by the ability to *throw an exception*. The error is then handled by an exception handler. In the example above insert methods may make the StringBuffer object too large. In this case an exception is raised and control will pass to the error handler.

We will look at a couple of example programs to clarify the above.

5.3.1.1 StringBuffer Example 1 – throwing an exception and catching

```
import java.lang.*;

class c0507
{
  public static void main(String[] args)
  {
    StringBuffer s1=new StringBuffer(“ this is the base
string”);
    StringBuffer s2=new StringBuffer(“ insert this”);
    System.out.println(s1);
    System.out.println(s2);
    try
    {
      s1.insert(1,s2);
      System.out.println(s1);
      s1.insert(10,s2);
      System.out.println(s1);
      s1.insert(20,s2);
      System.out.println(s1);
      s1.insert(99999,s2);
      System.out.println(s2);
```

```
    }
     catch (StringIndexOutOfBoundsException e)
     {
       System.out.println(" String too large");
     }
   }
}
```

Let us look in more details at some of the statements in this program.

```
StringBuffer s1=new StringBuffer(" this is the base string");
StringBuffer s2=new StringBuffer(" insert this");
```

These two statements define and create the two StringBuffer objects with initial values. There are only three constructors defined for the StringBuffer class compared to seven for the String class.

The creation of String and String Buffer objects within Java is quite different, e.g.

Creation Mechanism	**String**	**StringBuffer**
" Text in quotes "	Yes	No
+ and +=	Yes	No
explicit new	Yes	Yes

This means that code that we write for handling objects of these types is syntactically quite different.

We next print out the two StringBuffer objects.

```
try
{
   s1.insert(1,s2);
   System.out.println(s1);
   s1.insert(10,s2);
   System.out.println(s1);
   s1.insert(20,s2);
   System.out.println(s1);
   s1.insert(99999,s2);
   System.out.println(s2);
}

catch (StringIndexOutOfBoundsException e)
{
   System.out.println(" String index too large");
}
```

We next have the *try {...} catch* block. In the event of an error control will pass to the catch statement, and the statements following will be executed. In this case in the event of an *StringIndexOutOfBounds* error the message `String index too large` will appear.

Within the try catch block we have code which inserts one string within another and then prints out the value of the new string.Notice the object oriented way of doing this where we have the binding of StringBuffer object (in each case s1) with the insert action.

The the StringBuffer object s1 will grow successfully in size until we try providing a starting point for the insertion that is outside of the current size of s1.

This program highlights quite clearly the different way of thinking required when we approach things from an object oriented viewpoint.

5.3.1.2 StringBuffer Example 2 – throwing an exception and splat

```
import java.lang.*;

class c0508
{
   public static void main(String[] args)
   {
      StringBuffer s1=new
      StringBuffer(“ this is the base string”);
      StringBuffer s2=new StringBuffer(“ insert this”);
      System.out.println(s1);
      System.out.println(s2);
      s1.insert(1,s2);
      System.out.println(s1);
      s1.insert(10,s2);
      System.out.println(s1);
      s1.insert(20,s2);
      System.out.println(s1);
      s1.insert(999999,s2);
   }
}
```

This example is very similar to the previous but now we don’t have the enclosing try catch block. Try running both of the programs to see what happens.

We will come back to the concept of exception handling in much greater depth throughtout the course.

5.4 References

You should be aware by now of the difference between the built in basic Java data types (byte, short, int, long, etc) and objects. When we work with i/o for example we have the following conversion taking place when we need to get read in a floating point number:–

- String -> Float ->float

A String an object and whenever we use a String variable we are actually using a reference to the corresponding object. Object references are null when they don’t actually refer to anything. For people with a knowledge of pointers this will be quite familiar.

This has some interesting implications for the == operator. Try running the following program.

```
import java.lang.*;

class c0
{
   public static void main(String[] args)
   {
      String s1,s2;
      s1="Hello";
      s2="Hello";
      if (s1==s2) System.out.println(“ 1");
```

```
      s1=s2;
      if (s1==s2) System.out.println(“ 2");
      if (s1 == “Hello”) System.out.println(“ 3");
   }
}
```

What do you think the output will be?

5.5 Unicode

Java is of course Unicode based. Some knowledge of Unicode is required for successfully using Java with characters and strings.

The following provides some information regarding the languages supported under Unicode.

Arabic – ISO-8859-6

Catalan – ISO-8859-1

Chinese (Simplified) – GB2312

Chinese (traditional) BIG5

Danish – ISO-8859-1

Dutch – ISO-8859-1

English – ISO-8859-1

Esperanto – ISO-8859-3

Finnish – ISO-8859-1

French – ISO-8859-1

Georgian– UTF-8

German – ISO-8859-1

Hebrew – ISO-8859-1

Hungarian – ISO-8859-2

Irish Gaelic – ISO-8859-1

Italian – ISO-8859-1

Japanese – SHIFT_JIS

Korean – EUC_KR

Norwegian (Bokmal) – ISO-8859-1

Norwegian (Nynorsk) – ISO-8859-1

Occitan – ISO-8859-1

Portuguese (Brazil) – ISO-8859-1

Portuguese (Portugal) – ISO-8859-1

Romanian – ISO-8859-2

Russian – ISO-8859-5

Slovenian – ISO-8859-2

Spanish – ISO-8859-1

Swedish – ISO-8859-1

Yiddish – UTF-8

Other languages that I know are supported within Unicode include Gujariti, Telugu, Kannada, Mayalam, Thai, Lao

5.6 Summary

So far we've only looked at characters and Strings and StringBuffers. As Java is unicode based we are dealing with 16 bit characters, and each element of a string is 16 bit.

We can also look at character manipulation in a number of other ways, that may be more appropriate to the problem in hand. Thus we can have:–

- arrays of characters
- conversion to byte from both char and String and StringBuffer
- conversion from byte to String and StringBuffer
- arrays of bytes

and you should chose the representation that is most suited to your application.

5.7 Problems

1. Firstly try the examples out in this chapter.

2. Modify the StringBuffer example to repeatedly increase the size of the string. At what point does the program fail, i.e. how big is the string at the time it can't be inceased in size any more.

3. Write a Java program to print out , i.e. print out the character pi to the screen, not 3.14 etc.

6
Arrays in Java

‘Where shall I begin your Majesty’ he asked.
‘Begin at the beginning,’ the King said, gravely ‘and go
on until you come to the end then stop.’

Lewis Carroll, Alice’s Adventures in Wonderland.

Aims

The aims of this chapter are to:–

look at the basic array syntax in Java

look at the associated control structure, the for loop;

look at array element ordering in Java;

look forward to the use of some of the additional features of Java that enable us to make array handling more understandable and reliable.

6 Arrays In Java

In this chapter we will look at the basic features of Java that support arrays.

The first thing to know is that arrays start at 0, i.e. an array dimensioned to 12 has indices from 0 through 11.

The second thing to know is that arrays are objects.

Consider the following example:–

6.1 Example 1

For people who have attended the Fortran 90 and/or C++ courses the examples will look very familiar!

```
class c0601
{
  public static void main(String[] args)
  {
    float sum=(float)0.0,average=(float)0.0 ;
    float[] rainfall={1,2,3,4,5,6,7,8,9,10,11,12} ;
    int month ;
    System.out.println(“ Rainfall values are ”);
    for (month=0;month < 12 ; ++month)
      System.out.println( rainfall[month]);
    for (month=0;month < 12 ;++month)
      sum = sum + rainfall[month];
    average = sum/12;
    System.out.print(“ Average is ”);
    System.out.println( average);
  }
}
```

Let us look more closely at some of the statements in this program.

```
float sum=(float)0.0,average=(float)0.0 ;
```

This statement declares the variables sum and average to be of type float and it gives them initial values of 0.0 Remember that real constants are of type double by default, and as Java is strongly type we must cast to float.

```
float[] rainfall={1,2,3,4,5,6,7,8,9,10,11,12} ;
```

This declares the array rainfall to be of type float – rainfall is an array of float. We then assign initial values for the elements of the array. The values 1 through 12 have been chosen to make it easy to see whether the calculations give the correct results. The size of the array is worked out from the number of elements between the {} brackets.

```
for (month=0;month < 12 ; ++month)
  System.out.println( rainfall[month]);
```

This is a simple for loop in Java. It prints out each element of the array on a new line.

The next thing of interest is the array indexing. This goes from 0 through 11. A little odd at first, but don’t panic.

We are used to arrays starting at 1 – naturally, this is the first element. Zero didn’t exist in mathematics for quite some time. However BASIC is a programming language that offers arrays starting at 0 and 1 as an option in most implementations.

Lets look now at the for loop in Java. The general sytax is:–

```
for  (  inital statement
;
        expression 1     ;
        expression 2     )
        statement
```

The *initial statement* normally sets up an initial value for the loop counter. In this case month.

Expression 1 is the loop control mechanism. In this case we are interested in stopping once we have processed all 12 months.

Expression 2 is normally the loop counter increment mechanism. In this case increment by one.

statement is the statement that will be executed whilst expression 1 is true.

We will look in more detail at the for statement in a later chapter.

```
for (month=0;month < 12 ;++month)
   sum = sum + rainfall[month];
```

Here we calculate the sum using a simple for loop again.

Finally we calculate and print out the average.

6.2 Example 2 Variant on 1 using alternate syntax

```
class c0602
{
   public static void main(String[] args)
   {
      float sum=(float)0.0,average=(float)0.0 ;
      float[] rainfall=new float[12];
      int month ;
      System.out.println(“ Rainfal values are ”);
      for (month=0;month < rainfall.length ; ++month)
      {
         rainfall[month]=month+1;
         System.out.println( rainfall[month]);
         sum = sum + rainfall[month];
      }
      average = sum/12;
      System.out.print(“ Average is ”);
      System.out.println( average);
   }
}
```

The first difference occurs with the array declaration and creation of an array object. In the first example we had an implicit new using the assignment operator and {} to give initial values to the array.

```
float[] rainfall=new float[12];
```

has the same declaration syntax but a completely different syntax for the array object creation. The array is an object so we use new to create it. It is of type float and we want it to hold 12 values so we use `float[12]`

In this example we use { and } to bracket the statement under the control of the for loop. This enables us to repeatedly execute more than one statement. Now we include the initial-

isation, printing out of the values and the actual calculation of the sum. Note that the array index goes from 0 through 11, not 1 through 12.

We also make use of a new another feature of arrays in Java – their length attribute.

```
for (month=0;month < rainfall.length ; ++month)
```

So in this case we can terminate the loop when we have reached the length of the array. This is a simple example of the treatment of arrays as objects.

6.3 Example 3 – two dimensional arrays

The following looks at a two dimensional array problem in Java.

```
class c0603
{
  public static void  main(String[] args)
  {
    int latitude,longitude;
    float[][] height= new float[5][5];
    for ( latitude=0 ; latitude < 5 ; ++latitude)
      for (longitude=0;longitude < 5 ; ++longitude)
      {
        height[longitude][latitude]=latitude+longitude;
        System.out.println(height[longitude][latitude]);
      }
  }
}
float[][] height= new float[5][5];
```

The syntax for a 2 d array is a simple extension of the syntax for a 1 d array.

Things are now getting a little away from our original real world way of looking at problems. We now have 0–4 everywhere where the real world says 1–5.

6.4 Example 4 – 1 d array with real world -20 to +20

Now consider a simple physics example where in the real world voltage goes from -20 to +20.

```
class c0604
{
  public static void main(String[] args)
  {
    int voltage;
    float[] current=new float[41];
    float resistance=(float)100.0;
    for ( voltage=0 ; voltage <= 40 ; ++voltage )
    {
      current[voltage]=(voltage-20)/resistance;
      System.out.println(current[voltage]);
    }
  }
}
```

The solution to the problem in a programming language is now quite removed from the original problem. -20 through +20 has become 0 through 40. We are now having to do a few mental gymnastics to understand what is going on.

The **real** world	Java
-20	0
-19	1
-18	2
..	..
0	20
..	..
+19	39
+20	40

This means that there is quite a gap between the representation of a problem in the real world and its solution in Java (or C or c++ for that matter). As this is one of the major sources of errors getting into our programs it means that Java isn't well suited to the solution of many common mathematical problems.

6.5 Example 5 – 2 d array initialisation

In this example we look at array initialisation.

```
class c0605
{
  public static void main(String[] args)
  {
    int[][] a= {
                  { 1,2,3 },
                  { 4,5,6 }
                };
    for (int row=0;row<2;row++)
    {
      for (int column=0;column<3;column++)
      {
        System.out.print( a[row][column] );
        System.out.print(“ ”);
      }
    System.out.println();
    }
  }
}
```

6.6 Whole Array Manipulation

There is no whole array manipulation mechanism in Java, and it shares this with C++. However in C++ we can, through the use of classes and operator overloading, provide this functionality for one dimensional arrays, whilst still using the standard C++ array indexing notation, i.e. []. We can also extend this to multi-dimensional arrays in C++ using the () notation.

Thus Fortran 90 has very clear advantages over both Java and C++ in the mathematical area.

6.7 Summary

There are some problems with arrays in raw Java, and the main one is the machine oriented array dimensioning, rather than physical world array dimensioning, e.g. stepping from 0–360 instead of -180–+180. The major advance over C++ is the built in subscript checking, something which can only be achieved in C++ through operator overloading.

6.8 Problems

1. Type in and run some of the examples in this chapter. Modify the program to go outside of the array bound. What run time messages do you get?

2. Write a program to assign the following data values into a 3*3 array.

1	2	3
4	5	6
7	8	9

Produce totals for each row and column. It should produce output as shown below:

1	2	3	6
4	5	6	15
7	8	9	24
12	15	18	

After successfully doing this modify the program to produce averages for each row and column as well.

3. The data below has been taken from a group of first year undergraduates. It is their heights and weights.

```
1.85    85
1.80    76
1.85    85
1.70    90
1.75    69
1.67    83
1.55    64
1.63    57
1.79    65
1.78    76
```

Your body mass index is given by your height (in metres) squared divided by your weight in kilos. Calulate the BMI for each person.. Use arrays to hold the height and weight information.

Grades of obesity according to Garrow as follows:

Grade 0 (desirable) 20 - 24.9

Grade 1 (overweight) 25-29.9

Grade 2 (obese) 30-40

Grad 3 (morbidly obese) >40

Ideal BMI range,

Men, Range 20.1 to 25 kg/m^2

Women, Range 18.7 to 23.8 kg/m^2

If you know your own height and weight modify the above to calculate your own BMI.

7
Control Structures

Summarising: as a slow-witted human being I have a very small head and I had better learn to live with it and to respect my limitations and give them full credit, rather than try to ignore them, for the latter vain effort will be punished by failure.
Edsger W. Dijkstra, Structured Programming.

Aims

The aims of this chapter are to introduce:–

selection between various courses of action as part of the problem solution

the concepts and statements in Java needed to support the above. In particular:–

- logical expressions
- logical operators
- a *block* of statements
- several *blocks* of statements

the *if expression statement*

the i*f expression statement else statement;*

to introduce the *switch* statement with examples

iterative statements:–

- *while expression statement;*
- *do statement while expression;*
- the *for () statement*

the *break*, *continue* and *goto* statement

7 Control Structures

There are a reasonable range of control structures in Java. We need to review a number of other concepts before looking at them in depth.

7.1 Compound Statement or Block

A compound statement or block of statements is a sequence of statements enclosed in {}. A compound statement is treated as a single item and may appear anywhere that a single statement may occur.

7.2 Expression

An expression is made up of one or more operations. They in turn are a mixture of operands and operators. The evaluation of an expression typically means that one or more operations are carried out and a value is returned.

7.3 Boolean

Remember that within standard Java boolean exists as a built in type, with values of true and false.

7.4 if (expression) statement

Simple if statement. If the expression is true execute the statement that follows. Note that {} have to be used if it is necessary to execute multiple statements.

7.4.1 Example 1

```
if ( i>0 )
  System.out.println(“ Now above 0 ”);
```

If i is greater than 0 the cout statement is executed.

Note that you cannot make the classic C and C++ howler in Java. If you are familiar with C and C++ then you may be surprised to know that the following example won’t compile.

```
class c0703
{
  public static void main(String[] args)
  {
    int i=0;
    int j=10;
    System.out.println(i);
    System.out.println(j);
    if (i=1)
      j=99;
    System.out.println(i);
    System.out.println(j);
  }
}
```

7.5 if (expression) statement; else statement;

Standard extension to the if statement. Again {} have to be used if it is necessary to execute multiple statements.

7.5.1 Example 1

```
if ( i < 0 )
  System.out.println(“ Result not defined for negative val-
ues”);
else
  System.out.println(“ Calculating for positive i ”);
```

One or other of the cout statements will be executed. Note the semi-colons. If you are familiar with Pascal or Ada this will catch you out.

7.5.2 Example 2

```
if ( i < 0 )
  System.out.println(“ Entering negative region”);
else if ( i == 0 )
  System.out.println(“ crossover reached ”);
else
  System.out.println(“ Positive region entered ”);
```

Note the semi-colons. Again will catch the Pascal/Modula 2 programmer out.

7.6 switch (expression) statement

This is best illustrated with a simple example.

7.6.1 Example 1

```
import java.io.*;
import java.lang.Integer;

class c0704
{
  public static void main(String[] args)
  {
    int i=0;
    Integer I=new Integer(i);
    String Line;
    try
    {
      InputStream ip=System.in;
      DataInputStream inp=new DataInputStream(ip);
      System.out.println(“ Type in an integer value ”) ;
      Line=inp.readLine();
      I=Integer.valueOf(Line);
      i=I.intValue();
      switch (i)
      {
        case 1 : System.out.println(“ one entered ” );
        break;
        case 2 : System.out.println(“ two entered ” );
        break;
        case 3 : System.out.println(“ three entered ” );
        break;
        default: System.out.println(“ number other than 1,
2 or 3 entered”);
```

```
            break;
         }
      }
      catch(IOException e)
      {
         System.out.println(" Exceptions raised " + e);
      }
   }
}
```

Note the use of the break statement to exit from the switch statement, otherwise execution simply drops through!

Equivalent to the case statement in other languages.

7.7 while (expression) statement

Conventional while statement, i.e. statement may never be executed. This example highlights both the if else statement and the while statement. It is a direct translation of the Fortran 90 example.

7.7.1 Example 1

```
import java.io.*;
import java.lang.Integer;

class c0705
{
   public static void main(String[] args)
   {
      int counter=0;
      int mark=99;
      Integer Mark=new Integer(mark);
      int[] a={1,2,3,4,5,6,7,8,9,10,0};
      String Line;
      try
      {
         InputStream i=System.in;
         DataInputStream I=new DataInputStream(i);
         System.out.println(" Type in value to look for") ;
         Line=I.readLine();
         Mark=Integer.valueOf(Line);
         mark=Mark.intValue();
         a[a.length-1]=mark;
         while (mark != a[counter]) ++counter;
         if (counter == a.length-1)
            System.out.println(" Number not found");
         else
         {
            System.out.println(" Number found at position");
            System.out.println(counter);
         }
      }
      catch(IOException e)
```

```
      {
         System.out.println(“ Exceptions raised ” + e);
      }
   }
}
```

We will look at the use of a sentinel in a later example.

7.8 do statement while (expression);

Equivalent to repeat until statement, i.e. the loop is always executed at least once as the test is at the end of the loop. This example is the e^x function taken from the Fortran 90 course.

7.8.1 Example 1

```
import java.io.*;
import java.lang.Float;

class c0706
{
   public static void main(String[] args)
   {
      float tol=(float)1.0E-6;
      float term=(float)1.0,x=(float)0.0,etox=(float)1.0;
      int nterm=0;
      Float X=new Float(x);
      String Line;
      try
      {
         InputStream i=System.in;
         DataInputStream I=new DataInputStream(i);
         System.out.println(“ Type in a number”) ;
         Line=I.readLine();
         X=Float.valueOf(Line);
         x=X.floatValue();
         do
         {
            nterm+=1;
            term = (x/nterm) * term;
            etox+=term;
         }
         while (term > tol);
      }
      catch(IOException e)
      {
         System.out.println(“ Exceptions raised ” + e);
      }
      System.out.println(etox);
   }
}
```


7.9 for (init-statement;expression 1; expression 2) statement

Equivalent to the DO loop in Fortran or FOR loop in the Pascal family of languages. Note that {} must be used when multiple statements need to be executed.

ini-statement may be declaration or an expression. This enables us to introduce for loop control variables at the time we set up the for loop. Some people love the ability to introduce variables in this way, others hate it. It is the way it is.

The following highlight this point–

```
for (int i=0;
```

Declare i and initialise to 0.

```
for ( i=0;
```

i must have been declared prior to this statement.

expression 1 is the loop control mechanism. As long as this is true the for statement will be executed.

expression 2 is evaluated after each loop and is generally used to modify the for loop control variable.

7.9.1 Example 1

This complete program illustrates the above.

```
class c0707
{
  public static void main(String[] args)
  {
    float[] x={1,2,3,4,5,6,7,8,9,10};
    int i;
    System.out.println(" Numbers are");
    for ( i=0 ; i < 10 ; i++)
      System.out.println(x[i]);
  }
}
```

7.9.2 Example 2

```
class c0708
{
  public static void main(String[] args)
  {
    for(;;)
      System.out.println(" Hello world");
  }
}
```

What happens here?

7.10 break, continue, goto statements

The break statement can only occur within a switch or loop (for, do or while). Note however that you can only break from a single loop. If it is necessary to terminate an action within nested loops then the break won't do what we want.

The continue statment only makes sense with a loop. Control passes immediately to the appropriate statement depending on what type of loop we are in.

Statements may be labelled and we can use the break statement to jump to an appropriate point in the code.

There is no goto statement in Java.

The following is a complete program that illustrates the use of all three statements. Type the program in and run it to see what happens.

```
class c0710
{
   public static void main(String[] args)
   {
      int j=0;
      goto2:
      for (;;)
      {
         goto1:
         for (;;)
         {
            j=j+1;
            System.out.print(j);
            if (j>10)
               break goto2;
            if ((j>=5) & (j<=7))
               break goto1;
            else
            {
               System.out.println(" in the else clause");
               continue;
            }
         }
      }
      System.out.println(" When will we get here");
   }
}
```

Study the program to see if you can predict what the output will be. Now type the program in and compile and run it. Were you correct?

7.11 Summary

Java has a decent set of control structures. They provide us with most of the functionality we require.

7.12 Problems

1. Write a program to print out the 12 times table. Output should be roughly of the form

```
1     *     12     =     12
2     *     12     =     24
```

2. Write a program that produces a conversion table from litres to pints and vice versa. One litre is approximately 1 3/4 pints. The output should comprise three columns. The middle column should be an integer and the columns to the left and right should be the corresponding pints and litre values. This enables the middle column to be scanned quickly and the corresponding equivalent in litres or pints read easily.

3. Rewrite the program for the period of a pendulum. The new program should print out the length of the pendulum and period for lengths of the pendulum from 0 to 100 cm in steps of 0.5 cm.

The physical world has many examples where processes require some threshold to be overcome before they begin operation: critical mass in nuclear reactions, a given slope to be exceeded before friction is overcome, and so on. Unfortunately, most of these sorts of calculations become rather complex and not really appropriate here. The following problem tries to restrict the range of calculation, whilst illustrating the possibilities of decision making.

4. If a cubic equation is expressed as:

$$z^3 + a_2z_2 + a_1z + a0 = 0$$

and we let:

$$q=a_1/3-(a_2*a_2)/9$$

and:

$$r=(a_1*a_2-3a_0)/6-(a_2*a_2*a_2)/27$$

we can determine the nature of the roots as follows:

$q^3 + r^2 > 0$; one real root and a pair of complex;

$q^3 + r^2 = 0$; all roots real, and at least two equal;

$q^3 + r^2 < 0$; all roots real;

Incorporate this into a suitable program, to determine the nature of the roots of a cubic from suitable input.

5. The form of breaking waves on beaches is a continuum, but for convenience we commonly recognise three major types: surging, plunging and spilling. These may be classified empirically by reference to the wave period, T (seconds), the breaker wave height, H_b (metres), and the beach slope, m. These three variables are combined into a single parameter, B, where

$$B = H_b/(gmT^2)$$

g is the gravitational constant (981 cm sec^{-2}). If B is less than .003, the breakers are surging; if B is greater than 0.068, they are spilling, and between these values, plunging breakers are observed.

(i) On the east coast of New Zealand, the normal pattern of waves is swell waves, with wave heights of 1 to 2 metres, and wave periods of 10 to 15 seconds. During storms, the wave period is generally shorter, say 6 to 8 seconds, and the wave heights higher, 3 to 5 metres. The beach slope may be taken as about 0.1. What changes occur in breaker characteristics as a storm builds up?

(ii) Similarly, many beaches have a concave profile. The lower beach generally has a very low slope, say less than 1 degree (m=0.018), but towards the high tide mark, the slope increases dramatically, to say 10 degrees or more (m=0.18). What changes in wave type will be observed as the tide comes in?

6. Personal taxation is usually structured in the following way:–

no taxation on the first m_0 units of income;

taxation at t_1% on the next m_1 units;

taxation at t_2% on the next m_2 units;

taxation at t_3% on anything above.

 email: ian.chivers@kcl.ac.uk

For some reason, this is termed *progressive* taxation. Write a generalised program to determine net income after tax deductions. Write out the gross income, the deductions and the net income. You will have to make some realistic estimates of the tax thresholds m_i and the taxation levels t_i. You could use this sort of model to find out how sensitive revenue from taxation was in relation to cosmetic changes in thresholds and tax rates.

8. The specific heat capacity of water is 2009 J kg^{-1} K^{-1}; the specific latent heat of fusion (ice/water) is 335 kJ kg^{-1}, and the specific latent heat of vaporization (water/steam) is 2500 kJ kg^{-1}. Assume that the specific heat capacity of ice and steam are identical to that of water. Write a program which will read in two temperatures, and will calculate the energy required to raise (or lower) ice, water or steam at the first temperature, to ice, water or steam at the second. Take the freezing point of water as 273 K, and its boiling point as 373 K. For those happier with Celsius, 0^o C is 273 K, while 100^o c is 373 K. One calorie is 4.1868 J, and for the truly atavistic, 1 BTU is 1055 J (approximately).

7.13 Bibliography

Dahl O. J., Dijkstra E. W., Hoare C. A. R., *Structured Programming*, Academic Press, 1972.

- This is the original text, and a must. The quote at the start of the chapter by Dijkstra summarises beautifully our limitations when programming and the discipline we must have to successfully master programming.

Knuth D. E., *Structured Programming with GOTO Statements*, in Current Trends in Programming Methodology, Volume 1, Prentice Hall.

- The chapter by Knuth provides a very succinct coverage of the arguments for the adoption of structured programming, and dispells many of the myths concerning the use of the GOTO statement. Highly recommended.

8
Exception Handling

Don’t interrupt me while I’m interrupting.

Winston Churchill.

Aims

The primary aims of the chapter are:–

- to look generally at errors and how they can be handled in a programming language;
- to look at the mechanisms in Java for exception handling – try, catch, finally;
- to look at a complete realistic example that highlights the power of exception handling;

8 Exceptions

Errors always occur at run time in programs that do anything useful. The facilities for handling errors have gradually improved in programming languages. The concept of raising an exception and then passing control to an error handler is one that is now seen in a number of programming languages including C++, Ada 95 and Java.

Let us look at some common errors and solutions:–

- invalid input: ask the user to retype the data values
- file not found: ask the user to retype the file name
- numeric overflow: terminate the program with a warning;
- numeric underflow: terminate the program with a warning;
- array out of bounds: terminate the program with an error message;

and you will all be familiar with one or more of these common errors.

One of the major problems in languages that do not support the concept of raising an exception and passing control to an error handler is that the logic of something really quite simple can be lost completely because of the additional coding complexity.

So we have to chose between:–

- correctness by trapping all possible errors at one extreme with code that is very difficult to understand;

and

- probable erroneous output because we have only trapped a small number of errors with code that is relatively easy to understand.

Exceptions enable us to move towards overall program correctness without writing code that is unintelligible.-

Let us look at examples in a couple of programming languages of reading user input.

Fortran 90 for example has the concept of implicit gotos via the END= and ERR= options and also the concept of IOSTAT returing a value to let you know something has gone wrong.

Pascal and Modula 2 have the concept of EOLN and EOF.

The following are coding examples that read all input from a user until they type CTRL Z – end of file. The first is in Pascal, the second in Fortran 90, and the third in C++. The examples also highlight the quite different ways that languages have in their handling of end of line and end of file.

8.1 Linked List – Pascal

```
PROGRAM LinkedList(INPUT,OUTPUT);
TYPE Link = @ Node
   Node = RECORD
   C : CHAR;
   Next : Link;
   END;
VAR  Root    : Link;
     Current : Link;
BEGIN
   NEW(Root);
   READ(Root@.C);
```

```
   Current:=Root;
   WHILE NOT EOF DO
   BEGIN
   NEW(Current@.Next);
   Current:=Current@.Next);
   READ(Current@.C)
   END;
   Current@.Next:=NIL;
   Current:=Root;
   WHILE Current  NIL DO
   BEGIN
      WRITE(Current@.C);
      Current:=Current@.Next
   END;
END.
```

8.2 Linked List – Fortran 90

```
PROGRAM C20_01
!
! Simple linked list
!
TYPE Link
   CHARACTER :: C
   TYPE (Link) , POINTER   :: Next
END TYPE Link
TYPE (Link) , POINTER :: Root , Current
INTEGER :: IO_Stat_Number=0
   ALLOCATE(Root)
   READ (UNIT=*,FMT=10,ADVANCE='NO',&
   IOSTAT=IO_Stat_Number) Root%C
   10 FORMAT(A1)
   print *,io_stat_number
   IF (IO_Stat_Number == -1) THEN
      NULLIFY(Root%Next)
   ELSE
      ALLOCATE(Root%Next)
   ENDIF
   print *, 'At start of input DO WHILE'
   Current=Root
   DO WHILE (ASSOCIATED(Current%Next))
      Current=Current%Next
      READ (UNIT=*,FMT=10,ADVANCE='NO',&
      IOSTAT=IO_Stat_Number) Current%C
      print *,io_stat_number
      IF (IO_Stat_Number == -1) THEN
         NULLIFY(Current%Next)
      ELSE
         ALLOCATE(Current%Next)
      ENDIF
   END DO
```

```
   print *, 'At end of input DO WHILE'
   print *, 'At start of output DO WHILE'
   Current=Root
   DO WHILE (ASSOCIATED(Current%Next))
      PRINT * , Current%C
      Current=Current%Next
   END DO
   print *, 'At end of output DO WHILE'
END PROGRAM C20_01
```

8.3 Linked List – C++, old C syntax

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
struct link
{
   char c;
   struct link *next;
};
int main()
{
   char c;
   struct link *first = NULL;   /*  Start of list   */
   struct link *current;        /*  End of list     */
   struct link *ptr;            /*  Temporary       */

/* Loop reading characters entil the End Of File (EOF) */
/* Note that one cannot use eof() to check for EOF until*/
/* one has actually tried to read it - unlike Pascal.*/
   while (cin >> c ) /*  Loop for all characters  */
   {
      ptr = new (link);
      if (ptr == NULL)
      {
         cout << " Insufficient memory\n";
         exit(1);
      }

/*  Add new value and clear subsequent link  */
      ptr->c = c;
      ptr->next = NULL;

/*
Update pointers to add new value at end of list.  The "if"
statement could be omitted by making the first character en-
tered a "special" case - as in Example 10.1.4 - but
generallity is a good idea!
*/
      if (first == NULL) /*  If this is the first character
*/
```

```
      first = current = ptr;
    else                              /*  Otherwise. */
      current->next = ptr;
    current = ptr;
  }

/*  Now print out the list  */

  ptr = first;
  while (ptr != NULL)
  {
    cout << ptr-c;
    ptr = ptr->next;
  }
  cout << endl;
  return 0;
}
```

8.4 Discussion

It is apparent that the essential code is hidden to greater or lessor extent in the requirements to handle special conditions. The Pascal code is probably the cleanest. The beauty of exception handling is that the essential logic of the code is left alone and enclosed within a try statement. Let us look at one of the earlier examples to see how simple and straightforward this is in Java.

```
import java.io.*;
class c0302
{
  public static void main(String[] args)
  {
    try
    {
      InputStream i=System.in;
      DataInputStream in=new DataInputStream(i);
      String Line;
      System.out.println(" Type in a line of text ");
      Line=in.readLine();
      System.out.println(Line);
    }
    catch(IOException e)
    {System.out.println(" Exceptions raised: " + e); }
  }
}
```

If there is an error getting a line of input from the user within the body of the try block control will pass to the catch statement and in this case an informative message is issued about the error. The readLine() method can throw an IOException. We will come back to look at the above example after we have covered i/o in more depth.

The general syntax is:–

```
try
{
```

```
}
catch (Exception_1 e)
{
}
catch(Exception_2 e)
{
}
...
finally
{
}
```

8.4.1 try

The try block can have zero or more catch clauses. An abnormal exit is one of:–

- break
- continue
- return
- exception propagation

8.4.2 catch

These are the exception handlers. They actually carry out thw work of handling the exception that was raised in the try block. The argument of the catch must be of type Throwable (or subclass). We will look into this in more depth in the complete file copy program in this chapter.

8.4.3 finally

This part is executed:–

- after normal termination of the try block;
- after an exception handled in a catch clause;
- after an exception that has no catch clause;
- after a break, continue or return statement;

We will look into this in more depth in the complete file copy program in this chapter.

8.5 Array Subscript Errors

Array subscript checking is always on in Java, and any attempt to go out of the bounds of an array will raise an exception.

8.6 Anticipated Errors vs Unanticipated Errors

These are errors that one might reasonably expect to happen. End of file and an incorrectly typed file name are two obvious ones that spring to mind.

Unanticipated errors are ones that you don’t think of!

You can trap both kinds of errors with exception handling.

8.7 Complete Example – File copy program

The following program illustrates the power of exception handling. It also illustrates most of the syntax of how to use exception handling. It is taken directly from the Nutshell book by David Flanagan.

