Introduction à la programmation en Ada

23/04/2016 Courbevoie

Présenté par Guillaume Foliard (Ada-France)

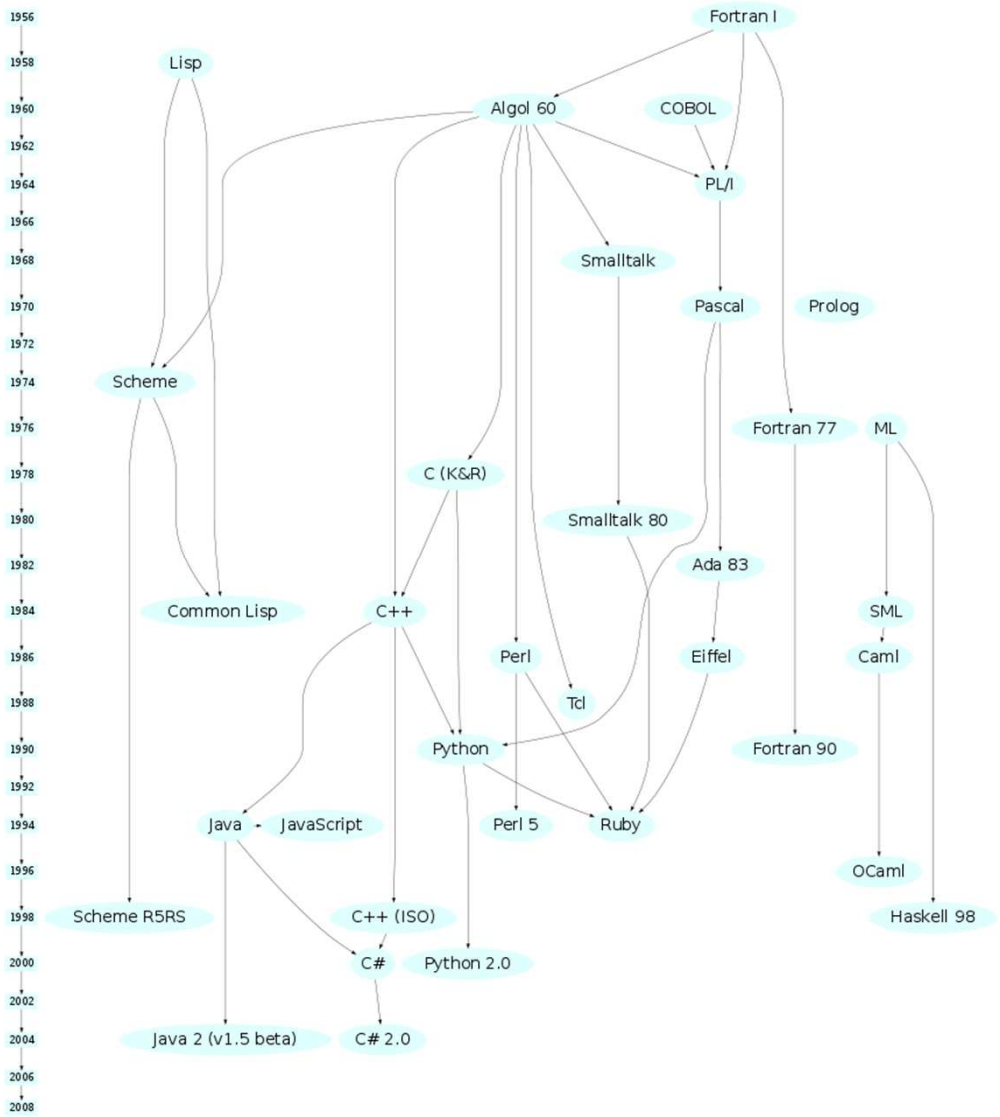
Remerciements à

- **Jean-Pierre Rosen (Ada-France, AdaLog)**
- **Xavier Grave (Ada-France)**

Objectifs

- Donner un aperçu des principaux concepts et outils mis à disposition par le langage Ada
  - Eléments de base
  - Définition de données
  - Structure d'un programme et modularité
  - Programmation Orientée Objet
  - Parallélisme
  - Programmation multi-langages
  - Bibliothèques et outils

