Introduction to Ada for Beginning and Experienced Programmers

Jean-Pierre Rosen
Adalog
www.adalog.fr

Ada
Time-tested, safe and secure

Brief History of Ada

- Named after Ada Byron, countess of Lovelace (1815-1852)
- 1983: The basis
  - First industrial language with exceptions, generics, tasking
- 1995: OOP, protected objects, hierarchical libraries
  - First standardized object-oriented language
- 2005: Interfaces, improving existing features
  - Putting it all together
- 2012 : Contracts, higher level expressions
  - Going formal

Ada 2022 coming soon !

A Free Language

- An international standard
  - ISO 8652:2012, freely available
  - Does not belong to any company
  - Entirely controlled by its users
- Free (and proprietary) compilers
- Many free resources
  - Components, APIs, tutorials…
  - http://www.adaic.com, http://getadanow.com, http://libre.adacore.com...
- A dynamic community
  - Newgroups : comp.lang.ada, fr.comp.lang.ada
  - LinkedIn, Reddit, IRC, Identi.ca, Stack Overflow, GNU Go Ada Initiative...

Who Uses Ada?

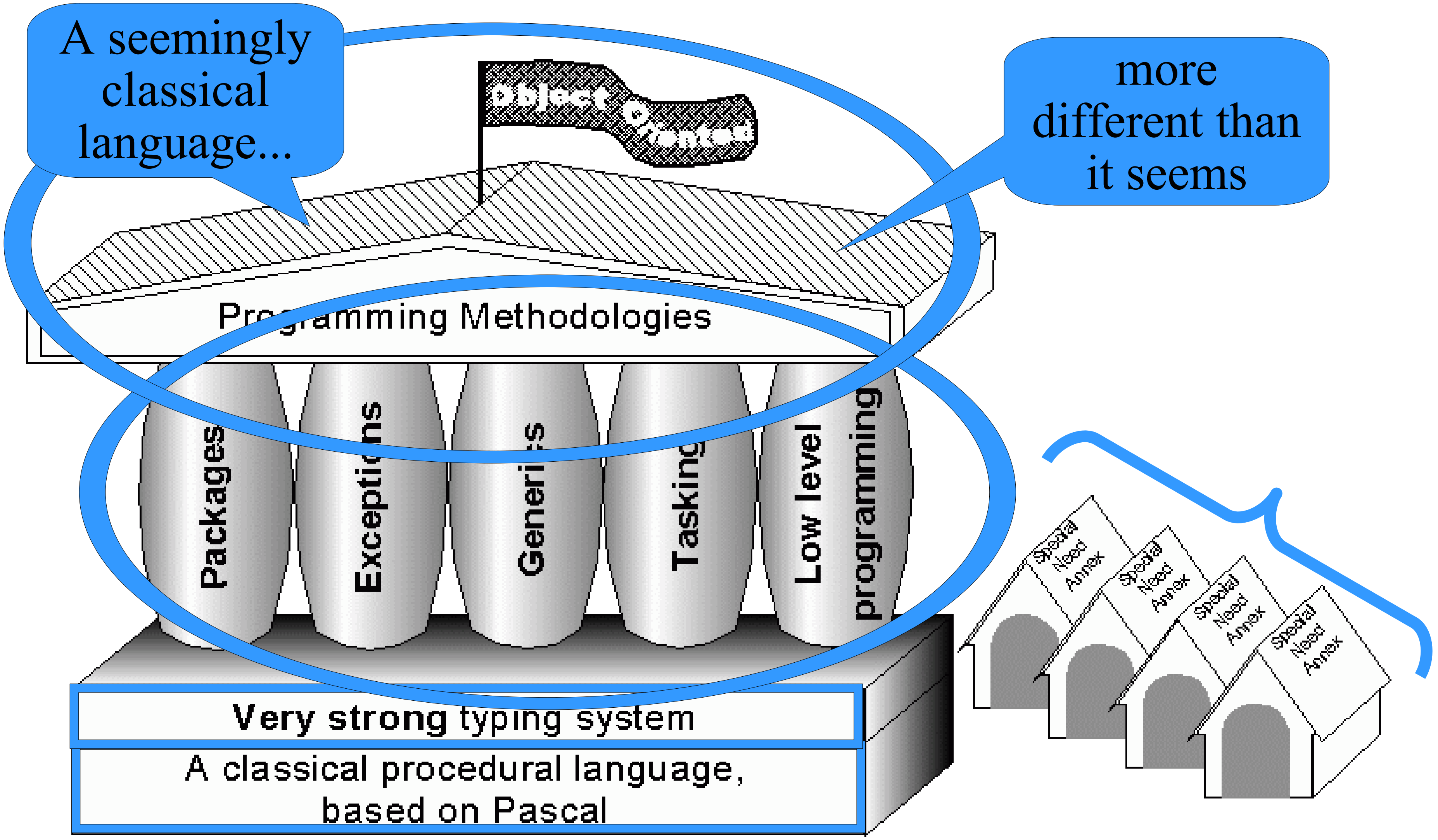
Why Use Ada?