```
// This example is from the book
```

```
// _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan.
// Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and
// distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY
// either expressed or implied.
import java.io.*;
public class FileCopy {
  public static void copy(
    String source_name,
    String dest_name)
    throws IOException
  {
    File source_file = new File(source_name);
    File destination_file = new File(dest_name);
    FileInputStream source = null;
    FileOutputStream destination = null;
    byte[] buffer;
    int bytes_read;

    try
    {
      // First make sure the specified source file
      // exists, is a file, and is readable.

      if (!source_file.exists() || !source_file.isFile())
        throw new FileCopyException(
          "FileCopy: no such source file: " +
source_name);
      if (!source_file.canRead())
        throw new FileCopyException(
          "FileCopy: source file " +
          "is unreadable: "
           + source_name);

      // If the destination exists, make sure
      // it is a writeable file
      // and ask before overwriting it.
      // If the destination doesn't
      // exist, make sure the directory exists
      // and is writeable.

      if (destination_file.exists())
      {
        if (destination_file.isFile())
        {
          DataInputStream in = new
            DataInputStream(System.in);
          String response;
```

```
            if (!destination_file.canWrite())
               throw new FileCopyException(
                  “FileCopy: destination ” +
                  “file is unwriteable: ” + dest_name);

            System.out.print(“File ”);
            System.out.print(dest_name);
            System.out.print(“ already exists. Overwrite?
(Y/N): ”);
            System.out.flush();
            response = in.readLine();
            if (!response.equals(“Y”)
               && !response.equals(“y”))
               throw new FileCopyException(
                  “FileCopy: copy cancelled.”);
         }
         else
            throw new FileCopyException(
               “FileCopy: destination ”
               + “is not a file: ”
               +  dest_name);
      }
      else
      {
         File parentdir = parent(destination_file);
         if (!parentdir.exists())
            throw new FileCopyException(
               “FileCopy: destination ”
               + “directory doesn’t exist: ” + dest_name);
         if (!parentdir.canWrite())
         throw new FileCopyException(
            “FileCopy: destination ”
            + “directory is unwriteable: ” + dest_name);
      }

      // If we’ve gotten this far,
      // then everything is okay; we can
      // copy the file.

      source = new FileInputStream(source_file);
      destination = new FileOutputStream(destination_file);
      buffer = new byte[1024];
      while(true) {
      bytes_read = source.read(buffer);
      if (bytes_read == -1) break;
      destination.write(buffer, 0, bytes_read);
      }
   }
```

```
   // No matter what happens
   // always close any streams we've opened.

   finally
   {
      if (source != null)
         try { source.close(); } catch (IOException e) { ; }
      if (destination != null)
         try { destination.close(); } catch (IOException e) {
; }
   }
}

   // File.getParent() can return null when
   // the file is specified without
   // a directory or is in the root directory.
   // This method handles those cases.

   private static File parent(File f)
   {
      String dirname = f.getParent();
      if (dirname == null)
      {
         if (f.isAbsolute())
            return new File(File.separator);
         else
            return new File(System.getProperty("user.dir"));
      }
      return new File(dirname);
   }

   public static void main(String[] args)
   {
      if (args.length != 2)
      {
         System.err.print("Usage: java FileCopy "
         System.err.print("[source_file][destination_file");
      }
      else
      {
         try
         {
            copy(args[0], args[1]);
         }
         catch (IOException e)
         {
            System.err.println(e.getMessage());
         }
      }
   }
```



```
}

class FileCopyException extends IOException
{
   public FileCopyException(String msg) { super(msg); }
}
```

Let us look at this example in some depth. Copying files is a frequent requirement so it makes sense to make the file copy method that does the work public. This means we can easily incorporate this fully tested method in another program with ease. In this example we have a main method and thus it is a standalone program.

There are a number of errors that can occur:–

- incorrectly typed source file name
- incorrectly type destination file name
- in the wrong part of the directory structure
- destination file already exists
- don't have write access to the file
- don't have write access to a directory

The program copes with all of these situations.

The file is treated as a sequence of bytes – the most flexible way of handling both text and binary files.

Let us look first at the FileCopy.copy method.

```
public class FileCopy
{
   public static void copy(
      String source_name,
      String dest_name)
      throws IOException
```

...

The copy method is public – we want it to be used.

The copy method is void – it does not have a return type. It is a subroutine or procedure in Fortran or Modula terminology.

It takes two arguments – the source and destination file names. These are of course of type String.

Lastly the copy method can throw an IOException. We will have to provide an error handler for this later in the code.

Let look now at the last part of the program.

```
class FileCopyException extends IOException
{
   public FileCopyException(String msg) { super(msg); }
}
```

IOException is a built in Java class. In this example we extend this class with our own class FileCopyException. So when we want to throw an exception we will be throwing a FileCopyException error, and control will pass to this error handler.

Our own FileCopyException procedure takes one argument of type String and then invokes the method within IOException. The terms super and sub class are used to indicate where

you are in a class hierarchy. We will come back to this in more depth when we look formally at class extension in a later chapter.

8.8 Java Errors and Exceptions

Java discriminates between errors and exceptions. An error is treated as irrecoverable and an exception is something that can be handled by the code writer if they want.

Note that errors and exceptions can occur both within the interpreter whilst you code is running and in your code itself.

We have the following two class hierarchies:–

Object

Throwable

Exception

and

Object

Throwable

Error

and the actual errors and exception methods are at the next level down in the tree.

8.9 Java On-line Documentation

Rather than cluuter up the notes with a complete list of all of the exceptions and errors that can arise in Java have a look at the documentation that comes with the Java 1.1 development kit. I've put this up on the College web server and you can download and install this file on a system that runs both Netscape and supports long file names.. This provides a very good on-line source of information regarding Java. I will try to get it installed on the Sun that we are using for the course also.

8.10 Summary

There are a number of examples that cover this area in more depth later in the course, as it is best to see exception handling done in a realistic context to appreciate the power and expressiveness that the concepts provide.

8.11 Problems

1. Try writing a Java version of the linked list examples in this chapter. There are a number of catches here.

2. Try running the example program in this chapter. I've put it up on the Web server. Base url is:–

`http://www.kcl.ac.uk/support/cc/fortran/home.html`

This method can easily be incorporated into any program that requires it very easily.

9
i/o

Winnie-the-Pooh read the two notices very carefully,
first from left to right, and afterwards,
in case he had missed some of it, from right to left.
A. A. Milne, Winnie-the-Pooh.

From a programmer's point of view the user is a peripheral that types when you issue a read request.
Peter Williams.

Aims

The aims of this chapter are to look at the mechanisms in Java for i/o. As Java is fully object oriented the way Java does i/o will appear strange at first. There is a coverage of streams and their use for both input and output.

9 i/o

There is a complete package for i/o called java.io. This contains paired streams. A stream is an ordered sequence of data. A stream has either a source or a destination.

There is no way at the moment within Java of getting formatted output. This means that you cannot:–

- define a minimum width on output
- define a maximum width on output
- define precision for floating point output

Thus it is impossible at present to achieve neat tabular output.

A subset is listed below.

InputStream	OutputStream
ByteArrayInputStream	ByteArrayOutputStream
PipedInputStream	PipedOutputStream
FilterInputStream	FilterOutputStream
DataInputStream	DataOutputStream
FileInputStream	FileOutputStream
RandomAccessFile	
DataInput	DataOutput
DataInputStream	DataOutputStream

The complete hierarchy is given below. This highlights many of the very important concepts involved in object oriented programming.

Object
- File
- FileDescriptor
- InputStream
  - ByteArrayInputStream
  - FileInputStream
  - FilterInputStream
    - BufferedInputStream
    - DataInputStream
    - LineNumberInputStream
    - PushBackInputStream
  - PipedInputStream
  - SequenceInputStream
  - StringBufferInputStream
- RandomAccessFile
  - DataInput
    - DataInputStream
  - DataOutput
    - DataOutputStream

 email: ian.chivers@kcl.ac.uk

OutputStream
ByteArrayOutputStream
FileOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
FilterOutputStream
PipedInputStream
StreamTokenizer

I recommend looking at the on-line documentation that comes with the Java 1.1 jdk. Use netscape to browse it. All of the information required is there. We will only look at a subset of the above in the notes.

Note that Java does not support file i/o from an applet when running on another system, i.e. if we were loading a Java applet from system A and running it on system B, then it would not have access to B's file system. This would comprimise the security of Java.

A Java program will have access to the underlying file system.

9.1 Class vs Interface

A class will provide both the definition and implementation of a set of methods an object supports.

An interface provides only a definition of the methods. The implementation must be provided by classes that extend the interface. This is the equivalent of an abstract class with virtual methods in C++ terminology.

9.2 Java.io.DataInput – interface

An interface that defines the methods for input of the base Java types. Note that these are object methods. You must convert from the base object to the underlying base Java type. A subset is given below.

public abstract boolean read Boolean() throws IOException, EOFException

public abstract byte readByte() throws IOException, EOFException

public abstract char readChar() throws IOException, EOFException

public abstract double readDouble() throws IOException, EOFException

public abstract float readFloat() throws IOException, EOFException

public abstract int readInt() throws IOException, EOFException

public abstract String readLine() throws IOException, EOFException

public abstract long readLong() throws IOException, EOFException

public abstract short readShort() throws IOException, EOFException

public abstract String readUTF() throws IOException, EOFException

public abstract int readUnsignedByte() throws IOException, EOFException

public abstract int readUnsignedShort() throws IOException, EOFException

9.2.1 UTF

Most text is 7 or 8 bit based. Most computer systems support 8 bit characters. Whilst Java uses 16 bit characters internally throughout a large part of the ineraction with the real world will be in terms of 8 bit characters.

To aid here Java supports UTF-8, which is a multibyte coding scheme. The standard ASCII characters only occupy 1 byte. Thus for text which is essentially ASCII we have a much more compact data representation.

9.3 java.io.DataInputStream – class

Note that this class returns a single Java base type in *binary* form. The value of 1 as an int is the following binary sequence of bits:–

00000000000000000001

This means that they cannot be used for user interaction. When we interact with users we do so via a stream of characters. The character value 1 is represented by the following sequence of bits:–

0000000000110001

Remember that Java characters are Unicode based and use 16 bits. In practical terms the interaction will take place in 8 bit ASCII for many people.

public final boolean read Boolean() throws IOException, EOFException

public final byte readByte() throws IOException, EOFException

public final char readChar() throws IOException, EOFException

public final double readDouble() throws IOException, EOFException

public final float readFloat() throws IOException, EOFException

public final int readInt() throws IOException, EOFException

public final String readLine() throws IOException, EOFException

public final long readLong() throws IOException, EOFException

public final short readShort() throws IOException, EOFException

public final String readUTF() throws IOException, EOFException

public final int readUnsignedByte() throws IOException, EOFException

public final int readUnsignedShort() throws IOException, EOFException

Note that all of these methods are final – they cannot be over-ridden.

9.4 java.io.DataOutput – interface

Matching interface for data output.

public abstract void writeBoolean() throws IOException

public abstract void writeByte() throws IOException

public abstract void writeChar() throws IOException

public abstract void writeDouble() throws IOException

public abstract void writeFloat() throws IOException

public abstract void writeInt() throws IOException

public abstract void writeLine() throws IOException

public abstract void writeLong() throws IOException

public abstract void writeShort() throws IOException

public abstract void writeUTF() throws IOException

public abstract void writeUnsignedByte() throws IOException

public abstract void writeUnsignedShort() throws IOException

9.5 java.io.DataOutputStream – class

Matching class for data output.

public final void writeBoolean() throws IOException

public final void writeByte() throws IOException

public final void writeChar() throws IOException

public final void writeDouble() throws IOException

public final void writeFloat() throws IOException

public final void writeInt() throws IOException

public final void writeLine() throws IOException

public final void writeLong() throws IOException

public final void writeShort() throws IOException

public final void writeUTF() throws IOException

public final void writeUnsignedByte() throws IOException

public final void writeUnsignedShort() throws IOException

Note that all of these methods are final – they cannot be over-ridden.

9.6 java.io.PrintStream – class

The most useful i/o stream for displaying text on the screen when running programs. For each print method there is a corresponding println method.

public synchronized void print(char[] s)

public synchronized void print(String s)

public void print(boolean b)

public void print(char c)

public void print(double d)

public void print(float f)

public void print(int i)

public void print(long l)

public synchronized void print(Object o)

public synchronized void println(char[] s)

public synchronized void println(String s)

public synchronized void println(boolean b)

public synchronized void println(char c)

public synchronized void println(double d)

public synchronized void println(float f)

public synchronized void println(int i)

public synchronized void println(long l)

Note that when we invoke the println method with an object there is a hidden call to the toString method associated with that object.

9.6.1 Synchronized

Java supports multiple threads. As such it is necessary to be able to indicate that a critical section of code or method must be allowed to complete. This is because the code segment or class is modifying the internal state of the class. The concepts introduced are not new

and anyone who has done an operating system course or real time programming would be familiar with the ideas.

Note that does not mean that the section of code cannot be timesliced. We have a lock set and access cannot be made to whatever the lock has been applied until it has completed.

If you would like more information see the bibliography at the end of the chapter on threads.

9.7 Example 1

The following is a C++ program to extract all of the ASCII characters from a Microsoft Word file.

```
#include <iostream.h>

int main()
{
unsigned char c;
unsigned int i=0;
  while (cin.get(c))
  {
    i=++i;
    if ( c < 32 || c   > 127)
      c = ‘ ‘;
      cout << c;
    if ( i > 60 && c == ‘ ‘)
    {
    cout << endl;
    i=0;
    }
  }
  return(0);
}
```

The following is a modification to the file copy program that does the same.

```
// This example is a simple modification of the earlier
// example on file copying.
//
// It has been rewritten to extract all of the ascii
// characters from a Microsoft Word document.
//

// This example is from the book _Java in a Nutshell_ by Da-
vid Flanagan.
// Written by David Flanagan.  Copyright (c) 1996 O’Reilly &
Associates.
// You may study, use, modify, and distribute this example
for any purpose.
// This example is provided WITHOUT WARRANTY either expressed
or implied.

import java.io.*;
```

```
public class convert {
  public static void copy(String source_name, String
dest_name)
    throws IOException
  {
    File source_file = new File(source_name);
    File destination_file = new File(dest_name);
    FileInputStream source = null;
    FileOutputStream destination = null;
    byte[] buffer;
    int bytes_read;

    try {
      // First make sure the specified source file
      // exists, is a file, and is readable.
      if (!source_file.exists() || !source_file.isFile())
        throw new convertException("convert: no such source
file: " +
                                   source_name);
      if (!source_file.canRead())
        throw new convertException("convert: source file "
+
                                   "is unreadable: " + source_name);

      // If the destination exists, make sure it is a
writeable file
      // and ask before overwriting it.  If the destination
doesn't
      // exist, make sure the directory exists and is
writeable.
      if (destination_file.exists()) {
        if (destination_file.isFile()) {
            DataInputStream in = new DataInputStream(Sys-
tem.in);
            String response;

            if (!destination_file.canWrite())
                throw new convertException("convert: des-
tination " +
                                           "file is unwriteable: "
+ dest_name);

            System.out.print("File " + dest_name +
                             " already exists.  Overwrite? (Y/N):
");
            System.out.flush();
            response = in.readLine();
            if (!response.equals("Y") && !re-
sponse.equals("y"))
```

```
                    throw new convertException("convert: copy
cancelled.");
          }
          else
               throw new convertException("convert: destina-
tion "
                                    + "is not a file: " +
dest_name);
        }
        else {
          File parentdir = parent(destination_file);
          if (!parentdir.exists())
               throw new convertException("convert: destina-
tion "
                                    + "directory doesn't exist:
" + dest_name);
          if (!parentdir.canWrite())
               throw new convertException("convert: destina-
tion "
                                    + "directory is unwriteable:
" + dest_name);
        }

        // If we've gotten this far, then everything is okay;
we can
        // copy the file.
        source = new FileInputStream(source_file);
        destination = new FileOutputStream(destination_file);
        int i=0;
        byte carriage_return=13;
        byte line_feed=10;
        char c;
        int byte_read;
        while(true)
        {
          byte_read = source.read();
          if (byte_read == -1) break;
          i++;
          if ( byte_read < 32 || byte_read > 127 )
            byte_read=32 ;
          if ( i>60 && byte_read==32 )
          {
            c=(char)carriage_return;
            destination.write(c);
            c=(char)line_feed;
            destination.write(c);
            i=0;
          }
          c=(char)byte_read;
          destination.write(c);
```

```
        }
      }
      // No matter what happens, always close any streams
we've opened.
      finally {
        if (source != null)
          try { source.close(); } catch (IOException e) { ; }
        if (destination != null)
          try { destination.close(); } catch (IOException e)
{ ; }
      }
    }

    // File.getParent() can return null when the file is spec-
ified without
    // a directory or is in the root directory.
    // This method handles those cases.
    private static File parent(File f) {
      String dirname = f.getParent();
      if (dirname == null) {
        if (f.isAbsolute()) return new File(File.separator);
        else return new File(System.getProperty("user.dir"));
      }
      return new File(dirname);
    }

    public static void main(String[] args) {
      if (args.length != 2)
        System.err.println("Usage: java convert " +
                   "<source file> <destination file>");
      else {
        try { copy(args[0], args[1]); }
        catch (IOException e) { Sys-
tem.err.println(e.getMessage()); }
      }
    }
}

class convertException extends IOException {
    public convertException(String msg) { super(msg); }
}
```

The key code is:

```
int i=0;
```

- Variable to count how many characters we have on a line. We have to break the lines somewhere.

```
byte carriage_return=13;
byte line_feed=10;
```

- Variables for carriage return and line feed.

```
char c;
```

- Variable to hold the character to be written.

```
int byte_read;
```

- Read returns integers.

```
while(true)
{
byte_read = source.read();
if (byte_read == -1) break;
```

- Read returns -1 at end of file.

```
i++;
```

- Increment the character count.

```
if ( byte_read < 32 || byte_read > 127 )
byte_read=32 ;
```

- Convert non-printing characters to blank.

```
if ( i>60 && byte_read==32 )
{
c=(char)carriage_return;
destination.write(c);
c=(char)line_feed;
destination.write(c);
i=0;
}
```

- Split the output line at the first blank character after position 60.

```
c=(char)byte_read;
destination.write(c);
```

- Convert to character and write out.

The program can be run with the following command line:

```
java convert before.doc after.txt
```

9.8 Problems

1. Write a program that writes a file that contains the numbers 1 through 10 as integer values. Examine this file with an editor. What do you notice?

2. Now write a program to read in these values and print them out to the screen.

3. Repeat problem one but now use single precision reals.

4. Repeat 2 with the file containing real numbers.

5. Chapter 17 in the third edition of the Deitel book has some i/o examples. Have a look at those to see how to create a sequential file and read it back in.

10
Threads

A person with one watch knows what time it is; a person with two watches is never sure
Proverb.

Aims

The aim of this chapter is to introduce the concepts and ideas involved in using threads in Java.:–

extends Thread;

implements Runnable;

sleep;

static variables;

synchronized;

priority;

init(), start(), stop(), paint(), update(), repaint(), yield();

10 Threads

The computer systems that you use runs many processes. It is not uncommon to:–

- be listening to a cd;
- downloading a file using ftp;
- printing a file;
- reading your mail;

at the same time.

If you are using a unix system try:–

- ps -ef

If you use Windows try

- [CTRL] [ESC]

to bring up a list of tasks.

Some programming languages offer support for writing programs that can multitask. Modula 2 offers coroutines, Ada has tasks, and Java has threads.

This will enable us to write Java programs that run each of the following as a separate thread:–

- download data from a remote system
- play sound
- interact with a user
- carry out calculations
- display the output from these calculations as the calculations run

10.1 Example 1 – extends Thread

The first example creates two threads and starts them running. The main loop of the program then gets the current thread name and prints it out.

```
public class thread01 extends Thread
{

   public static void main(String[] args)
   {
      thread01 t1=new thread01();
      thread01 t2=new thread01();
      t1.start();
      t2.start();
   }

   public void run()
   {
      for(;;)
      {
         System.out.println(Thread.currentThread().getName());
      }
   }
}
```

Let us look at the program in more depth.

```
public class thread01 extends Thread
```

This creates our own class called thread01 that is an extension of the inbuilt class thread.

```
     thread01 t1=new thread01();
     thread01 t2=new thread01();
```

These two statements create two objects of type thread01. We use the class name as the constructor.

```
     t1.start();
     t2.start();
```

These two statements start the threads running.

```
        System.out.println(Thread.currentThread().getName());
```

This is the only statement that is in the run procedure. It will get the name of the currently executing thread and print it out.

The output of this program will depend on the system that you run it on.

10.2 Example 2 – Extends Thread

This example is a simple variant of the first.

```
public class thread02 extends Thread
{
   int delay;

   thread02(int t)
   {delay=t;}

   public static void main(String[] args)
   {
      thread02 t1=new thread02(50);
      thread02 t2=new thread02(100);
      t1.start();
      t2.start();
   }

   public void run()
   {
      try
      {
         for(;;)
         {
            Sys-
tem.out.println(Thread.currentThread().getName());
            sleep(delay);
         }
      }
      catch(InterruptedException e)
      {  return;   }
   }
}
```


The first thing of interest is the addition of a class variable delay. Whenever we create an object of this class we will also create a variable called delay.

```
thread02(int t)
{delay=t;}
```

We now supply our own constructor for an object of type thread02. The constructor tales one argument and will be used to provide a value for the class variable delay.

```
thread02 t1=new thread02(50);
thread02 t2=new thread02(100);
```

We now create two thread objects and provide initial values for delay. The time is in milli-seconds.

```
          Sys-
tem.out.println(Thread.currentThread().getName());
          sleep(delay);
```

The run method gets the name of the cuurently executing thread and then calls the sleepmethod with the corresponding value for delay.

10.3 Example 3 – implements Runnable

The next two examples are simple variants of the first two examples. They utilise the second way of using threads in Java. The first way uses simple class extension. This way is not appropriate for many problems. The following example allows us to subclass another class if we want. The drawback with this way of doing things is that the Runnable interface does not provide start, stop, suspend and resume methods – the only method implemented is the run method.

```
public class thread03 implements Runnable
{

   public static void main(String[] args)
   {
      Runnable t1=new thread03();
      Runnable t2=new thread03();
      new Thread(t1).start();
      new Thread(t2).start();
   }

   public void run()
   {
      for(;;)
      {
         System.out.println(Thread.currentThread().getName());
      }
   }
}
```

Let us look at this example now.

10.4 Example 4 – Implements Runnable

This is the equivalent way of doing the second example.

```
public class thread04 implements Runnable
{
  int delay;

  thread04(int t)
  {
    delay=t;
  }

  public static void main(String[] args)
  {
    Runnable t1=new thread04(50);
    Runnable t2=new thread04(100);
    new Thread(t1).start();
    new Thread(t2).start();
  }

  public void run()
  {
    try
    {
      for(;;)
      {
        System.out.println(Thread.currentThread().getName());
        Thread.sleep(delay);
      }
    }
    catch(InterruptedException e)
    {
      return;
    }
  }
}
```

10.5 Example 5 – static variable in a thread

This example looks at some of the problems that can occur when using static variables

```
public class thread05 extends Thread
{
  static int i;

  public static void main(String[] args)
  {
    thread05 t1=new thread05();
    thread05 t2=new thread05();
    t1.start();
    t2.start();
  }
```



```
  public void run()
  {
    for(;;)
    {
      i=0;
      i++;
      if(i==0)
      {
        System.out.print(Thread.currentThread().getName());
        System.out.print("  ");
        System.out.println(i);
      }
    }
  }
}
```

10.6 Example 6 – synchronized

This example shows how to avoid potential problems when updating static class variables.

```
public class thread07 extends Thread
{
  static int i;

  public static void main(String[] args)
  {
    thread07 t1=new thread07();
    thread07 t2=new thread07();
    t1.start();
    t2.start();
  }

  public void run()
  {
    for(;;)
    {
      synchronized (this)
      {
        i=0;
        i++;
        if(i==0)
        {
          Sys-
tem.out.print(Thread.currentThread().getName());
          System.out.print("  ");
          System.out.println(i);
        }
      }
    }
  }
```

```
}
```

10.7 Example 7 –yield

This example shows how to get a degree of predictability in the behaviour of programs that use threads.

```
public class thread08 extends Thread
{

  public static void main(String[] args)
  {
    thread08 t1=new thread08();
    thread08 t2=new thread08();
    t1.start();
    t2.start();
  }

  public void run()
  {
    for(;;)
    {
      System.out.println(Thread.currentThread().getName());
      Thread.yield();
    }
  }
}
```

10.8 Example 8 – thread priority

This example looks at using threads and providing them with priorities.

```
public class thread09 extends Thread
{

  public static void main(String[] args)
  {
    thread09 t1=new thread09();
    thread09 t2=new thread09();
    t1.setPriority(10); t1.start();
    t2.setPriority(1);  t2.start();
  }

  public void run()
  {
    for(;;)
    {
      System.out.println(Thread.currentThread().getName());
      Thread.yield();
    }
  }
}
```


10.9 Problems

The important thing with these examples is to see them actually running. See if you can predict what the output will be.

The next thing is to try all of the examples out on another platform. Do you think that you will get the same results.

Compare the actual output for each of the examples on the various systems that you have access to.

Chapter 15 of the Deitel book has a coverage of multithreading. They have a number of examples including one implementing a circular buffer.

10.10 Bibliography

The two main types of sources are texts on operating systems and programming langauges that support real time programming. Ada 95 and Modula 2 offer support in this area.

Barnes J., *Programming in Ada 95*, Addsion Wesley.

The chapter on tasking provides a description of the way that Ada handles multitasking.

Christian K., *A Guide to Modula 2*, Springer Verlag.

The chapter on coroutines is worth looking at.

Deitel H. M., *An Introduction to Operating Systems*, Addison Wesley.

There are several chapters that look at the whole area of multi-processing, asynchronous and snychronous processes. Easy read. Very comprehensive bibliography.

The following faq is excellent.

- http://www.best.com/~bos/threads-faq/

Searching Amazon will reveal a fascinating spread of books on threads. Try it!

11
Introduction to Graphics Programming

A picture paints a thousand words.
anon

Aims

The aim of this chapter is to introduce the concepts and ideas involved in using the facilities offered in Java for graphical output. The coverage looks at what was available in the original 1.0.x JDK and the facilities in later releases. There is a coverage of:

Basic graphics concepts:

vector vs raster graphics, pixels, bit maps – gif vs jpg, fonts, coordinate space, user space, screen resolution, colour and colour models, scanning, integer arithmetic, real arithmetic, flicker, double buffering, selective erasure, device contexts, clipping, rendering.

java.awt

java.awt.Graphics

java.awt.Graphics2D

The imaging models supported by the Java 2D API:

The original producer consumer model

The immediate model introduced in the Java 2 JDK

The pipeline model in the Java Advanced Imaging API.

11 Introduction to Graphics Programming

This chapter looks first at a small number of concepts that are essential for successful use of computer generated graphics.

We start with a coverage of what was available in the original 1.0.x release of Java, and also the developments that have taken place since.

Java has moved on a long way from the first offerings in the 1.0.x JDK.

11.1 Vector vs Raster Graphics

There are two main ways of doing graphics. One involves using x-y coordinates and then drawing lines between them. The second involves actually looking at points and whether or not they are displayed or not.

Consider a 10*10 display space. With vector graphics we could consider drawing a line between 1,1 and 9,9. With raster graphics we consider each row of 10 points and whether a point is on or not.

11.2 Pixels

With each x-y point we have the additional feature of a gray scale value or colour value.

11.3 Bit maps – gif vs jpg

These are the two bit map formats currently supported by Java. As jpg is smaller I recommend using this bit map format rather than gif.

11.4 Screen resolution

Screens come in a number of display sizes. The following are commonly available on the pc:–

- 640*480 – standard vga
- 800*600 – super vga
- 1024*768 – you need at least a 15 inch monitor.
- 1280*1024 – just about ok with a 17 inch, but I would consider 19 or 21 inch.

It is not worth considering the higher screen resolutions unless one has a 15 or 17 inch monitor. The text is very difficult to read on smaller monitors.

11.4.1 Interlaced vs non-interlaced

To give a stable display a computer screen must be refreshed within quite short times, i.e. the picture must be redrawn completely 60 times a second. With monitors with low scan rates it will often only be possible to support higher screen resolutions if the screen is interlaced. I had a cheap 14 inch monitor at home and I drove it in 1024*768 mode, but I could only do that by doing it in interlaced mode. It flickered quite badly.

In the public rooms the pc’s are all driven in 800*600 mode. Very clear and legible.

Most people find little flicker above 70 Hz.

11.5 Colour Models

There are several colour models. Some commnly used ones are:–

- RGB – red, green, blue: tends to be used with displays, additive;
- CMY – cyan, magenta, yellow: tends to be used with printers, subtractive;

If one needs faithful printed colour then one should use software that offers pantone colour support. You would typically print out test sheets with the name/number of each colour and then use that to provide the on-screen colour. This may not match very closely the printed colour.

11.6 Scanning

Take care when scanning. Take a 5*7 inch colour photo with 300 dpi resolution.

5*300 * 7*300 = 35*9 * 10,000 = 3,150,000

11.7 Coordinate spaces

The java coordinate space is screen based in java.awt 1.0.x release and the top left is 0,0.

There is also the concept of the problem coordinate space. We then need to map between the two.

11.8 Fonts

There is a limited choice of fonts with the early versions of the JDK and but we can use them with a range of settings. Things have changed considerably with later versions of the JDK.

11.9 Aliasing and Antialiasing

With simple raster plottng we get jagged edges or staircasing.

Naturally enough the term used to get rid of aliasing is antialiasing. We get better quality images if we use antialiasing techniques. Later versions of Java supported antialiasing. See Foley et al for a coverage of this area.

11.10 Device context

A concepts that is very useful in the graphics and windows programming area is that of a device context. All drawing can be done to a device context and this might be virtual or physical, e.g. the screen or a printer. In Java we use the Graphics object as our device context. You have seen this throughout the examples already. We will look at this area in more depth in this chapter.

11.11 Clipping

Clipping refers to the way in which a graphics object is displayed on the current area. Some of the object will be outside the clip region and will not be displayed. See Foley et al for a coverage of this subject.

11.12 Rendering

Rendering is the production of the actual output on a graphics device.

11.13 Putting it all togethor

A Java graphics context enables drawing on the screen. A Graphics object manages a graphics context by controlling how things are drawn. The applets so far have have used generally used the Graphics object g – which is the argument to paint.

Things have changed in this area. What follows is based on the early 1.0.x jdk. We will come back to this whole area in later chapters.

11.14 History

Things are more complicated than they need be in this area, due to the way that Java has evolved. As you know we have had three major release of the Java jdk.

- jdk 1.0.x
- jdk 1.1.x
- jdk 1.2.x

and Sun have announced jdk 1.3 on February 15th.

There have also been several release of Swing. As problems are discovered with the current versions Sun try and fix the problems. This makes working out the best way of doing something quite difficult in the graphics and windows programming area.

If you look at the books and examples that exist you will see several ways of achieving what appears to be the same end result.

This chapter starts by looking at ways of doing things using the original jdk 1.0.x style. This (hopefully) will show some of the problems that exist with the original release and why Sun had to make changes.

11.15 Example 1 – Bouncing Balls

```
import java.awt.*;

// Simple graphics example of a bouncing ball.
// Things to look at are the amount of flicker
// and the amount of time that the thread is put
// to sleep. This is in milliseconds. It is worth
// experimenting with this as the applet will behave
// differently on different systems.

public class graphic01 extends java.applet.Applet implements
Runnable
{
  double x,y,deltax,deltay;
  double xl=300;
  double yl=300;
  Thread ball;

  public void init()
  {
    setBackground(Color.white);
    deltax=3;
    deltay=3;
    x=Math.random()*xl;
    y=Math.random()*yl;
  }

  public void start()
  {
    if(ball==null)
    {
```

```
      ball=new Thread(this);
      ball.start();
    }
  }

  public void stop()
  {
    if(ball!=null)
    {
      ball.stop();
      ball=null;
    }
  }

  public void run()
  {
    for(;;)
    {
      x+=deltax;
      y+=deltay;
      if ( (x>=xl) | (x<=0) ) deltax=-deltax;
      if ( (y>=yl) | (y<=0) ) deltay=-deltay;
      repaint();
      try
      {
        Thread.sleep(1);
      }
      catch (InterruptedException e)
      {
      }
    }
  }

  public void paint(Graphics g)
  {
    g.setColor(Color.blue);
    g.fillOval((int)x,(int)y,20,20);
  }
}
```

Compiling with JDK1.2.2 generates a warning about size and stop. Try running this example on more than one platform if possible. Also try running within a web browser as well as using the appletviewer.

11.16 Example 2 – Bouncing Balls with integer arithmetic

Note that the naming convention doesn't match the example numbering. If I get time I will rename and renumber all examples. However this involves quite a lot of work on more than one platform.

```
import java.awt.*;
```

```
// Simple graphics example of a bouncing ball.
// Things to look at are the amount of flicker
// and the amount of time that the thread is put
// to sleep. This is in milliseconds. It is worth
// experimenting with this as the applet will behave
// differently on different systems.

// this example redoes the first example by using
// integer arithmetic throughout. things to look at
// are the differences in the speed of the ball
// and also looking at varying the sleep time.

public class graphic03 extends java.applet.Applet implements
Runnable
{
   int x,y,deltax,deltay;
   int xl=300;
   int yl=300;
   Thread ball;

   public void init()
   {
      setBackground(Color.white);
      deltax=3;
      deltay=3;
      x=(int)(Math.random()*xl);
      y=(int)(Math.random()*yl);
   }

   public void start()
   {
      if(ball==null)
      {
         ball=new Thread(this);
         ball.start();
      }
   }

   public void stop()
   {
      if(ball!=null)
      {
         ball.stop();
         ball=null;
      }
   }

   public void run()
   {
      for(;;)
```

```
    {
      x+=deltax;
      y+=deltay;
      if ( (x>=xl) | (x<=0) ) deltax=-deltax;
      if ( (y>=yl) | (y<=0) ) deltay=-deltay;
      repaint();
      try
      {
        Thread.sleep(1);
      }
      catch (InterruptedException e)
      {
      }
    }
  }

  public void paint(Graphics g)
  {
    g.setColor(Color.blue);
    g.fillOval(x,y,20,20);
  }
}
```

Compiling with -deprecation and JDK 1.2.2 generates a warning about stop.

11.17Example 3 – Bouncing Balls with double buffering

```
import java.awt.*;

// simple bouncing ball. this example uses an off screen
// area to do the update. this technique is common in graph-
ics
// and is called double buffering. the other technique that
this example
// show is that of so called selective erasure. This is an-
other
// common programming technique in the graphics area.
// rather than repaint the whole screen only the part that
// has to change is redrawn.
// again worth looking at the sleep times.
// the example has one small snag. the selective erase
// doesn’t work correctly.

public class graphic02 extends java.applet.Applet implements
Runnable
{
  double x,y,deltax,deltay,oldx,oldy;
  double xl=300;
  double yl=300;
  Thread ball;
  Image bgi;
  Graphics bgg;
```

```
public void init()
{
   setBackground(Color.white);
   deltax=3;
   deltay=3;
   x=Math.random()*xl;
   y=Math.random()*yl;
   bgi=createImage(this.size().width,this.size().height);
   bgg=bgi.getGraphics();
}

public void start()
{
   if(ball==null)
   {
      ball=new Thread(this);
      ball.start();
   }
}

public void stop()
{
   if(ball!=null)
   {
      ball.stop();
      ball=null;
   }
}

public void paint(Graphics g)
{
   bgg.setColor(Color.white);
   bgg.fillOval((int)oldx,(int)oldy,20,20);

   bgg.setColor(Color.blue);
   bgg.fillOval((int)x,(int)y,20,20);

   g.drawImage(bgi,0,0,this);
   oldx=x;
   oldy=y;
}

public void update(Graphics g)
{
   paint(g);
}

public void run()
{
```

```
    for(;;)
    {
      x+=deltax;
      y+=deltay;
      if ( (x>=xl) | (x<=0) ) deltax=-deltax;
      if ( (y>=yl) | (y<=0) ) deltay=-deltay;
      repaint();
      try
      {
        Thread.sleep(1);
      }
      catch (InterruptedException e)
      {
      }
    }
  }
}
```

Compiling with JDK 1.2.2 and -deprecation generates warning about size and stop. Running the applet on Gum using the appletviewer is very slow. Running on a P166 with 64 Mb of memory is faster, but there are still problems with *trails* of where the ball has been. Running off the college web server with Netscape on the pc has the same trail behaviour.

11.18 Examle 4 – Bouncing Balls with integer arithmetic and double buffering

```
import java.awt.*;

// simple bouncing ball. this example uses an off screen
// area to do the update. this technique is common in graph-
ics
// and is called int buffering. the other technique that
this example
// show is that of so called selective erasure. This is an-
other
// common programming technique in the graphics area.
// rather than repaint the whole screen only the part that
// has to change is redrawn.
// again worth looking at the sleep times.
// the example has one small snag. the selective erase
// doesn't work correctly.