Eléments de base



Eléments de base

- Syntaxe
  - A base de mots-clés, peu de symboles comme en C
  - Privilégie la lisibilité, proche du langage naturel
  - Non ambigüe

```
function Calcule
   (Valeur_A : in Integer;
    Valeur_B : in Integer)
    return Integer
is
   Resultat : Integer;
begin
   Resultat := Valeur_A + Valeur_B;
   return Resultat;
end Calcule;
```

Eléments de base

- Structures de contrôle
  - Boucles

```
for <variable> in <domaine>
loop
   <séquence d'instructions>
end loop;
```

```
while <condition>
loop
   <séquence d'instructions>
end loop;
```

```
for <variable> of <conteneur>
loop
   <séquence d'instructions>
end loop;
```

```
loop
   <séquence d'instructions>
   exit when <condition>;
   <séquence d'instructions>
end loop;
```

  - Alternatives

```
if <condition>
then
   <séquence d'instructions>
elsif <condition>
then
   <séquence d'instructions>
else
   <séquence d'instructions>
end if;
```

```
case <variable> is
   when <domaine> =>
   <séquence d'instructions>

   when <domaine> =>
   <séquence d'instructions>
end case;
```

Eléments de base

- Exemples

```
for Colour in Palette loop
   Index := Index + 1;
end loop;

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

Main_Loop :
loop
   Index := Index + 1;

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

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

case Index is
   when 1 .. 10  =>
      Result := Red;
   when 11 .. 20 =>
      Result := Green;
   when 21 .. 30 =>
      Result := Blue;
   when others   =>
      Result := Red;
   -- all cases must be processed
end case;
```

Définition de données

- Un concept central du langage Ada est celui de typage fort
- Le typage des données permet de décrire leur nature et leur structure
- Le fait qu'il soit fort est un outil supplémentaire au service du programmeur qui permet au compilateur de rechercher automatiquement les contradictions qui sont alors considérées comme des erreurs

Définition de données

- Définition de la nature des données

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

Mon_Age       : Age;
Dernier_Etage : Etage;
   ...
Mon_Age       := 10;             -- OK
Dernier_Etage := Etage'Last;     -- OK
Mon_Age       := Dernier_Etage; -- Incorrect !
```

- Par défaut le compilateur détermine automatiquement la représentation physique la plus adaptée
  - Octet, mot de 16 bits, 32 bits, etc…
  - Le programmeur peut influer sur ce choix

Définition de données

- Structuration des données
  - Types simples

```
type <nom_type> is range <domaine>;
```

```
type <nom_type> is (<liste_enumérés>);
```

```
type <nom_type> is access <nom_type>;
```

  - Types complexes

```
type <nom_type> is array (<nom_type>) of <nom_type>;
```

```
type <nom_type> is record
   <nom_donnée> : <nom_type>);
   <nom_donnée> : <nom_type>);
   ...
end record;
```

```
type <nom_type> is (<nom_discriminant> : <nom_type>) record
   case <nom_discriminant> is
      when <valeur> =>
         <nom_donnée> : <nom_type>);
      when <valeur> =>
         <nom_donnée> : <nom_type>);
      ...
   end case;
end record;
```

Définition de données

- Dérivation de types
  - Tout type peut être dérivé en sous-types avec des contraintes affinées

```
type Age is range 1 .. 25000;

subtype Age_Homme is Age range 1 .. 125;

subtype Age_Tortue is Age range 1 .. 200;
```

  - Un type et ses sous-types sont compatibles entre eux, les contraintes seront vérifiées à l'exécution.