- When failure is not an option
  - Of course, Ada is used in safety critical systems...
- Other systems should not fail!
  - Buffer overflows are still the most common origin of security breaches
  - Arithmetic overflow, illegal pointers, memory leaks...
- Ada checks a lot at compile time
  - Bad design doesn't compile!

What's important in a language is not what it allows

What's important in a language is what it forbids

What's in Ada


A seemingly classical language...
more different than it seems
Object Oriented
Programming Methodologies
Packages
Exceptions
Generics
Tasking
Low level programming
Very strong typing system
A classical procedural language, based on Pascal
Special Need Annex
Special Need Annex
Special Need Annex
Special Need Annex

The Building-Block Approach

The Building-Block Approach

Readable, Pascal-Like Syntax

```
for C in Colour loop
   I := I + 1;
end loop;

while I > 1 loop
   I := I - 1;
end loop;

Main_Loop :
loop
   I := I + 1;

   exit Main_Loop when I = 100;
   I := I + 2;
end loop Main_Loop;
```

Cannot cheat with loop control

```
if I in 1 .. 10 then
   Result := Red;
elsif I in 11 .. 20 then
   Result := Green;
elsif I in 21 .. 30 then
   Result := Blue;
end if;

case I is
   when 1 .. 10   =>
      Result := Red;
   when 11 .. 20 =>
      Result := Green;
   when 21 .. 30 =>
      Result := Blue;
   when others    =>
      Result := Red;
end case;
```

All possible cases must be given

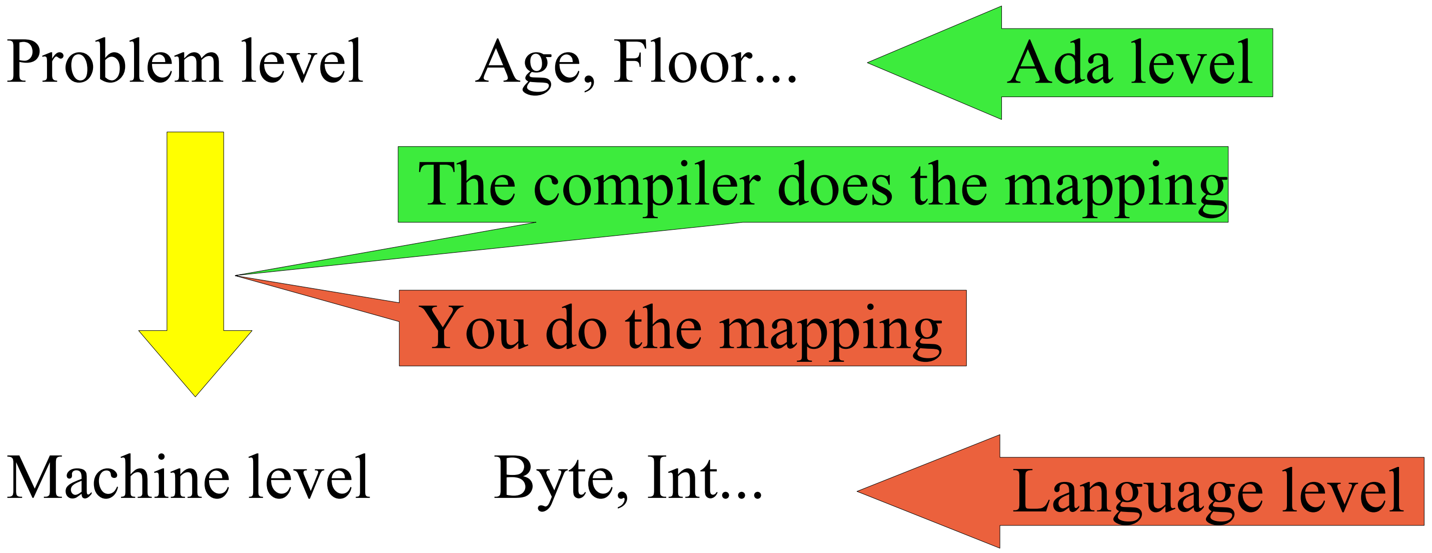
```
Mat := ((1, 0, 0),
        (0, 1, 0),
        (0, 0, 1));

Head := new Node'(Value=> 10_000,
                  Next => new Node'(Value=> 2009, Next=> null);
```

