Mälardalens University
Department of Computer Science and Electronics
CD5130 Object Oriented programming

Ada 95

Abdul Rahim Rao (ral04001@student.mdh.se)
Muhammad Asif(amd04003@student.mdh.se)
Supervisor: Martin Skogeval

Abstract

It is rightly said by someone that current age is an age of information overload and that's why it is easier to write computer codes using different object oriented programming.Because Today there are many Object Oriented Programming (OOP) languages which are used to create software that are more realistically modeled, just like the real-world entities. OOP has been available for developers for about 15 years, because of the popularity of the C++, and C heritage, C++ is not completely object-oriented compared to Java which is completely object-oriented. Like Java, there are many languages that strictly follow OO features, for example,C++,C#,Java, Ada and many more. The main objective of our report is to emphasize on introduction of Ada, its basic concepts,usage of Ada in this modern period,what are the reasons that world's most popular companies and Departments willing to use the Ada based programs in their applications. A comparison between two languages will also be discussed in this report.

Table of Contents

Introduction

In the early days of computing, from the mid 1940 through the early 1960s, computers were very costly and were used primarily for research by government agencies and universities. After that period, the size and cost of computers decreased while their computing abilities increased dramatically.

We may define a program as a specific set of instructions that direct the computer to perform some computation. Programs are written in some specific programming languages. Originally, these languages were very cumbersome since they were written to be used for particulars computers. It became clear quite early that writing programs in such languages was tedious and easily error prone. The first step in the development of new programming languages was made by replacing these machine-language instructions by those that replaced strings of bits by alphabetic and numeric symbols. High level languages, the first of which was developed in 1952and called FORTRAN I, are those that are used most often by programmers. The major advantage of high level languages is that their instructions are easier to read and are portable to a wide variety of computers.

History of Ada

In the 1970s, a lot of programming languages were hardware independent, and none of them supported safe modular programming. In 1975 the Higher Order Language Working Group (HOLWG) was formed with the intent of reducing this number by finding or creating a programming language generally suitable for the department's requirements. And the result of this investigation was the Ada language. The US Department of Defense (DOD) required the use of Ada for every software project where new code was more than 30% of result, though exceptions to this rule were often granted. This requirement was effectively removed in 1997, as the DOD began to embrace Commercial Off The Shelf (COTS) technology. This version of the language is commonly known as Ada 83, from the date of its adoption by ANSI. Ada 95, the joint ISO/ANSI standard is the latest standard for Ada. Ada 95 the first ISO standard object-oriented programming language is accepted in 1995. To help with the standard revision and future acceptance, the US Air Force funded the

development of the GNAT Compiler. The GNAT Compiler is part of the GNU Compiler Collection.

Work continues on improving and updating the technical content of the Ada programming language. A Technical Corrigendum to Ada 95 was published in October 2001.

What Is Ada

Ada is a modern programming language designed for large, long-lived applications - and embedded systems in particular - where reliability and efficiency are essential. It was originally developed in the early 1980s (this version is generally known as Ada 83) and then revised and enhanced in an upward compatible fashion in the mid 1990s. The resulting language, Ada 95, was the first internationally standardized (ISO) Object-Oriented Language. Under the auspices of ISO, the language is currently undergoing another (minor) revision, scheduled for completion in the form of an amendment to the standard in 2005.

The name "Ada" is not an acronym; it was chosen in honour of Augusta Ada Lovelace (1815-1852), a mathematician who is sometimes regarded as the world's first programmer because of her work with Charles Babbage. She was also the daughter of the poet Lord Byron.