// arithmetic now made integer throughout.
// again look at the sleep time and the speed or the
// bouncing ball.

public class graphic04 extends java.applet.Applet implements
Runnable
{
  int x,y,deltax,deltay,oldx,oldy;
  int xl=300;
  int yl=300;
```

```
Thread ball;
Image bgi;
Graphics bgg;

public void init()
{
   setBackground(Color.white);
   deltax=3;
   deltay=3;
   x=(int)(Math.random()*xl);
   y=(int)(Math.random()*yl);
   bgi=createImage(this.size().width,this.size().height);
   bgg=bgi.getGraphics();
}

public void start()
{
   if(ball==null)
   {
      ball=new Thread(this);
      ball.start();
   }
}

public void stop()
{
   if(ball!=null)
   {
      ball.stop();
      ball=null;
   }
}

public void paint(Graphics g)
{
   bgg.setColor(Color.white);
   bgg.fillOval(oldx,oldy,20,20);

   bgg.setColor(Color.blue);
   bgg.fillOval(x,y,20,20);

   g.drawImage(bgi,0,0,this);
   oldx=x;
   oldy=y;
}
public void update(Graphics g)
{
   paint(g);
}
```

```
  public void run()
  {
    for(;;)
    {
      x+=deltax;
      y+=deltay;
      if ( (x>=xl) | (x<=0) ) deltax=-deltax;
      if ( (y>=yl) | (y<=0) ) deltay=-deltay;
      repaint();
      try
      {
        Thread.sleep(1);
      }
      catch (InterruptedException e)
      {
      }
    }
  }
}
```

This is now quite jerky running on the appletviewer on the pc.

11.19 Example 6 – Loading jpg images – static display

```
import java.awt.Graphics;
import java.awt.Image;

public class ian01  extends java.applet.Applet
{
  Image image01;

  public void init()
  {
    image01=getImage(getCodeBase(),"ian01.jpg");
  }

  public void paint(Graphics g)
  {
    g.drawImage(image01,10,10,this);
  }
}
```

Use Netscape to find some jpegs and ftp them to Gum. Have a look at their size. Other things to consider are colour information. The three most common are 8 bit (256 colours) 16 bit and 24 bit colour. 8 bit colour isn't too bad, especially for thumbnails.

11.20 Example 7 – Loading image – simple scaling

```
import java.awt.Graphics;
import java.awt.Image;

public class jane extends java.applet.Applet
{
```



```
   Image janeimg;

   public void init()
   {
      janeimg=getImage(getCodeBase(),"janet.gif");
   }

   public void paint(Graphics g)
   {
      int iwidth  = janeimg.getWidth(this);
      int iheight = janeimg.getHeight(this);

   int xpos=10;

   // 25%

   g.drawImage(janeimg,xpos,10,iwidth/4,iheight/4,this);

   // 50%

   xpos += (iwidth/4)+10;

   g.drawImage(janeimg,xpos,10,iwidth/2,iheight/2,this);

   // 100%

   xpos += (iwidth/2)+10;

   g.drawImage(janeimg,xpos,10,this);

   }

}
```

11.21 Example 8 – Moving image

```
import java.awt.Graphics;
import java.awt.Image;

public class joan02 extends java.applet.Applet
{
   Image img;

   public void init()
   {
      img=getImage(getCodeBase(),"joan01.gif");
   }
```

```
   public void paint(Graphics g)
   {
      int iwidth  = img.getWidth(this);
      int iheight = img.getHeight(this);
      int xpos;
      int ypos=10;

      for (xpos=10;xpos<400;xpos+=50)
      {
         g.drawImage(img,xpos,ypos,this);
         ypos += +50;
         try {Thread.sleep(100);}
         catch(InterruptedException e) {}
      }
   }
}
```

11.22 Basic Drawing Methods

There are a number of basic drawing methods and they are given below with examples.

11.22.1 Lines – g.drawline(x1,y1,x2,y2)

```
import java.awt.*;

public class c1140 extends java.applet.Applet
{

   public void init()
   {
      setBackground(Color.white);
   }

   public void paint(Graphics g)
   {
      int x1=0;
      int y1=0;
      int x2=100;
      int y2=200;
      g.setColor(Color.blue);
      g.drawLine(x1,y1,x2,y2);
   }
}
```

11.22.2 Rectangles – g.drawRect(xstart,ystart,width,height) g.fillRect(x,y,w,h)

```
import java.awt.*;

public class c1142 extends java.applet.Applet
{
```



```
   public void init()
   {
      setBackground(Color.white);
   }

   public void paint(Graphics g)
   {
      g.setColor(Color.blue);
      g.drawRect(10,10,100,100);
   }
}
import java.awt.*;

public class c1143 extends java.applet.Applet
{

   public void init()
   {
      setBackground(Color.white);
   }

   public void paint(Graphics g)
   {
      g.setColor(Color.blue);
      g.drawRect(10,10,100,100);
      g.fillRect(20,20,90,90);
   }
}
```

11.22.3 Rounded Rectangles – g.drawRoundRect(xstart,ystart,w,h,xcurve,ycurve)

```
import java.awt.*;

public class c1144 extends java.applet.Applet
{

   public void init()
   {
      setBackground(Color.white);
   }

   public void paint(Graphics g)
   {
      g.setColor(Color.blue);
      g.drawRoundRect(10,10,100,100,20,20);
   }
}
```

11.22.4 3D Effects – g.draw3Drect(x,y,w,h,true)

This method has to be repeated several times to obtain the desired effect. true raises the rectangle and false gives the effect of a pushed button.

```
import java.awt.*;

public class c1145 extends java.applet.Applet
{

  public void init()
  {
    setBackground(Color.white);
  }

  public void paint(Graphics g)
  {
    g.setColor(Color.blue);
    g.draw3DRect(10,10,100,100,true);
    g.draw3DRect(11,11,98,98,true);
    g.draw3DRect(12,12,96,96,true);
    g.draw3DRect(13,13,94,94,true);
  }
}
```

11.22.5 Polygons

There are two ways of doing this:–

- using arrays
- using the polygon class

The following two examples show both methods.

```
import java.awt.*;

public class c1146 extends java.applet.Applet
{

  public void init()
  {
    setBackground(Color.white);
  }

  public void paint(Graphics g)
  {
    int x[]={20,30,40,40,30,20,10,10,20};
    int y[]={10,10,20,30,40,40,30,20,10};
    g.setColor(Color.blue);
    g.drawPolygon(x,y,9);
  }
}

import java.awt.*;
```

```
public class c1147 extends java.applet.Apple
{

  public void init()
  {
    setBackground(Color.white);
  }

  public void paint(Graphics g)
  {
    Polygon octagon=new Polygon();
    octagon.addPoint(20,10);
    octagon.addPoint(30,10);
    octagon.addPoint(40,20);
    octagon.addPoint(40,30);
    octagon.addPoint(30,40);
    octagon.addPoint(20,40);
    octagon.addPoint(10,30);
    octagon.addPoint(10,20);
    octagon.addPoint(20,10);
    g.setColor(Color.blue);
    g.drawPolygon(octagon);
  }
}
```

11.22.6 Ovals – g.drawOval(x,y,w,h) and g.fillOval(x,y,w,h)

Left as an exercise.

11.22.7 Arcs – g.drawArc(x,y,w,h,start,end) and g.fillArc(x,y,w,h,s,e)

Left as an exercise.

11.22.8 Colour – Color

The default colour model supported by Java is the RGB model. R stands for red, g for green and b for blue! Each colour can have an integer value in the range 0 to 255. This gives 16 million colours. Your monitor of course may not be able to display that number of colours. It is an additive system. There are the following predefined colours in Java:–

Colour	Red	Green	Blue
Color.white	255	255	255
Color.black	0	0	0
Color.lightgray	192	192	192
Color.gray	128	128	128
Color.darkgray	64	64	64
Color.red	255	0	0
Color.green	0	255	0
Color.blue	0	0	255
Color.yellow	255	255	0
Color.magenta	255	0	255

Color.cyan	0	255	255
Color.pink	255	175	175
Color.orange	255	200	0

The other supported colour model is the hsb model (hue, saturation, brightness). There are additional methods to support this model.

The following program cycles through the colours in steps of 10.

```
import java.awt.*;

public class c1160 extends java.applet.Applet
{

  public void init()
  {
    setBackground(Color.white);
  }

  public void paint(Graphics g)
  {
    for (int red=0;red<256;red=red+10)
    {
      for (int green=0;green<256;green=green+10)
      {
        for (int blue=0;blue<256;blue=blue+10)
        {
          Color yeuk=new Color(red,green,blue);
          g.setColor(yeuk);
          g.fillOval(10,10,100,100);
        }
      }
    }
  }
}
```

11.22.9 Texts and Fonts

Java provides support for a number of widely available fonts. The following table gives names on three platforms.

Java name	X-Windows	Windows	Macintosh
Helvetica	adobe-helvetica	arial	
TimesRoman	adobe-times	times new roman	
Courier	adobe courier	courier new	
Dialog	b&h-lucida	ms sans serif	
DialogInput	b&h-lucidatypewriter	ms sans serif	
ZapfDingbats	itc-zapfdingbats	windings	
default	misc-fixed	arial	

We can also use the fonts in four variations:–

- PLAIN

- BOLD
- ITALIC
- BOLDITALIC

The following example prints out some text in some of the above fonts in each of the four variations.

```
import java.awt.*;
public class c1170 extends java.applet.Applet
{
  String s;

  public void showfont(String s)
  {
    Font f=new Font(s,Font.PLAIN,20);
    setFont(f);
    add(new Label (“ plain”));
    f=new Font(s,Font.BOLD,20);
    setFont(f);
    add(new Label (“ bold”));
    f=new Font(s,Font.ITALIC,20);
    setFont(f);
    add(new Label (“ italic”));
    f=new Font(s,Font.BOLD + Font.ITALIC,20);
    setFont(f);
    add(new Label (“ bold italic”));
  }

  public void start()
  {
    s="Helvetica";
    showfont(s);
    s="TimesRoman";
    showfont(s);
    s="Courier";
    showfont(s);
    s="Dialog";
    showfont(s);
  }
}
```

11.23AWT 1.0.x

The package java.awt contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. Some components fire events when a user interacts with the components. The AWTEvent class and its subclasses are used to represent the events that AWT components can fire. A container is a component that can contain components and other containers. A container can also have a layout manager that controls the visual placement of components in the container. The AWT package contains several layout manager classes and an interface for building your own layout manager.

11.23.1 Interface Summary

ActiveEvent

- An interface for events that know how dispatch themselves.

Adjustable

- The interface for objects which have an adjustable numeric value contained within a bounded range of values.

Composite

- The Composite interface, along with CompositeContext, defines the methods to compose a draw primitive with the underlying graphics area.

CompositeContext

- The CompositeContext interface defines the encapsulated and optimized environment for a compositing operation.

ItemSelectable

- The interface for objects which contain a set of items for which zero or more can be selected.

LayoutManager

- Defines the interface for classes that know how to layout Containers.

LayoutManager2

- Defines an interface for classes that know how to layout Containers based on a layout constraints object.

MenuContainer

- The super class of all menu related containers.

Paint

- This Paint interface defines how color patterns can be generated for Graphics2D operations.

PaintContext

- The PaintContext interface defines the encapsulated and optimized environment to generate color patterns in device space for fill or stroke operations on a Graphics2D.

PrintGraphics

- An abstract class which provides a print graphics context for a page.

Shape

- The Shape interface provides definitions for objects that represent some form of geometric shape.

Stroke

- The Stroke interface allows a Graphics2D object to obtain a Shape that is the decorated outline, or stylistic representation of the outline, of the specified Shape.

Transparency

- The Transparency interface defines the common transparency modes for implementing classes.

11.23.2 Class Summary

AlphaComposite

- This AlphaComposite class implements the basic alpha compositing rules for combining source and destination pixels to achieve blending and transparency effects with graphics and images.

AWTEvent

- The root event class for all AWT events.

AWTEventMulticaster

- A class which implements efficient and thread-safe multi-cast event dispatching for the AWT events defined in the java.awt.event package.

AWTPermission

- This class is for AWT permissions.

BasicStroke

- The BasicStroke class defines a basic set of rendering attributes for the outlines of graphics primitives.

BorderLayout

- A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.

Button

- This class creates a labeled button.

Canvas

- A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.

CardLayout

- A CardLayout object is a layout manager for a container.

Checkbox

- A check box is a graphical component that can be in either an “on” (true) or “off” (false) state.

CheckboxGroup

- The CheckboxGroup class is used to group together a set of Checkbox buttons.

CheckboxMenuItem

- This class represents a check box that can be included in a menu.

Choice

- The Choice class presents a pop-up menu of choices.

Color

- A class to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.

Component

- A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.

ComponentOrientation

- The ComponentOrientation class encapsulates the language-sensitive orientation that is to be used to order the elements of a component or of text.

Container

- A generic Abstract Window Toolkit(AWT) container object is a component that can contain other AWT components.

Cursor

- A class to encapsulate the bitmap representation of the mouse cursor.

Dialog

- A Dialog is a top-level window with a title and a border that is typically used to take some form of input from the user.

Dimension

- The Dimension class encapsulates the width and height of a component (in integer precision) in a single object.

Event

- Event is a platform-independent class that encapsulates events from the platform's Graphical User Interface in the Java 1.0 event model.

EventQueue

- EventQueue is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes.

FileDialog

- The FileDialog class displays a dialog window from which the user can select a file.

FlowLayout

- A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph.

Font

- The Font class represents fonts.

FontMetrics

- The FontMetrics class defines a font metrics object, which encapsulates information about the rendering of a particular font on a particular screen.

Frame

- A Frame is a top-level window with a title and a border.

GradientPaint

- The GradientPaint class provides a way to fill a Shape with a linear color gradient pattern.

Graphics

- The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

Graphics2D

- This Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

GraphicsConfigTemplate

- The GraphicsConfigTemplate class is used to obtain a valid GraphicsConfiguration.

GraphicsConfiguration

- The GraphicsConfiguration class describes the characteristics of a graphics destination such as a printer or monitor.

GraphicsDevice

- The GraphicsDevice class describes the graphics devices that might be available in a particular graphics environment.

GraphicsEnvironment

- The GraphicsEnvironment class describes the collection of GraphicsDevice objects and Font objects available to a Java(tm) application on a particular platform.

GridBagConstraints

- The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

GridBagLayout

- The GridBagLayout class is a flexible layout manager that aligns components vertically and horizontally, without requiring that the components be of the same size.

GridLayout

- The GridLayout class is a layout manager that lays out a container's components in a rectangular grid.

Image

- The abstract class Image is the superclass of all classes that represent graphical images.

Insets

- An Insets object is a representation of the borders of a container.

Label

- A Label object is a component for placing text in a container.

List

- The List component presents the user with a scrolling list of text items.

MediaTracker

- The MediaTracker class is a utility class to track the status of a number of media objects.

Menu

- A Menu object is a pull-down menu component that is deployed from a menu bar.

MenuBar

- The MenuBar class encapsulates the platform's concept of a menu bar bound to a frame.

MenuComponent

- The abstract class MenuComponent is the superclass of all menu-related components.

MenuItem

- All items in a menu must belong to the class MenuItem, or one of its subclasses.

MenuShortcut

- A class which represents a keyboard accelerator for a MenuItem.

Panel

- Panel is the simplest container class.

Point

- A point representing a location in (x, y) coordinate space, specified in integer precision.

Polygon

- The Polygon class encapsulates a description of a closed, two-dimensional region within a coordinate space.

PopupMenu

- A class that implements a menu which can be dynamically popped up at a specified position within a component.

PrintJob

- An abstract class which initiates and executes a print job.

Rectangle

- A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's top-left point (x, y) in the coordinate space, its width, and its height.

RenderingHints

- The RenderingHints class contains rendering hints that can be used by the Graphics2D class, and classes that implement BufferedImageOp and Raster.

RenderingHints.Key

- Defines the base type of all keys used to control various aspects of the rendering and imaging pipelines.

Scrollbar

- The Scrollbar class embodies a scroll bar, a familiar user-interface object.

ScrollPane

- A container class which implements automatic horizontal and/or vertical scrolling for a single child component.

SystemColor

- A class to encapsulate symbolic colors representing the color of GUI objects on a system.

TextArea

- A TextArea object is a multi-line region that displays text.

TextComponent

- The TextComponent class is the superclass of any component that allows the editing of some text.

TextField

- A TextField object is a text component that allows for the editing of a single line of text.

TexturePaint

- The TexturePaint class provides a way to fill a Shape with a texture that is specified as a BufferedImage.

Toolkit

- This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit.

Window

- A Window object is a top-level window with no borders and no menubar.

11.23.3 Exception Summary

AWTException

- Signals that an Absract Window Toolkit exception has occurred.

IllegalComponentStateException

- Signals that an AWT component is not in an appropriate state for the requested operation.

11.23.4 Error Summary

AWTError

- Thrown when a serious Abstract Window Toolkit error has occurred.

The original AWT has some problems associated with it, and you can see now why it was necessary to develop a better windowing development system. We will look at Swing in a later chapter. We will next look at some of the developments that have taken place with the original AWT.

11.23.5 java.awt.Graphics

The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on devices. You need to know about what is in this package as the other more recent packages will inherit from this package. A Graphics object has state information needed for the basic rendering operations that Java supports. This state information includes the following properties:

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function (XOR or Paint).

- The current XOR alternation color (see setXORMode(java.awt.Color)).

Coordinates are infinitely thin and lie between the pixels of the output device. Operations that draw the outline of a figure operate by traversing an infinitely thin path between pixels with a pixel-sized pen that hangs down and to the right of the anchor point on the path. This means:

- If you draw a figure that covers a given rectangle, that figure occupies one extra row of pixels on the right and bottom edges as compared to filling a figure that is bounded by that same rectangle.
- If you draw a horizontal line along the same y coordinate as the baseline of a line of text, that line is drawn entirely below the text, except for any descenders.

Operations that fill a figure operate by filling the interior of that infinitely thin path. Operations that render horizontal text render the ascending portion of character glyphs entirely above the baseline coordinate.

All coordinates that appear as arguments to the methods of this Graphics object are considered relative to the translation origin of this Graphics object prior to the invocation of the method. All rendering operations modify only pixels which lie within the area bounded by the current clip, which is specified by a Shape in user space and is controlled by the program using the Graphics object. This user clip is transformed into device space and combined with the device clip, which is defined by the visibility of windows and device extents. The combination of the user clip and device clip defines the composite clip, which determines the final clipping region. The user clip cannot be modified by the rendering system to reflect the resulting composite clip. The user clip can only be changed through the setClip or clipRect methods. All drawing or writing is done in bsthe current color, using the current paint mode, and in the current font.

11.23.5.1 Constructor Summary

Graphics()

- Constructs a new Graphics object.

11.23.5.2 Method Summary

abstract void clearRect(int x, int y, int width, int height)

- Clears the specified rectangle by filling it with the background color of the current drawing surface.

abstract void clipRect(int x, int y, int width, int height)

- Intersects the current clip with the specified rectangle.

abstract void copyArea(int x, int y, int width, int height, int dx, int dy)

- Copies an area of the component by a distance specified by dx and dy.

abstract Graphics create()

- Creates a new Graphics object that is a copy of this Graphics object.

Graphics create(int x, int y, int width, int height)

- Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.

abstract void dispose()

- Disposes of this graphics context and releases any system resources that it is using.

void draw3DRect(int x, int y, int width, int height, boolean raised)

- Draws a 3-D highlighted outline of the specified rectangle.

abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

- Draws the outline of a circular or elliptical arc covering the specified rectangle.

void drawBytes(byte[] data, int offset, int length, int x, int y)

- Draws the text given by the specified byte array, using this graphics context's current font and color.

void drawChars(char[] data, int offset, int length, int x, int y)

- Draws the text given by the specified character array, using this graphics context's current font and color.

abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)

- Draws as much of the specified image as is currently available.

abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)

- Draws as much of the specified image as is currently available.

abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)

- Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)

- Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)

- Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.

abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)

- Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.

abstract void drawLine(int x1, int y1, int x2, int y2)

- Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

abstract void drawOval(int x, int y, int width, int height)

- Draws the outline of an oval.

abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

- Draws a closed polygon defined by arrays of x and y coordinates.

void drawPolygon(Polygon p)

- Draws the outline of a polygon defined by the specified Polygon object.

abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)

- Draws a sequence of connected lines defined by arrays of x and y coordinates.

void drawRect(int x, int y, int width, int height)

- Draws the outline of the specified rectangle.

abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

- Draws an outlined round-cornered rectangle using this graphics context's current color.

abstract void drawString(AttributedCharacterIterator iterator, int x, int y)

- Draws the text given by the specified iterator, using this graphics context's current color.

abstract void drawString(String str, int x, int y)

- Draws the text given by the specified string, using this graphics context's current font and color.

void fill3DRect(int x, int y, int width, int height, boolean raised)

- Paints a 3-D highlighted rectangle filled with the current color.

abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

- Fills a circular or elliptical arc covering the specified rectangle.

abstract void fillOval(int x, int y, int width, int height)

- Fills an oval bounded by the specified rectangle with the current color.

abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)

- Fills a closed polygon defined by arrays of x and y coordinates.

void fillPolygon(Polygon p)

- Fills the polygon defined by the specified Polygon object with the graphics context's current color.

abstract void fillRect(int x, int y, int width, int height)

- Fills the specified rectangle.

abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

- Fills the specified rounded corner rectangle with the current color.

void finalize()

- Disposes of this graphics context once it is no longer referenced.

abstract Shape getClip()

- Gets the current clipping area.

abstract Rectangle getClipBounds()

- Returns the bounding rectangle of the current clipping area.

Rectangle getClipBounds(Rectangle r)

- Returns the bounding rectangle of the current clipping area.

Rectangle getClipRect()

- Deprecated. As of JDK version 1.1, replaced by getClipBounds().

abstract Color getColor()

- Gets this graphics context's current color.

abstract Font getFont()

- Gets the current font.

FontMetrics getFontMetrics()

- Gets the font metrics of the current font.

abstract FontMetrics getFontMetrics(Font f)

- Gets the font metrics for the specified font.

boolean hitClip(int x, int y, int width, int height)

- Returns true if the specified rectangular area intersects the bounding rectangle of the current clipping area.

abstract void setClip(int x, int y, int width, int height)

- Sets the current clip to the rectangle specified by the given coordinates.

abstract void setClip(Shape clip)

- Sets the current clipping area to an arbitrary clip shape.

abstract void setColor(Color c)

- Sets this graphics context's current color to the specified color.

abstract void setFont(Font font)

- Sets this graphics context's font to the specified font.

abstract void setPaintMode()

- Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.

abstract void setXORMode(Color c1)

- Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.

String toString()

- Returns a String object representing this Graphics object's value.

abstract void translate(int x, int y)

- Translates the origin of the graphics context to the point (x, y) in the current co-ordinate system

11.24Package java.awt.Graphics2D – JDK 1.2

This Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. This is the fundamental class for rendering 2-dimensional shapes, text and images on the Java platform. All coordinates passed to a Graphics2D object are specified in a device-independent coordinate system called User Space, which is used by applications. The Graphics2D object contains an AffineTransform object as part of its rendering state that defines how to convert coordinates from user space to device-dependent coordinates in Device Space.

Where possible use Graphics2D in preference to Graphics.

11.24.1 Rendering

The Rendering Process can be broken down into four phases that are controlled by the Graphics2D rendering attributes:

- Determine what to render.

- Constrain the rendering operation to the current Clip
- Determine what colors to render.
- Apply the colors to the destination drawing surface using the current Composite attribute in the Graphics2D context.

The three types of rendering operations are:

- Shape operations
- Text operations
- Image Operations

applied to a Graphics2D context.

11.24.2 Compatability

Whilst the rendering models are different between jdk 1.1.x and jdk 1.2.x (this supports anti aliasing) Sun have attempted to ensure backwards compatability when running legacy code under 1.2.x.

11.24.3 Constructor Summary

protected Graphics2D()

- Constructs a new Graphics2D object.

11.24.4 Method Summary

abstract void addRenderingHints(Map hints)

- Sets the values of an arbitrary number of preferences for the rendering algorithms.

abstract void clip(Shape s)

- Intersects the current Clip with the interior of the specified Shape and sets the Clip to the resulting intersection.

abstract void draw(Shape s)

- Strokes the outline of a Shape using the settings of the current Graphics2D context.

void draw3DRect(int x, int y, int width, int height, boolean raised)

- Draws a 3-D highlighted outline of the specified rectangle.

abstract void drawGlyphVector(GlyphVector g, float x, float y)

- Renders the text of the specified GlyphVector using the Graphics2D context's rendering attributes.

abstract void drawImage(BufferedImage img, BufferedImageOp op, int x, int y)

- Renders a BufferedImage that is filtered with a BufferedImageOp.

abstract boolean drawImage(Image img, AffineTransform xform, ImageObserver obs)

- Renders an image, applying a transform from image space into user space before drawing.

abstract void drawRenderableImage(RenderableImage img, AffineTransform xform)

- Renders a RenderableImage, applying a transform from image space into user space before drawing.

abstract void drawRenderedImage(RenderedImage img, AffineTransform xform)

- Renders a RenderedImage, applying a transform from image space into user space before drawing.

abstract void drawString(AttributedCharacterIterator iterator, float x, float y)

- Renders the text of the specified iterator, using the Graphics2D context's current Paint.

abstract void drawString(AttributedCharacterIterator iterator, int x, int y)

- Renders the text of the specified iterator, using the Graphics2D context's current Paint.

abstract void drawString(String s, float x, float y)

- Renders the text specified by the specified String, using the current Font and Paint attributes in the Graphics2D context.

abstract void drawString(String str, int x, int y)

- Renders the text of the specified String, using the current Font and Paint attributes in the Graphics2D context.

abstract void fill(Shape s)

- Fills the interior of a Shape using the settings of the Graphics2D context.

void fill3DRect(int x, int y, int width, int height, boolean raised)

- Paints a 3-D highlighted rectangle filled with the current color.

abstract Color getBackground()

- Returns the background color used for clearing a region.

abstract Composite getComposite()

- Returns the current Composite in the Graphics2D context.

abstract GraphicsConfiguration getDeviceConfiguration()

- Returns the device configuration associated with this Graphics2D.

abstract FontRenderContext getFontRenderContext()

- Get the rendering context of the Font within this Graphics2D context.

abstract Paint getPaint()

- Returns the current Paint of the Graphics2D context.

abstract Object getRenderingHint(RenderingHints.Key hintKey)

- Returns the value of a single preference for the rendering algorithms.

abstract RenderingHints getRenderingHints()

- Gets the preferences for the rendering algorithms.

abstract Stroke getStroke()

- Returns the current Stroke in the Graphics2D context.

abstract AffineTransform getTransform()

- Returns a copy of the current Transform in the Graphics2D context.

abstract boolean hit(Rectangle rect, Shape s, boolean onStroke)

- Checks whether or not the specified Shape intersects the specified Rectangle, which is in device space.

abstract void rotate(double theta)

- Concatenates the current Graphics2D Transform with a rotation transform.

abstract void rotate(double theta, double x, double y)

- Concatenates the current Graphics2D Transform with a translated rotation transform.

abstract void scale(double sx, double sy)

- Concatenates the current Graphics2D Transform with a scaling transformation Subsequent rendering is resized according to the specified scaling factors relative to the previous scaling.

abstract void setBackground(Color color)

- Sets the background color for the Graphics2D context.

abstract void setComposite(Composite comp)

- Sets the Composite for the Graphics2D context.

abstract void setPaint(Paint paint)

- Sets the Paint attribute for the Graphics2D context.

abstract void setRenderingHint(RenderingHints.Key hintKey, Object hintValue)

- Sets the value of a single preference for the rendering algorithms.

abstract void setRenderingHints(Map hints)

- Replaces the values of all preferences for the rendering algorithms with the specified hints.

abstract void setStroke(Stroke s)

- Sets the Stroke for the Graphics2D context.

abstract void setTransform(AffineTransform Tx)

- Sets the Transform in the Graphics2D context.

abstract void shear(double shx, double shy)

- Concatenates the current Graphics2D Transform with a shearing transform.

abstract void transform(AffineTransform Tx)

- Composes an AffineTransform object with the Transform in this Graphics2D according to the rule last-specified-first-applied.

abstract void translate(double tx, double ty)

- Concatenates the current Graphics2D Transform with a translation transform.

abstract void translate(int x, int y)

- Translates the origin of the Graphics2D context to the point (x, y) in the current coordinate system.

Methods inherited from class java.awt.Graphics include clearRect, clipRect, copyArea, create, create, dispose, drawArc, drawBytes, drawChars, drawImage, drawImage, drawImage, drawImage, drawImage, drawImage, drawLine, drawOval, drawPolygon, drawPolygon, drawPolyline, drawRect, drawRoundRect, fillArc, fillOval, fillPolygon, fillPolygon, fillRect, fillRoundRect, finalize, getClip, getClipBounds, getClipBounds, getClipRect, getColor, getFont, getFontMetrics, getFontMetrics, hitClip, setClip, setClip, setColor, setFont, setPaintMode, setXORMode, toString

11.25 Package java.awt.geom – JDK 1.2

Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry. Some important features of the package include classes for manipulating geometry, such as AffineTransform and the PathIterator interface which is implemented by all Shape objects, classes that implement the Shape interface, such as CubicCurve2D, Ellipse2D, Line2D, Rectangle2D, and GeneralShape, the Area class which provides mechanisms for add (union), subtract, intersect, and exclusiveOR operations on other Shape objects. Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

11.26 Package java.awt.im – JDK 1.2

Provides classes and an interface for the input method framework. This framework enables all text editing components to receive Japanese, Chinese, or Korean text input through input methods. An input method lets users enter thousands of different characters using keyboards with far fewer keys. Typically a sequence of several characters needs to be typed and then converted to create one or more characters. Text editing components can use this package and related classes in java.awt.event to support the on-the-spot input style.

11.27 Package java.awt.image.renderable – JDK 1.2

Provides classes and interfaces for producing rendering-independent images.

11.28 Package java.awt.print – JDK 1.2

Provides classes and interfaces for a general printing API. The API includes such features as the ability to specify document types, mechanisms for control of page setup and page formats, the ability to manage job control dialogs.

We will now look at rewriting our earlier examples to use the 1.2.x offerings in the graphics area. We need to cover a number of concepts first.

11.29 Java 2D API Overview

The Java 2DTM API enhances the graphics, text, and imaging capabilities of the Abstract Windowing Toolkit (AWT), enabling the development of richer user interfaces and new types of Java applications.

Along with these richer graphics, font, and image APIs, the Java 2D API supports enhanced color definition and composition, hit detection on arbitrary geometric shapes and text, and a uniform rendering model for printers and display devices.

The Java 2D API also enables the creation of advanced graphics libraries, such as CAD-CAM libraries and graphics or imaging special effects libraries, as well as the creation of image and graphic file read/write filters.

When used in conjunction with the Java Media Framework and other Java Media APIs, the Java 2D APIs can be used to create and display animations and other multimedia presentations. The Java Animation and Java Media Framework APIs rely on the Java 2D API for rendering support.

11.29.1 Enhanced Graphics, Text, and Imaging

Early versions of the AWT provided a simple rendering package suitable for rendering common HTML pages, but not full-featured enough for complex graphics, text, or imaging. As a simplified rendering package, the early AWT embodied specific cases of more general rendering concepts. The Java 2DTM API provides a more flexible, full-featured rendering package by expanding the AWT to support more general graphics and rendering operations.

For example, through the Graphics class you can draw rectangles, ovals, and polygons. Graphics2D enhances the concept of geometric rendering by providing a mechanism for rendering virtually any geometric shape. Similarly, with the Java 2D API you can draw styled lines of any width and fill geometric shapes with virtually any texture.

11.29.2 Rendering Model

The basic graphics rendering model has not changed with the addition of the Java 2D APIs. To render a graphic, you set up the graphics context and invoke a rendering method on the Graphics object.

The Java 2D API class Graphics2D extends Graphics to support more graphics attributes and provide new rendering methods.

The Java 2D API automatically compensates for differences in rendering devices and provides a uniform rendering model across different types of devices. At the application level, the rendering process is the same whether the target rendering device is a display or a printer.

11.29.3 Backward Compatibility and Platform Independence

The Java 2DTM API maintains backward compatibility with JDK 1.1 software. It is also architected so that applications can maintain platform-independence.

To ensure backward compatibility, the functionality of existing JDK graphics and imaging classes and interfaces was maintained. Existing features were not removed and no package designations were changed for existing classes. The Java 2D API enhances the functionality of the AWT by implementing new methods in existing classes, extending existing classes, and adding new classes and interfaces that don't affect the legacy APIs.

The Java 2D API functionality is delivered through an expanded graphics context, Graphics2D. To provide this extended graphics context while maintaining backward compatibility, Graphics2D extends the Graphic s class from the JDK 1.1 release.

The usage model of the graphics context remains unchanged. The AWT passes a graphics context to an AWT Component through the following methods:

- paint
- paintAll
- update
- print
- printAll
- getGraphics

A JDK 1.1 applet interprets the graphics context that's passed in as an instance of Graphics. To gain access to the new features implemented in Graphics2D, a Java 2D API-compatible applet casts the graphics context to a Graphics2D object:

