UNIVERSIDADE FEDERAL DE ALAGOAS
CAMPUS A.C. SIMÕES
Instituto de Computação - IC
Ciências da Computação

Ada

Lucas A. Lisboa
João Victor Ribeiro Ferro

Maceió - AL
2021

1 - Introdução

Na década de 1970, o Departamento de Defesa dos Estados Unidos (DoD) iniciou o desenvolvimento da linguagem Ada, com o objetivo de reduzir os custos com o uso excessivo de linguagens diferentes (mais de 450 eram utilizadas pelo Departamento na época) e criar uma linguagem de alto nível para sistemas embarcados (SEBESTA,2012). O projeto foi liderado por Jean Ichbiah da CII Honeywell Bull e teve seu nome dado em homenagem a Ada Lovelace, a primeira programadora.

Em 1983, houve a primeira padronização (Ada 83), feita no modelo ANSI (ANSI/MIL-STD 1815A), e, em 1987, foi adaptado para o padrão ISO (ISO-8652:1987). Após isso, foram criadas mais três versões, todas lideradas por Tucker Taft,:

1. Ada 95;
2. Ada 2005;
3. Ada 2012.

Este trabalho irá avaliar a versão mais recente da linguagem (Ada 2012).

As principais contribuições da linguagem são:

- Suporte para o uso de abstração de dados no projeto de programas;
- Alto número de recursos para tratamento de exceção;
- Possibilidade de unidades de programas genéricas;
- Execução concorrente de unidade de programas especiais (mecanismo *rendezvous*).

2 - Declaração

2.1 - Identificadores

Ada não possui limite de tamanho para nome de identificador, mas algumas implementações permitem isto (sendo o limite estipulado maior/igual a 200 caracteres).

Pelo padrão Unicode, um identificador deve iniciar com qualquer caracter das seguintes categorias (ADA-EUROPE, 2012):

- *Uppercase Letter (Maiúsculas);*
- *Lowercase Letter (Minúsculas);*
- *Titlecase Letter;*
- *Modifier Letter (Caracteres Especiais);*
- *Letter Number;*
- *Other Letter.*

Também pelo padrão Unicode, um identificador pode conter qualquer caractere das seguintes categorias (além das categorias já citadas):

- *Connector Punctuation;*

- *Nonspacing Mark;*
- *Spacing Mark;*
- *Decimal Number (Numeral Decimal).*

OBS$_1$: Um identificador não pode conter dois caracteres seguidos da categoria *Connector Punctuation*, nem terminar com algum caractere desta categoria.

OBS$_2$: Ada não é sensível à capitalização, ou seja, *Count* e *count* são a mesma variável.

2.2 - Variáveis

A linguagem permite a criação de novo tipo de dado, tipo derivado de outro e subtipo. A declaração de uma variável é feita de maneira *implícita* ou *explícita.* Esta última é feita da seguinte forma (ADA-EUROPE, 2012):

identificador : tipo

3 - Construtores e Destrutores

Em Ada, não existe uma sintaxe única para construtores e destruidores. Há duas maneiras mais comuns para definir um **construtor** em Ada (ADACORE, 2021):

1. Atribuir uma função a um atributo.
   Exemplo:

```
type T is tagged record
F : Integer := Compute_Default_F;
end record;

function Compute_Default_F return Integer is
begin
   Put_Line ("Compute");
   return 0;
end Compute_Default_F;

V1 : T;
V2 : T := (F => 0);
```

   Na declaração de V1, é utilizado a inicialização padrão para F definida pela função *Compute_Default_F.* Já no caso da inicialização de V2, é atribuído 0 a F manualmente, sem executar a função padrão.

2. Utilizar o tipo *Ada.Finalization.Controlled* e sobrescrevendo *Inicialize*

Exemplo:

```
type T is new Ada.Finalization.Controlled with record
   F : Integer;
end record;

procedure Initialize (Self : in out T) is
begin
   Put_Line ("Compute");
   Self.F := 0;
end Initialize;

V1 : T;
V2 : T := (F => 0);
```

Na declaração de V1, é utilizado a inicialização padrão para F definida pela função *Inicialize* sobrescrita. Já no caso da inicialização de V2, é atribuído 0 a F manualmente, sem executar a função padrão.