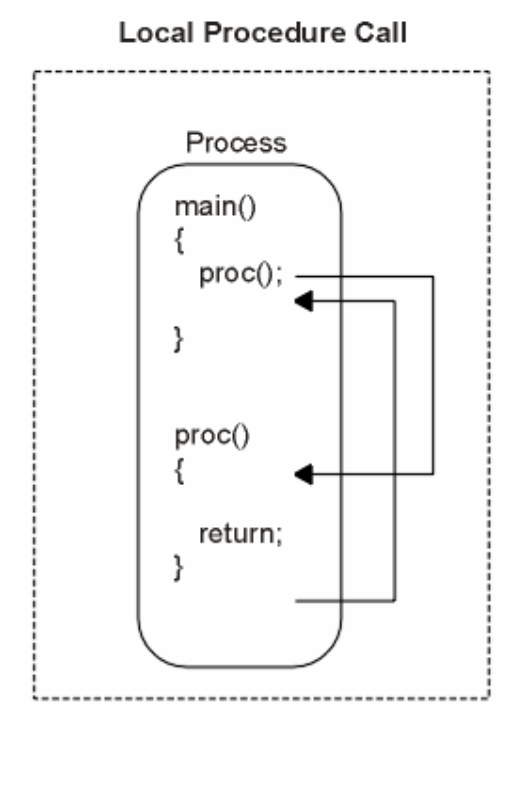
Ada 95 is seeing significant usage worldwide in the high-integrity and safety-critical domains including commercial and military aircraft avionics, air traffic control, railroad systems, and medical devices. With its embodiment of modern software engineering principles Ada is an excellent teaching language for both introductory and advanced computer science courses, and it has been the subject of significant university research especially in the area of real-time technologies.

AdaCore has a long and intimate history with the Ada programming language. Company members worked on the original Ada 83 designs and review, played key roles in the Ada 95 project, and continue to be deeply involved with the Ada 05 revision process. The initial GNAT compiler was essential to the growth of Ada 95;

the company delivered it at the time of the language's standardization, thus guaranteeing that users would have a quality implementation for transitioning to Ada 95 from Ada 83 or other languages.

Ada is a structured, statically typed programming language while it addresses much the same tasks as C or C++, but it has the type-safety of a language like Java.

Ada is a procedural language. It is called a procedural language since the programmers when programming in Ada, break their program into smaller programs or procedures in order to handle different parts of the given problem. These programs operate by calling the procedures one after another to solve the entire problem.



The figure given above shows the procedure Call mechanism in Ada

Language Overview

The Ada programming language was designed in response to a problem posed in the mid 1970 by the US Department of defence. Ada is multi-faceted. From one perspective it is a classical stack-based general-purpose language, not tied to any specific development methodology. It has a simple syntax, structured control statements, flexible data composition facilities, strong type checking, traditional features for code modularization ("subprograms"), and a mechanism for detecting and responding to exceptional run-time conditions ("exception handling").

One of the important concepts in Ada is the idea of encapsulating items together to form a package which may be re-used in other programs. The library package Ada.Text_Io is provided on Ada systems to allow the input and output of textual information to and from the user's program.
Ada is a programming language designed to support the construction of long-lived, highly reliable software systems. The language includes facilities to define packages of related types, objects, and operations. The packages may be parameterized and the types may be extended to support the construction of libraries of reusable, adaptable software components. The operations may be implemented as subprograms using conventional sequential control structures, or as entries that include synchronization of concurrent threads of control as part of their invocation. The language treats modularity in the physical sense as well, with a facility to support separate compilation.

The language includes a complete facility for the support of real-time, concurrent programming. Errors can be signalled as exceptions and handled explicitly. The language also covers systems programming; this requires precise control over the representation of data and access to system-dependent properties. Finally, a predefined environment of standard packages is provided, including facilities for,

among others, input-output, string manipulation, numeric elementary functions, and random number generation.

Comparison of Ada to different languages

A simple overview of comparison of ada among most popular languages because Some people believe that Java is "like" C++. However, usual C++ idioms do not carry over to Java because Java is not a superset of C++. Java is not even a subset of C++; it can at best be seen as a derivation with many modifications and extensions. Thus the syntax of Java and C++ may seem similar at a first glance, but the semantics and philosophy of these two languages is very different. [4]

Java is closer to Ada 95 than to C++, syntax notwithstanding.

A minimal comparison of Java with C++ and Ada 95

	Java	C++	Ada 95
Inheritance	Single (but with multiple subtyping)	Multiple	Single (but supports MI)
Preprocessor	No	Yes	No
Separate Interface/Implementation	No (interface generated from code)	Yes (header files)	Yes (specifications)
Garbage Collection	Yes	No	Yes
Operator Overloading	No	Yes	Yes
Pointer Arithmetic	No	Yes	No
Generics	No (but extensive polymorphism)	Yes ("templates")	Yes
Exceptions	Yes	Yes	Yes
Native Multi-threading	Yes	No	Yes