```
public void Paint (Graphics g)
{
   Graphics2D g2 = (Graphics2D) g;
   ...
   ...
   g2.setTransform (t);
}
```

Note that we still invoke paint with Graphics and switch to Graphics2D with an explicit cast.

To enable the development of platform-independent applications, the Java 2D API makes no assumptions about the resolution, color space, or color model of the target rendering device. Nor does the Java 2D API assume any particular image file format.

Truly platform-independent fonts are possible only when the fonts are built-in (provided as part of the JDK software), or when they are mathematically or programmatically generated. The Java 2D API does not currently support built-in or mathematically generated fonts, but it does enable the programmatic definition of entire fonts through their glyph set. Each glyph can in turn be defined by a Shape that consists of line segments and curves. Many fonts of particular styles and sizes can be deriv ed from a single glyph set.

11.29.4 Setting Up the Graphics2D Context

To configure the Graphics2D context for rendering, you use the Graphics2D set methods to specify attributes such as the RenderingHints, Stroke, Paint, clipping path, Composite, and Transform.

11.29.5 Rendering Graphics Primitives

Graphics2D provides rendering methods for Shapes, Text, and Images:

- draw—strokes a Shape's path using the Stroke and Paint objects in the Graphics2D context.
- fill—fills a Shape using the Paint in the Graphics2D context.
- drawString—renders the specified text string using the Paint in the Graphics2D context.
- drawImage—renders the specified image.

To stroke and fill a shape, you must call both the draw and fill methods.

Graphics2D also supports the draw and fill methods from previous versions of the JDK software, such as drawOval and fillRect.

11.29.6 Managing and Manipulating Rasters

A BufferedImage object uses a Raster to manage its rectangular array of pixel data. The Raster class defines fields for the image's coordinate system—width, height, and origin. A Raster object itself uses two objects to manage the pixel data, a DataBuffer and a SampleModel. The DataBuffer is the object that stores pixel data for the raster, and the SampleModel provides the interpretation of pixel data from the DataBuffer.

11.29.7 Geometries

The Java 2DTM API provides several classes that define common geometric objects, such as points, lines, curves, and rectangles. These new geometry classes are part of the java.awt.geom package. For backward compatibility, the geometry classes tha t existed in previous versions of the JDK software, such as Rectangle, Point, and Polygon, remain in the java.awt package.

The Java 2D API geometries such as GeneralPath, Arc2D, and Rectangle2D implement the Shape interface defined in java.awt. Shape provides a common protocol for describing and inspecting geometric path objects. A new interface, PathIterator, defines methods for retrieving elements from a geometry.

11.29.8 Fonts and Text Layout

You can use the Java 2DTM API transformation and drawing mechanisms with text strings. In addition, the Java 2D API provides text-related classes that support fine-grain font control and sophisticated text layout. These include an enhanced Font c lass and the new TextLayout class.

11.29.9 Imaging

The Java 2DTM API supports three imaging models

- The producer/consumer (push) model provided in previous versions of the JDK software.
- The immediate mode model introduced in the Java 2 SDK software release.
- The pipeline (pull) model compatible with the immediate mode model and that will be fully implemented in the forthcoming Java Advanced Imaging API.

The following table contrasts the features of each of these imaging models.

	Push	Immediate	Pull
Class Interface	Image ImageProducer ImageConsumer ImageObserver	BufferedImage Ratser BufferedImageOp RatserOp	RenderableImage RenderableImageOp RenderedOp RenderableOp TiledImage
Pros	Driven by image availability processed incrementally	Simplest Programming interface Commonly used	Required data only Lazy evaluation
Cons	Transfer required more complex api	memory required for whole image complete image processing	more complex interface and implementation

This API supports accessing image data in a variety of storage formats and manipulating image data through several types of filtering operations.

11.29.10 Color

Color imaging is one of the fundamental components of any graphics system, and it is often a source of great complexity in the imaging model. The Java 2D API provides support for high-quality color output that is easy to use and allows advanced clients to make sophisticated use of color.

The key color management classes in the Java 2D API are ColorSpace, Color, ColorModel:

A ColorSpace represents a system for measuring colors, typically using three separate numerical values or components. The ColorSpace class contains methods for converting between the color space and two standard color spaces, CI EXYZ and RGB.

A Color is a fixed color, defined in terms of its components in a particular ColorSpace. To draw a Shape in a color, such as red, you pass a Color object representing that color to the Graphics2D cont ext. Color is defined in the java.awt package.

A ColorModel describes a particular way that pixel values are mapped to colors. A ColorModel is typically associated with an Image or BufferedImage and provides the information necessary to correctly interpret the pixel values. ColorModel is defined in the java.awt.image package.

11.29.11 ColorModels and Color Data and the BufferedImage Class

In addition to the Raster object for managing image data, the BufferedImage class includes a ColorModel for interpreting that data as color pixel values. The abstract ColorModel class defines methods for turning an image's pixel data into a color value in its associated ColorSpace.

The java.awt.image package provides four types of color models:

- PackedColorModel – An abstract ColorModel that represents pixel values that have color components embedded directly in the bits of an integer pixel. A DirectColorModel is a subclass of PackedColorModel.
- DirectColorModel – a ColorModel that represents pixel values that have RGB color components embedded directly in the bits of the pixel itself. DirectColorModel model is similar to an X11 TrueColor visual.
- ComponentColorModel – a ColorModel that can handle an arbitrary ColorSpace and an array of color components to match the ColorSpace.
- IndexColorModel – a ColorModel that represents pixel values that are indices into a fixed color map in the sRGB color space.

ComponentColorModel and PackedColorModel are new in the JavaTM 2 SDK software release.

11.29.12 Printing

The Java Printing API enables applications to:

- Print all AWT and Java 2DTM graphics, including composited graphics and images.
- Control document-composition functions such as soft collating, reverse order printing, and booklet printing.
- Invoke printer-specific functions such as duplex (two-sided) printing and stapling.
- Print on all platforms, including Windows and Solaris. This includes printers directly attached to the computer as well as those that the platform software is able to access using network printing protocols.

Not all of these features are supported in the Java 2 SDK Printing API and implementation. The API will be extended to support all of these features in future releases. For example, additional printer controls will be added by augmenting the set of named properties of a print job that the application can control.

Sun also make available a lot of sample programs. This is available at their web site. It was about 350 Kb in March 2000. I use these as a last resort. If you get stuck then have a look to see if there is an example that does some or all of what you are trying to do.

Redoing the earlier examples

We will now look at redoing some of the earlier examples to use the newer, preferred methods.

11.30 Simple bouncing ball

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class graphic11 extends java.applet.Applet implements
Runnable
{
   double x,y,deltax,deltay;
   double xl=300;
   double yl=300;
   Thread ball;

   public void init()
   {
      setBackground(Color.white);
      deltax=Math.random();
      deltay=Math.random();
      x=Math.random()*xl;
      y=Math.random()*yl;
   }

   public void start()
   {
      if(ball==null)
      {
         ball=new Thread(this);
         ball.start();
      }
   }

   public void stop()
   {
      if(ball!=null)
      {
         ball.stop();
         ball=null;
      }
   }

   public void run()
   {
      for(;;)
      {
         x+=deltax;
```



```
        y+=deltay;
        if ( (x>=xl) | (x<=0) ) deltax=-deltax;
        if ( (y>=yl) | (y<=0) ) deltay=-deltay;
        repaint();
        try
        {
          Thread.sleep(1);
        }
        catch (InterruptedException e)
        {
        }
      }
    }

    public void paint(Graphics g)
    {
      Graphics2D g2d = (Graphics2D) g;
      g2d.setColor(Color.blue);
      g2d.fillOval((int)x,(int)y,20,20);
    }

    public static void main(String argv[])
    {
      final graphic11 g11 = new graphic11();
      g11.init();
      JFrame jf = new JFrame(" Using Graphics2D ");
      jf.addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });
    }
}
```

In this example we add imports for the new event handling and we are also using components from Swing.

The first difference is that we use objects of ype Graphics2D now, rather than the original Graphics. The examples Sun make available still callpaint with an object of type Graphics and then explicitly cast within paint to Graphics2D.

The second difference is the addition of the main routine at the end. Within this routine we:

- create a variable of type graphic11
- initialise the variable
- create a JFrame variable
- register the JFrame as a window listener
- override windowClosing

11.30.1 Initialisation

public void init()

Called by the browser or applet viewer to inform this applet that it has been loaded into the system. It is always called before the first time that the start method is called.

A subclass of Applet should override this method if it has initialization to perform. For example, an applet with threads would use the init method to create the threads and the destroy method to kill them. The implementation of this method provided by the Applet class does nothing.

11.30.2 JFrame

public class JFrame extends Frame

implements WindowConstants, Accessible, RootPaneContainer

An extended version of java.awt.Frame that adds support for interposing input and painting behavior in front of the frame's children (see glassPane), support for special children that are managed by a LayeredPane (see rootPane) and for Swing MenuBars.

The JFrame class is slightly incompatible with java.awt.Frame. JFrame contains a JRootPane as it's only child. The contentPane should be the parent of any children of the JFrame.

11.30.3 addWindowListener

public void addWindowListener(WindowListener l)

Adds the specified window listener to receive window events from this window. If l is null, no exception is thrown and no action is performed.

11.30.4 Class WindowAdapter

public abstract class WindowAdapter extends Object implements WindowListener

An abstract adapter class for receiving window events. The methods in this class are empty. This class exists as convenience for creating listener objects.

Extend this class to create a WindowEvent listener and override the methods for the events of interest. (If you implement the WindowrListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you can only have to define methods for events you care about.)

Create a listener object using the extended class and then register it with a Window using the window's addWindowListener method. When the window's status changes by virtue of being opened, closed, activated or deactivated, iconified or deiconified, the relevant method in the listener object is invoked, and the WindowEvent is passed to it.

Experiment with the sleep time.

We will look into the above in more detail in later chapters.

11.31 Bouncing balls with selective erase

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class graphic12 extends java.applet.Applet implements
Runnable
```

```
{
   double x,y,deltax,deltay,oldx,oldy;
   double xl=300;
   double yl=300;
   Thread ball;
   BufferedImage bgi;
   Graphics bgg;

   public void init()
   {
      setBackground(Color.white);
      deltax=Math.random();
      deltay=Math.random();
      x=Math.random()*xl;
      y=Math.random()*yl;
      bgi=(BufferedImage)
createImage(this.size().width,this.size().height);
      bgg=bgi.getGraphics();
   }

   public void start()
   {
      if(ball==null)
      {
         ball=new Thread(this);
         ball.start();
      }
   }

   public void stop()
   {
      if(ball!=null)
      {
         ball.stop();
         ball=null;
      }
   }

   public void paint(Graphics g)
   {
      Graphics2D g2d = (Graphics2D) g;

      bgg.setColor(Color.white);
      bgg.fillOval((int)oldx,(int)oldy,20,20);

      bgg.setColor(Color.blue);
      bgg.fillOval((int)x,(int)y,20,20);

      g2d.drawImage(bgi,0,0,this);
      oldx=x;
```

```
      oldy=y;
   }

   public void update(Graphics g)
   {
      paint(g);
   }

   public void run()
   {
      for(;;)
      {
         x+=deltax;
         y+=deltay;
         if ( (x>=xl) | (x<=0) ) deltax=-deltax;
         if ( (y>=yl) | (y<=0) ) deltay=-deltay;
         repaint();
         try
         {
            Thread.sleep(10);
         }
         catch (InterruptedException e)
         {
         }
      }
   }

   public static void main(String argv[])
   {
      final graphic12 g12 = new graphic12();
      g12.init();
      JFrame jf = new JFrame(" Using Graphics2D ");
      jf.addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent e)
         {
            System.exit(0);
         }
      });
   }

}
```

The differences are:

- The init method creates a BufferedImage from an Image. An explicit cast is required.
- The paint method uses Graphics2D variables, and again we have an explicit cast.
- A main method has been added, which is identical with that used in 11.30.

Experiment with the sleep time.

11.32 Simple jpeg display

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class ian11  extends javax.swing.JApplet
{
  Image image01;
  BufferedImage bimage01;

  public void init()
  {
    image01 = getImage(getCodeBase(),"ian01.jpg");
  }

  public void displayimage(int w, int h,Graphics2D g2d)
  {
    int iw;
    int ih;
    BufferedImage bi  = (BufferedImage) createImage (w,h);
    Graphics2D    big = bi.createGraphics();
    iw=image01.getWidth(this);
    ih=image01.getHeight(this);
    big.drawImage(image01,10,10,iw,ih,this);
    g2d.drawImage(bi,0,0,this);
  }

  public void paint(Graphics g)
  {
    Graphics2D g2d = (Graphics2D) g;
    Dimension    d = getSize();
    displayimage(d.width,d.height,g2d);
  }

  public static void main(String argv[])
  {
    final ian11 i11 = new ian11();
    i11.init();
    JFrame jf = new JFrame(" Java 2d version ");
    jf.addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        System.exit(0);
```

```
        }
      });
   }
}
```

Differences include:

- We are now working with buffered images
- paint creates a Graphics2D context from a Graphics context.
- We pick up the size
- We call dispalyimage with the current Graphics2d context and size
- displayimage creates a buffered image and graphics 2d context based on the size from the calling routine.
- We get the height and width of the jpg file associated image.
- Draw the image
- Force the display with g2d.drawImage

The main method is identical to the last two examples in this section.

11.33 Simple line drawing

```
import java.awt.*;

public class c11401 extends java.applet.Applet
{

   public void init()
   {
      setBackground(Color.white);
   }

   public void paint(Graphics g)
   {
      Graphics2D g2d = (Graphics2D) g;
      int x1=0;
      int y1=0;
      int x2=100;
      int y2=200;
      g2d.setColor(Color.blue);
      g2d.drawLine(x1,y1,x2,y2);
   }
}
```

Very simple example. Minimal changes have been made in this case.

11.34 Summary

There is a lot to this subject. I have only provided a very brief coverage. It is essential to get a background to the field of computer graphics if you are going to use Java for graphical output.

11.35 Problems

Try the examples out in this chapter.

I would also have a look at the examples that come with the Sun JDK. These came with an early AWT based version.

- ArcTest
- BarChart
- Blink
- BouncingHeads
- CardTest
- DitherTest
- DrawTest
- Fractal
- GraphLayout
- GraphicsTest
- ImageMap
- ImageTest
- JumpingBox
- MoleculeViewer
- NervousText
- ScrollingImages
- SimpleGraph
- SpreadSheet
- TicTacToe
- TumblingDuke
- UnderConstruction
- WireFrame

They can all be found on the College web server in the demo directory, under the Java home page.

The following is the source of the bouncing heads example. Taking this as a basis modify the program to replace the images with your own. Look at the images with Netscape and look at the size information that Netscape provides. The width and height variables contain details of the expected sizes.

```
/*
 * %W% %E%
 *
 * Copyright (c) 1994-1995 Sun Microsystems, Inc. All Rights
Reserved.
 *
 * Permission to use, copy, modify, and distribute this
software
```

```
 * and its documentation for NON-COMMERCIAL or COMMERCIAL
purposes and
 * without fee is hereby granted.
 * Please refer to the file http://java.sun.com/copy_trade-
marks.html
 * for further important copyright and trademark information
and to
 * http://java.sun.com/licensing.html for further important
licensing
 * information for the Java (tm) Technology.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIA-
BLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MOD-
IFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 *
 * THIS SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE OR RESALE AS
ON-LINE
 * CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING
FAIL-SAFE
 * PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES,
AIRCRAFT
 * NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DI-
RECT LIFE
 * SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAIL-
URE OF THE
 * SOFTWARE COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY,
OR SEVERE
 * PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES").
SUN
 * SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF
FITNESS FOR
 * HIGH RISK ACTIVITIES.
 */

import java.util.Hashtable;
import java.applet.*;
import java.io.*;
import java.awt.*;
import java.net.*;

/**
```

```
 * @author        Jonathan Payne
 * @version      %I%, %G%
 */

class BounceImage {
     static float inelasticity = .96f;
     static float Ax = 0.0f;
     static float Ay = 0.0002f;
     static float Ar = 0.9f;

     public float x = 0;
     public float y = 0;
     public int width;
     public int height;
     public float Vx = 0.1f;
     public float Vy = 0.05f;
     public int index;
     public float Vr = 0.005f + (float)Math.random() *
0.001f;
     public float findex = 0f;

     BounceItem  parent;
     static boolean  imagesReadIn = false;

     public void play(int n) {
          if (parent.sounds[n] != null) {
               parent.sounds[n].play();
          }
     }

     public BounceImage(BounceItem parent) {
          this.parent = parent;
          width = 65;
          height = 72;
     }

     public void move(float x1, float y1) {
          x = x1;
          y = y1;
     }

     public void paint(Graphics g) {
          int i = index;

          if (parent.bounceimages[i] == null) {
               i = 0;
          }
          g.drawImage(parent.bounceimages[i], (int)x, (int)y,
null);
     }
```

```
    public void step(long deltaT) {
        boolean collision_x = false;
        boolean collision_y = false;

        float jitter = (float)Math.random() * .01f - .005f;

        x += Vx * deltaT + (Ax / 2.0) * deltaT * deltaT;
        y += Vy * deltaT + (Ay / 2.0) * deltaT * deltaT;
        if (x <= 0.0f) {
            x = 0.0f;
            Vx = -Vx * inelasticity + jitter;
            collision_x = true;
            play((int)(Math.random() * 3));
        }
        Dimension d = parent.size();
        if (x + width >= d.width) {
            x = d.width - width;
            Vx = -Vx * inelasticity + jitter;
            collision_x = true;
            play((int)(Math.random() * 3));
        }
        if (y <= 0) {
            y = 0;
            Vy = -Vy * inelasticity + jitter;
            collision_y = true;
            play((int)(Math.random() * 3));
        }
        if (y + height >= d.height) {
            y = d.height - height;
            Vx *= inelasticity;
            Vy = -Vy * inelasticity + jitter;
            collision_y = true;
        }
        move(x, y);
        Vy = Vy + Ay * deltaT;
        Vx = Vx + Ax * deltaT;

        findex += Vr * deltaT;
        if (collision_x || collision_y) {
            Vr *= Ar;
        }

        while (findex <= 0.0) {
            findex += parent.bounceimages.length;
        }
        index = ((int) findex) % parent.bounceimages.length;

    }
}
```



```
public class BounceItem extends Applet implements Runnable {
     boolean images_initialized = false;
     BounceImage images[];

     boolean time_to_die;
     AudioClip music;
     AudioClip sounds[];
     Image bounceimages[];

     public BounceItem() {
     }

     void makeImages(int nimages) {

          bounceimages = new Image[8];
          for (int i = 1 ; i <= 8 ; i++) {
               bounceimages[i-1] = getImage(getCodeBase(),
“images/jon/T” + i + “.gif”);
               //System.out.println(“d = ” +
bounceimages[i-1].getWidth() + “,” +
bounceimages[i-1].getHeight());
          }

          images = new BounceImage[nimages];
          for (int i = 0; i < nimages; i++) {
               BounceImage img = images[i] = new
BounceImage(this);
               img.move(1 + img.width * .8f * (i % 3) + i /
3 * .3f * img.width,
                         img.height * .3f + (i % 3) * .3f *
img.height);
          }

          sounds = new AudioClip[4];
          sounds[0] = getAudioClip(getCodeBase(), “au-
dio/ooh.au”);
          sounds[1] = getAudioClip(getCodeBase(), “au-
dio/ah.au”);
          sounds[2] = getAudioClip(getCodeBase(), “au-
dio/dah.au”);
          sounds[3] = getAudioClip(getCodeBase(), “au-
dio/gong.au”);
          music = getAudioClip(getCodeBase(), “au-
dio/spacemusic.au”);
     }

     public void run() {
          long lasttime;
```

```
          try {
               if (images == null) {
                    System.out.println(“Making images ...”);
                    makeImages(4);
               }

               if (music != null) {
                    music.loop();
               }
               lasttime = System.currentTimeMillis();
               while (!time_to_die) {
                    int i;
                    long now = System.currentTimeMillis();
                    long deltaT = now - lasttime;
                    boolean active = false;
                    Dimension d = size();

                    for (i = 0; i < images.length; i++) {
                         BounceImage      img = images[i];

                         img.step(deltaT);
                         if (img.Vy > .05 || -img.Vy > .05 ||
img.y + img.width < d.height - 10) {
                              active = true;
                         }
                    }
                    if (!active && images.length != 0) {
                         for (i = 0; i < images.length; i++) {
                              BounceImage img = images[i];

                              img.Vx = (float)Math.random() /
4.0f - 0.125f;
                              img.Vy = -(float)Math.random() /
4.0f - 0.2f;
                              img.Vr = 0.05f - (float)Math.ran-
dom() * 0.1f;
                         }
                         if (sounds[3] != null) {
                              sounds[3].play();
                         }
                    }
                    repaint();
                    lasttime = now;
                    try {
                         Thread.sleep(100);
                    } catch (InterruptedException e) {
                         return;
                    }
               }
          } finally {
```



```
            if (music != null) {
                music.stop();
            }
        }

    }

    public void init() {
        Dimension d = size();
        if ((d.width <= 100) || (d.height <= 100)) {
            resize(500, 300);
        }
    }

    public void start() {
        time_to_die = false;
        (new Thread(this)).start();
    }

    public void stop() {
        time_to_die = true;
        music.stop();
    }

    public void paint(Graphics g) {
        Dimension d = size();
        g.setColor(Color.gray);
        g.drawRect(0, 0, d.width - 1, d.height - 1);
        if (images != null) {
            for (int i = 0; i < images.length; i++) {
                if (images[i] != null) {
                    images[i].paint(g);
                }
            }
        }
    }
}
```

If you feel really adventurous replace the sounds too.

Also have a look at Chapter 11 of the Deitel book.

11.36 Bibliography

One of the best books I've found is:

Foley, van Dam, Feiner, Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley.

> There is a good coverage of most of the things that you need to know. Not cheap, about 35-40 uk pounds the last time I looked.

The other source of information is of course the on-line JDK documentation. This is a mixture of actual inforation that actually is installed on the College web server or your own pc, with links to the Sun site.

The documentation tells you what is on-line and what is on the Sun web server. A modem is therefore very useful when working at home.

Sun also make available a lot of examples on their web site. I normally bookmark some of the key components to make it easier to use with Netscape.

11.36.1 Scanning

The following is a very useful source of information regarding scanning. If you make sure that your iamges look good then I strongly recommend having a look at this FAQ.

http://www.infomedia.net/scan/The-Scan-FAQ.html

http://www.infomedia.net/scan/The-Scan-FAQ.html

11.36.2 Fonts

If you want information about fonts then I've put up a pdf version of the Norman Walsh comp.fonts.FAQ Address is

- http://mountain-ash.cnit.kcl.ac.uk/fonts/cffq215_ps.pdf

Note it doesn't look great on screen but will print ok. After reading and digesting the information in the FAQ you may be able to explain why this is the case. :-)

11.36.3 Microsoft

A couple of their urls are:

- http://www.microsoft.com/typography/default.asp
- http://www.microsoft.com/typography/users.htm
- http://www.microsoft.com/typography/history/history.htm

11.36.4 Non-Microsoft

- http://www.truetype.demon.co.uk/tthist.htm

12
AWT Based Windows Programming

‘When I use a word,’ Humpty Dumpty said, in a rather scornful tone, ‘it means just what I choose it to mean - neither more nor less’
‘The question is,’ said Alice, ‘whether you can make words mean so many different things.’

Lewis Carroll, Through the Looking Glass and What Alice found there.

Aims

The aims are to introduce some of facilities provided within the Abstract Windows Toolkit for Windows programming. The examples are based on the early 1.0.x jdk. There is a coverage of:–

- buttons
- labels
- buttons and lables
- scrollbars
- checkboxes
- checkboxgroups
- list items
- text fields
- passwords
- text fields on multiple lines
- layout managers
    - flowlayout
    - gridlayout
    - gridbag layout with constraints

12 AWT Based Windows Programming

Java provides a range of methods for writing and developing window based programs. This chapter looks at what was available in the earliest releases and provides a simple example of each. Note that complete examples that actually do something useful are given after we have covered events. The later chapter on Swing looks at the more powerful and easier to use facilties provided in later versions. There is a quick coverage of some of the common graphuical user interface components.

12.1 Button

```
import java.awt.*;
public class c1110 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new Button(“ QPR”));
    add(new Button(“ Arsenal”));
    add(new Button(“ Spurs”));
  }
}
```

12.2 Label

```
import java.awt.*;
public class c1111 extends java.applet.Apple
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new Label(“ QPR”));
    add(new Label(“ Arsenal”));
    add(new Label(“ Spurs”));
  }
}
```

12.3 Button and Label

```
import java.awt.*;
public class c1112 extends java.applet.Apple
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new Button(“ qpr”));
    add(new Button(“ arsenal”));
    add(new Button(“ spurs”));
    add(new Button(“ chelsea”));
    add(new Label(“ QPR”));
```

```
    add(new Label(“ Arsenal”));
    add(new Label(“ Spurs”));
  }
}
```

12.4 Scrollbar

```
import java.awt.*;
public class c1113 extends java.applet.Applet
{
  public void init()
  {
    add(new Scrollbar());
    add(new Scrollbar(Scrollbar.HORIZONTAL));
  }
}
```

12.5 Scrollbar with size information

```
import java.awt.*;
public class c1114 extends java.applet.Applet
{
  public void init()
  {
    add(new Scrollbar());
    add(new Scrollbar(Scrollbar.HORIZONTAL,10,50,1,50));
  }
}
```

12.6 Checkbox

```
import java.awt.*;
public class c1115 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new Checkbox(“ QPR”,null,true));
    add(new Checkbox(“ Arsenal”));
    add(new Checkbox(“ Spurs”));
  }
}
```

12.7 Checkbox with Grouping

```
import java.awt.*;
public class c1116 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    CheckboxGroup g=new CheckboxGroup();
```

```
    add(new Checkbox(“ QPR”,g,false));
    add(new Checkbox(“ Arsenal”,g,false));
  }
}
```

12.8 List

```
import java.awt.*;
public class c1117 extends java.applet.Applet
{
  public void init()
  {
//    Font f=new Font(“Arial”,Font.PLAIN,20);
//    setFont(f);
    List l=new List(3,false);
    l.addItem(“ Qpr”);
    l.addItem(“ Arsenal”);
    l.addItem(“ Spurs”);
    l.addItem(“ Chelsea”);
    l.addItem(“ Crystal Palace”);
    add(l);
  }
}
```

12.9 TextField

```
import java.awt.*;
public class c1118 extends java.applet.Applet
{
  public void init()
  {
    add(new TextField(“ Who are you favourite London
team?”,50));
    add(new Label(“ Who is your favourite player”));
    add(new TextField(20));
  }
}
```

12.10 Passwords

```
import java.awt.*;
public class c1119 extends java.applet.Applet
{
  public void init()
  {
    add(new Label(“ Type in your password”));
    TextField t=new TextField(20);
    t.setEchoCharacter(‘*’);
    add(t);
  }
}
```

12.11TextArea

```
import java.awt.*;
public class c1120 extends java.applet.Applet
{
   public void init()
   {
      String s=" This is some text that is going\n" +
                “ to spread over several lines\n” +
                “ and use the control characters for \n” +
                “ end of line - as in c and c++”;
      add(new TextArea(s,10,10));
   }
}
```

12.12Layout

There are a number of layout managers in Java. We look at each in turn and privide simple examples.

The following methods apply to all 5 layout classes:–

add(String,component)	add the component
remove(component)	remove the component
layoutContainer()	reshape the components in the container to meet the requirements of the BorderLayout object
minimumLayoutSize(container)	return the minimum dimensions needed to layout the components
preferredLayoutSize(container)	return the preferred dimensions needed to layout the components

12.12.1 Panels

The first thing to look at is the concept of dividing the screen up into panels. We can then use the various layout managers within these panels.

12.12.2 FlowLayout

This is the simplest layout manager. The objects are displayed from left to right in the order they are added to the screen.

```
import java.awt.*;
public class c1130 extends java.applet.Applet
{
   public void init()
   {

   // layout using the FlowLayout mananger

      Font f=new Font(“Arial”,Font.PLAIN,20);
      setFont(f);
      setLayout(new FlowLayout(FlowLayout.LEFT));
      add(new Button(“ QPR”));
      add(new Button(“ Arsenal”));
```

```
    add(new Button(“ Spurs”));
    add(new Button(“ Chelsea”));
    add(new Button(“ West Ham”));
    add(new Button(“ Crystal Palace”));
  }
}
```

12.12.3 GridLayout

This layout manager divides the screen into a grid of rows and columns. Objects are added at the top left and we move horizantally until the row is full.

```
import java.awt.*;
public class c1131 extends java.applet.Applet
{
  public void init()
  {

  // layout using the GridLayout mananger

    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    setLayout(new GridLayout(4,2));
    add(new Button(“ QPR”));
    add(new Button(“ Arsenal”));
    add(new Button(“ Spurs”));
    add(new Button(“ Chelsea”));
    add(new Button(“ West Ham”));
    add(new Button(“ Crystal Palace”));
  }
}
```

12.12.4 Gridlayout with size

Similar to GridLayout, but now the components can vary in size.

```
import java.awt.*;
public class c1132 extends java.applet.Applet
{
  public void init()
  {

  // layout using the GridLayout mananger

    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    setLayout(new GridLayout(4,2,10,10));
    add(new Button(“ QPR”));
    add(new Button(“ Arsenal”));
    add(new Button(“ Spurs”));
    add(new Button(“ Chelsea”));
    add(new Button(“ West Ham”));
    add(new Button(“ Crystal Palace”));
  }
```

```
}
```

12.12.5 GridBagLayout

```
import java.awt.*;
public class c1133 extends java.applet.Applet
{

  public void nb(String s,
                  GridBagLayout gb,
                  GridBagConstraints gbc)
  {
    Button b=new Button(s);
    gb.setConstraints(b,gbc);
    add(b);
  }

  public void init()
  {

  // layout using the GridBagLayout mananger
  // plus constraints

    Font f=new Font("Arial",Font.PLAIN,18);
    setFont(f);
    GridBagLayout gb = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();
    setLayout(gb);
    gbc.gridwidth=4;
    gbc.gridheight=4;
    gbc.weightx=4.0;
    gbc.fill=GridBagConstraints.HORIZONTAL;
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    nb("Goalkeeper",gb,gbc);
    gbc.weightx=1.0;
    gbc.fill=GridBagConstraints.NONE;
    gbc.gridwidth=4;
    nb("Right back  ",gb,gbc);
    nb("Centre Half ",gb,gbc);
    nb("Centre Half ",gb,gbc);
    gbc.fill=GridBagConstraints.HORIZONTAL;
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    nb("Left back   ",gb,gbc);
  }
}
```

12.12.6 CardLayout

This layout manager stacks things like a deck of cards. Only the top is visible.

```
import java.awt.CardLayout;
import java.awt.*;
public class c1134 extends java.applet.Applet
```

```
{

   // layout using the card layout mananger

   Font f=new Font(“Arial”,Font.PLAIN,20);
   Panel p=new Panel();

   public void init()
   {
      setFont(f);
      setLayout(new CardLayout());
      add(new Label(“ QPR”));
      add(new Label(“ Arsenal”));
      add(new Label(“ Spurs”));
      add(new Label(“ Chelsea”));
      add(new Label(“ West Ham”));
      add(new Label(“ Crystal Palace”));
   }

   public void start()
   {
   }
}
```

12.13Putting it all togethor

We will look at two examples on the College web server. These are ones that are provided by Sun and are often included in third party Java compilers. URLs are

http://www.kcl.ac.uk/kis/support/cit//fortran/java/demo/GraphLayout/

and

http://www.kcl.ac.uk/kis/support/cit//fortran/java/demo/GraphicsTest/

You run the them by clicking on the example1.html file.

We will not be doing anything more complex with this way of doing things as they have been superceded by later versions of the jdk.

12.14Problems

Try the examples out in this chapter.

13
Events

For Madmen Only

Hermann Hesse, Steppenwolf.

Aims

The aims of this chapter are to introduce some of the ideas involved in the use of events in Java. This is an area where there have been major changes since the original Java 1.0.x JDK.

13 Events

Java supports windows programming and therefore has to offer facilities for handling events. This is another area where Java differs from conventional procedural programming. In the first part of this chapter we will look at event handling based on the early 1.0.x jdk.

13.1 AWT Events

We will look at AWT based event handling first.

There are two kinds of AWT event handling mechanisms provided in Java, one for the keyboard and the other for handling the mouse. We will look at each in turn.

13.1.1 Mouse Events

public boolean mouseDown(Event e, int x,int y)

public boolena mouseUp(Event e,int x,int y)

public boolean mouseMove(Event e, int x,int y)

public boolena mouseExit(Event e,int x,int y)

public boolean mouseEnter(Event e, int x,int y)

public boolena mouseDrag(Event e,int x,int y)

13.1.2 Keyboard events

public boolean controlDown()

bublic boolean metaDown()

public boolean shiftDown

public final static int LEFT

public final static int RIGHT

public final static int END

public final static int HOME

public final static int PGDN

public final static int PGUP

public final static int Fx

Function keys 1 through 12.

13.1.3 Example 1 – Cut and paste text

This example illustrates some of the above.

```
import java.applet.Applet;
import java.awt.*;

public class c1401 extends Applet
{
  private TextArea t1,t2;
  private Button b;

  public void init()
  {
    String s="This is some text that spans\n" +
             “several line. We use the control n \n” +
             “character to do this.”;
```

```
    t1=new TextArea(5,10);
    t1.setText(s);

    t2=new TextArea(5,20);
    b=new Button(“ copy text -> ”);

    setLayout(new FlowLayout(FlowLayout.LEFT,5,5));

    add(t1);
    add(b);
    add(t2);

  }

  public boolean action(Event e,Object o)
  {
    if (e.target == b)
    {
      t2.setText(t1.getSelectedText());
      return true;
    }

    return false;
  }
}
```

13.1.4 Example 2 – Simple mouse tracking

```
import java.applet.Applet;
import java.awt.*;

public class c1402 extends Applet
{
  public boolean mouseMove(Event e,int x,int y)
  {
    showStatus(“ Mouse at (” + x + “ , ” + y + “)”);
    return true;
  }
}
```

13.1.5 Example 3 – Mouse with drag

```
import java.applet.Applet;
import java.awt.*;

public class c1403 extends Applet
{
  private int x,y;
  private boolean firsttime;

  public void init()
  {
```

```
    firsttime=true;
  }

  public void paint(Graphics g)
  {
    if(!firsttime)
    {
      g.fillOval(x,y,4,4);
    }
  }

  public void update(Graphics g)
  {
    paint(g);
  }

  public boolean mouseDrag(Event e,int xx,int yy)
  {
    x=xx;
    y=yy;

    firsttime=false;

    repaint();

    showStatus(" Event mouse drag ");

    return true;
  }
}
```

13.1.6 Example 4 – Key up and key down

```
import java.applet.Applet;
import java.awt.*;

public class c1404 extends Applet
{
  private Font f;
  private String letter;
  private boolean first;

  public void init()
  {
    f=new Font("Courier",Font.BOLD,30);
    first=true;
  }

  public void paint(Graphics g)
  {
```

```
     g.setFont(f);
     if (!first)
     {
       g.drawString(letter,50,50);
     }
   }

   public boolean keyDown(Event e,int key)
   {
     showStatus(“ key down the ” + (char) key + “ was
pressed ”);

     letter=String.valueOf((char)key);
     first=false;
     repaint();

     return true;
   }

   public boolean keyUp(Event e,int key)
   {
     showStatus(“ key up the ” + (char)key + “ was released
”);

     return true;
   }
}
```

13.2 Swing Event Handling – As of JDK 1.2.2

I have included a brief coverage of all that exist at this release. Consult the on-line documentation for more details.