Strong Typing System

```
type Age   is range 0..125;
type Floor is range -5 .. 15;

My_Age   : Age;
My_Floor : Floor;
  ...
My_Age   := 10;       -- OK
My_Floor := 10;       -- OK
My_Age   := My_Floor; -- FORBIDDEN !
```



Packages (1)

```
package Colour_Manager is
   type Colour is private;
   type Density is delta 1.0/256.0 range 0.0 .. 1.0;

   Red, Green, Blue : constant Colour;

   function "+" (Left, Right : Colour) return Colour;
   function "*" (Coeff: Density; Origin : Colour) return Colour;

private
   type Colour is
      record
         R_Density, G_Density, B_Density : Density;
      end record;
   Red   : constant Colour := (1.0, 0.0, 0.0);
   Green : constant Colour := (0.0, 1.0, 0.0);
   Blue  : constant Colour := (0.0, 0.0, 1.0);
end Colour_Manager;
```

```
package body Colour_Manager is
   ...
end Colour_Manager;
```

Packages (2)

```
with Colour_Manager;
procedure Paint is
   use Colour_Manager;
   My_Colour : Colour := 0.5*Blue + 0.5*Red;
begin
   -- Make it darker
   My_Colour := My_Colour * 0.5;
   My_Colour := My_Colour / 2.0; -- Forbidden (or define "/")
   ...
end Paint;
```

Abstractions are enforced

Dependences are explicit
➔ no makefiles!

Discriminated Types

```
type Major  is (Letters, Sciences, Technology);
type Grade is delta 0.1 range 0.0 .. 20.0;

type Student_Record (Name_Length : Positive;
                     With_Major  : Major)
is record
   Name    : String(1 .. Name_Length); --Size depends on discriminant
   English : Grade;
   Maths   : Grade;

   case With_Major is     -- Variant part, according to discriminant
      when Letters =>
         Latin : Grade;
      when Sciences =>
         Physics   : Grade;
         Chemistry : Grade;
      when Technology =>
         Drawing : Grade;
   end case;
end record;
```