Ada At Work

The use of Ada language in this modern era in most famous and secure regions, convert our attention that Ada language is still most popular language as compare to other modern languages i.e. C++.Java,XML,.Net,COM etc.Programmers can use this language most confidently.Practicaly usage of Ada language in some most intelligent fields is under:

- **Flight**
  - Boeing 777
  - Helicoptors
  - Flight control
  - Landing Gear
  - BE-200

- **Transportation**
  - New York City Subway
  - Paris Subway
  - Cairo & Calcutta Metros
  - European Train Control
  - Ship System 2000
  - Trains in Class
  - GPS

- **Space**
  - Space Station Robot Embeds Ada (.pdf)

- **Manufacturing**
  - Weirton Steel Mill
  - Furniture Making

- **Safety & Testing**
  - Pratt & Whitney
  - Nuclear Power Plant

- **Other**
  - Pollution Monitoring
  - Video Security
  - Ada Cuts Time 99.5%
  - For MS Windows
  - Radio Telescope
  - Editing Videos [2]

Programming in the large

The original Ada 83 design introduced the package construct, a feature that supports encapsulation ("information hiding") and modularization, and that allows the developer to control the namespace that is accessible within a given compilation unit. Ada 95 introduced the concept of "child units," adding considerably flexibility and easing the design of very large systems

Generic templates

A key to reusable components is a mechanism for parameterizing modules with respect to data types and other program entities, for example a stack package for an arbitrary element type. Ada meets this requirement through a facility known as "generics"; since the parameterization is done at compile time, run-time performance is not penalized.

A generic unit is a program unit that is either a generic subprogram or a generic package. A generic unit is a template, which can be parameterized, and from which

corresponding (nongeneric) subprograms or packages can be obtained. The resulting program units are said to be instances of the original generic unit.

A generic unit is declared by a generic_declaration. This form of declaration has a generic_formal_part declaring any generic formal parameters. An instance of a generic unit is obtained as the result of a generic_instantiation with appropriate generic actual parameters for the generic formal parameters. An instance of a generic subprogram is a subprogram. An instance of a generic package is a package.

Generic units are templates. As templates they do not have the properties that are specific to their nongeneric counterparts. For example, a generic subprogram can be instantiated but it cannot be called. In contrast, an instance of a generic subprogram is a (nongeneric) subprogram; hence, this instance can be called but it cannot be used to produce further instances

Object-Oriented Programming (OOP)

Ada 83 was object-based, allowing the partitioning of a system into modules corresponding to abstract data types or abstract objects. Full OOP support was not provided since, first, it seemed not to be required in the real-time domain that was Ada's primary target, and, second, the apparent need for automatic garbage collection in an OO language would have interfered with predictable and efficient performance.

However, large real-time systems often have components such as GUIs that do not have real-time constraints and that could be most effectively developed using OOP features. In part for this reason, Ada 95 supplies full support for OOP, through its "tagged type" facility: classes, polymorphism, inheritance, and dynamic binding. Ada 95 does not require automatic garbage collection but rather supplies definitional features allowing the developer to supply type-specific storage reclamation operations ("finalization").

Ada is methologically neutral and does not impose a "distributed overhead" for OOP. If an application does not need OOP, then the OOP features do not have to be used, and there is no run-time penalty.

Concurrent programming

Ada supplies a structured, high-level facility for concurrency. The unit of concurrency is a program entity known as a "task." Tasks can communicate implicitly via shared data or explicitly via a synchronous control mechanism known as the rendezvous. A shared data item can be defined abstractly as a "protected object" (a feature introduced in Ada 95), with operations executed under mutual exclusion when invoked from multiple tasks. Asynchronous task interactions are also supported, specifically timeouts and task termination. Such asynchronous behavior is deferred during certain operations, to prevent the possibility of leaving shared data in an inconsistent state.

Systems programming

Both in the "core" language and the Systems Programming Annex, Ada supplies the necessary features to allow the programmer to get close to the hardware. For example, you can specify the bit layout for fields in a record, define the alignment and size, place data at specific machine addresses, and express specialized or time-critical code sequences in assembly language. You can also write interrupt handlers in Ada, using the protected type facility.