We now look at the concept of an event listener and an event handler. There are two places to look at:

- java.awt.event

and

- javax.swing.event

and some of the event clases are interfaces – equivalent to a C++ abstract class, and hence you must provide the code to do the event handling.

With the advent of Swing there came several new event types.

We will look at complete examples that illustrate the above in the Swing chapter.

13.2.1 Interface Summary

AncestorListener

- AncestorListener Interface to support notification when changes occur to a JComponent or one of its ancestors.

CaretListener

- Listener for changes in the caret position of a text component.

CellEditorListener

- CellEditorListener defines the interface for an object that listens to changes in a CellEditor

ChangeListener

- Defines an object which listens for ChangeEvents.

DocumentEvent

- Interface for document change notifications.

DocumentEvent.ElementChange

- Describes changes made to a specific element.

DocumentListener

- Interface for an observer to register to receive notifications of changes to a text document.

HyperlinkListener

- HyperlinkListener

InternalFrameListener

- The listener interface for receiving internal frame events.

ListDataListener

- ListDataListener

ListSelectionListener

- The listener that’s notified when a lists selection value changes.

MenuDragMouseListener

- Defines a menu mouse-drag listener.

MenuKeyListener

- MenuKeyListener

MenuListener

- Defines a listener for menu events.

MouseInputListener

- A listener implementing all the methods in both the MouseListener and MouseMotionListener interfaces.

PopupMenuListener

- A popup menu listener

TableColumnModelListener

- TableColumnModelListener defines the interface for an object that listens to changes in a TableColumnModel.

TableModelListener

- TableModelListener defines the interface for an object that listens to changes in a TableModel.

TreeExpansionListener

- The listener that’s notified when a tree expands or collapses a node.

TreeModelListener

- TreeChangeListener defines the interface for an object that listens to changes in a TreeModel.

TreeSelectionListener

- The listener that's notified when the selection in a TreeSelectionModel changes.

TreeWillExpandListener

- The listener that's notified when a tree expands or collapses a node.

UndoableEditListener

- Interface implemented by a class interested in hearing about undoable operations.

13.2.2 Class Summary

AncestorEvent

- An event reported to a child component that originated from an ancestor in the component hierarchy.

CaretEvent

- CaretEvent is used to notify interested parties that the text caret has changed in the event source.

ChangeEvent

- ChangeEvent is used to notify interested parties that state has changed in the event source.

DocumentEvent.EventType

- Enumeration for document event types

EventListenerList

- A class which holds a list of EventListeners.

HyperlinkEvent

- HyperlinkEvent is used to notify interested parties that something has happened with respect to a hypertext link.

HyperlinkEvent.EventType

- Defines the ENTERED, EXITED, and ACTIVATED event types, along with their string representations, returned by toString().

InternalFrameAdapter

- An abstract adapter class for receiving internal frame events.

InternalFrameEvent

- InternalFrameEvent: an AWTEvent which adds support for JInternalFrame objects as the event source.

ListDataEvent

- Defines an event that encapsulates changes to a list.

ListSelectionEvent

- An event that characterizes a change in the current selection.

MenuDragMouseEvent

- MenuDragMouseEvent is used to notify interested parties that the menu element has received a MouseEvent forwarded to it under drag conditions.

MenuEvent

- MenuEvent is used to notify interested parties that the menu which is the event source has been posted, selected, or canceled.

MenuKeyEvent

- MenuKeyEvent is used to notify interested parties that the menu element bshas received a KeyEvent forwarded to it in a menu tree.

MouseInputAdapter

- The adapter which receives mouse events and mouse motion events.

PopupMenuEvent

- PopupMenuEvent only contains the source of the event which is the JPoupMenu sending the event

SwingPropertyChangeSupport

- This subclass of java.beans.PropertyChangeSupport is identical in functionality — it sacrifices thread-safety (not a Swing concern) for reduce memory consumption, which helps performance (both big Swing concerns).

TableColumnModelEvent

- TableColumnModelEvent is used to notify listeners that a table column model has changed, such as a column was added, removed, or moved.

TableModelEvent

- TableModelEvent is used to notify listeners that a table model has changed.

TreeExpansionEvent

- An event used to identify a single path in a tree.

TreeModelEvent

- Encapsulates information describing changes to a tree model, and used to notify tree model listeners of the change.

TreeSelectionEvent

- An event that characterizes a change in the current selection.

UndoableEditEvent

- An event indicating that an operation which can be undone has occurred.

13.2.3 Package javax.swing.event

Provides for events fired by Swing components. It contains event classes and corresponding event listener interfaces for events fired by Swing components in addition to those events in the java.awt.event package.

13.3 ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

13.4 ActionEvent

A semantic event which indicates that a component-defined action occured. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every every ActionListener object that registered to receive such events using the component's addActionListener method.

The object that implements the ActionListener interface gets this ActionEvent when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

13.5 Example 1

This is taken from the Java Swing book. It is in several files.

```
/*
 * CCPHandler.java
 * A Cut, Copy, and Paste event handler.  Nothing too
 * fancy, just define
 * some constants that can be used to set
 * the actionCommands on buttons.
 */

import java.awt.event.*;

public class CCPHandler implements ActionListener {

   public final static String CUT   = "cut";
   public final static String COPY  = "copy";
   public final static String PASTE = "paste";

   public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();
      if (command == CUT) { // we can do this since we're
comparing statics
         System.out.println("Got Cut event");
      }
      else if (command == COPY) {
         System.out.println("Got Copy event");
      }
      else if (command == PASTE) {
         System.out.println("Got Paste event");
      }
   }
}
```

This is the second file.

```
/*
 * LnFListener.java
 * A listener that can swing the look and feel of a frame
based on
 * the actionCommand of an ActionEvent object.
 * Supported look and feels are:
```

```
 *   * Metal
 *   * Windows
 *   * Motif
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LnFListener implements ActionListener {
   Frame frame;

   public LnFListener(Frame f) {
      frame = f;
   }

   public void actionPerformed(ActionEvent e) {
      String lnfName = null;

      if (e.getActionCommand().equals("Metal")) {
         lnfName = "javax.swing.plaf.metal.MetalLookAndFeel";
      } else if (e.getActionCommand().equals("Motif")) {
         lnfName = "com.sun.java.swing.plaf.mo-
tif.MotifLookAndFeel";
      } else {
         lnfName = "com.sun.java.swing.plaf.win-
dows.WindowsLookAndFeel";
      }

      try {
         UIManager.setLookAndFeel(lnfName);
         SwingUtilities.updateComponentTreeUI(frame);
      }
      catch (UnsupportedLookAndFeelException ex1) {
         System.err.println("Unsupported LookAndFeel: " +
lnfName);
      }
      catch (ClassNotFoundException ex2) {
         System.err.println("LookAndFeel class not found: " +
lnfName);
      }
      catch (InstantiationException ex3) {
         System.err.println("Could not load LookAndFeel: " +
lnfName);
      }
      catch (IllegalAccessException ex4) {
         System.err.println("Cannot use LookAndFeel: " +
lnfName);
      }
   }
```

```
}
```

This is the third file.

```
/*
 * ToolbarFrame4.java
 * The Swing-ified button example.
 * The buttons in this toolbar all carry
 * images but no text.
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ToolbarFrame4 extends Frame {

   // This time, let's use JButtons!
   JButton cutButton, copyButton, pasteButton;
   JButton winButton, javaButton, motifButton;

   public ToolbarFrame4() {
      super("Toolbar Example (Swing no text)");
      setSize(450, 250);
      addWindowListener(new BasicWindowMonitor());

      // JPanel works much like Panel does, so we'll use it
      JPanel toolbar = new JPanel();
      toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));

      CCPHandler handler = new CCPHandler();

      cutButton = new JButton(new ImageIcon("cut.gif"));
      cutButton.setActionCommand(CCPHandler.CUT);
      cutButton.addActionListener(handler);
      toolbar.add(cutButton);

      copyButton = new JButton(new ImageIcon("copy.gif"));
      copyButton.setActionCommand(CCPHandler.COPY);
      copyButton.addActionListener(handler);
      toolbar.add(copyButton);

      pasteButton = new JButton(new ImageIcon("paste.gif"));
      pasteButton.setActionCommand(CCPHandler.PASTE);
      pasteButton.addActionListener(handler);
      toolbar.add(pasteButton);

      add(toolbar, BorderLayout.NORTH);

      // Add the look and feel controls
      JPanel lnfPanel = new JPanel();
```

```
    LnFListener lnfListener = new LnFListener(this);
    javaButton = new JButton("Metal");
    javaButton.addActionListener(lnfListener);
    lnfPanel.add(javaButton);
    motifButton = new JButton("Motif");
    motifButton.addActionListener(lnfListener);
    lnfPanel.add(motifButton);
    winButton = new JButton("Windows");
    winButton.addActionListener(lnfListener);
    lnfPanel.add(winButton);
    add(lnfPanel, BorderLayout.SOUTH);
  }

  public static void main(String args[]) {
    ToolbarFrame4 tf4 = new ToolbarFrame4();
    tf4.setVisible(true);
  }
}
```

Points to note:

- a separate handler class.
- the import of java.awt.event
- the import of javax.swing
- the class extends Frame
- each of the buttons is a JButton
- explicit call to super class constructor
- add the window listener
- create a panel with layout
- set up the three buttons
- create a new panel
- set up the three look and feel buttons

13.5.1 Frames

public class Frame extends Window implements MenuContainer

A Frame is a top-level window with a title and a border.

13.5.2 super – Constructor Chaining

super is a reserved word in Java. One of its uses is to call the constructor of a superclass. This is what we have in this example.

13.6 Example 2

This is taken from the Deitel book.

```
// Fig. 6.9: Craps.java
// Craps
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class Craps extends JApplet implements ActionListener
{
    // constant variables for status of game
    final int WON = 0, LOST = 1, CONTINUE = 2;

    // other variables used in program
    boolean firstRoll = true;    // true if first roll
    int sumOfDice = 0;            // sum of the dice
    int myPoint = 0;    // point if no win/loss on first roll
    int gameStatus = CONTINUE;  // game not over yet

    // graphical user interface components
    JLabel die1Label, die2Label, sumLabel, pointLabel;
    JTextField firstDie, secondDie, sum, point;
    JButton roll;

    // setup graphical user interface components
    public void init()
    {
        Container c = getContentPane();
        c.setLayout( new FlowLayout() );

        die1Label = new JLabel( “Die 1" );
        c.add( die1Label );
        firstDie = new JTextField( 10 );
        firstDie.setEditable( false );
        c.add( firstDie );

        die2Label = new JLabel( “Die 2" );
        c.add( die2Label );
        secondDie = new JTextField( 10 );
        secondDie.setEditable( false );
        c.add( secondDie );

        sumLabel = new JLabel( “Sum is” );
        c.add( sumLabel );
        sum = new JTextField( 10 );
        sum.setEditable( false );
        c.add( sum );

        pointLabel = new JLabel( “Point is” );
        c.add( pointLabel );
        point = new JTextField( 10 );
        point.setEditable( false );
        c.add( point );

        roll = new JButton( “Roll Dice” );
        roll.addActionListener( this );
        c.add( roll );
```

```
    }

    // call method play when button is pressed
    public void actionPerformed( ActionEvent e )
    {
        play();
    }

    // process one roll of the dice
    public void play()
    {
        if ( firstRoll ) {               // first roll of the
dice
            sumOfDice = rollDice();

            switch ( sumOfDice ) {
                case 7: case 11:          // win on first roll
                    gameStatus = WON;
                    point.setText( "" );  // clear point text
field
                    break;
                case 2: case 3: case 12: // lose on first roll
                    gameStatus = LOST;
                    point.setText( "" );  // clear point text
field
                    break;
                default:                    // remember point
                    gameStatus = CONTINUE;
                    myPoint = sumOfDice;
                    point.setText( Integer.toString( myPoint )
);
                    firstRoll = false;
                    break;
            }
        }
        else {
            sumOfDice = rollDice();

            if ( sumOfDice == myPoint )    // win by making
point
                gameStatus = WON;
            else
                if ( sumOfDice == 7 )        // lose by roll-
ing 7
                    gameStatus = LOST;
        }

        if ( gameStatus == CONTINUE )
            showStatus( "Roll again." );
        else {
```

```
            if ( gameStatus == WON )
                showStatus( “Player wins. ” +
                    “Click Roll Dice to play again.” );
            else
                showStatus( “Player loses. ” +
                    “Click Roll Dice to play again.” );

            firstRoll = true;
        }
    }

    // roll the dice
    public int rollDice()
    {
        int die1, die2, workSum;

        die1 = 1 + ( int ) ( Math.random() * 6 );
        die2 = 1 + ( int ) ( Math.random() * 6 );
        workSum = die1 + die2;

        firstDie.setText( Integer.toString( die1 ) );
        secondDie.setText( Integer.toString( die2 ) );
        sum.setText( Integer.toString( workSum ) );

        return workSum;
    }
}

/**************************************************************
*************
 * (C) Copyright 1999 by Deitel & Associates, Inc. and
Prentice Hall.      *
 * All Rights Reserved.
*
 *
*
 * DISCLAIMER: The authors and publisher of this book have
used their      *
 * best efforts in preparing the book. These efforts include
the            *
 * development, research, and testing of the theories and
programs         *
 * to determine their effectiveness. The authors and pub-
lisher make        *
 * no warranty of any kind, expressed or implied, with re-
gard to these     *
 * programs or to the documentation contained in these
books. The authors *
 * and publisher shall not be liable in any event for inci-
dental or        *
```

```
 * consequential damages in connection with, or arising out
of, the         *
 * furnishing, performance, or use of these programs.
*


**********************************************************
***********/
```

Things to note:

- we now extend JApplet
- we in turn implement ActionListener – compare this with the previous example.
- set up some variables
- set up the graphical components within init()
- over ride actionPerformed() – this calls play()
- play actually plays the game.
- roll the dice – this updates the text atrea.

These examples show two of the different ways that we can program event handling using actionListener. We will come back to this area in more depth in later chapters.

13.7 Summary

Event driven programming takes a while to get used to. We will look at how this all works in later chapters.

13.8 Problems

Try the examples out to see what happens.

14
Swing

A man should keep his brain attic stacked with all the furniture he is likely to use, and the rest he can put away in the lumber room of his library, where he can get at it if he wants.

Sir Arthur Conan Doyle, Five Orange Pips.

Aims

The aims of this chapter are to look at Swing.

14 Swing

It should be obvious by now that graphics programming requires some effort. It was realised that the AWT was not adequate, and if Java was to succeed in the graphics area then this had to be addressed. The AWT were tied to the local platform. The Java Foundation Classes or JFC were created to ease the design and implementation of the user interface. These include things that people take for granted with windows based software, e.g. menus, dialog boxes etc. There is a coverage of each of them in this chapter. We will first look at the history of the JFC and Swing.

14.1 History

The original JDK release that these examples were based on was 1.0.2. As you have seen during the course some of the examples generate warning messages when compiled. This is due to the evolution that is taking place with Java.

One of the major drawbacks of the early releases was the platform specific behaviour of many program. Hardly write once and run anywhere.

JDK 1.1 fixed some of the problems. The new event model was a massive step forward, but the AWT was still platform specific. Now we had an event subscriber model rather than a chain based model. This eliminated a lot of the overhead of event propagation.

In April 1997 the JFC were announced. One of the new components were given the name Swing. They JFC comprises:–

- Swing

  We will cover this in this chapter;

- AWT

  The basic gui;

- Accessibility

  This package is for people who find access using a keyboard, mouse and screen a problem;

- 2D API

  Package for painting, complex shapes, fonts etc;

- Drag and Drop

  This package allows users to implement a visual interface to their code.

Swing was based on both Netscape's Internet Foundation Classes and input from IBM's Taligent division and Lighthouse Design.

Swing sits on top of AWT. Swing is written entirely in Java and has a consistent look and feel across platforms.

14.2 What do I need?

I would recommend getting hold of the 1.2 JDK as this will have everything you require.

I would also recommend getting hold of the on-line documentation. This is up on the College web server. The url is:–

http://www.kcl.ac.uk/kis/support/cc/fortran/java/jdk1.2.2/docs/

If you program on a pc then I would install this in conjunction with JDK 1.2.

14.3 Swing Packages

There is a problem here is that Sun have had more than one resting place for the Swing package. It was first released in:

- com.sun.java.swing

and has since moved to:

- javax.swing

The IBM VisualAge version I use expects them in the first place. The Sun version I use expects them in the second. I can't find them at all within Microsoft J++.

The following is a complete list of what is in Swing.

14.3.1 javax.accessibility

Support for people who have difficulty using the traditional user interface methods, i.e. keyboard, mouse, screen.

14.3.2 javax.swing

The major part of Swing.

14.3.3 javax.swing.border

Fancy borders.

14.3.4 javax.swing.colorchooser

Colour access.

14.3.5 javax.swing.event

Event handling.

14.3.6 javax.swing.filechooser

File access.

14.3.7 javax.swing.pending

Stuff waiting to be formally released.

14.3.8 javax.swing.plaf

Pluggable look and feel.

14.3.9 javax.swing.table

Table support.

14.3.10 javax.swing.text

Text manipulation.

14.3.11 javax.swing.text.html

Html support.

14.3.12 javax.swing.tree

Tree handling.

14.3.13 javax.swing.undo

Undoable operation support.

14.4 Enter Microsoft Stage Left

As developments were taking place coordinated by Sun, Microsoft set off on their own. An outcome of their work was AFC or Application Foundation Classes. This was intended to do very similar things to Swing, and worked with the 1.0.2 release. The two parts to it are:

- UI – the user interface component
- FX – the classes for control of graphics etc

Legal battles have been taking place. If you have Microsoft Visual J++ then you can see what Microsoft have to offer.

14.5 Pluggable Look and Feel

There are a number of gui interfaces around, and they include:

- Apple Mac
- PC and Windows
- Unix and Motif

The aim of Plaf is to provide the user with the choice of which look and feel they want. The default look and feel is called *metal*.

14.6 Lightweight Components

One meaning of the term lightweight component is that the component has the ability to render itself onto the screen. This allows developers to draw the look and feel of the application at run-time, instead of the host operating system.

14.7 Model–View–Controller (MVC) Architecture

Swing uses the MVC architecture as the basis of the design of the components.

14.7.1 Model

The state data for the component. There are several models here depending on the component. A menu is different to a scrollbar.

14.7.2 View

The way it looks on the screen.

14.7.3 Controller

How the components interact via events – mouse clicks, gaining or losing focus, keyboard etc.

14.8 Multithreading

You have already seen some of the problems that can occur with threads in an earlier chapter. A swing component paints itself based on the state of the component. If the state changes during the paint process what should happen? Incorrectly painting itself is obviously not acceptable. To handle this there is an event dispatch queue. This is a system thread that handles the communication of evenets to other components.

14.9 Components

There a number of Swing components and they include:

- Menus
- Buttons
- Dialog Boxes
- List Boxes
- Combo Boxes
- Layout Managers

- Tables
- Trees
- Undo
- Look and Feel

We will look at each in turn and use the examples from the AWT chapter as our starting point.

14.10 Simple Examples

Given the sheer size of Swing we are only going to touch briefly on what can be done. In this section we will simply rewrite the AWT examples to use Swing classes where possible and simply compare the differences. What we will try in most cases is a minimal set of changes to the original examples. Some will work, some won't.

We will look at the new ways of doing things later on in the chapter.

14.10.1 JButton

In this case we only have to make very minor changes. The first is the extra import line and the simple replacement of Button with JButton.

```
import java.awt.*;
import javax.swing.*;

public class s1410 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font("Arial",Font.PLAIN,20);
    setFont(f);
    add(new JButton(" QPR"));
    add(new JButton(" Arsenal"));
    add(new JButton(" Spurs"));
  }
}
```

Compile and run the program. Compare this example with the AWT equivalent.

14.10.2 JLabel

```
import java.awt.*;
import javax.swing.*;

public class s1411 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font("Arial",Font.PLAIN,20);
    setFont(f);
    add(new JLabel(" QPR"));
    add(new JLabel(" Arsenal"));
    add(new JLabel(" Spurs"));
  }
}
```

Compile and run this example. Compare with the AWT equivalent.

14.10.3 Button and Label

```
import java.awt.*;
import javax.swing.*;

public class s1412 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new JButton(“ qpr”));
    add(new JButton(“ arsenal”));
    add(new JButton(“ spurs”));
    add(new JButton(“ chelsea”));
    add(new JLabel(“ QPR”));
    add(new JLabel(“ Arsenal”));
    add(new JLabel(“ Spurs”));
  }
}
```

14.10.4 JScollBar

```
import java.awt.*;
import javax.swing.*;

public class s1413 extends java.applet.Applet
{
  public void init()
  {
    add(new JScrollBar());
    add(new JScrollBar(JScrollBar.HORIZONTAL));
  }
}
```

14.10.5 JScrollBar with size information

```
import java.awt.*;
import javax.swing.*;
public class s1414 extends java.applet.Applet
{
  public void init()
  {
    add(new JScrollBar());
    add(new JScrollBar(JScrollBar.HORIZONTAL,10,30,1,50));
  }
}
```

The parameters changed for the second form of the constructor. Check the on-line documentation for details.

14.10.6 CheckBox

```
import java.awt.*;
import javax.swing.*;
public class s1415 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    add(new JCheckBox(“ QPR”,null,true));
    add(new JCheckBox(“ Arsenal”));
    add(new JCheckBox(“ Spurs”));
  }
}
```

14.10.7 CheckBox with Grouping

```
import java.awt.*;
import javax.swing.*;
public class s1416 extends java.applet.Applet
{
  public void init()
  {
    Font f=new Font(“Arial”,Font.PLAIN,20);
    setFont(f);
    JCheckBoxGroup g=new JCheckBoxGroup();
    add(new JCheckBox(“ QPR”,g,false));
    add(new JCheckBox(“ Arsenal”,g,false));
  }
}
```

This example won’t compile. Look at the on-line documentation and work out what we need from Swing to replace what we have in the above AWT based example.

14.10.8 List

```
import java.awt.*;
import javax.swing.*;
public class s1417 extends java.applet.Applet
{
  public void init()
  {
//    Font f=new Font(“Arial”,Font.PLAIN,20);
//    setFont(f);
    JList l=new JList(3,false);
    l.addItem(“ Qpr”);
    l.addItem(“ Arsenal”);
    l.addItem(“ Spurs”);
    l.addItem(“ Chelsea”);
    l.addItem(“ Crystal Palace”);
    add(l);
  }
}
```


This compiles with 6 error messages. Do you think it will run? Try it what. What do you think has happened?

14.10.9 TextField

```
import java.awt.*;
import javax.swing.*;
public class s1418 extends java.applet.Applet
{
  public void init()
  {
    add(new JTextField(“ Who are you favourite London
team?”,50));
    add(new JLabel(“ Who is your favourite player”));
    add(new JTextField(20));
  }
}
```

14.10.10 Passwords

```
import java.awt.*;
import javax.swing.*;
public class s1419 extends java.applet.Applet
{
  public void init()
  {
    add(new JLabel(“ Type in your password”));
    JTextField t=new JTextField(20);
    t.setEchoCharacter(‘*’);
    add(t);
  }
}
```

This example won’t compile. Have a look at the on-line documentation and make the necessary changes to make it work.

14.10.11 TextArea

```
import java.awt.*;
import javax.swing.*;

public class s1420 extends java.applet.Applet
{
  public void init()
  {
    String s=" This is some text that is going\n" +
             “ to spread over several lines\n” +
             “ and use the control characters for \n” +
             “ end of line - as in c and c++”;
    add(new JTextArea(s,10,10));
  }
}
```

14.11Layout

14.11.1 Panels

A Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

14.11.2 FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class s1430 extends java.applet.Applet
{
  public void init()
  {

  // layout using the FlowLayout mananger

    Font f=new Font("Arial",Font.PLAIN,20);
    setFont(f);
    setLayout(new FlowLayout(FlowLayout.LEFT));
    add(new JButton(" QPR"));
    add(new JButton(" Arsenal"));
    add(new JButton(" Spurs"));
    add(new JButton(" Chelsea"));
    add(new JButton(" West Ham"));
    add(new JButton(" Crystal Palace"));
  }
}
```

14.11.3 GridLayout

Not available at this time.

14.11.4 Gridlayout with size

Not available at this time.

14.11.5 GridBagLayout

Not available at this time.

14.11.6 CardLayout

Not available at this time.

14.11.7 Simple Graph Plotting – AWT Based

This example shows how to plot a sine curve, and it does so using an AWT based approach.

```
import java.awt.*;
import java.lang.Math.*;

public class c1180 extends java.applet.Applet
{

  public void init()
  {
    setBackground(Color.white);
  }
```

```
  public void paint(Graphics g)
  {
    int x=0;
    int y=180;

    int angled;
    double angler;

    g.setColor(Color.blue);

    for (angled=0;angled<361;++angled)
    {

    x       = angled;

    g.drawLine(x,y,x,y);

    angler = angled*2*Math.PI/360;

    y       = 180 + (int) (180 * Math.sin(angler));

    }
  }
}
```

Things to note include:

- The origin at the top left causes a few problems.
  - We are forced to shift the y values down and in this example I have chosen 180.
  - I have also scaled the values returned by the sine function.
- We are forced to qualify both PI and the sine function with Math. This is a little verbose.

14.11.8 Simple Graph Plotting – Swing Based

This is the same example but now we use the new Swing way of doing things.

```
import java.awt.*;
import javax.swing.JApplet;
import java.lang.Math.*;

public class c1184 extends javax.swing.JApplet
{

  public void init()
  {
    setBackground(Color.white);
  }

  public void paint(Graphics g)
```

```
  {
    int x=0;
    int y=180;

    int angled;
    double angler;

    g.setColor(Color.blue);

    for (angled=0;angled<361;++angled)
    {

    x       = angled;

    g.drawLine(x,y,x,y);

    angler = angled*2*Math.PI/360;

    y       = 180 + (int) (180 * Math.sin(angler));

    }
  }
}
```

Things to note include:

- This example will not run under Netscape 4.5 It generates a class not found error for Javax.Swing.JApplet.
- It will run under the Appletviewer that comes with the 1.2.2 JDK.
- We are now using the enhanaced functionality provided by JApplet over the original Applet. This is done using:
  - import javax.swing.JApplet;
- We extend JApplet using:
  - public class c1184 extends javax.swing.JApplet

The rest of the code is identical.

14.12 Inheritence Revisited

You have now seen the power of OO programming and the benefits it has to offer. We have been able to accept the default behaviour of most of the classes we have used and rely on the fact that a lot of complexity is hidden from us.

You have also seen that there is more than one way to acheive something within Java in a lot of cases. Part of this is due to the fact that Sun released Java as an incomplete language and it has evolved in response to the demands that people have made on it. Java is now being used in areas well beyond what it was originally capable of doing.

We are now going to look at two ways of doing things within Swing. The first will inherit from JApplet. The second will inherit from JComponent

14.13 JApplet

JApplet is well down the Java class hierarchy, and this is shown below.

java.lang.Object –> java.awt.Component –> java.awt.Container –> java.awt.Panel –> java.applet.Applet –> javax.swing.JApplet

An extended version of java.applet.Applet that adds support for interposing input and painting behavior in front of the applets children (see glassPane), support for special children that are managed by a LayeredPane (see rootPane) and for Swing MenuBars.

The JApplet class is slightly incompatible with java.applet.Applet. JApplet contains a JRootPane as it's only child. The contentPane should be the parent of any children of the JApplet. This is different than java.applet.Applet, e.g. to add a child to an an java.applet.Applet you'd write:

```
applet.add(child);
```

However using JApplet you need to add the child to the JApplet's contentPane instead:

```
applet.getContentPane().add(child);
```

The same is true for setting LayoutManagers, removing components, listing children, etc. All these methods should normally be sent to the contentPane() instead of the JApplet itself. The contentPane() will always be non-null. Attempting to set it to null will cause the JApplet to throw an exception. The default contentPane() will have a BorderLayout manager set on it.

14.14Swing Containers and JComponent

JComponent is the base class for the Swing components. JComponent provides:

- A "pluggable look and feel" (l&f) that can be specified by the programmer or (optionally) selected by the user at runtime.
- Components that are designed to be combined and extended in order to create custom components.
- Comprehensive keystroke-handling that works with nested components.
- Action objects, for single-point control of program actions initiated by multiple components.
- A border property that implicitly defines the component's insets.
- The ability to set the preferred, minimim, and maximum size for a component.
- ToolTips — short descriptions that pop up when the cursor lingers over a component.
- Autoscrolling — automatic scrolling in a list, table, or tree that occurs when the user is dragging the mouse.
- Simple, easy dialog construction using static methods in the JOptionPane class that let you display information and query the user.
- Slow-motion graphics rendering using debugGraphics so you can see what is being displayed on screen and whether or not it is being overwritten.
- Support for Accessibility.
- Support for international Localization.

The following are sub-classes:

- AbstractButton, BasicInternalFrameTitlePane, JColorChooser, JComboBox, JFileChooser, JInternalFrame, JInternalFrame.JDesktopIcon, JLabel, JLayeredPane, JList, JMenuBar, JOptionPane, JPanel, JPopupMenu,

JProgressBar, JRootPane, JScrollBar, JScrollPane, JSeparator, JSlider, JSplitPane, JTabbedPane, JTable, JTableHeader, JTextComponent, JToolBar, JToolTip, JTree, JViewport

For more information on these subjects, see the Swing package description

14.15 Examples

Chapters 12 and 13 of the Deitel book look at basic and advanced gui programming. I would recommend looking at some of these examples to see the newer and better ways of doing things.

The Eckstein book has a very comprehensive coverage. Over 1200 pages.

Sun provide a very comprehensive demonstration program that can be found on the College web server. The url is:

http://www.kcl.ac.uk/kis/support/cit//fortran/java/jdk1.2.2/jfc/SwingSet/

I would recommend trying to run this. You will need to get all the source java files and then compile them and run under the appletviewer. If you have the jdk installed on your machine they are in the demo directory hanging off the root of the installation. There are about 30 files, taking up 320Kb on my local hard disk.

14.16 Problems

0. Try the programs out in this chapter, especially the SwingSet demo.

14.17 Bibliography

The major sources I used during the development of these examples include:

- The on-line jdk documentation.
    - This is essential. The information you need will be there but digging it out may take a long time.
- Deitel and Deitel, *Java: How to Program*, Prentice Hall, various versions.
    - Very good general coverage of the whole of the Java language. The two chapters on graphical user interface programming are enough to get you started.
- Eckstein, Loy, Wood, *Java Swing*, O'Reilly
    - Really good coverage of Swing. Much deeper coverage than the first.
- Foley, van Dam, Feiner, Hughes, Computer Graphics, Addison Wesley
    - Java now offers very good functionality in the graphics area. I'd get hold of a copy to fill in the missing gaps.

15
JavaBeans

Common sense is the best distributed commodity in the world, for every man is convinced that he is well supplied with it.

Descartes.

Aims

The aims of this chapter are look at JavaBeans.

15 JavaBeans

When we look at programming with older langauges (Fortran 66 and Fortran 77, C, the original Pascal) they support so called procedural programming. Fortran 90 adds object based programming and C++ adds object oriented programming. Java is completely object based. With the addition of Java Beans we have added one more programming paradigm – reusabe software components. They enable us to construct solutions based on using building blocks.

JavaBeans can be used to create applications or applets. They can be simple, such as a text box or label, or they can be complex, such as a mail tool or a data inspector. Beans typically have a set of:

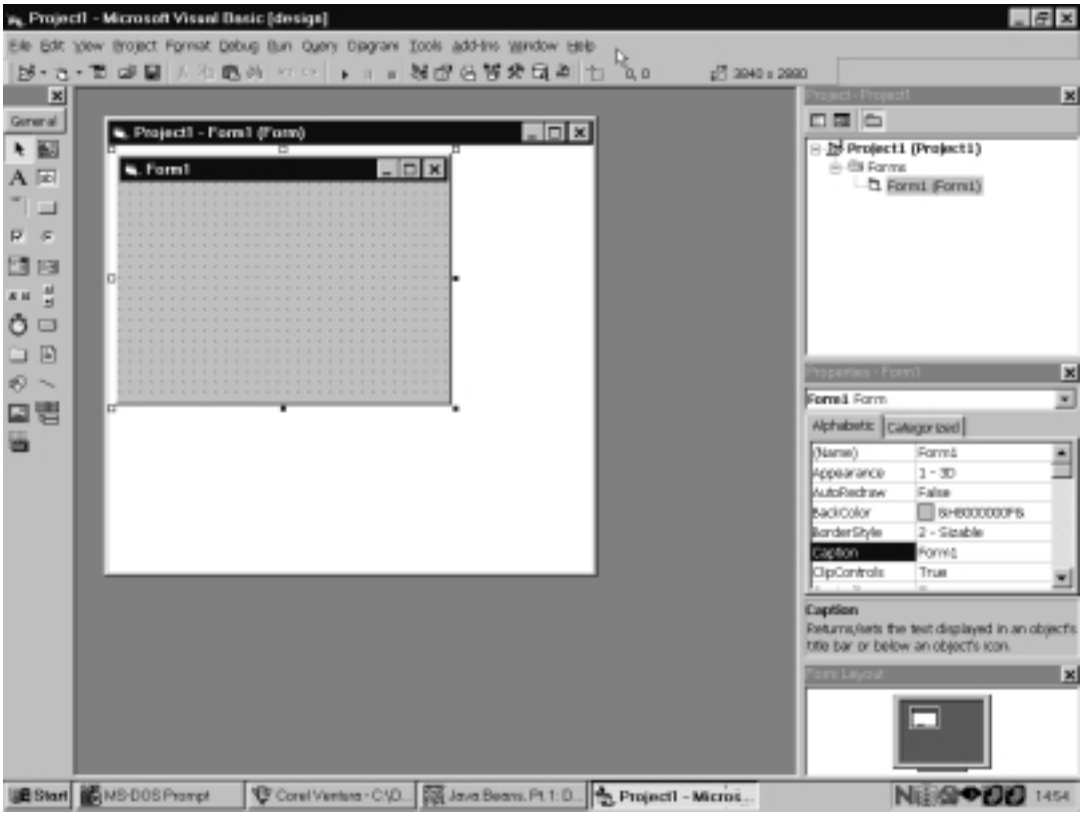
- properties: are the attributes exposed by a bean
- methods: are the actions that a bea makes available for use by other beans.
- events. the events that the bean makes happen.

JavaBeans is a component architecture for the Java platform. ActiveX is Microsoft's offering.

The following is taken verbatim from the Sun blurb about JavaBeans. I couldn't resist!

> THE CHOICE IS CLEAR: Write to the JavaBeansTM Component Architecture.
>
> Throughout the industry, JavaBeans component architecture is the architecture of choice. More than 1,000,000 developers around the world have already embraced the JavaTM platform. And no wonder. The Java platform has opened up an entirely new world of opportunities for building fully portable network-aware applications. Yet many developers are not yet sure how best to take advantage of the capabilities and benefits the Java platform delivers without sacrificing their existing investment in legacy applications. The Answer is JavaBeans Component Architecture
>
> JavaBeans component architecture is the platform-neutral architecture for the Java application environment. It's the ideal choice for developing or assembling network-aware solutions for heterogeneous hardware and operating system environments—within the enterprise or across the Internet. In fact, it's the only component architecture you should consider if you're developing for the Java platform.
>
> JavaBeans component architecture extends "Write Once, Run AnywhereTM" capability to reusable component evelopment. In fact, the JavaBeans architecture takes interoperability a major step forward—your code runs on every OS and also within any application environment. A beans developer secures a future in the emerging network software market without losing customers that use proprietary platforms, because JavaBeans components interoperate with ActiveX. JavaBeans architecture connects via bridges into other component models such as ActiveX. Software components that use JavaBeans APIs are thus portable to containers including Internet Explorer, Visual Basic, Microsoft Word, Lotus Notes, and others.
>
> The JavaBeans specification – which was completed ahead of schedule – defines a set of standard component software APIs for the Java platform. The specification was developed by Sun with a number of leading industry partners and was then refined based on broad general input from developers, customers, and end-users during a public review period.

Better than apple pie, mum, etc.



If you have used Visual Basic or Delphi you will be familiar with the way in which you can develop a program using a visual interface. Look at the screen shot on the next page when reading the following. With Visual Basic you drag and drop tools from the toolbar on the left and drop them onto the form in the middle. You then alter the state of the object by using the propertires displayed on the right hand side. This process is similar in all visual development environments, just the words and phrases change.

This is called visual development. You don't actually have to write any code – or so the claims go.

JavaBeans in conjunction with java.awt.dnd opens up this possibility with Java.

To do this properly you obviously need a graphical work environment. We have been compiling from the unix or dos prompt so far.

I recommend getting hold of IBM Visual Age for Java and installing that. The entry level version is free of charge.

I would also recommmend getting hold of the BeanBox Development Kit. This enables you to test out your beans. The url is given later in this chapter.

Java Beans will obviously vary in functionality but will have one or more of the following attributes:

- Introspection: looking inside and finding out how a bean works.
- Customization: how to alter the appearance and behavior of a bean.
- Event handling: events they can handle and events they generate
- Properties: beans can be programmed and customised
- Persistence: saving state information, e.g. saving a word processing document.

Beans can be programmed as well as handled by visual tools.

In the simple case you need only add a pair of methods to an existing class definition in order to make it a Bean. This can be done either by modifying the source for an existing class, or by extending the behavior of an existing class by subclassing it.

15.1 Package java.beans – JDK 1.1

Contains classes related to Java Beans development. A few of the classes are used by beans while they run in an application. For example, the event classes are used by beans that fire property and vetoable change events (see PropertyChangeEvent). However, most of the classes in this package are meant to be used by a bean editor (that is, a development environment for customizing and putting together beans to create an application). In particular, these classes help the bean editor create a user interface that the user can use to customize the bean. For example, a bean may contain a property of a special type that a bean editor may not know how to handle. By using the PropertyEditor interface, a bean developer can provide an editor for this special type. To minimize the resources used by a bean, the classes used by bean editors are loaded only when the bean is being edited. They are not needed while the bean is running in an application and therefore not loaded. This information is kept in what's called a bean-info.

15.2 Package java.beans.beancontext

Provides classes and interfaces relating to bean context. A bean context is a container for beans and defines the execution environment for the beans it contains. There can be several beans in a single bean context, and a bean context can be nested within another bean context. This package also contains events and listener interface for beans being added and removed from a bean context.

15.3 Example 1

This example is a complete reworking of the earlier example of a moving graphics image. Rather than use AWT I've moved over to the swing way of doing things. I have also used the networking component to use a url to locate the image.