```
Age_Jeanne    : Age_Homme;
Age_Godzilla  : Age;
Age_Jeanzilla : Age_Tortue;

Age_Jeanne    := 122;
Age_Godzilla  := 10_000;
Age_Jeanzilla := Age_Jeanne + Age_Godzilla; -- Compile, mais erreur de contrainte à l'exécution
```

Structure et modularité

- Pour structurer les programmes et faciliter la réutilisation de modules Ada offre plusieurs concepts
  - Paquetages
  - Sous-programmes
  - Généricité

Structure et modularité

- Paquetage
  - Unité de module contenant des déclarations de type, de variable, de sous-programmes
  - Imbricables
  - Espace de nommage

```
package <nom_paquetage> is
   ...
end <nom_paquetage>;
```

  - Les modules comme les paquetages peuvent se référencer entre eux

```
with Paquetage_1;

package Paquetage_2 is
   ...
   Paquetage_1.Sous_Programme_1
end Paquetage_2;
```

Structure et modularité

- Partie privée
  - Un paquetage peut offrir une vue incomplète à ces clients pour cacher des détails d'implémentation et ainsi diminuer le couplage

```
package <nom_paquetage> is
   ...
   type <nom_type> is private;
   ...
private
   type <nom_type> is range <domaine>;
end <nom_paquetage>;
```

- Hiérarchies de paquetage
  - Un paquetage peut avoir des enfants
  - Permet d'organiser, d'étendre, etc…

Structure et modularité

- Sous-programmes
  - Procédure ou fonction
  - Imbricables

```
procedure <nom_procédure>
   (<nom_paramètre> : in out <type_paramètre>,
    <etc...>)
is
   <déclarations>
begin
   <séquence d'instructions>
end <nom_procédure>;
```

```
function <nom_fonction>
   (<nom_paramètre> : in <type_paramètre>,
    <etc...>)
    return <type_valeur_de_retour>
is
   <déclarations>
begin
   <séquence d'instructions>
   return <résultat>;
end <nom_fonction>;
```

Structure et modularité

- Généricité
  - Permet le paramétrage d'unité (paquetage ou sous-programmes) afin de faciliter la réutilisation

```
generic
   type <nom_type> is range <>;
procedure <nom_procédure>
   (<nom_paramètre> : in <nom_type>)
is
   <déclarations>
begin
   <séquence d'instructions>
end <nom_procédure>;

procedure <nom_instance> is
   new <nom_procédure> (<nom_type> => <nom_type_effectif>);
```

Programmation Orientée Objet

- Apparue à partir de Ada95
  - Implémentation des principes de la POO par extension du concept de type: type étiqueté

```
type <nom_type> is tagged <définition>;
```

  - Les sous-programmes attachés à un types étiqueté doit être déclarés dans le même paquetage (pas directement de notion de classe regroupant les deux comme en C++)
  - Contrairement à beaucoup de langages, en Ada il est possible de faire de la POO sans pointeurs

Programmation Orientée Objet

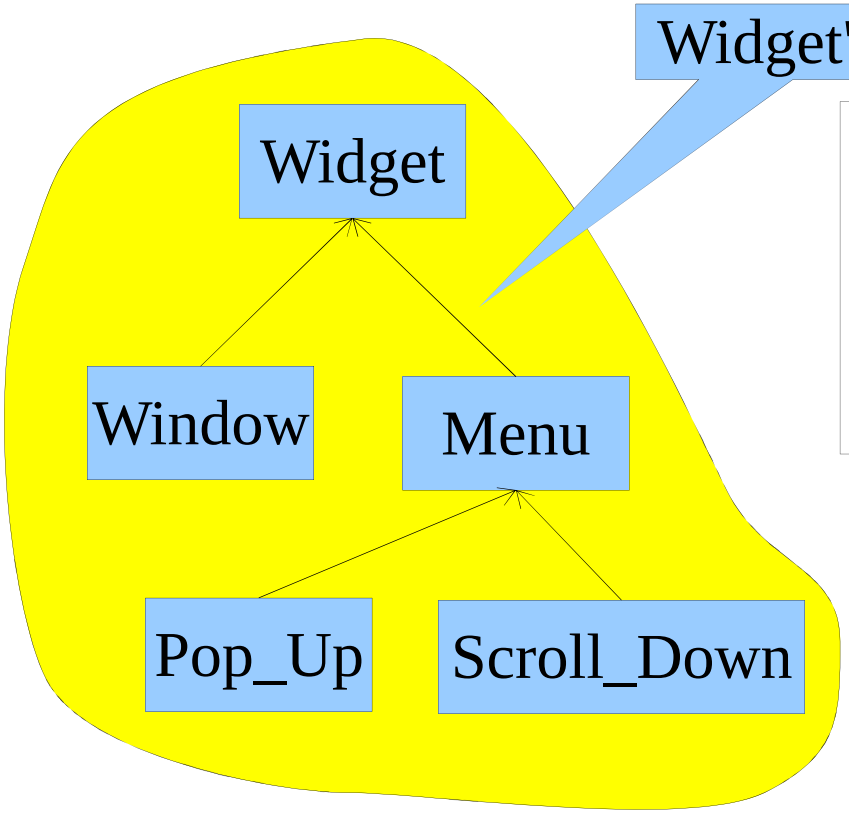
- Les paquetages permettent l'encapsulation
- Les types étiquetés permettent les liens dynamiques
- Une classe = encapsulation + lien dynamique