Real-time programming

Ada's tasking features allow you to express common real-time idioms (periodic tasks, event-driven tasks), and the Real-Time Annex provides several facilities that allow you to avoid unbounded priority inversions. A task dispatching policy is defined that basically requires tasks to run until blocked or preempted. A protected object locking policy is defined that uses priority ceilings; this has an especially efficient implementation in Ada (mutexes are not required) since protected operations are not allowed to block.

High-integrity systems

With its emphasis on sound software engineering principles Ada supports the development of high-integrity applications, including those that need to be certified against safety standards such as DO-178B. For example, strong typing means that data intended for one purpose will not be accessed via inappropriate operations; errors such as treating pointers as integers (or vice versa) are prevented.
However, the full language is inappropriate in a safety-critical application, since the generality and flexibility may interfere with traceability / certification requirements. Ada addresses this issue by supplying a compiler directive, pragma Restrictions, that allows you to constrain the language features to a well-defined subset (for example, excluding exception handlers). One of the most interesting restrictions is the Ravenscar Profile, a collection of concurrency features that are powerful enough for real-time programming but simple enough to make certification practical.

Ada Benefits Summary

- Helps you design safe and reliable code
- Reduces development costs
- Supports new and changing technologies
- Facilitates development of complex programs
- Helps make code readable and portable
- Reduces certification costs for safety-critical software

Ada Features Summary

- Object orientated programming
- Strong typing
- Abstractions to fit program domain
- Generic programming/templates
- Exception handling
- Facilities for modular organization of code
- Standard libraries for I/O, string handling, numeric computing
- Systems programming
- Concurrent programming

- Real-time programming
- Distributed systems programming
- Interfaces to other languages (C, COBOL, Fortran)

In brief, Ada is an internationally standardized language combining object-oriented programming features, well-engineered concurrency facilities, real-time support, and built-in reliability. An appropriate tool for addressing the real issues facing software developers today, Ada is used throughout a number of major industries to design software that protects businesses and lives.

Ada data types:

A data type is a collection of values together with operations that are used to manipulate those values in some way. An example of a data type is the set of all even integers with the operations of addition, subtraction and multiplication defined on these integers. In adda data types classify into two general categories: predefined and programmer defined.

Predefined data types

The predefined data types are built into the implementation

Float type

The float data type stores numeric values that are expressed to several decimal places of accuracy. Such values are commonly called floating point numbers. The advantage of having FLOAT values in part lies in the ability to express a wider range of numbers at the possible expense of less precision.

Boolean type

Ada has a predefined BOOLEAN data type for dealing with such situations named after the 19th century British mathematician and logician, George Boole. The BOOLEAN type has only two values: TRUE and FALSE.

Programmer defined data types

Any programmer defined type must be originate with the programmer, who creates the set of values and defines all operations that apply

Control statement and program flow- conditional and block flow

- Loops
- Subprograms
- Arrays and strings
- Applications of array processing
- Records
- Ada types and their declaration
- Packages: design and Implementation
- Exception handling and Text file processing
- Generics
- Access types
- Recursive Methods

- Concurrency and task structures

EXAMPLE PROGRAMS

Here are some sample programs designed in Ada language.

```
package CharStak is

procedure Push(In_Char : in CHARACTER);  -- In_Char is added to the
                                         -- stack if there is room.

procedure Pop(Out_Char : out CHARACTER); -- Out_Char is removed from
                                         -- stack and returned if a
                                         -- character is on stack.
                                         -- else a blank is returned

function Is_Empty return BOOLEAN;        -- TRUE if stack is empty

function Is_Full return BOOLEAN;         -- TRUE if stack is full

function Current_Stack_Size return INTEGER;

procedure Clear_Stack;                   -- Reset the stack to empty

end CharStak;
```

```
package body CharStak is

Maximum_Size : constant := 25;
Stack_List : STRING(1..Maximum_Size); -- The stack itself, purposely
                                      -- defined very small.
Top_Of_Stack : INTEGER := 0;          -- This will always point to
                                      -- the top entry on the stack.

procedure Push(In_Char : in CHARACTER) is
begin
   if not Is_Full then
      Top_Of_Stack := Top_Of_Stack + 1;
      Stack_List(Top_Of_Stack) := In_Char;
   end if;
end Push;


procedure Pop(Out_Char : out CHARACTER) is
begin
   if Is_Empty then
      Out_Char := ' ';
   else
      Out_Char := Stack_List(Top_Of_Stack);
      Top_Of_Stack := Top_Of_Stack - 1;
   end if;
end Pop;
```

```
function Is_Empty return BOOLEAN is
begin
   return Top_Of_Stack = 0;
end Is_Empty;


function Is_Full return BOOLEAN is
begin
   return Top_Of_Stack = Maximum_Size;
end Is_Full;


function Current_Stack_Size return INTEGER is
begin
   return Top_Of_Stack;
end Current_Stack_Size;


procedure Clear_Stack is
begin
   Top_Of_Stack := 0;
end Clear_Stack;

end CharStak;
```