```
package jbean01;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

public class bean01 extends JPanel
   implements ActionListener, Serializable
{

   protected ImageIcon img;

   public bean01()
   {
      img = new ImageIcon();

      URL url;
```

```
    url = getClass().getResource("joan01.gif");

    img = new ImageIcon(url);

  }

  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);

    int xpos;
    int ypos=10;

    if ( img.getImageLoadStatus() == MediaTracker.COMPLETE)
    {
      for (xpos=10;xpos<=400;xpos+=50)
      {
        img.paintIcon(this,g,xpos,ypos);
        ypos += +50;

        try {Thread.sleep(50);}

        catch(InterruptedException e) {}
      }
    }
  }

  public void actionPerformed(ActionEvent e)
  {
    repaint();
  }

  public static void main(String args[])
  {
    bean01 joanbean = new bean01();

    JFrame app = new JFrame(" Bean test ");

    app.getContentPane().add(joanbean , BorderLayout.CENTER);

    app.addWindowListener
    (new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      }
    );
```

```
      app.setSize(400,400);
      app.show();

   }
}
```

Note that the first line of the program is a package statement. We put classes into a package when working with beans. There is also an additional compilation option required, and this would be in the above case:

```
javac -d . bean01.java
```

where . is the current directory. The development environment is set up to create a subdirectory based on your package name. In this case you will end up with the compiled class file in jbean01.

After successful compilation we then need to create the Java Archive File or .jar file. You can put more than one bean into a jar file. We do this with the following command:

```
jar cfm bean01.jar manifest.tmp jbean01\*.*
```

and note again that everything will refers to this sub-directory.

The manifest file is required and I've followed the Sun naming convention, calling it manifest.tmp. The manifest file from this example is given below:

```
Main-Class: jbean01.bean01

Name: bean01.class
Java-Bean: True
```

The next major difference occurs when defing our own class bean01. In this case we now have an additioal element in the function header and this is Serializable. This means that the state of the bean is remembered. Another term for this is persistence.

When developing this example I didn't rebuild the jar file after recompiling the Java source and it ran as before.

You can examine the archive with:

```
jar tvf bean01.jar
```

We run the program with the following command:

```
java -jar bean01.jar
```

15.4 Summary

I've only briefly touched on what beans can do in this chapter. They really come into their own when used in conjuction with a visual development environment.

We are currently trying to install Netbeans on Gum. I will notify people when we have successully installed it and I'll provide details on how to use the environment.

You also have details of IBM Visual Age for Java and I would recommed installing that on your pc and working through the pdf documentation. More details are in the chapter on IBM Visual Age for Java.

15.5 Useful addresses

I would visit the following and print out some of the material. You mileage will vary.

http://java.sun.com/beans/

- Sun's home page and a very good place to start.

http://java.sun.com/beans/training.html

- links to both instructor led and on line training resources specifically for JavaBeans component developers.

http://developer.java.sun.com/developer/onlineTraining/index.html

- Links to Java Beans training material.

http://developer.java.sun.com/developer/onlineTraining/Beans/EJBTutorial/index.html

- Enterprise JavaBeansTM Tutorial: Building Your First Stateless Session Bean

15.6 Problems

Try converting some of your existing Java programs to a bean. Be prepared to devote some time to this!

16
Overview of Development Environments

The good teacher is a guide who helps others dispense with his services.

R. S. Peters, Ethics and Education

Aims

The aims of this chapter are look at some of the development environments available, both free and commercial.

16 Overview of Development Environments

In this chapter we will look at the various options open to us for Java programming.

16.1 Edit, Compile and Run

Sun make the development kits freely available for a numer of platforms including Solaris and Windows. The development kit comprises:–

- the compiler
- the interpreter or Java Virtual machine

and this is one of the reasons that people have shown such an interest in Java as you can try things out for zero cost in monetary terms.

We then use whatever native editor we like and are involved in the traditional programming cycle of:–

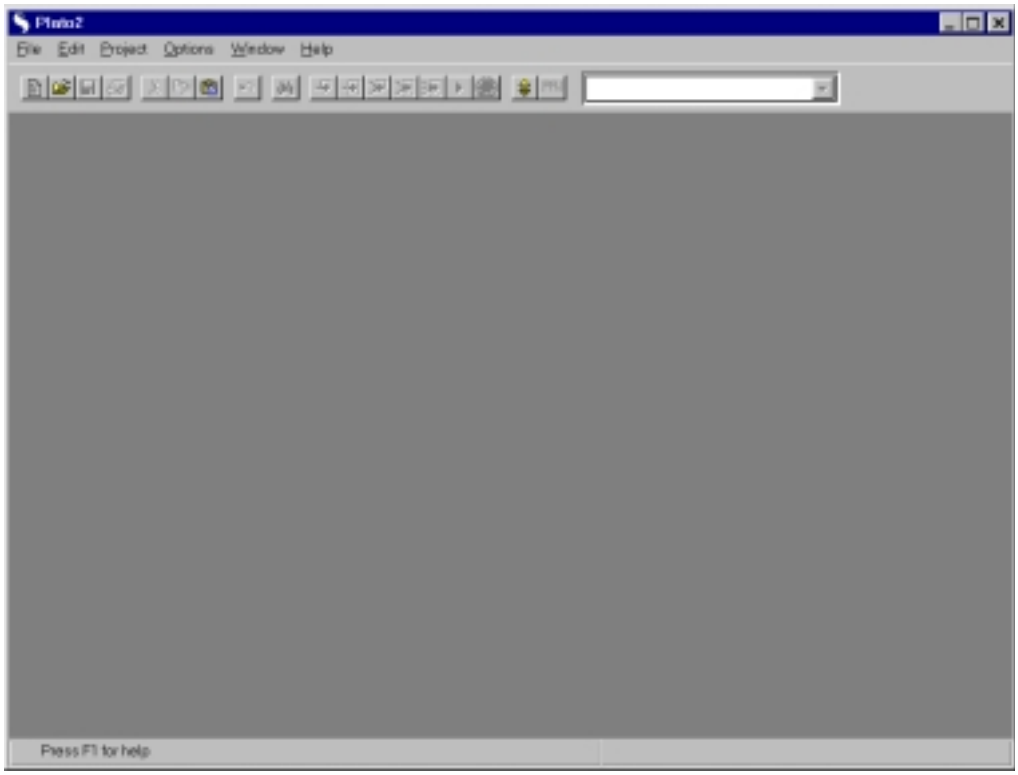
- edit
- compile
- run

that we see with the Fortran and C families of languages.

16.2 Workbench or IDE

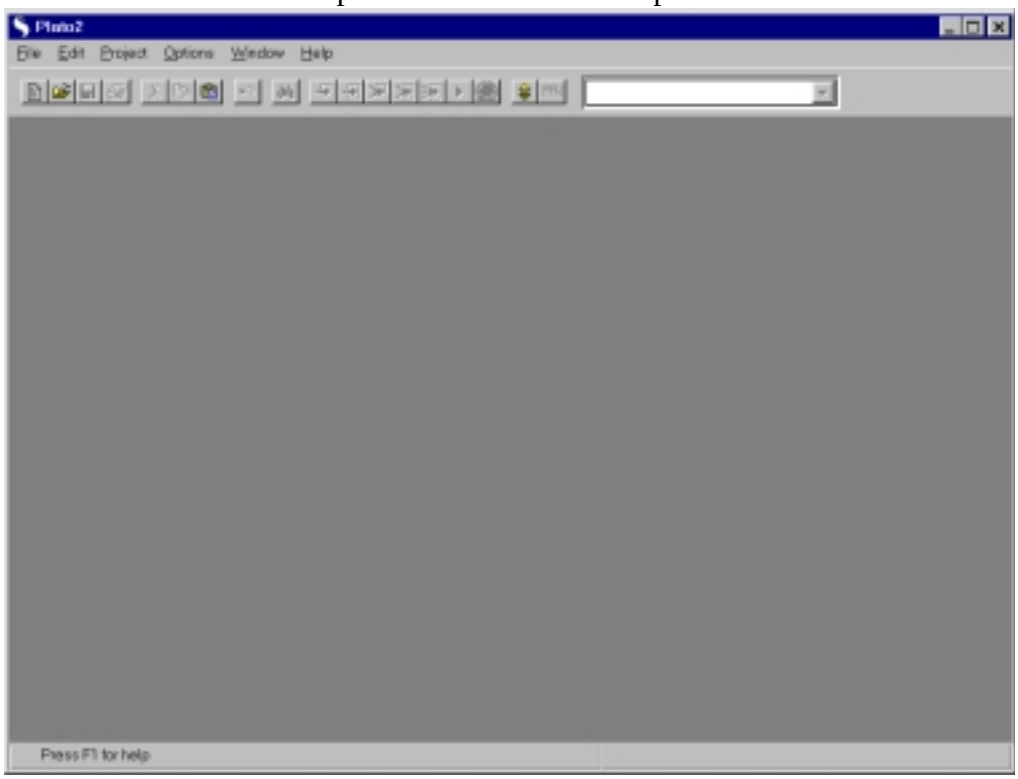
The next level of sophistication is to provide and integrated environment. Terms use include:–

- workbench
- ide – integrated development environment



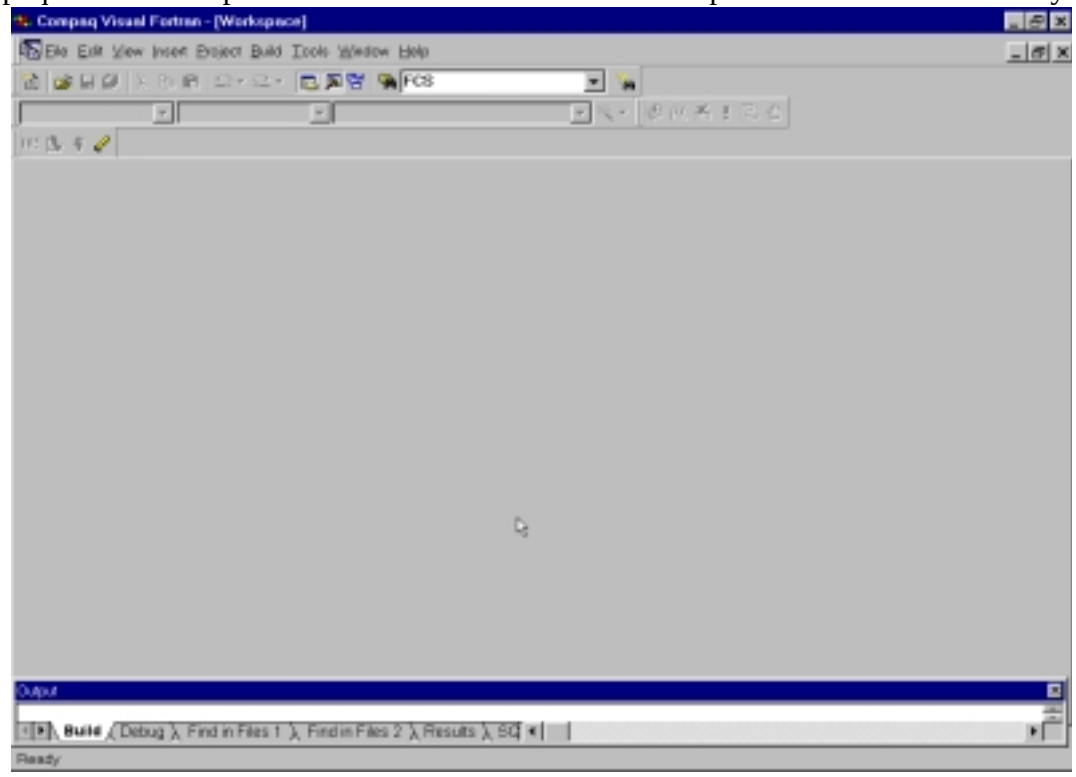
Salford Fortran Development Environment

and they vary quite considerably from the very straighforward to extremely complex. The Salford Fortran 90 and 95 compilers come with a simple environment called Plato. The



Salford Fortran Development Environment

Compaq Fortran compiler works within Microsoft Developer Studio which is a very com-

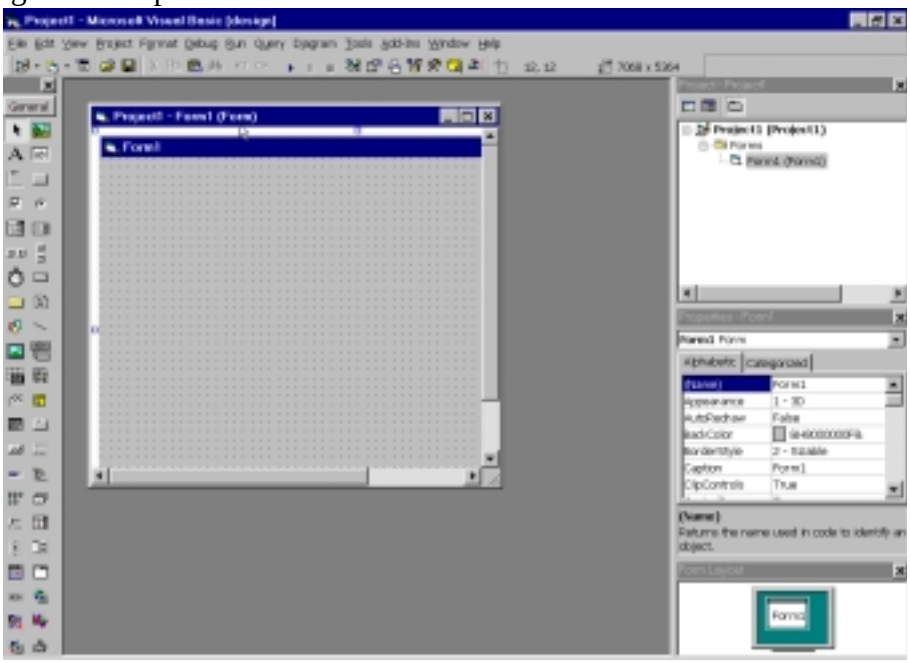


Compaq Fortran Development Environment

plex environment that takes some time to get on top off. The following screen shots show these.

16.3 Visual Development Tools

The next level of sophisticaion comes with visual development tools. Microsoft Visual Basic s a very good example of what can be achieved.



Visual Basic Development Environment

These are normaly the most costly, but they do offer considerable benefits (eventually) in terms of speed of development. Putting a visual interace onto a program is obviously going to be much easier using visual development tools, rather than raw programming.

16.4 Problems

17
Forte for Java

Make it as simple as possible, but no simpler.
Albert Einstein

Aims

The aims of this chapter are to provide an overview of Sun’s offerings in this area.

17 Forte for Java

So far we have only looked at a simple command line interface to Java programming. We have been able to achieve quite a lot with this approach, and working under Unix or Dos is almost identical. We are now going to look at another way of working – with a complete development environment. There are two components to a complete environment and these comprise:–

- the JDK
- the graphical interface

and you need both, and they need to work togethor.

The early Sun development environments were not very good. They then took an interest in a third party product called Netbeans and bought the product and rebadged it. It is now called Forte for Java. The following urls are useful:

- http://java.sun.com/j2se/1.3/download-windows.html
    - provides details of several IDEs for Java.
- http://www.sun.com/forte/ffj/ce/
    - is the Forte home page.

The IDE is written entirely in Java. The idea is that through a set of wizards, utilities, and templates you can write code quicker. The IDE provides the ability to design a graphical user interface using a form editor.

17.1 Forte Recommended Configurations

Sun's recommendation on an Intel platform is Intel Pentium II processor with a 300-MHz CPU, 128 Mbytes of memory, and 30Mbytes of disk space. I've tried the following:

- P120 with 64 Mb of memory, Windows 95: runs like a dog.
- P II 266 with 128 Mb of memory, Windows NT: ok.
- P II 350, 192 Mb of memory, Windows 98: fine.

Be patient whichever version you use!

17.2 The JDK

You need to install a compatible version of the JDK.

17.3 Documentation

Sun provide 3 documents that can be printed in the release I installed:

- QuickStart – 36 pages
- Tutorials – 38
- UserGuide – 299

and these are all Adobe PDF format. I recommend printing them, and working through the first two to help you familiarise yourself with the environment.

There is extensive on-line help. More difficult to work with though.

18
Microsoft Visual J++

‘Can you do addition?’ the White Queen asked. ‘What’s one and one and one and one and one and one and one and one and one and one?’ ‘I don’t know’ said Alice. ‘I lost count.’ ‘She can’t do addition,’ the Red Queen interrupted.

Lewis Carroll, Through the Looking Glass and What Alice Found There.

Aims

The aims of this chapter are to provide a brief coverage of Microsoft Visual J++.

18 Microsoft Visual J++

This chapter looks at Microsoft Visual J++.

18.1 Availability and Versions

There are two versions:

- Standard
- Professional

I've been using the latter and it provides the following additional features over the standard edition:

- Database access
- Visual data controls
- Professional visual development tools
- Visual component manager
- Remote debugging

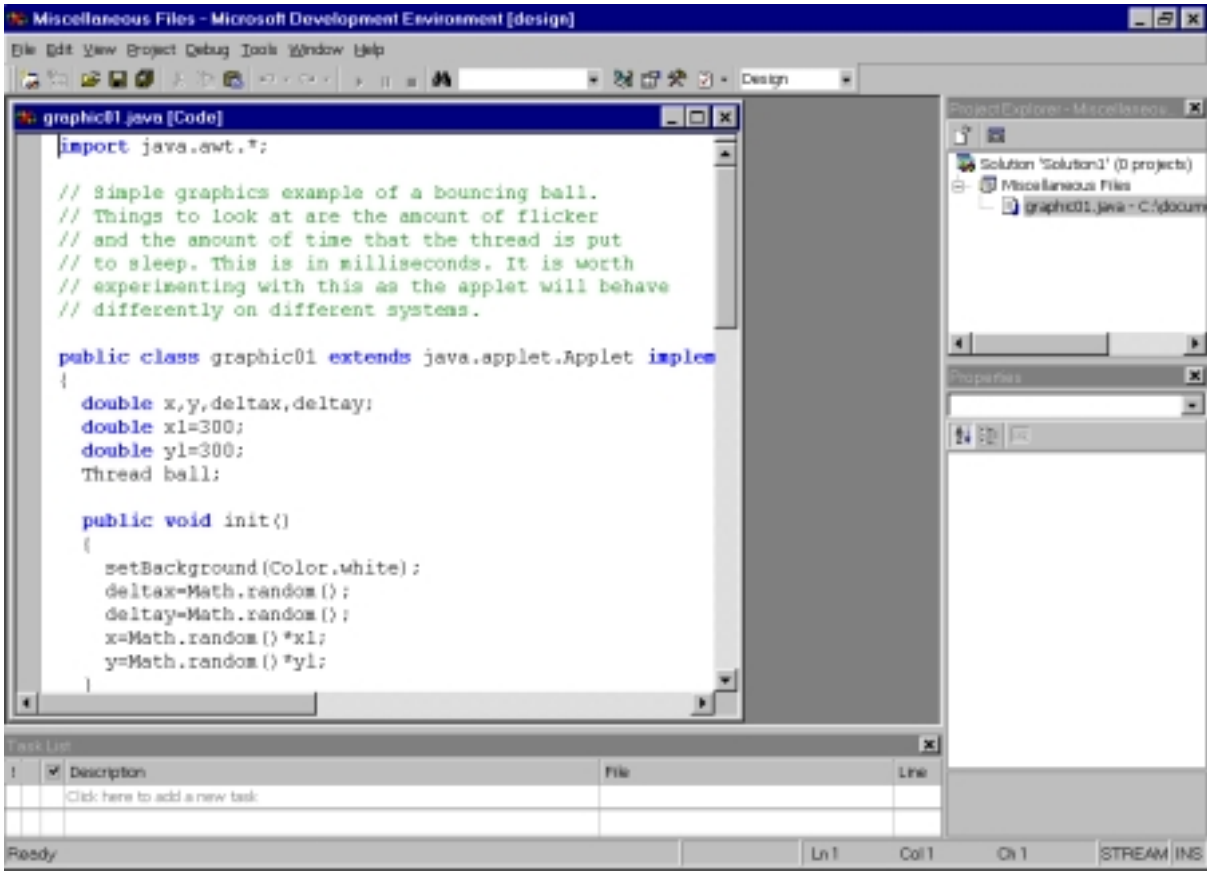
Both provide programming wizards and access to the Windows Foundation Classes. The on-line documentation is adequate. I've provided details of third party books in the bibliography.

The emphasis throughout is on creating Windows based applications or programs using Java. If you don't mind sticking to the Windows platform then this doesn't matter. However if you want an application that can run on any platform then this is a major headache, and you would be better off with another development environment or compiler.

18.2 The Development Environment

If you have used Visual Basic or Visual C++ then you will be quite at home. If you haven't don't panic. Work through the examples provided. I would start by using the help menu and

following the links to the Java documentation and work through that. The Cowell book is another option.



Microsoft J++

18.3 Working practices

Work is directory based. Do not mix code in one directory. I have developed examples in one directory as I started learning Java using the Sun JDK. This working habit is a disaster with both Microsoft Visual J++ and IBM VisualAge for Java.

18.4 Documentation Map

Below is an overview of what you'll find in the Visual J++ documentation:

- What's New: Highlights the new features in Visual J++.
- Getting Started: Explains how to get up and running with Visual J++.
- User's Guide: Shows how to use Visual J++.
- Programmer's Guide Provides in-depth information on using the Windows Foundation Classes for Java (WFC), and discusses other advanced programming techniques.
- WFC and Java Reference: Contains detailed reference information for the WFC API and controls, the Java API, and the Java Language Specification.

- Visual J++ Reference: Contains comprehensive reference information for the Visual J++ Code Model, extensibility, conditional compilation, reserved words, and compiler errors.
- Samples: Provides a rich collection of ready-to-run samples that illustrate key technologies.

Located elsewhere in the MSDN Library, the following documentation sets can be found:

- Java Package Manager
- ActiveX Data Objects (ADO)
- Remote Data Service (RDS)
- Database Designer
- Query Designer
- VBScript
- JScript
- Active Server Pages (ASP) and server script
- Microsoft Transaction Server

18.5 Getting Started with Visual J++ 6.0

Microsoft Visual J++ is an integrated Windows-hosted development tool for Java programming. If you have used Visual Basic then you will feel quite comfortable with the development environment. Visual J++ 6.0 introduces the Windows Foundation Classes for Java (WFC). This new application framework accesses the Microsoft Windows API, enabling you to write full-featured Windows applications with the Java programming language. WFC also wraps the Dynamic HTML object model implemented in Internet Explorer 4.0, which allows you to dynamically manipulate HTML on both the client and the server.

18.5.1 Creating a WFC Application

When you create a Windows application with WFC, your project contains a form that is the main window of the application. You can then add WFC controls to the form to design the graphical user interface.

Use the Forms Designer to modify your form. The RAD features of the Forms Designer allow you to quickly drop controls onto your form, configure their properties, and add event handlers.

Use Class Outline and the Text editor to modify your code. Class Outline provides a dynamic view of the contents and structure of your Java classes, and can assist you in adding methods and member variables. The Text editor supports IntelliSense features, such as Statement Completion, to help you write code.

Use the WFC data controls and components toa ccess data from your form. WFC uses ADO to retrieve data and perform simple data binding.

18.5.2 Building and Running Your Application

When you build your application, any compilation errors appear in the Task List. After you correct these errors, you can run your application from within the development environment.

18.5.3 Debugging Your Application

Although your application may compile without errors, it may not run as expected. The process of finding and fixing logic and run-time errors is known as debugging. Using theintegrated debugger, you can set breakpoints to step through your code one statement at a time and view the values of variables and properties.

18.5.4 Packaging Your Application

When you have finished modifying and debugging your application, you canpackage it to into an .exe or .cab file and deploy it to the Web.

18.6 Getting Going

Either follow the steps provided or get hold of the Cowell book.

18.7 Bibliography

Cowell J., *Essential Visual J++ 6.0* Fast, Springer.

Quick introduction. Walks you through the use of the development environment and worth it for that alone. No coverage of Swing of course.

19
IBM Visual Age for Java

Aims

The aim of this chapter is to provide a brief introduction to IBM Visual Age for Java.

19 IBM VisualAge for Java

The aim of this chapter is to provide a brief introduction to IBM Visual Age for Java.

19.1 Health Warning

The major part of the development work in this chapter has been done on a 350 MHz Pentium PII with 64 Mb of memory. I've also tried the entry level on a Cyrix clone (between a Pentium 133 and Pentium 166) with 32Mb of memory. This is slow.

19.2 Versions and Availability

Home page for VisualAge Developer Domain is:–

`http://www7.software.ibm.com/vad.nsf`

Three versions are available:–

- Entry
- Professional
- Enterprise

The first is free. At the time of writing these notes VisualAge 2.0 Entry Edition was available. This includes support for:–

- JDK 1.1.6
- Swing 1.0.2

You have to register to get a user id and password that allows you to then download the various offerings. Don't forget our password – I've forgotten mine!

The Professional Edition costs about 80 uk pounds. It comes with a couple of additional CDs containing a variety of material.

I've included some details of what you get below.

19.2.1 VisualAge® Object Connection Partners CD Version 2.0.1.

This CD provides a number of Java products built with or built for IBM VisualAge for Java. It contains software, including JavaBeans and tools, and documentation provided by IBM's Object Connection program members. It is provided solely for evaluation purposes, much of it is in trial version form.

19.2.2 MindQ: Introduction to VisualAge for Java

This is produced by MindQ Publishing. It is a multimedia CD. A little confusing at first.

19.2.3 AlphaWorks

Home url is

http://www.alphaworks.ibm.com/

19.2.3.1 alphaWorks History: The Launch

I quote *IBM alphaWorks began its life in 1996 in Armonk, NY, spear-headed by John Patrick (IBM Vice President, Internet Division). Patrick's vision was to create a Web site that surfaced IBM's hottest Internet technologies from research, and established a cutting-edge Web presence for IBM. Three college supplementals and five IBM'ers built the alphaWorks Web site in 30 days. The alphaWorks Web site launched on August 26, 1996, with press coverage and attention, attracting curious developers to download "alpha-code" technologies from IBM. alphaWorks was even featured in IBM Chairman, Louis V. Gerstner's keynote address at Fall Internet World in October 1996. John Patrick then decided to move the team closer to the epicenter of technology and Internet innovation - Silicon Valley.*

alphaWorks moved into the IBM Almaden Research Center in January 1997, and relocated once more to the IBM Center for Java Technology in Cupertino, CA.

19.2.4 Other Offerings

A beta version of VisualAge Professional Edition 3.0 is also available. Typically we are looking at 80-100 Mb. I don't recommend downloading this at home over the telephone line.

I have a copy of:–

Nilsson D.R., Jakab P.M., *Developing JavaBeans Using VisualAge for Java*, Wiley

and the CD has the following on it:–

- VisualAge for Java Entry Edition 1.0
- VisualAge for Java Patch Set 1
- Sun Java Developers Toolkit 1.1.5
- Sun Java Runtime Environment 1.1.5

Take care when buying a book with a CD to get hold of as recent a version as you can.

19.3 Documentation

IBM provide a lot of documentation in Adobe Acrobat Portable Document Format. Acrobat readers are free over the web. The following url:–

http://www.kcl.ac.uk/www/adoacr.html

provides more information.

19.3.1 IDE Basics: Concepts and Tasks: 34 pages

How to use the integrated development environment. Print this guide if you can. Have a quick browse to get a feel for some of the things you will need to know to develop a Java applet.

19.3.2 Getting Started: 144 pages

General introduction to using the basic features of Visual Age for Java. Print this guide if you can. I would work through the following chapters:

- Chapter 2: The Basics
- Chapter 3: Building your first applet

at least. I would also work through:–

- Chapter 4: Adding State Checking to your Applet
- Chapter 5: Enhancing the To-Do List Program
- Chapter 6: What Else can you do with the Visual Composition Editor

Chapter 6 takes you through the use of working with Beans including:

- manipulating beans
- moving them around
- changing their properties
- connecting beans

There is also a coverage of working with relational using the select bean.

19.3.3 Visual Composition: Concepts and Tasks: 267 pages

IBM provide (and I quote) *a state of the art object oriented visual composition editor for assembling programs visually from JavaBeans components.*

19.3.4 Data Access: Concepts and Tasks: 61 pages

Web access to data is becoming increasingly important. This takes you through some of what IBM have done to make this possible with Java.

19.3.5 SCM Tools: Concepts and Tasks: 16 pages

Software Configuration Management.

19.3.6 AgentRunner: Concepts, Tasks and Samples: 25 pages

Create and run Domino based agents.

19.3.7 Tool Integrators: 20 pages

Integrate file-based code within the IDE.

19.4 Installation

Follow the installation guidelines for which ever version you have.

19.5 Overview

If you are used to thinking in terms of a program being made up of one or more source files and going through the following cycle:–

- edit
- compile
- link – if no errors
- run

then the first time you meet a development environment can be quite a shock. Both Microsoft Visual Basic and Developer Studio take a while to get used to. Visual Basic also introduces the concept of *visual development* where one can drap and drop visual components. Well VisualAge for Java takes things one step further.

The workbench and components are object based, rather than file based. You need to think in terms of

- projects
- packages
- classes
- interfaces
- methods

and the hierarchy associated with these will provide a structure.

I am used to working with directories when programming and typically organise my work into:

- f90: fortran 90 code
- f95: fortran 95 code
- java: java code
- cxx: c++ code

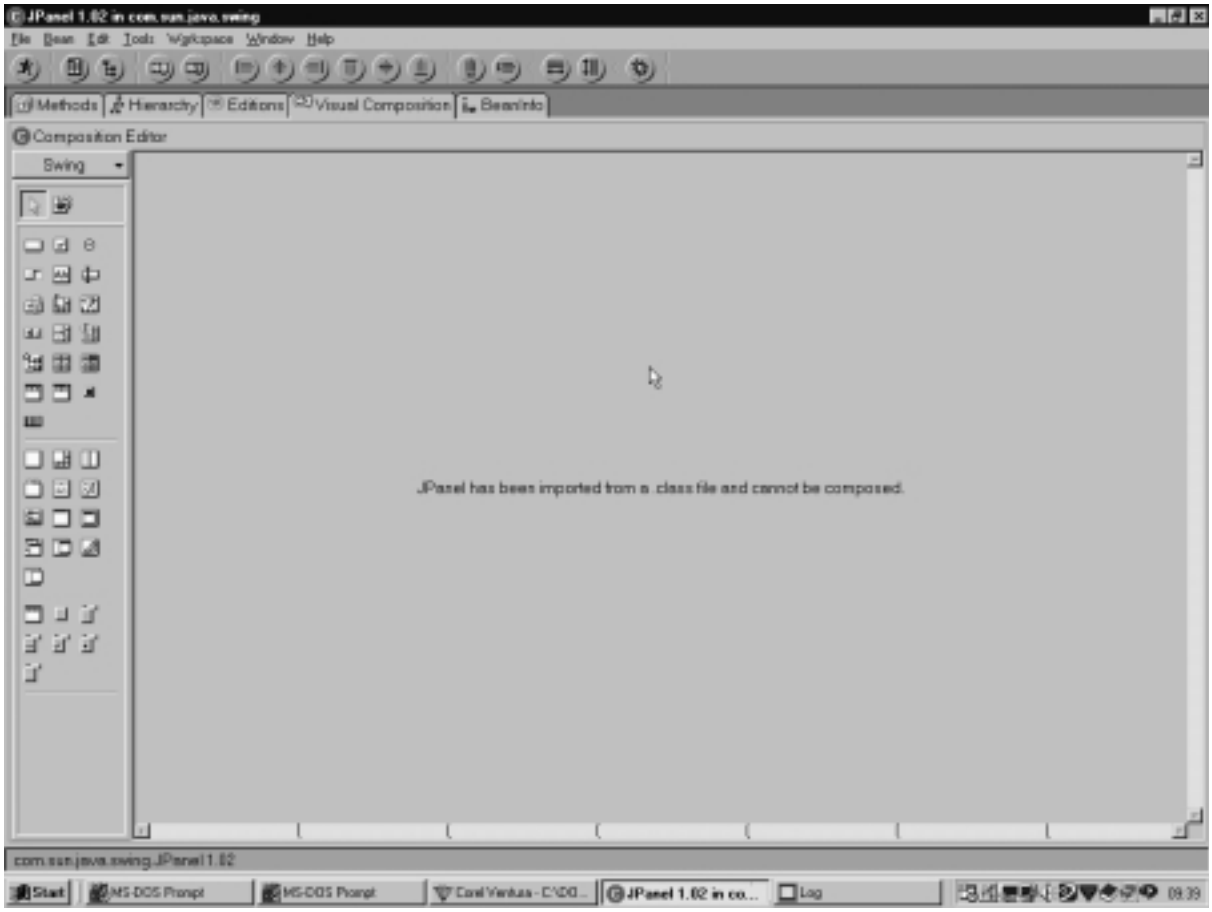
This no longer works very well when working with a development environment. You can use the above but you have to add directories below the above to keep related work files. This is true when working with:

- Compaq/Dec Fortran 95 under Microsoft Visual Studio
- Microsoft Visual C++ under Visual Studio
- Microsoft Visual J++
- IBM VisualAge for Java

Don't point one of the Java programming environments at your Java directory. It will try reading everything in and compiling it!

19.6 Starting up Visual Age for Java

The following is the opening screen.



1 Opening Screen

As you can see we have a similar layout to Visual Basic which we looked at in an earlier chapter. The key things to note are:

- the left hand side has the tools - in this case Beans
- the middle is where we can lay things out.
- there are a variety of toolbars, menus etc to enable us to work with the environment and develop our applet.

The composition editor provides with access to four sets of beans. These are:

- swing
- awt
- other
- available, and under available we have:
    - IBM Data Access Beans: Select and DB Navigator
    - Domino Java Class Library 4.6.1
    - Sun BDK Examples 1.0
    - IBM Domino Examples 2.0
    - Sun JDK Examples 1.1.6
    - Sun JFC Examples 1.0.2
    - IBM Java Examples 2.0

19.7 Summary

Follow the instructions in the two documents and be patient. It takes a while to get used to working in this new way. The key feature is the way that everything is based around beans. This is a major step forward when developing professional Java applets. The learning curve is steep, but your productivity will increase.

20
Multimedia

Aims

The aims of this chapter are to provide a brief coverage of standardisation efforts, implementation differences and future developments.

20 Multimedia

For more general capabilities you will need to download the Sun Java Media Framework. Visit

`http://java.sun.com/products/java-media/jmf/`

There are verisons for Windows and Solaris.

The JavaTM Media Framework (JMF) is an application programming interface (API) for incorporating time-based media into Java applications and applets. The JMF 1.0 API (the Java Media Player API) enabled programmers to develop Java programs that presented time-based media. The JMF 2.0 API extends the framework to provide support for capturing, storing, and broadcasting media data, using custom codecs, and manipulating media data before it is rendered.

The JMF 2.0 API was developed by Sun Microsystems, Inc. and IBM. The JMF 1.0 API was developed by Sun Microsystems, Inc., Silicon Graphics Inc., and Intel Corporation.

JMF 2.0 supports a wide array of media types, including

- protocols: FILE, HTTP, FTP, RTP
- audio: AIFF, AU, AVI, GSM, MIDI, MP2, MP3, QT, RMF, WAV
- video: AVI, MPEG-1, QT
- other: Flash 2, HotMedia

20.1 Playing Audio Clips

This chapter only looks at audio clips at the moment.

20.2 java.applet

20.2.1 Interface AudioClip: Since: JDK1.0

public interface AudioClip

The AudioClip interface is a simple abstraction for playing a sound clip. Multiple AudioClip items can be playing at the same time, and the resulting sound is mixed together to produce a composite.

20.2.2 Method Summary

void loop()

- Starts playing this audio clip in a loop.

void play()

- Starts playing this audio clip.

void stop()

- Stops playing this audio clip.

20.2.3 Method Detail

play

public void play()

- Starts playing this audio clip. Each time this method is called, the clip is re-started from the beginning.

loop

public void loop()

- Starts playing this audio clip in a loop.

stop

public void stop()

- Stops playing this audio clip.

getAudioClip

public AudioClip getAudioClip(URL url)

- Returns the AudioClip object specified by the URL argument.
- This method always returns immediately, whether or not the audio clip exists. When this applet attempts to play the audio clip, the data will be loaded.
- Parameters: url - an absolute URL giving the location of the audio clip.
- Returns: the audio clip at the specified URL.

getAudioClip

public AudioClip getAudioClip(URL url, String name)

- Returns the AudioClip object specified by the URL and name arguments.
- This method always returns immediately, whether or not the audio clip exists. When this applet attempts to play the audio clip, the data will be loaded.
- Parameters: url - an absolute URL giving the base location of the audio clip. name - the location of the audio clip, relative to the url argument.
- Returns: the audio clip at the specified URL.

newAudioClip

public static final AudioClip newAudioClip(URL url)

- Get an audio clip from the given URL
- Parameters: url - Points to the audio clip

20.3 Example – Audio

This is example 2 from chapter 16 of the Deitel book.