Discriminants

Discriminants are to data
what parameters are to subprograms

Object Oriented Programming

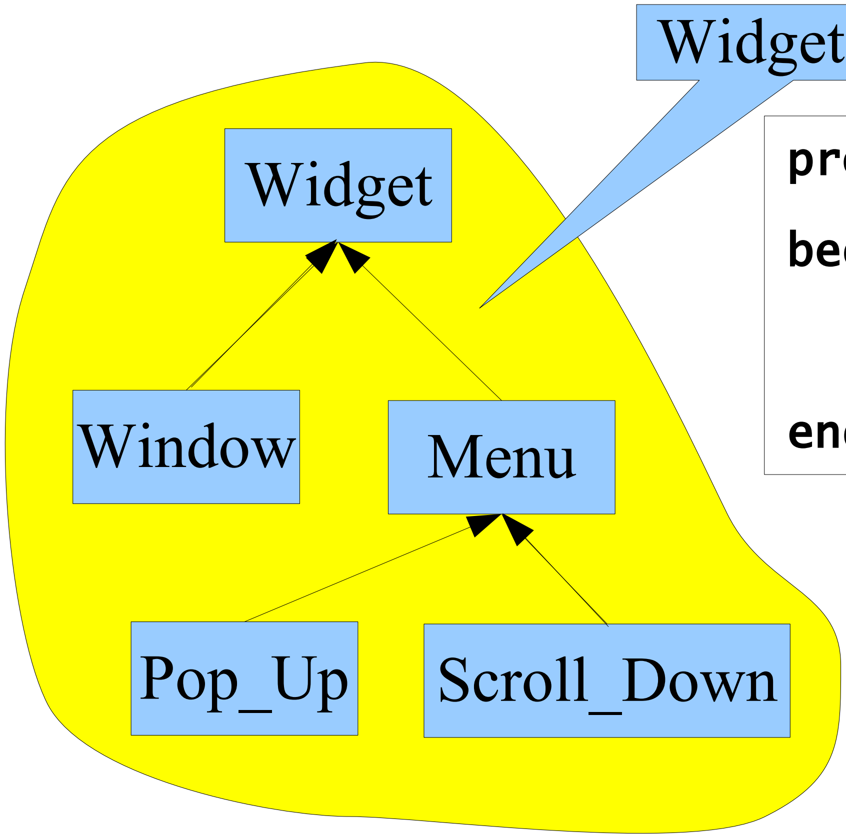
- Packages support encapsulation
- Tagged types support dynamic binding
- A class = Encapsulation + dynamic binding
  - Design pattern: a tagged type in a package

```
package Widget is
   type Instance is tagged private;
   procedure Paint (Self : Instance);
   ...
private
   ...
end Widget;
```

```
package Menu is
   type Instance is new Widget.Instance with private;
   procedure Paint (Self : Instance);
   ...
private
   ...
end Widget;
```

Object Oriented Programming

- Differentiate *specific* type from *class-wide* type



```
procedure Move (Item : Widget'Class;
                X, Y : Coordinates) is
begin
   Erase (Item);
   Set_Position (Item, X, Y);
   Paint (Item);
end Move;
```

```
   P : Widget.Menu.Pop_Up.Instance;
   W : Widget.Window.Instance;
begin
   Move (P, X => 23, Y => 45);
   Move (W, Y => 19, X => 23);

   -- Ada 2005:
   P.Move (X => 23, Y => 45);
   W.Move (Y => 19, X => 23);
   ...
```

Interfaces (Ada 2005+)

- A type can be derived from one tagged type and several interfaces
  - Methods of an interface are abstract <u>or null</u>

```
with Ada.Text_IO; use Ada.Text_IO;
package Persistance is
   type Services is interface;

   procedure Read  (F : File_Type; Item : out Services) is abstract;
   procedure Write (F : File_Type; Item : in  Services) is abstract;
end Persistance;
```

```
type Persistant_Window is
   new Widget.Window.Instance and Persistance.Services;
```

Exceptions

- Every run-time error results in an exception
  - Buffer overflow
  - Dereferencing null
  - Device error
  - Memory violation (in C code!)
  - ...
- Every exception can be handled

Once you've taken care of the unexpected...

..take care of the unexpected unexpected

Generics

- Provide algorithms that work on any data type with a *required* set of properties

```
generic
   type Item is private;
procedure Swap (X, Y : in out Item);

procedure Swap (X, Y : in out Item) is
   Temp : Item;
begin
   Temp := X;
   X    := Y;
   Y    := Temp;
end Swap;
```

```
   procedure Swap_Age is new Swap (Age);
   My_Age, His_Age : Age;
begin
   Swap_Age (My_Age, His_Age);
```