The above program code illustrates how we can define our own stack for using in our programs. A Stack can be defined as list of homogeneous items which always grows and shrinks in this way that an item will be removed from stack which will enter in last with LIFO(Last In First Out) method.

In order to keep it simple, we will only allow the stack to store **CHARACTER** type variables, although it could be defined to store any type of variables we desired, even arrays or records.[3]

The Use of Dynamic Strings

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_text_IO; use Ada.Integer_Text_IO;
with DynStrng; use DynStrng;

procedure TryStrng is

   Try_This : STRING(1..13);
   Name     : DYNAMIC_STRING(0..15);
   Stuff    : DYNAMIC_STRING(0..35);
   Result   : BOOLEAN;
   Neat     : constant STRING := "XYZ";
```

```
   Good3    : STRING(1..3);
   Good4    : STRING(1..4);
   Column   : INTEGER;

begin

   Name(0) := CHARACTER'VAL(3);
   Stuff(0) := CHARACTER'VAL(7);

   Put(Size_Of(Name));
   Put(Size_Of(Stuff));
   Put(Length(Name));
   Put(Length(Stuff));
   New_Line;

   Try_This := "ABCDEFGHIJKL$";
   Copy(Try_This,Stuff,Result);
   Put(Size_Of(Stuff));
   Put(Length(Stuff));
   Put(Stuff); Put(Stuff);
   New_Line(2);

   Copy(Stuff,Name,Result);
   Put(Name); Put(Name); Put(Name); New_Line;

   Concat(Name,Name,Stuff,Result);
   Put(Stuff); New_Line;

   Delete(Stuff,5,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line;
   Delete(Stuff,6,3,Result);
   Put(Stuff); New_Line(2);

   Try_This := "1234567890123";
   Copy(Try_This,Stuff,Result);
   Copy(Neat,Name,Result);
   Put(Stuff); Put(Name); New_Line;

   Insert(Stuff,Name,5,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,50,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,2,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,24,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,5,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,5,Result);
```

```
   Put(Stuff); New_Line;
   Insert(Stuff,Name,5,Result);
   Put(Stuff); New_Line;
   Insert(Stuff,Name,5,Result);
   Put(Stuff); New_Line(2);

   Good3 := "123";
   Try_This := "1234567890123";
   Copy(Try_This,Stuff,Result);
   Copy(Good3,Name,Result);
   Pos(Stuff,Name,1,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;
   Pos(Stuff,Name,2,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;
   Pos(Stuff,Name,7,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;
   Pos(Stuff,Name,12,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;
   Pos(Stuff,Name,18,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;
   Pos(Stuff,Name,50,Column,Result);
   Ada.Text_IO.Put("Found in column number"); Put(Column); New_Line;

end TryStrng;
```

The Above code is designed to use the dynamic string package in various ways by defining strings, inserting characters or strings, deleting portions of strings, and displaying the results. This program was written to test the **DynStrng** package so it does a lot of silly things. There is nothing new or innovative in this utilitarian program, so we will be left on our own to understand, compile, and execute it.[3]

Conclusion

Ada is among the one of the most progressing and emerging object oriented language. As it has the strong typing and abstraction to fit program domain as well it is also supports exception handling and generic programming templates. As it is excellent exception handling and supports standard libraries. Ada has also proved it self for the system, concurrent, real-time and distributed system programming. Ada is also flexible as it supports the interface to other languages.

References:

[1] A first course in computer science with ADA , Nicholas J.Delillo.

[2] http://www.adaic.org/atwork/index.html

[3] http://www.infres.enst.fr/~pautet/Ada95/chap16.htm

[4] http://www.adahome.com/Resources/Languages/chart3.html