```
// Fig. 16.2: LoadAudioAndPlay.java
// Load an audio clip and play it.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LoadAudioAndPlay extends JApplet {
    private AudioClip sound1, sound2, currentSound;
    private JButton playSound, loopSound, stopSound;
    private JComboBox chooseSound;

    // load the image when the applet begins executing
    public void init()
    {
        Container c = getContentPane();
        c.setLayout( new FlowLayout() );
```



```
        String choices[] = { “Welcome”, “Hi” };
        chooseSound = new JComboBox( choices );
        chooseSound.addItemListener(
            new ItemListener() {
                public void itemStateChanged( ItemEvent e )
                {
                    currentSound.stop();

                    currentSound =
                        chooseSound.getSelectedIndex() == 0 ?
                            sound1 : sound2;
                }
            }
        );
        c.add( chooseSound );

        ButtonHandler handler = new ButtonHandler();
        playSound = new JButton( “Play” );
        playSound.addActionListener( handler );
        c.add( playSound );
        loopSound = new JButton( “Loop” );
        loopSound.addActionListener( handler );
        c.add( loopSound );
        stopSound = new JButton( “Stop” );
        stopSound.addActionListener( handler );
        c.add( stopSound );

        sound1 = getAudioClip(
                    getDocumentBase(), “welcome.wav” );
        sound2 = getAudioClip(
                    getDocumentBase(), “hi.au” );
        currentSound = sound1;
    }

    // stop the sound when the user switches Web pages
    // (i.e., be polite to the user)
    public void stop()
    {
        currentSound.stop();
    }

    private class ButtonHandler implements ActionListener {
        public void actionPerformed( ActionEvent e )
        {
            if ( e.getSource() == playSound )
                currentSound.play();
            else if ( e.getSource() == loopSound )
                currentSound.loop();
            else if ( e.getSource() == stopSound )
                currentSound.stop();
```

```
        }
    }
}

/**************************************************************
*************
 * (C) Copyright 1999 by Deitel & Associates, Inc. and
Prentice Hall.      *
 * All Rights Reserved.
*
 *
*
 * DISCLAIMER: The authors and publisher of this book have
used their      *
 * best efforts in preparing the book. These efforts include
the            *
 * development, research, and testing of the theories and
programs         *
 * to determine their effectiveness. The authors and pub-
lisher make        *
 * no warranty of any kind, expressed or implied, with re-
gard to these     *
 * programs or to the documentation contained in these
books. The authors *
 * and publisher shall not be liable in any event for inci-
dental or        *
 * consequential damages in connection with, or arising out
of, the        *
 * furnishing, performance, or use of these programs.
*

**************************************************************
***********/
```

Try it out. Obviously you will need a sound card and speakers to hear anything.

There are two sound file format files used in this example:

- wav
- au

If you have a pc try:

- dir /s /p *.wav

from the root of your c drive to see what wav files you have.

Java 1.3 increases the support considerably in this area.

20.4 Problems

Try the example out on the pc. You obviously need a sound card and speakers. Have a look at what sounds Microsoft provide. Can you get them to play?

21
Simple Networking

Aims

The aims of this chapter are to provide a brief coverage of networking.

21 Simple Networking

This chapter looks at using Java for networking.

21.1 Package java.net: Since: JDK1.0

Provides the classes for implementing networking applications. Using the socket classes, you can communicate with any server on the Internet or implement your own Internet server. A number of classes are provided to make it convenient to use Universal Resource Locators (URLs) to retrieve data on the Internet.

21.1.1 Interface Summary

ContentHandlerFactory: This interface defines a factory for content handlers.

FileNameMap: A simple interface which provides a mechanism to map between between a file name and a MIME type string.

SocketImplFactory: This interface defines a factory for socket implementations.

SocketOptions: Interface of methods to get/set socket options.

URLStreamHandlerFactory: This interface defines a factory for URL stream protocol handlers.

21.1.2 Class Summary

Authenticator: The class Authenticator represents an object that knows how to obtain authentication for a network connection.

ContentHandler: The abstract class ContentHandler is the superclass of all classes that read an Object from a URLConnection.

DatagramPacket: This class represents a datagram packet.

DatagramSocket: This class represents a socket for sending and receiving datagram packets.

DatagramSocketImpl: Abstract datagram and multicast socket implementation base class.

HttpURLConnection: A URLConnection with support for HTTP-specific features.

InetAddress: This class represents an Internet Protocol (IP) address.

JarURLConnection: A URL Connection to a Java ARchive (JAR) file or an entry in a JAR file.

MulticastSocket: The multicast datagram socket class is useful for sending and receiving IP multicast packets.

NetPermission: This class is for various network permissions.

PasswordAuthentication: The class PasswordAuthentication is a data holder that is used by Authenticator.

ServerSocket: This class implements server sockets.

Socket: This class implements client sockets (also called just “sockets”).

SocketImpl: The abstract class SocketImpl is a common superclass of all classes that actually implement sockets.

SocketPermission: This class represents access to a network via sockets.

URL: Class URL represents a Uniform Resource Locator, a pointer to a “resource” on the World Wide Web.

URLClassLoader: This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.

URLConnection: The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.

URLDecoder: The class contains a utility method for converting from a MIME format called “x-www-form-urlencoded” to a String

URLEncoder: The class contains a utility method for converting a String into a MIME format called “x-www-form-urlencoded” format.

URLStreamHandler: The abstract class URLStreamHandler is the common superclass for all stream protocol handlers.

21.1.3 Exception Summary

BindException: Signals that an error occurred while attempting to bind a socket to a local address and port.

ConnectException: Signals that an error occurred while attempting to connect a socket to a remote address and port.

MalformedURLException: Thrown to indicate that a malformed URL has occurred.

NoRouteToHostException: Signals that an error occurred while attempting to connect a socket to a remote address and port.

ProtocolException: Thrown to indicate that there is an error in the underlying protocol, such as a TCP error.

SocketException: Thrown to indicate that there is an error in the underlying protocol, such as a TCP error.

UnknownHostException: Thrown to indicate that the IP address of a host could not be determined.

UnknownServiceException: Thrown to indicate that an unknown service exception has occurred.

21.2 Examples

There are a couple of examples in this chapter.

21.2.1 Manipulating urls

```
import java.awt.*;
import java.net.*;
import java.applet.Applet;

public class network01 extends Applet {
    Site sites[];

    public void init()
    {
     sites = new Site[6];

     sites[0]=new Site(“Home”,
       “http://www.kcl.ac.uk/kis/support/cc/fortran/”);

     sites[1]=new Site(“Fortran 90",
       “http://www.kcl.ac.uk/kis/support/cc/for-
tran/f90home.html”);

     sites[2]=new Site(“C++”,
```

```
       "http://www.kcl.ac.uk/kis/support/cc/for-
tran/cpp/cpp.html");

     sites[3]=new Site("Java",
       "http://www.kcl.ac.uk/kis/support/cc/for-
tran/java/javahome.html");

     sites[4]=new Site("Compiler support on Gum",
       "http://www.kcl.ac.uk/kis/support/cc/fortran/com-
piler.html");

     sites[5]=new Site("Scientific Data Management",
       "http://www.kcl.ac.uk/kis/support/cc/fortran/data-
base/database.html");

       for ( int i = 0; i < sites.length; i++ )
           add( new Button( sites[ i ].getTitle() ) );
   }

   public boolean action( Event e, Object arg )
   {
       if ( e.target instanceof Button ) {
           String title;
           URL location;

           for ( int i = 0; i < sites.length; i++ ) {
               title = sites[ i ].getTitle();
               location =
                   sites[ i ].getLocation();

               if ( title.equals( arg.toString() ) ) {
                   gotoSite( location );
                   return true;  // event handled
               }
           }
       }

       return false;     // event not handled yet
   }

   public void gotoSite( URL loc )
   {
       // this must be executed in a broswer such as
Netscape
       getAppletContext().showDocument( loc );
   }
}

class Site extends Button {
   private String title;
```

```
    private URL location;

    public Site( String siteTitle, String siteLocation )
    {
        title = siteTitle;

        try {
            location = new URL( siteLocation );
        }
        catch ( MalformedURLException e ) {
            System.err.println( “Invalid URL: ” + siteLocation
);
        }
    }

    public String getTitle() { return title; }

    public URL getLocation() { return location; }
}
```

21.2.2 Reading a file on a web server

```
import java.awt.*;
import java.net.*;
import java.io.*;
import java.applet.Applet;

public class network02 extends Applet
{
    URL fileURL;
    TextArea contents;
    InputStream input;
    DataInputStream dataInput;

    public void init()
    {
        contents = new TextArea( “Connecting ...”, 400,400 );
        add( contents );

        try
        {
            fileURL = new URL(
                “http://www.kcl.ac.uk/kis/sup-
port/cc/staff/ian.html” );
        }

        catch ( MalformedURLException e )
        {
            showStatus( “Exception: ” + e.toString() );
        }
    }
```

```
    public void start()
    {
        String text;

        try
        {
            input = fileURL.openStream();
            dataInput = new DataInputStream( input );
            contents.setText( “Replace with your ego trip:\n”
);

            while ( ( text = dataInput.readLine() ) != null )

                contents.appendText( text + “\n” );

            dataInput.close();
        }

        catch ( IOException e )
        {
            showStatus( “Exception: ” + e.toString() );
        }
    }
}
```

What do you notice about the output?

The original source file is on the college web server. Save the file and have a look at it in an editor.

21.3 Problems

Try out the examples in this chapter. Try them out on Gum and your own pc. They are also on the College web server. Run them from there using Netscape.

22
Web Data Access

Aims

The aims of this chapter are to provide a brief coverage of web data access.

22 Web Data Access

22.1 Background

I've done this in the past using a product called Tango. This is freely available on the Internet and can be installed on your own machine. I recommend an NT server.

The company that sells Tango can be found at:

- http://www.pervasive.com/

and their download offerings are to be found at:

- http://www.pervasive.com/products/download/

The product is made up of a number of components.

22.1.1 The Visual Development Environment

If you are familiar with Visual Basic then you will appreciate the ease of use that Tango offers. You drag the items you want from a toolbox on the left and drop them onto a form in the middle. You then have the database connectivity handled by a visual interface on the right.

It uses ODBC to handle database access. Register the database within the ODBC manager under NT and the data sources will appear in the Tango visual interface. Clicking on the datbase will then throw up the tables. You can then drag and drop the columns you want onto the form.

22.1.2 The Web Server

It will install a web add on to your server which will handle all web access. It passes the request onto the database via the odbc driver and will then dynamically generate the html on the fly as the data is returned.

I have accessed the following data sources in this way:

- Microsoft Access
- RBTI Rbase
- Oracle
- dBase

Oracle requires the installation of an Oracle data manager.

22.2 Java

Java obviously has to work in a similar fashion. So you need at least:

- a database on a web accessible machine
- a jdbc driver for that database
- a web server process to handle the jdbc request

22.3 Data Sources

You will obviously need a data source. This is best done using an SQL based relational database management system.

22.3.1 Oracle

You obviously need an Oracle data source. You can get a personal copy of Oracle from the Oracle site. They also offer it on a cd.

22.3.2 Microsoft

If you have Office then you may already have Access. This combined with Microsoft Visual J++ should be enough to get you started.

Another possibility would be SQL server, but this would imply NT as well. I am not sure of the cost of SQL server for NT.

22.3.2.1 Some entries from the FAQ

Are there any ODBC drivers that do not work with the JDBC-ODBC Bridge?

- Most ODBC 2.0 drivers should work with the Bridge. Since there is some variation in functionality between ODBC drivers, the functionality of the bridge may be affected. The bridge works with popular PC databases, such as Microsoft Access and FoxPro.

How can I use the JDBC API to access a desktop database like Microsoft Access over the network?

- Most desktop databases currently require a JDBC solution that uses ODBC underneath. This is because the vendors of these database products haven't implemented all-Java JDBC drivers. The best approach is to use a commercial JDBC driver that supports ODBC and the database you want to use. See the JDBC drivers page for a list of available JDBC drivers. The JDBC-ODBC bridge from Sun's Java Software does not provide network access to desktop databases by itself. The JDBC-ODBC bridge loads ODBC as a local DLL, and typical ODBC drivers for desktop databases like Access aren't networked. The JDBC-ODBC bridge can be used together with the RMI-JDBC bridge , however, to access a desktop database like Access over the net. This RMI-JDBC-ODBC solution is free.

Does the JDBC-ODBC Bridge work with Microsoft J++?

- No, J++ does not support the JDBC-ODBC bridge since it doesn't implement the Java Native Interface (JNI). Any all-Java JDBC driver should work with J++, however.

22.3.3 IBM

DB2 would imply using OS/2 as your data source and Visual Age for Java.

22.4 JDBC API

The JDBCTM API provides universal data access from the JavaTM programming language. Using the JDBC 2.0 API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

The JDBC 2.0 API includes two packages: the java.sql package, known as the JDBC 2.0 core API, and the javax.sql package, known as the JDBC Standard Extension.

The JavaTM 2 SDK, Standard Edition, includes the JDBC 2.0 core API and the JDBC-ODBC Bridge.

The JavaTM 2 SDK, Enterprise Edition, includes the JDBC 2.0 core API and also the JDBC 2.0 Standard Extension. If you do not need everything that is included in the Enterprise Edition, you can download the JDBC Standard Extension separately.

Note that if the javax.sql package is bundled with your JDBC 2.0 technology driver, you will not need to download it.

JDBC Technology Drivers

To use the JDBC API with a particular database management system, you need a JDBC technology-based driver to mediate between JDBC technology and the database. Depending on various factors, a driver might be written purely in the Java programming language or in a mixture of the Java programming language and Java Native Interface (JNI) native methods. The JDBC web site maintains a list of vendors with drivers currently available or under development.

The latest SDK includes the JDBC-ODBC Bridge. This JDBC technology-based driver makes most Open Database Connectivity (ODBC) drivers available to programmers using the JDBC API. The JDBC-ODBC Bridge Guide describes the current status of this software. Drivers: Types of JDBC technology drivers

JDBC technology drivers fit into one of four categories:

- 1. A JDBC-ODBC bridge provides JDBC API access via one or more ODBC drivers. Note that some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver. Hence, this kind of driver is generally most appropriate when automatic installation and downloading of a Java technology application is not important. For information on the JDBC-ODBC bridge driver provided by Sun, see JDBC-ODBC Bridge Driver.
- 2. A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.
- 3. A net-protocol fully Java technology-enabled driver translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all of its Java technology-based clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC API alternative. It is likely that all vendors of this solution will provide roducts suitable for Intranet use. In order for these products to also support Internet access they must handle the additional requirements for security, access through firewalls, etc., that the Web imposes. Several vendors are adding JDBC technology-based drivers to their existing database middleware products.
- 4. A native-protocol fully Java technology-enabled driver converts JDBC technology calls into the network protcol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary the database vendors themselves will be the primary source for this style of driver. Several database vendors have these in progress.

22.5 Package java.sql – JDK 1.1

Provides the JDBC package. JDBC is a standard API for executing SQL statements. It contains classes and interfaces for creating SQL statements, and retrieving the results of executing those statements against relational databases. JDBC has a framework whereby different ‘’drivers’’ can be installed dynamically to access different databases.

22.5.1 Interface Summary

22.5.1.1 Array – JDBC 2.0

22.5.1.2 Blob – JDBC 2.0

22.5.1.3 CallableStatement

The interface used to execute SQL stored procedures.

22.5.1.4 Clob – JDBC 2.0

The mapping in the JavaTM programming language for the SQL CLOB type.

22.5.1.5 Connection

A connection (session) with a specific database.

22.5.1.6 DatabaseMetaData

Comprehensive information about the database as a whole.

22.5.1.7 Driver

The interface that every driver class must implement.

22.5.1.8 PreparedStatement

An object that represents a precompiled SQL statement.

22.5.1.9 Ref – JDBC 2.0

A reference to an SQL structured type value in the database.

22.5.1.10 ResultSet

A ResultSet provides access to a table of data.

22.5.1.11 ResultSetMetaData

An object that can be used to find out about the types and properties of the columns in a ResultSet.

22.5.1.12 SQLData – JDBC 2.0

The interface used for the custom mapping of SQL user-defined types.

22.5.1.13 SQLInput – JDBC 2.0

A input stream that contains a stream of values representing an instance of an SQL structured or distinct type.

22.5.1.14 SQLOutput – JDBC 2.0

The output stream for writing the attributes of a user-defined type back to the database.

22.5.1.15 Statement

The object used for executing a static SQL statement and obtaining the results produced by it.

22.5.1.16 Struct – JDBC 2.0

22.5.2 Class Summary

22.5.2.1 Date

A thin wrapper around a millisecond value that allows JDBC to identify this as a SQL DATE.

22.5.2.2 DriverManager

The basic service for managing a set of JDBC drivers.

22.5.2.3 DriverPropertyInfo

Driver properties for making a connection.

22.5.2.4 Time

A thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIME value.

22.5.2.5 Timestamp

This class is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIMESTAMP value.

22.5.2.6 Types

The class that defines constants that are used to identify generic SQL types, called JDBC types.

22.5.3 Exception Summary

22.5.3.1 BatchUpdateException – JDBC 2.0

22.5.3.2 DataTruncation

An exception that reports a DataTruncation warning (on reads) or throws a DataTruncation exception (on writes) when JDBC unexpectedly truncates a data value.

22.5.3.3 SQLException

An exception that provides information on a database access error.

22.5.3.4 SQLWarning

An exception that provides information on database access warnings.

22.6 Package javax.sql

22.6.1 Interface Summary

22.6.1.1 ConnectionEventListener

A ConnectionEventListener is an object that registers to receive events generated by a PooledConnection.

22.6.1.2 ConnectionPoolDataSource

A ConnectionPoolDataSource object is a factory for PooledConnection objects.

22.6.1.3 DataSource

A DataSource object is a factory for Connection objects.

22.6.1.4 PooledConnection

A PooledConnection object is a connection object that provides hooks for connection pool management.

22.6.1.5 RowSet

The RowSet interface adds support to the JDBC API for the JavaBeans(TM) component model.

22.6.1.6 RowSetInternal

A rowset object presents itself to a reader or writer as an instance of RowSetInternal.

22.6.1.7 RowSetListener

The RowSetListener interface is implemented by a component that wants to be notified when a significant event happens in the life of a RowSet

22.6.1.8 RowSetMetaData

The RowSetMetaData interface extends ResultSetMetaData with methods that allow a metadata object to be initialized.

22.6.1.9 RowSetReader

An object implementing the RowSetReader interface may be registered with a RowSet object that supports the reader/writer paradigm.

22.6.1.10 RowSetWriter

An object that implements the RowSetWriter interface may be registered with a RowSet object that supports the reader/writer paradigm.

22.6.1.11 XAConnection

An XAConnection object provides support for distributed transactions.

22.6.1.12 XADataSource

A factory for XAConnection objects.

22.6.2 Class Summary

22.6.2.1 ConnectionEvent

The ConnectionEvent class provides information about the source of a connection related event.

22.6.2.2 RowSetEvent

A RowSetEvent is generated when something important happens in the life of a rowset, like when a column value changes.

22.7 Examples

Deitel provides three examples. You will need to register the Access database with the ODCB manager which is in the control panel. They are in chapter 18. You will need to download all of the files and try them out on a pc. Register them as a User Data Source Name. Choose the Access driver.

Look carefully at the Deitel source. Yoy need to ensure that you register the database with the same names as those used in the Java source.

You need a basic understanding of SQL and the relational model. I've provided references at the end of this chapter.

I have Access installed and used this to look at the databases provided. This enables to see much more about the databases concerned.

Note that

22.8 Summary

If you have Microsoft Visual J++ then you can try out jdbc using this.

If you are going to try out anything original then you will need Access to create your own databases. Various versions of Microsoft Office include Access.

22.9 Bibliography

Cannen S., Otten G., *SQL: The Standard Handbook*, McGraw Hill

- This is a very good reference text to the SQL standard.

Date C., *An Introduction to Database Systems,* Addison Wesley.

- One of the most prolific writers about database systems.

Pratt P.J., *A Guide to SQL,* Boyd and Fraser.

- Very good gentle introduction with clear examples.

Rbase Technologies Inc., Rbase 2000, Release 6.5, Rbase Technologies Inc.

- This is the relational database management system I recommend on the pc. It is SQL level 2 conformant and when used in conjunction with Tango provides a very good web database development solution. Sales in the UK handled by Aspen Software. Requires an NT server.

23
Servlets

Aims

The aims of this chapter are to provide a coverage of servlets.

23 Servlets

The basic idea of a servlet is that of a program that runs on a web server and provides a service to other programs that make requests to it.

They are written in Java and have access to the Java API. This obviously includes the JDBC API to access databases and Sun saw their initial use to provide secure web-based access to data which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques.

Sun also see servlets as a popular choice for building interactive web applications. Third-party Servlet containers are available for Apache Web Server, iPlanet Web Server (formerly Netscape Enterprise Server), Microsoft IIS, and others. Servlet containers can also be integrated with web-enabled application servers, such as BEA WebLogic Application Server, IBM WebSphere, Netscape Application Server, and others.

23.1 Getting started

You will need to download and install a number of things. I recommend starting at:

- http://java.sun.com/products/servlet/index.html

and downloading the following:–

- the Java servlet api documentation
- the Java servlet development kit

and following the instructions.

I have had mixed results with the install depending on what I've already got installed. I would not recommend installing:

- jdk1.3
- Forte for Java, which currently requires the 1.3 jdk.
- IBM Visual Age for Java

I would recommend installing in the directory where you have the JDK.

The 2.1 JSDK will create the following directory structure and associated files on you system.

```
.:
       8978 Apr 27  1999 LICENSE.txt
       3028 Apr 28  1999 README.txt
        837 Apr 27  1999 default.cfg
      92904 Apr 21  1999 server.jar
      22562 Apr 21  1999 servlet.jar
       1397 Apr 21  1999 startserver
        664 Apr 27  1999 startserver.bat
       1162 Apr 21  1999 stopserver
        648 Apr 27  1999 stopserver.bat
          0 Jul 11 13:45 unix.txt

./etc:
       1696 Apr 27  1999 SimpleStartup.java

./examples:

./examples/WEB-INF:
```

```
         237 Apr 27  1999 mappings.properties
         219 Apr 27  1999 mime.properties
         350 Apr 27  1999 servlets.properties
         324 Apr 27  1999 webapp.properties

./examples/WEB-INF/jsp:

./examples/WEB-INF/jsp/beans:

./examples/WEB-INF/jsp/beans/cal:
        1630 Apr 21  1999 Entries.class
        1088 Apr 27  1999 Entries.java
         771 Apr 21  1999 Entry.class
         586 Apr 27  1999 Entry.java
        3112 Apr 21  1999 JspCalendar.class
        2949 Apr 27  1999 JspCalendar.java
        2018 Apr 21  1999 TableBean.class
        1890 Apr 27  1999 TableBean.java

./examples/WEB-INF/jsp/beans/colors:
        1858 Apr 21  1999 ColorGameBean.class
        2788 Apr 27  1999 ColorGameBean.java

./examples/WEB-INF/jsp/beans/error:
         463 Apr 21  1999 Smart.class
         231 Apr 27  1999 Smart.java

./examples/WEB-INF/servlets:
        3265 Apr 21  1999 CookieExample.class
        3398 Apr 27  1999 CookieExample.java
        1721 Apr 21  1999 HelloWorldExample.class
        1714 Apr 27  1999 HelloWorldExample.java
        1354 Apr 27  1999 LocalStrings.properties
        2251 Apr 21  1999 RequestHeaderExample.class
        2153 Apr 27  1999 RequestHeaderExample.java
        2464 Apr 21  1999 RequestInfoExample.class
        2814 Apr 27  1999 RequestInfoExample.java
        2661 Apr 21  1999 RequestParamExample.class
        2973 Apr 27  1999 RequestParamExample.java
        3407 Apr 21  1999 SessionExample.class
        3574 Apr 27  1999 SessionExample.java

./examples/images:
         292 Apr 21  1999 code.gif
        1242 Apr 21  1999 execute.gif
        1231 Apr 21  1999 return.gif

./examples/servlets:
        1895 Apr 21  1999 cookies.html
        1854 Apr 21  1999 helloworld.html
```

```
       4451 Apr 21  1999 index.html
       1414 Apr 21  1999 reqheaders.html
       2818 Apr 21  1999 reqinfo.html
       1684 Apr 21  1999 reqparams.html
       2475 Apr 21  1999 sessions.html

./src:

./src/javax:

./src/javax/servlet:
      12920 Apr 27  1999 GenericServlet.java
        234 Apr 27  1999 LocalStrings.properties
       4634 Apr 27  1999 RequestDispatcher.java
       6846 Apr 27  1999 Servlet.java
       3147 Apr 27  1999 ServletConfig.java
      16714 Apr 27  1999 ServletContext.java
       3957 Apr 27  1999 ServletException.java
       3169 Apr 27  1999 ServletInputStream.java
       8939 Apr 27  1999 ServletOutputStream.java
      12239 Apr 27  1999 ServletRequest.java
       6269 Apr 27  1999 ServletResponse.java
       1813 Apr 27  1999 SingleThreadModel.java
       5802 Apr 27  1999 UnavailableException.java

./src/javax/servlet/http:
      15649 Apr 27  1999 Cookie.java
      32100 Apr 27  1999 HttpServlet.java
      14856 Apr 27  1999 HttpServletRequest.java
      18425 Apr 27  1999 HttpServletResponse.java
      10584 Apr 27  1999 HttpSession.java
       2950 Apr 27  1999 HttpSessionBindingEvent.java
       1966 Apr 27  1999 HttpSessionBindingListener.java
       1895 Apr 27  1999 HttpSessionContext.java
       8826 Apr 27  1999 HttpUtils.java
        720 Apr 27  1999 LocalStrings.properties

./src/javax/servlet/jsp:
        622 Apr 27  1999 HttpJspPage.java
        660 Apr 27  1999 JSPPage.java

./tmp:

./tmp/1b6f5a49:

./tmp/35475a49:

./webpages:
       1967 Apr 28  1999 index.html
```

```
./webpages/WEB-INF:
        235 Apr 27  1999 mappings.properties
       2138 Apr 27  1999 mime.properties
        300 Apr 27  1999 servlets.properties
        324 Apr 27  1999 webapp.properties

./webpages/WEB-INF/servlets:
       1047 Jul 11 13:16 HTTPGetServlet.class
       5293 Apr 21  1999 SnoopServlet.class
       4898 Apr 27  1999 SnoopServlet.java

./webpages/docs:

./webpages/docs/api:
       2506 Apr 21  1999 allclasses-frame.html
      10258 Apr 21  1999 deprecated-list.html
       7789 Apr 21  1999 help-doc.html
      83183 Apr 21  1999 index-all.html
        756 Apr 21  1999 index.html
        942 Apr 21  1999 overview-frame.html
       4289 Apr 21  1999 overview-summary.html
       7247 Apr 21  1999 overview-tree.html
         33 Apr 21  1999 package-list
        618 Apr 21  1999 packages.html
       7183 Apr 21  1999 serialized-form.html
       1240 Apr 21  1999 stylesheet.css

./webpages/docs/api/javax:

./webpages/docs/api/javax/servlet:
      27030 Apr 21  1999 GenericServlet.html
      12099 Apr 21  1999 RequestDispatcher.html
      15433 Apr 21  1999 Servlet.html
      10447 Apr 21  1999 ServletConfig.html
      33416 Apr 21  1999 ServletContext.html
      12917 Apr 21  1999 ServletException.html
      10649 Apr 21  1999 ServletInputStream.html
      22170 Apr 21  1999 ServletOutputStream.html
      27468 Apr 21  1999 ServletRequest.html
      15027 Apr 21  1999 ServletResponse.html
       6147 Apr 21  1999 SingleThreadModel.html
      15644 Apr 21  1999 UnavailableException.html
       1939 Apr 21  1999 package-frame.html
       7671 Apr 21  1999 package-summary.html
       6194 Apr 21  1999 package-tree.html
      10677 Apr 21  1999 package-use.html

./webpages/docs/api/javax/servlet/class-use:
       5830 Apr 21  1999 GenericServlet.html
       6043 Apr 21  1999 RequestDispatcher.html
```

```
      10034 Apr 21  1999 Servlet.html
       9758 Apr 21  1999 ServletConfig.html
       7072 Apr 21  1999 ServletContext.html
      17706 Apr 21  1999 ServletException.html
       7584 Apr 21  1999 ServletInputStream.html
       5991 Apr 21  1999 ServletOutputStream.html
      10510 Apr 21  1999 ServletRequest.html
      10430 Apr 21  1999 ServletResponse.html
       4324 Apr 21  1999 SingleThreadModel.html
       4337 Apr 21  1999 UnavailableException.html

./webpages/docs/api/javax/servlet/http:
      27615 Apr 21  1999 Cookie.html
      36459 Apr 21  1999 HttpServlet.html
      33325 Apr 21  1999 HttpServletRequest.html
      53488 Apr 21  1999 HttpServletResponse.html
      21842 Apr 21  1999 HttpSession.html
      12208 Apr 21  1999 HttpSessionBindingEvent.html
       9306 Apr 21  1999 HttpSessionBindingListener.html
       8509 Apr 21  1999 HttpSessionContext.html
      13297 Apr 21  1999 HttpUtils.html
       1556 Apr 21  1999 package-frame.html
       7080 Apr 21  1999 package-summary.html
       6360 Apr 21  1999 package-tree.html
       7362 Apr 21  1999 package-use.html

./webpages/docs/api/javax/servlet/http/class-use:
       6737 Apr 21  1999 Cookie.html
       4338 Apr 21  1999 HttpServlet.html
      11913 Apr 21  1999 HttpServletRequest.html
      10684 Apr 21  1999 HttpServletResponse.html
       8635 Apr 21  1999 HttpSession.html
       6799 Apr 21  1999 HttpSessionBindingEvent.html
       4451 Apr 21  1999 HttpSessionBindingListener.html
       6137 Apr 21  1999 HttpSessionContext.html
       4324 Apr 21  1999 HttpUtils.html
```

23.1.1 Notes

23.1.1.1 Jar files

Copy the:

- server.jar

and

- servlet.jar

files in the root directory into:

- F:\jdk1.2.2\jre\lib\ext

This is the JDK extensions directory. You can't compile and run servlets without doing this.

23.1.1.2 Start the server

You must start the server. This is done by running the

- startserver.bat

file which is in the following directory:

- f:\jdk1.2.2\jsdk2.1

on my system.

I get variable results with this. Error messages include:

```
Can’t set up server admin
java.rmi.server.ExportException: Listen failed on port: 1109;
nested exception is:
java.net.SocketException: Descriptor not a socket: listen
failed
```

23.1.1.3 Compiled class files

You must copy all servlet .class files to

- F:\jdk1.2.2\jsdk2.1\webpages\WEB-INF\servlets

to get them to run.

23.1.1.4 Incorrect example link

The Java Server Development kit documentation home page is:

- f:\jdk1.2.2\jsdk2.1\webpages\index.html

The servlet example link is incorrect. Try

- f:\jdk1.2.2\jsdk2.1\examples\servlets\index.html

instead.

The examples links don’t work.

23.1.1.5 Complete source code

The source code listed is incomplete. The complete code is in:

- F:\jdk1.2.2\jsdk2.1\examples\WEB-INF\servlets

and so are the class files. Don’t forget to copy them to

- F:\jdk1.2.2\jsdk2.1\webpages\WEB-INF\servlets

HTML files to run the examples

The html files do not run the servlets. Example html files are not included.!

23.1.1.6 Calling Servlets From a Browser

The URL for a servlet has the following general form, where servlet-name corresponds to the name you have given your servlet:

- http://machine-name:port/servlet/servlet-name

The following:

- http://localhost:8080/servlet/HTTPGetServlet

runs the first Deitel Java servlet example.

Servlet URLs can contain queries, such as for HTTP GET requests. For example, the servlet that delivers details about a particuar book takes the stock-number of the book as a query. The servlet’s name is bookdetails; the URL for the servlet to GET and display all the information about the bookstore’s featured book is:

- http://localhost:8080/servlet/bookdetails?bookId=203

23.1.1.7 Calling Servlets from an HTML page

The following html file runs the first Deitel servlet example.