```
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;
```

Programmation Orientée Objet

- Différence entre un type spécifique et un type « class-wide »: aiguillage à l'exécution




```
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);
   ...
```

Parallélisme

- Le tasking est une partie intégrante du langage
  - Ce n'est pas une bibliothèque
- Les tâches (threads) sont des objets haut niveau
  - Similaire à des variables de données
- Communication et synchronisation haut niveau
  - Rendez-vous (modèle client/serveur)
  - Objets protégés (moniteurs passifs)
- Le tasking est facile à utiliser
  - N'hésitez pas à mettre des tâches dans vos programmes !

Parallélisme

- Exemple

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Task_Example
is
   task Server is
      entry Are_You_Here;
   end Server;

   task body Server
   is
   begin
      Put_Line ("Server starting");
      accept Are_You_Here;
      Put_Line ("Server going on");
   end Server;

begin
   Put_Line ("Main starting");
   Server.Are_You_Here;
   Put_Line ("Main going on");
end Task_Example;
```

Le serveur attend le client

Le client appelle le serveur

Programmation multi-langages

- Exemple d'une fonction système en C appelée en Ada

```
with Ada.Text_IO;
with Interfaces.C;

procedure Test_Gettimeofday
is
   type Timeval is record
      Tv_Sec  : Interfaces.C.Long;
      Tv_Usec : Interfaces.C.Long;
   end record
   with Convention => C;

   type Timezone is record
      Tz_Minuteswest : Interfaces.C.Int;
      Tz_Dsttime     : Interfaces.C.Int;
   end record
   with Convention => C;

   function Get_Time_Of_Day
     (Tv : in out Timeval;
      Tz : access Timezone := null)
      return Interfaces.C.Int;

   pragma Import (C, Get_Time_Of_Day, "gettimeofday");

   Time   : Timeval;
   Status : Interfaces.C.Int;
begin
   Status := Get_Time_Of_Day (Time);
   Ada.Text_IO.Put_Line (Time.Tv_Sec'Img & " " & Time.Tv_Usec'Img);
end Test_Gettimeofday;
```

```
struct timeval {
    time_t      tv_sec;  /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
```

```
struct timezone {
    int tz_minuteswest; /* minutes west of Greenwich */
    int tz_dsttime;     /* type of DST correction */
};
```

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Bibliothèques et outils

- Bibliothèques
  - Hiérarchie de paquetages Ada.*
    - ✓ Conteneurs
    - ✓ Entrées/Sorties: fichiers, chaînes de caractères
    - ✓ Gestion des dates
    - ✓ etc…
  - Hiérarchie de paquetages GNAT.*
    - ✓ Algorithmes
    - ✓ Expression régulières
  - AWS: Ada Web Server
  - GtkAda (binding avec GTK)
  - etc...

Bibliothèques et outils

- Outils
  - Compilateur GNAT
  - Environnement de développement
    - ✓ GPS
    - ✓ Support sous Emacs, etc…
  - Vérification de règles de codage
    - ✓ AdaControl
    - ✓ GNATcheck
  - Calcul de métriques de code
    - ✓ GNATmetric
  - etc...

AdaFrance

Questions ?