Tasking

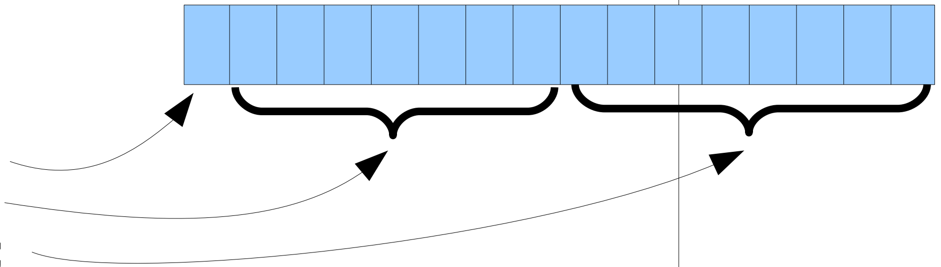
- Tasking is an integral part of the language
  - Not a library
- Tasks (*threads*) are high level objects
- High level communication and synchronization
  - Rendezvous (client/server model)
  - Protected objects (passive monitors)
- Tasking is easy to use
  - Don't hesitate to put tasks in your programs!

Access to Low Level

- Let the compiler do the hard work
  - You describe the high level view
  - You describe the low level view
  - You work at high level, and get what you want at low level

```
type BitArray is array (Natural range <>) of Boolean;
type Monitor_Info is
   record
      On     : Boolean;
      Count  : Natural range 0..127;
      Status : BitArray (0..7);
   end record;

for Monitor_Info use
   record
      On     at 0 range 0 .. 0;
      Count  at 0 range 1 .. 7;
      Status at 0 range 8 .. 15;
   end record;
```



Access to Low Level

- Let the compiler do the hard work
  - You describe the high level view
  - You describe the low level view
  - You work at high level, and get what you want at low level

```
type BitArray is array (Natural range <>) of Boolean;
type Monitor_Info is
   record
      On     : Boolean;
      Count  : Natural range 0..127;
      Status : BitArray (0..7);
   end record;

for Monitor_Info use
   record
      On     at 0 range 0 .. 0;
      Count  at 0 range 1 .. 7;
      Status at 0 range 8 .. 15;
   end record;
```

```
   MI : Monitor_info;
begin
   MI.Status(3) := False;
```

```
ANDB [BP-11],-9
```

Really Low Level

```
KBytes : constant := 1024;

Memory : Storage_Array (0..640*KBytes-1);
for Memory'Address use To_Address(0);

procedure Poke (Value : Byte; Into : Storage_Offset) is
begin
   Memory (Into) := Value;
end Poke;

function Peek (From : Storage_Offset) return Byte is
begin
   return Memory (From);
end Peek;
```

- You can include machine code...
- You can handle interrupts...

Everything can be done in Ada,
provided it is stated **clearly**

Special Needs Annexes

- An annex is an extension of the standardisation for specific problem domains.
  - An annex contains no new syntax. An annex may define only packages, pragmas or attributes.
- System Programming Annex
- Real-Time Annex
- Distributed Systems Annex
- Information Systems Annex
- Numerics Annex
- Safety and Security Annex

A Portable Language

- Really portable!
  - Configure/automake/conditional compilation... only compensate for the lack of portability
  - The virtual machine concept is just a workaround for the lack of portability of programming languages.
  - But there are Ada compilers for the JVM and .net as well…
- All compilers implement *exactly* the same language
  - and are checked by passing a conformity suite
- High level constructs protect from differences between systems

Linux, Windows: 100% same code

Ease of Writing

- Try GNAT's error messages!

```
procedure Error is
   Lines : Integer;
begin
   Line  := 3;
   Lines = 3;
end Error;
```

```
error.adb:4:04: "Line" is undefined
error.adb:4:04: possible misspelling of "Lines"
```

```
error.adb:5:10: "=" should be ":="
```

- The language protects you from many mistakes
  - Strong typing is not a pain, it's a help!
  - If it compiles, it works...
  - Spend your time on *designing*, not chasing stupid bugs

Components and Tools

- Ada interfaces easily with other languages
  - Bindings are available for most usual components
    - Posix, Win32, X, Motif, Gtk, Qt, Tcl, Python, Lua, Ncurses, Bignums, Corba, MySQL, PostGres…
- Unique to Ada:
  - AWS (Ada Web Server)
    - A complete web development framework
  - ASIS (Ada Semantic Interface Specification)
    - Makes it easy to write tools to process and analyze Ada sources
  - Many more…

Conclusion

Try Ada !

…and discover what higher level programming means