```
<!— Fig. 19.6: HTTPGetServlet.html —>
<HTML>
    <HEAD>
        <TITLE>
            Servlet HTTP GET Example
        </TITLE>
    </HEAD>
    <BODY>
        <FORM
        ACTION="http://localhost:8080/servlet/HTTPGetServlet"
        METHOD="GET">
        <P>Click the button to have the servlet send
          an HTML document</P>
        <INPUT TYPE="submit" VALUE="Get HTML Document">
        </FORM>
    </BODY>
</HTML>
```

23.2 Package java.servlet

23.2.1 Interfaces

RequestDispatcher

Servlet

ServletConfig

ServletContext

ServletRequest

ServletResponse

SingleThreadModel

23.2.2 Classes

GenericServlet

ServletInputStream

ServletOutputStream

23.2.3 Exceptions

ServletException

UnavailableException

23.3 Package java.servlet.http

23.3.1 Interfaces

HttpServletRequest

HttpServletResponse

HttpSession

HttpSessionBindingListener
HttpSessionContext

23.3.2 Classes

Cookie
HttpServlet
HttpSessionBindingEvent
HttpUtils

23.4 Package java.servlet.jsp

23.4.1 Interfaces

HttpJspPage
JspPage

23.4.2 Classes

JspEngineInfo
JspFactory
JspWriter
PageContext

23.4.3 Exceptions

JspException
JspTagException

23.5 Package java.servlet.jsp.tagtext

23.5.1 Interfaces

BodyTag
Tag

23.5.2 Classes

BodyContent
BodyTagSupport
TagAttributeInfo
TagData
TagExtraInfo
TagInfo
TagLibraryInfo
TagSupport
VariableInfo

We will look at java server pages in more depth in the next chapter. In this chapter we will concentrate on some simple examples looking at:

23.6 Bibliography

Probably the best place to start is

- http://java.sun.com/products/servlet/index.html

Follow the links.

The following provides on-line documentation.

- http://java.sun.com/products/servlet/2.2/javadoc/index.html

The following is a white paper on servlets.

- http://java.sun.com/products/servlet/whitepaper.html

Also try

- http://java.sun.com/docs/books/tutorial/servlets/TOC.html

for an on-line tutorial.

24
JavaServer Pages

Aims

The aims of this chapter are to provide a coverage of the package java.util.

24 JavaServer Pages

JavaServer pages (JSP) are Sun's response to Microsoft's Active Server Pages. Both are designed to create dynamic web pages. ASP is restricted to the Microsoft platform. ASP are found on Microsoft Internet Information Server (IIS) and work under Windows NT Server. The following url:–

- http://mountain-ash.cnit.kcl.ac.uk/

will be the system we will be using for some of the following examples.

JSP is Sun's platform independent variant of ASP. JSP technology uses XML-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page. Additionally, the application logic can reside in server-based resources (e.g. JavaBeans) that the page accesses with these tags and scriptlets. Any and all formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it fast and easy (if you believe that you'll believe anything!) to build web-based applications.

24.1 Bibliography

24.1.1 JSP

Visit

- http://java.sun.com/products/jsp/

Also have a look at:

- http://www.serverpages.com/Java_Server_Pages/

24.1.2 HTML

Visit:–

- http://members.aol.com/htmlguru/

or just try putting in html to AskJeeves.

24.1.3 XML

Visit:–

- http://www.xml.org/

and

- http://www.xml.org/xmlorg_resources/whitepapers.shtml

25
Package java.util

Aims

The aims of this chapter are to provide a coverage of the package java.util.

25 Package java.util

This chapter looks at some of the the java.util package. It should be apparent by now that a working knowledge of the class library is essential for a good understanding of any object oriented language.

25.1 Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

25.1.1 Interface Summary

25.1.1.1 Collection

The root interface in the collection hierarchy.

25.1.1.2 Comparator

A comparison function, which imposes a total ordering on some collection of objects.

25.1.1.3 Enumeration

An object that implements the Enumeration interface generates a series of elements, one at a time.

25.1.1.4 EventListener

A tagging interface that all event listener interfaces must extend.

25.1.1.5 Iterator

An iterator over a collection.

25.1.1.6 List

An ordered collection (also known as a sequence).

25.1.1.7 ListIterator

An iterator for lists that allows the programmer to traverse the list in either direction and modify the list during iteration.

25.1.1.8 Map

An object that maps keys to values.

25.1.1.9 Map.Entry

A map entry (key-value pair).

25.1.1.10 Observer

A class can implement the Observer interface when it wants to be informed of changes in observable objects.

25.1.1.11 Set

A collection that contains no duplicate elements.

25.1.1.12 SortedMap

A map that further guarantees that it will be in ascending key order, sorted according to the natural ordering of its keys (see the Comparable interface), or by a comparator provided at sorted map creation time.

25.1.1.13 SortedSet

A set that further guarantees that its iterator will traverse the set in ascending element order, sorted according to the natural ordering of its elements (see Comparable), or by a Comparator provided at sorted set creation time.

25.1.2 Class Summary

25.1.2.1 AbstractCollection

This class provides a skeletal implementation of the Collection interface, to minimize the effort required to implement this interface.

25.1.2.2 AbstractList

This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a “random access” data store (such as an array).

25.1.2.3 AbstractMap

This class provides a skeletal implementation of the Map interface, to minimize the effort required to implement this interface.

25.1.2.4 AbstractSequentialList

This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a “sequential access” data store (such as a linked list).

25.1.2.5 AbstractSet

This class provides a skeletal implementation of the Set interface to minimize the effort required to implement this interface.

25.1.2.6 ArrayList

Resizable-array implementation of the List interface.

25.1.2.7 Arrays

This class contains various methods for manipulating arrays (such as sorting and searching).

25.1.2.8 BitSet

This class implements a vector of bits that grows as needed.

25.1.2.9 Calendar

Calendar is an abstract base class for converting between a Date object and a set of integer fields such as YEAR, MONTH, DAY, HOUR, and so on.

25.1.2.10 Collections

This class consists exclusively of static methods that operate on or return collections.

25.1.2.11 Date

The class Date represents a specific instant in time, with millisecond precision.

25.1.2.12 Dictionary

The Dictionary class is the abstract parent of any class, such as Hashtable, which maps keys to values.

25.1.2.13 EventObject

The Event class is the abstract root class from which all event state objects shall be derived.

25.1.2.14 GregorianCalendar

GregorianCalendar is a concrete subclass of Calendar and provides the standard calendar used by most of the world.

25.1.2.15 HashMap

Hash table based implementation of the Map interface.

25.1.2.16 HashSet

This class implements the Set interface, backed by a hash table (actually a HashMap instance).

25.1.2.17 Hashtable

This class implements a hashtable, which maps keys to values.

25.1.2.18 LinkedList

Linked list implementation of the List interface.

25.1.2.19 ListResourceBundle

ListResourceBundle is a abstract subclass of ResourceBundle that manages resources for a locale in a convenient and easy to use list.

25.1.2.20 Locale

A Locale object represents a specific geographical, political, or cultural region.

25.1.2.21 Observable

This class represents an observable object, or “data” in the model-view paradigm.

25.1.2.22 Properties

The Properties class represents a persistent set of properties.

25.1.2.23 PropertyPermission

This class is for property permissions.

25.1.2.24 PropertyResourceBundle

PropertyResourceBundle is a concrete subclass of ResourceBundle that manages resources for a locale using a set of static strings from a property file.

25.1.2.25 Random

An instance of this class is used to generate a stream of pseudorandom numbers.

25.1.2.26 ResourceBundle

Resource bundles contain locale-specific objects.

25.1.2.27 SimpleTimeZone

SimpleTimeZone is a concrete subclass of TimeZone that represents a time zone for use with a Gregorian calendar.

25.1.2.28 Stack

The Stack class represents a last-in-first-out (LIFO) stack of objects.

25.1.2.29 StringTokenizer

The string tokenizer class allows an application to break a string into tokens.

25.1.2.30 TimeZone

TimeZone represents a time zone offset, and also figures out daylight savings.

25.1.2.31 TreeMap

Red-Black tree based implementation of the SortedMap interface.

25.1.2.32 TreeSet

This class implements the Set interface, backed by a TreeMap instance.

25.1.2.33 Vector

The Vector class implements a growable array of objects.

25.1.2.34 WeakHashMap

A hashtable-based Map implementation with weak keys.

25.1.3 Exception Summary

25.1.3.1 ConcurrentModificationException

This exception may be thrown by methods that have detected concurrent modification of a backing object when such modification is not permissible.

25.1.3.2 EmptyStackException

Thrown by methods in the Stack class to indicate that the stack is empty.

25.1.3.3 MissingResourceException

Signals that a resource is missing.

25.1.3.4 NoSuchElementException

Thrown by the nextElement method of an Enumeration to indicate that there are no more elements in the enumeration.

25.1.3.5 TooManyListenersException

The TooManyListenersException Exception is used as part of the Java Event model to annotate and implement a unicast special case of a multicast Event Source.

25.2 Bibliography

26
Package java.awt.dnd

Aims

The aims of this chapter are to provide a coverage of some of the the package java.awt.dnd.

26 Package java.awt.dnd

This chapter looks at some of the the java.awt.dnd package.

26.1 Package java.awt.dnd – JDK 1.2

Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI. Normally driven by a physical gesture of a human user using an appropriate input device, Drag and Drop provides both a mechanism to enable continuous feedback regarding the possible outcome of any subsequent data transfer to the user during navigation over the presentation elements in the GUI, and the facilities to provide for any subsequent data negotiation and transfer.

This package defines the classes and interfaces necessary to perform Drag and Drop operations in Java. It defines classes for the drag-source and the drop-target, as well as events for transferring the data being dragged. This package also provides a means for giving visual feedback to the user throughout the duration of the Drag and Drop operation.

26.1.1 Interface Summary

26.1.1.1 Autoscroll

During DnD operations it is possible that a user may wish to drop the subject of the operation on a region of a scrollable GUI control that is not currently visible to the user.

26.1.1.2 DragGestureListener

This interface is sourced from a DragGestureRecognizer and is invoked when an object of that (sub)class detects a drag initiating gesture.

26.1.1.3 DragSourceListener

The DragSourceListener defines the event interface for originators of Drag and Drop operations to track the state of the user’s gesture, and to provide appropriate “drag over” feedback to the user throughout the Drag and Drop operation.

26.1.1.4 DropTargetListener

The DropTargetListener interface is the callback interface used by the DropTarget class to provide notification of DnD operations that involve the subject DropTarget.

26.1.2 Class Summary

26.1.2.1 DnDConstants

This class contains constant values representing the type of action(s) to be

performed by a Drag and Drop operation.

26.1.2.2 DragGestureEvent

A DragGestureEvent is passed to DragGestureListener’s dragGestureRecognized() method when a particular DragGestureRecognizer detects that a platform dependent drag initiating

gesture has occurred on the Component that it is tracking.

26.1.2.3 DragGestureRecognizer

The DragGestureRecognizer is an abstract base class for the specification of a platform-dependent listener that can be associated with a particular Component in order to identify platform-dependent drag initiating gestures.

26.1.2.4 DragSource

The DragSource is the entity responsible for the initiation of the Drag and Drop operation, and may be used in a number of scenarios: 1

default instance per JVM for the lifetime of that JVM.

26.1.2.5 DragSourceContext

The DragSourceContext class is responsible for managing the initiator side of the Drag and Drop protocol.

26.1.2.6 DragSourceDragEvent

The DragSourceDragEvent is delivered from the DragSourceContextPeer, via the DragSourceContext, to the currently registered DragSourceListener.

26.1.2.7 DragSourceDropEvent

The DragSourceDropEvent is delivered from the DragSourceContextPeer, via the DragSourceContext, to its currently registered DragSourceListener's dragDropEnd() method.

26.1.2.8 DragSourceEvent

This class is the base class for DragSourceDragEvent and DragSourceDropEvent.

26.1.2.9 DropTarget

The DropTarget is associated with a Component when that Component wishes to accept drops during Drag and Drop operations.

26.1.2.10 DropTarget.DropTargetAutoScroller

this protected nested class implements autoscrolling

26.1.2.11 DropTargetContext

A DropTargetContext is created whenever the logical cursor associated with a Drag and Drop operation coincides with the visible geometry of a Component associated with a DropTarget.

26.1.2.12 DropTargetDragEvent

The DropTargetDragEvent is delivered to a DropTargetListener via its dragEnter() and dragOver() methods.

26.1.2.13 DropTargetDropEvent

The DropTargetDropEvent is delivered via the DropTargetListener drop() method.

26.1.2.14 DropTargetEvent

The DropTargetEvent is the base class for both the DropTargetDragEvent and the DropTargetDropEvent.

26.1.2.15 MouseDragGestureRecognizer

This abstract subclass of DragGestureRecognizer defines a DragGestureRecognizer for mouse based gestures.

26.1.3 Exception Summary

26.1.3.1 InvalidDnDOperationException

This exception is thrown by various methods in the java.awt.dnd package.

A typical Drag and Drop operation can be decomposed into the following states (not entirely sequentially):

- A DragSource comes into existence, associated with some presentation element (Component) in the GUI, to initiate a Drag and Drop of some potentially Transferable data.

- 1 or more DropTarget(s) come into/go out of existence, associated with presentation elements in the GUI (Components), potentially capable of consuming Transferable data types.
- A DragGestureRecognizer is obtained from the DragSource and is associated with a Component in order to track and identify any Drag initiating gesture by the user over the Component.
- A user makes a Drag gesture over the Component, which the registered DragGestureRecognizer detects, and notifies its DragGestureListener of.
- Note: Although this API consistently refers to the stimulus for a drag and drop operation being a physical gesture by a human user, this does not preclude a programmatically driven DnD operation given the appropriate implementation of a DragSource. This package contains the abstract class MouseDragGestureRecognizer for recognizing mouse device gestures. Other abstract subclasses may be provided by the platform to support other input devices or particular Component class semantics.
- The DragGestureListener causes the DragSource to initiate the Drag and Drop operation on behalf of the user, perhaps animating the GUI Cursor and/or rendering an Image of the item(s) that are the subject of the operation.
- As the user gestures navigate over Component(s) in the GUI with associated DropTarget(s), the DragSource receives notifications in order to provide “Drag Over” feedback effects, and the DropTarget(s) receive notifications in order to provide “Drag Under” feedback effects based upon the operation(s) supported and the data type(s) involved.

The gesture itself moves a logical cursor across the GUI hierarchy, intersecting the geometry of GUI Component(s), possibly resulting in the logical “Drag” cursor entering, crossing, and subsequently leaving Component(s) and associated DropTarget(s).

The DragSource object manifests “Drag Over” feedback to the user, in the typical case by animating the GUI Cursor associated with the logical cursor.

DropTarget objects manifest “Drag Under” feedback to the user, in the typical case, by rendering animations into their associated GUI Component(s) under the GUI Cursor.

The determination of the feedback effects, and the ultimate success or failure of the data transfer, should one occur, is parameterized as follows:

- By the transfer “operation” selected by the user, and supported by both the DragSource and DropTarget: Copy, Move or Reference(link).
- By the intersection of the set of data types provided by the DragSource and the set of data types comprehensible by the DropTarget.
- When the user terminates the drag operation, normally resulting in a successful Drop, both the DragSource and DropTarget receive notifications that include, and result in the type negotiation and transfer of, the information associated with the DragSource via a Transferable object.

26.2 Bibliography

27
IEEE Arithmetic

‘Can you do Addition?’ the White Queen asked. ‘What’s one and one and one and one and one and one and one and one and one?’

‘I don’t know,’ said Alice. ‘I lost count.’

Through the Looking Glass, Lewis Carroll.

Aims

The aims of this chapter are to look in more depth at arithmetic and in particular what support Java has for the IEEE 754 standard. There is a coverage of:

Hardware support for arithmetic.

Integer formats.

Floating point formats: single and double.

Special values: denormal, infinity and not a number - NAN.

Exceptions and flags: divide by zero, inexact, invalid, overflow, underflow.

27 IEEE Arithmetic

This chapter is based on material from:

Chivers I.D., Sleightholme J., *Introducing Fortran 95*, Springer Verlag.

The literature contains details of the IEEE 754 standard and the bibliography contains details of a number of printed and on-line sources.

27.1 History

When we use programming languages to do arithmetic two major concerns are the ability to develop reliable **and** portable numerical software. Arithmetic is done in hardware and there are number of things to consider:

The range of hardware available both now and in the past.

The evolution of hardware.

and there has been a very considerable change in arithmetic units since the first computers. The following is a list of hardware and computing systems that the authors have some used or have heard of. It is not exhaustive or definitive. It reflects the authors age and experience.

- CDC
- Cray
- IBM
- ICL
- Fujitsu
- DEC
- Compaq
- Gateway
- Sun
- Silicon Graphics
- Hewlett Packard
- Data General
- Honeywell
- Elliot
- Mostek
- National Semiconductors
- Intel
- Zilog
- Motorola
- Signetics
- Amdahl
- Texas Instruments
- Cyrix

Some of the operating systems include:

- NOS
- NOS/BE
- Kronos
- Unix
- VMS
- Dos
- Windows
- MVS
- VM
- CP/M

and again the list is not exhaustive or definitive. The intention is to provide with some idea of wide range of hardware, computer manufacturers and operating systems that have existed in the last 50 years.

To help with the anarchy that existed in this area Doctor Robert Stewart (acting on behalf of the IEEE) convened a meeting which led to the birth of IEEE 754.

The first draft was prepared by Willam Kahan, Jerome Coonen and Harold Stone, called the KCS draft and eventually adopted as IEEE 754. A fascinating account of the development of this standard can be found in *An Interview with the Old Man of Floating Point*, and the bibliography provides a web address of this interview. Kahan went on to get the ACM Turing Award in 1989 for his work in this area

This has become a de facto standard amongst arithmetic units in modern hardware. Note that it is not possibe to precisely describe the answers a program will give and the authors of the standard knew this. This goal is virtually impossible to achieve when one considers floating point arithmetic. Reasons for this include:

- The conversions of numbers between decimal and binary formats.
- The use of elementary library functions.
- Results of calculations may be in hardware inaccessible to the programmer.
- Intermediate results in subexpressions or arguments to procedures.

The bibliography contains details of a paper that addresses this issue in much greater depth – *Differences Among IEEE 754 Implementions*.

Fortran is one of a small number of languages to provide access to IEEE arithmetic, and it achives this via TR1880 which will become an integral part of Fortran 2000. The C standard (C9X) addresses this issue and Java offers limited IEEE arithmetic support. More information can be found in the references at the end of the chapter.

27.2 IEEE 754 Specifications

The standard specifies a number of things including:

- Single precision floating point format.
- Double precision floating point format.
- Two classes of extended floating point formats.
- Accuracy requirements on the following floating point operations:
    - Add.

- Subtract.
- Multiply.
- Divide.
- Square root.
- Remainder.
- Round numbers in floating point format to integer values.
- Convert between different floating point formats.
- Convert between floating point and integer format.
- Compare

- Base conversion - i.e. when converting between decimal and binary floating point formats and vice versa.
- Exception handling for:
  - Divide by zero.
  - Overflow.
  - Underflow.
  - Invalid operation.
  - Inexact.
- Rounding directions.
- Rounding precisions.

and we will look briefly at each of these requirements.

27.2.1 Single precision floating point format.

This is a 32 bit quantity made up of a sign bit, 8 bit baised exponent and 23 bit mantissa. The standard also specifies that certain of the bit patterns are set aside and do not represent normal numbers. This means that valid numbers are in the range 3.40282347E+38 to 1.17549435E-38 and the precision is between 6 and 9 digits depending on the numbers.

The special bit patterns provide the following:

- +0
- -0
- subnormal numbers in the range 1.17549421E-38 to 1.40129846-45
- + infinity
- - infinity
- quiet NaN (Not a Number)
- signalling NaN

One of the first systems that the authors worked with that had special bits patterns set aside were the CDC 6000 range of computers that had negative indefinite and infinity. The ideas are not new therefore, as this was in the late 1970s.

The support of positive and negative zero means that certain problems can be handled correctly including:

- The evaluation of the log function which has a discontinuity at zero.

- The equation $\sqrt{1/z}$ $1/\sqrt{z}$ can be solved when z = -1.

See also the Kahan paper *Branch Cuts for Complex Elementary Functions, or Much Ado About Nothing's Sign Bit* for more details.

Subnormals, which permit gradual underflow, fill the gap between 0 and the smallest normal number.

Simply stated underflow occurs when the result of an arithmetic operation is so small that it is subject to a larger than normal rounding error when stored. The existence of subnormals means that greater precision is available with these small numbers than normal numbers. The key features of gradual underflow are:

- When underflow does occur there should never be a loss of accuracy any greater than from ordinary roundoff.
- The operations of addition, subtraction comparision and remainder are always exact.
- Algorithms written to take advanage of subnormal numbers have smaller error bounds than other systems.
- If x and y are within a factor of 2 then x-y is error free, which is used in a number of algorithms that increase the precision at critical regions.

The combination of postive and negative zero and subnormal numbers means that when x and y are small and x-y has been flushed to zero the evaluation of:

- $1/(x\ \ y)$

can be flagged and located.

Certain arithmetic operations cause problems including:

- 0 *
- 0 / 0
- $\sqrt{x}$ when x < 0

and the support for NaN handles these cases.

The support for positive and negative infinity allows the handling of:

- x / 0 when x is non-zero and of either sign

and the outcome of this means that we write our programs to take the appropriate action. In some cases this would mean recalculating using another approach.

For more information see the references in the bibliography.

27.2.2 Double precision floating point format.

This is a 64 bit quantity made up of a sign bit, 11 bit biased exponent and 52 bit mantissa. As with single precision the standard specifies that certain of the bit patterns are set aside and do not represent normal numbers. This means we have valid numbers in the range 1.7976931348623157E308 to 2.2250738585072014E-308 and precsion between 15 and 17 digits depending on the numbers.

As with single precision there are bit patterns set aside for the same special conditions.

Note that his does not mean that the hardware has to handle the manipulation of this 64 bit quantity in an identical fashion. The Sparc and Intel family handle the above as two 32 bit quantities but the order of the 2 component parts is reversed – so called big endian and little endian.

27.2.3 Two classes of extended floating point formats.

These formats are not mandatory. A number of variants of double extended exist including:

Sun – 4 32 bit words, one sign bit, 15 bit biased exponent and 112 bit mantissa, numbers in the range 3.362E-4932 to 1.189E4932, 33 to 36 digits of significance.

Intel – 10 bytes – one sign bit, 15 bit biased exponent, 63 bit mantissa, numbers in the range 3.362E-4932 to 1.189E4932, 18-21 digits of significance.

PowerPC – as Sun.

27.2.4 Accuracy requirements

Remainder and compare must be exact. The rest should return the exact result if possible. If not there are well defined rounding rules to apply.

27.2.5 Base conversion - i.e. when converting between decimal and binary floating point formats and vice versa.

These results should be exact if possible, if not the results must differ by tolerances that depend on rounding mode.

27.2.6 Exception handling

It must be possible to signal to the user the occurence of the following conditions or exceptions.

- Divide by zero.
- Overflow.
- Underflow.
- Invalid operation.
- Inexact.

The ability to detect the above is a big step forward in our ability to write robust and portable code. These operations do occur in calculations and it is essential to have user programmer control over what action to take.

27.2.7 Rounding directions.

Four rounding directions are available:

- nearest – the default
- down
- up
- chop

Access to directed rounding can be used to implement interval arithmetic for example.

27.2.8 Rounding precisions.

The only mandatory part here is that machines that computations in extended mode let the programmer control the precision via a control word. This means that if software is being developed on machines that support extended modes they can be switched to a mode that would enable the software to run on a system that didn't support extended mode. This area looks like a can of worms. Look at the Kahan paper for more information – *Lecture Notes on the Status of IEEE 754*.

27.3 Resumé

The above has provided a quick tour on IEEE 754.

27.4 ematics

This is the best place to start:

http://www.cs.berkeley.edu/~darcy/Borneo/

Abstract

The design of Java relies heavily on experiences with programming languages past. Major Java features, including garbage collection, object-oriented programming, and strong static type checking, have all proved their worth over many years. However, Java breaks with tradition in its floating point support: instead of accepting whatever floating point format a machine might provide, Java mandates use of the nearly ubiquitous IEEE Standard for Binary Floating Point Arithmetic (IEEE 754-1985). Unfortunately, Java's specification creates several problems for numerical computation. Only a proper subset of IEEE 754's required features are supported by Java; useful IEEE 754 features are either explicitly forbidden or omitted from the Java specification. Java does not allow use of the IEEE 754 recommended double extended format on the x86. Using the double extended format often protects simple numerical formulas from floating point anomalies. Strict adherence to Java's floating point semantics leads to significant performance penalties on popular architectures, including both the x86 and PowerPC.

To address these problems, the Borneo language changes and extends Java so that all IEEE 754 features can be expressed and so that new numeric types can be easily created. Borneo allows either better hardware use than Java or (nearly) exact reproducibility while in all cases being predictable.

Summary

Unlike other languages designed to support IEEE 754 features (such as Modula-3, C9X, and RealJava), Borneo does not just add library functions to set and query the floating point state. Borneo has scoped language declarations to control the rounding mode, sticky flags, and trapping status. Lexical scoping permits more optimizations and makes reasoning about the program easier. One major change to Java not directly related to floating point is Borneo's addition of operator overloading. Besides the ability to overload existing operators, as in C++, Borneo also lets novel, user-defined, operators be defined and overloaded. Except for some new keywords, Borneo is upwards compatible with Java. Given a Java class P compiled to bytecode, another Java class cannot determine whether P was compiled under Borneo semantics or Java semantics. (Borneo semantics disallow some floating point optimizations permitted in Java).

Also have a look at

http://math.nist.gov/javanumerics/reports/jgfnwg-01.html

27.5 Bibliography

Hauser J.R., *Handling Floating Point Exceptions in Numeric Programs,* ACM Transaction on Programming Languages and Systems, Vol. 18, No. 2, March 1996, Pages 139-174.

The paper looks at a number of techniques for handling floating point exceptions in numeric code. One of the conclusions is for better structured support for floating point exception handling in new programming languages, or of course later standards of existing languages.

IEEE, *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*, Institute of Electrical and Electronic Engineers Inc.

The formal definition of IEEE 754.

Knuth D., *Seminumerical Algorithms*, Addison Wesley.

There is a coverage of floating point arithmetic, multiple precision arithmetic, radix conversion and rational arithmetic.

Sun, *Numerical Computation Guide*, SunPro.

Very good coverage of the numeric formats for IEEE Standard 754 for Binary Floating-Point Arithmetic. All SunPro compiler products support the features of the IEEE 754 standard.

27.5.1 Web based sources

http://validgh.com/goldberg/addendum.html

Differences Among IEEE 754 Implementations. The material in this paper will eventually be included in the Sun Numerical Computation Guide as an addendum to Appendix D, David Goldberg's *What Every Computer Scientist Should Know about Floating Point Arithmetic.*

http://docs.sun.com/

Follow the links to the *Floating Point and Common Tools AnswerBook.* The *Numerical Computation Guide* can be browsed on-line or downloaded as a pdf file. The last time we checked it was about 260 pages. Good source of information if you have Sun equipment.

http://www.validgh.com/

This web site contains technical and business information relating to the validgh professional consulting practice of David G. Hough. Contains links to the Goldber paper and the above addendum by Doug Priest.

http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html

Brief coverage of IEEE arithmetic with pointers to further sources. There is also a coverage of the storage layout and ranges of floating point numbers. Computer Science 341 is an introduction to the design of a computer's hardware, particularly the CPU and memory systems.

http://www.nag.co.uk/nagware/NP/TR.html

NAG provide coverage of TR 15580 and TR 15581. The first is the support Fortran has for IEEE arithmetic.

http://www.cs.berkeley.edu/~wkahan/

Willam Kahan home page.

http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

An Interview with the Old Man of Floating Point. Reminiscences elicited from William Kahan by Charles Severance, which appears in an issue of IEEE Computer - March 1998 (not confirmed).

http://www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps

Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic. Well worth a read.

http://www.stewart.cs.sdsu.edu/cs575/labs/l3floatpt.html

CS 575 Supercomputing – Lab 3: Floating Point Arithmetic. CS 575 is an interdisciplinary course to introduce students in the sciences and engineering to advanced computing techniques using the supercomputers at the San Diego Supercomputer Center (SDSC).

http://www.mathcom.com/nafaq/index.html

FAQ: Numerical Analysis and Associated Fielda Resource Guide. A summary of Internet resources for a number of fields related to numercial analysis.

http://www.math.psu.edu/dna/disasters/ariadne.html

The Explosion of the Ariane 5: A 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,768, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.

27.5.2 Hardware Sources

Osbourne A., Kane G., *4 and 8 Bit Microprocessor Handbook*, Osbourne/McGraw Hill.

Good source of information on 4 and 8 bit microprocessors.

Osbourne A., Kane G., *16 Bit Microprocessor Handbook*, Osbourne/McGraw Hill.

Ditto 16 bit microprocessors.

Intel, *386 DX Microprocessor Hardware Reference Manual*, Intel

The first Intel offering with 32 bit addressing.

Intel, *80386 System Software Writer's Guide*, Intel

Developer's Guide to the above.

http://www.intel.com/

Intel's home page.

http://developer.intel.com/design/pentiumiii/

Details of the Pentium III processor.

http://www.cyrix.com/

Cyrix home page.

Bhandarkar D.P., *Alpha Implementations and Architceure: Complete Reference and Guide*, Digital Press

Looks at some of the trade offs and design philosophy behind the alpha chip. The author worked with VAX, MicroVAX and VAX vectors as well as the Prism. Also looks at the GEM compiler technology that DEC/Compaq use.

http://www.digital.com/alphaserver/workstations/

Home page for the Compaq/DEC Alpha systems.

http://www.sgi.com/

Silicon Graphics home page.

http://www.sun.com/

Sun home page.

http://www.ibm.com/

IBM home page.

27.5.3 Operating Systems

Deitel H.M., *An Introduction to Operating Systems*, Addison Wesley.

The revised first edition includes case studies of Unix, VMS, CP/M, MVS and VM.

27.5.4 Java and IEEE 754

http://www.cs.berkeley.edu/~darcy/Borneo/

Borneo Language Homepage: Borneo is a dialect of the Java language designed to have true support for the IEEE 754 floating point standard. Well worth reading.

27.5.5 C and IEEE 754

http://wwwold.dkuug.dk/JTC1/SC22/WG14/

The official home of JTC1/SC22/WG14 - C. The C programming language standard ISO/IEC 9899 was adopted by ISO in 1990. ANSI then replaced their first standard X3.159 by the ANSI/ISO 9899 standard identical to ISO/IEC 9899:1990.

http://www.c9x.org/

Another source of information regarding C9X. There is a draft of the standard available and Annex F, G and H contain details of the changes concerning arithmetic.

28
Language Standardisation

Aims

The aims of this chapter are to provide a brief coverage of standardisation efforts, implementation differences and future developments.

28 Language Standardisation

This chapter looks at standardisation efforts, standard conformance, implementation differences and future developments.

28.1 Sun

Sun are obviously one of the major driving forces behind Java. However other companies quickly saw the potential that Java had and there was an attempt to get the language standardised. This has had mixed success. Currently Sun has pulled out of the standardisation exercise and there have been legal battles involving Microsoft being the most widely known.

The first place to start is at Sun's site. They make available the Java Language Specification, and the second edition draft is now available for public review. The draft includes all changes, clarifications and amendments made to the Java programming language since the publication of the first edition of the language specification in 1996. Of particular note is the full integration of the changes made in the 1.1 release of the Java platform into the specification, especially nested classes and interfaces. The document is available in the following formats:

- HTML (tar.Z, ~865K)
- HTML (zip, ~600K)
- PDF (~3706K)
- PostScript (tar.Z, ~1765K)
- PostScript (zip, ~1267K)

You've also seen what else they make available during the course by using the on-line documentation.