Utilizando o tipo *Ada.Finalization.Controlled*, também é possível sobrescrever a função *Finalize*, a qual funciona de maneira análoga a um **destrutor**.

Exemplo:

```
with Ada.Finalization;

package P is
     type T is new Ada.Finalization.Controlled with
private;
function Make return T;

private
     type T is ... end record;
     overriding procedure Finalize (This: in out T);
end P;
```

4 - Atributos

Um atributo é uma característica de uma entidade que pode ser requisitado através de uma referência do atributo ou referência de alcance. A referência de um atributo inicia-se com um prefixo, seguido do símbolo ‘ e pelo seu nome. O compilador determina pelo contexto se o símbolo ‘ define o início de um atributo, caractere literal ou uma expressão quantificada (WIKIBOOKS, 2021).

Exemplo:

```
A : Character := Character'Val (32) -- A is now a space
```

5 - Ocultação de Informação

Diferente de outras linguagens, a ocultação de informação em Ada se dá limitando a visibilidade da entidade para fora do pacote (*package*), enquanto, em outras linguagens, como Java, o *private* já torna a entidade não visível fora da classe. Dessa forma, em Ada, ao atribuir a uma entidade a cláusula *private,* ela é tornada parcialmente visível fora do pacote, sendo fornecido para acesso apenas elementos básicos da interface, como o nome da entidade, não sendo possível visualizar o conteúdo.

É possível atribuir a cláusula *limited* antes de *private*, sendo sua inclusão opcional. Seu uso evita que comparações e atribuições sejam feitas de maneira direta, isto é, não é possível utilizar os operadores :=, =, /=, >, >=, < e <= para entidades *limited private* (ADACORE, 2012).

Exemplos:

```
type Tipo_Criado is private;
type Tipo_Criado2 is limited private;
```

6 - Tipo Abstrato e Interface

Em Ada, é necessário explicitar que o método, classe ou tipo *tagged* é abstrato com o uso da cláusula *abstract*. Todos os métodos abstratos devem ser implementados quando um tipo concreto se basear num tipo abstrato (ADACORE, 2021).

Exemplo:

```
package P is

     type T is abstract tagged private;

     procedure Method (Self : T) is abstract;
 private
     type T is abstract tagged record
     F1, F2 : Integer;
     end record;

end P;
```

Ada permite que um tipo tagged implemente múltiplas interfaces.

Exemplo:

```
type Root is tagged record
   F1 : Integer;
end record;
procedure M1 (Self : Root);
```

```
type I1 is interface;
procedure M2 (Self : I1) is abstract;

type I2 is interface;
procedure M3 (Self : I2) is abstract;

type Child is new Root and I1 and I2 with record
   F2 : Integer;
end record;

-- M1 implicitly inherited by Child
procedure M2 (Self : Child);
procedure M3 (Self : Child);
```

7 - Herança

Em Ada, os tipos derivados fornecem herança. As classes derivadas podem estender somente as classes pai e subtipos. Ada não oferece herança múltipla (ADACORE, 2021).

Exemplo:

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

procedure Primitives is
  package Week is
     type Days is (Monday, Tuesday, Wednesday, Thursday,
                   Friday, Saturday, Sunday);

     --  Print day is a primitive of the type Days
     procedure Print_Day (D : Days);
  end Week;

  package body Week is
     procedure Print_Day (D : Days) is
     begin
     Put_Line (Days'Image (D));
     end Print_Day;
  end Week;

  use Week;
  type Weekend_Days is new Days range Saturday .. Sunday;
```

```
   --   A procedure Print_Day is automatically inherited
here. It is as if
  --  the procedure
  --
  --  procedure Print_Day (D : Weekend_Days);
  --
  --  has been declared with the same body

  Sat : Weekend_Days := Saturday;
begin
   Print_Day (Sat);
end Primitives;
```

8 - Polimorfismo

Em Ada, o polimorfismo mediante despacho dinâmico (*dynamic dispatching*) e ocorre para *class-wide type* , isto é, um descendente de um tipo *tagged*. Dessa forma, a entidade pai terá seu método sobrescrito pela entidade filho (WIKIBOOKS, 2021).

Exemplo:

```
declare
   Someone : Person.Object'Class := ...; -- to be
expanded later
begin
   Someone.Put; -- dynamic dispatching
end;
```

No caso acima, o método Put especificado em Someone, o qual sobrescreveu o *Put* de *Person*.

9 - Encapsulamento

Ada tem a capacidade de reunir coleções de subprogramas, de tipos e de dados em unidades que podem ser compiladas separadamente, ou seja, que suas informações de interface são salvas pelo compilador e usadas para verificação de interface quando usada por outra unidade. Além disso, tem o mecanismo de controle de acesso para as entidades nessas unidades. A construção de encapsulamento é chamada de pacotes, os mesmos são divididos em duas partes:

- Pacote de especificação (interface)
- Pacote de corpo (implementação da maioria das entidades nomeadas no pacote).

A especificação e o corpo associado compartilham o mesmo nome. Utiliza-se a palavra reservada *“body”* para identificar o início do pacote de corpo. A

compilação do pacote de corpo e especificação podem ser feitas separadamente. Outra coisa, o código do cliente pode ser compilado antes do pacote de corpo ser compilado ou escrito. O pacote de especificação tem duas partes: pública e privada, o nome do tipo abstrato aparece na parte pública e a representação de um tipo abstrato aparece na parte privada.

Ada detém um designer que define um tipo de dados, pode optar por fazer o tipo inteiramente visível para os clientes ou fornecer apenas as informações de interface. Existem duas formas de esconder a representação de clientes na especificação do pacote:

- A primeira é incluir duas secções do pacote especificação, na qual as entidades são visíveis para os clientes e que escondem o seu conteúdo. Para um tipo de dados abstrato, uma declaração aparece na parte visível da especificação, proporcionando apenas o nome do tipo e do fato de a sua representação está escondida. A representação do tipo aparece numa parte da especificação chamada privada, que é introduzida pela palavra reservada *"private"* , essa cláusula vem no final da especificação do pacote. A cláusula particular é visível para o compilador, mas não para as unidades de programa do cliente.
- A segunda é esconder a representação e definir os dados abstratos tipo um ponteiro e proporcionar uma definição da estrutura apontada no corpo do pacote, cujo conteúdo integral estão escondidos dos clientes.

Além disso, o pacote Ada pode incluir qualquer número de declarações de dados e subprogramas nas suas sessões públicas e privadas, na qual podem incluir as interfaces para qualquer número de tipos de dados abstratos, assim como qualquer recurso do programa. Assim um pacote é um construtor do encapsulamento de tipo múltiplo.

10 - Exemplo de Programa

Fibonacci (PROGOPEDIA, 2021):

```
with Ada.Text_IO, Ada.Integer_.Text_IO;

procedure Fibonacci is
begin
     declare
          function Fib (N: Integer) return Integer is
          begin
          if N < 3 then
               return 1;
          else
               return Fib(N-1) + Fib(N-2);
          end if
```

```
          end if;
     I: Integer := 1;
     begin
          loop
               Ada.Integer_Text_IO_Put  (Item  =>  Fib(i),  Width
=> 1);
               Ada.Text_IO.Put (“, “);
               i : i +1;
               exit when i = 17;
          end loop
          ada.Text_IO.Put (“...”);
     end;
end Fibonacci;
```

Referências

ADA-EUROPE. **Ada 2012 Reference Manual**: 2012 Edition. 2012. Disponível em: http://www.ada-auth.org/standards/12rm/html/RM-TTL.html. Acesso em: 19 maio 2021.

ADACORE. **LEARN.ADACORE.COM**. 2021. Disponível em: https://learn.adacore.com/index.html. Acesso em: 19 maio 2021.

PROGOPEDIA**. Fibonacci numbers in Ada.** Disponível em: http://progopedia.com/example/fibonacci/128/ Acesso em: 20 maio 2021

SEBESTA, Robert W. . **Concepts of Programming Languages**. Artima Inc, 10th edition, 2012.

WIKIBOOKS**. Ada Programming / Object Orientation.** Disponível em: https://en.wikibooks.org/wiki/Ada_Programming/Object_Orientation. Acesso em: 20 maio 2021