Grado en Ingeniería en Computadores

Sistemas en Tiempo Real

L2. - Programación a pequeña escala

Departamento de Automática
Universidad de Alcalá

2 de febrero del 2022

Índice

Tipos

Estructuras de control

Atributos

Subprogramas

Excepciones

Tipos

Estructuras de control

Atributos

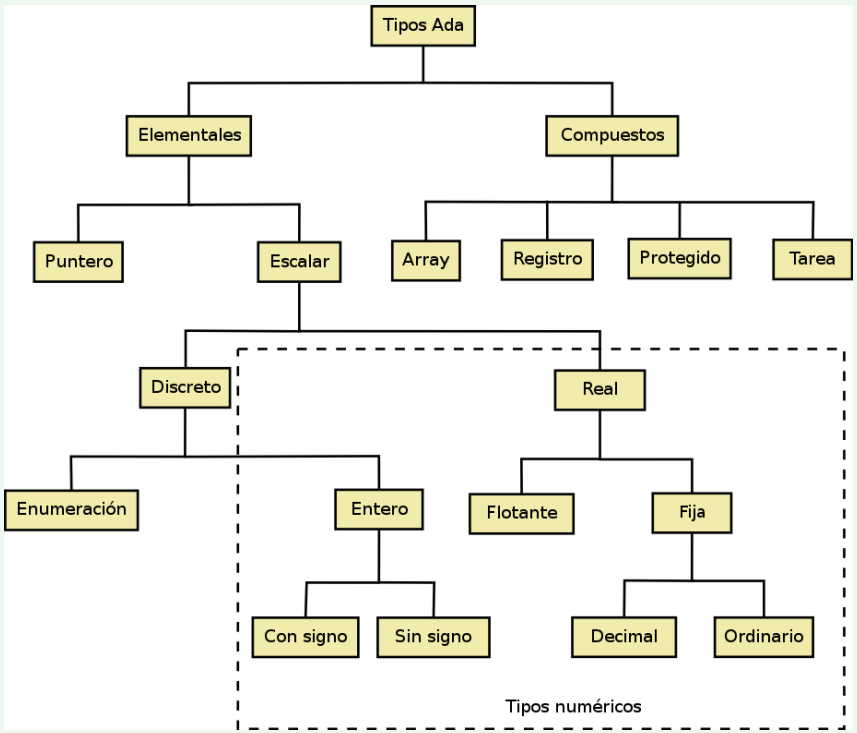
Subprogramas

Excepciones

Tipos como Objetos

En Ada los tipos se caracterizan por:

- Un conjunto de valores válidos (Fuertemente tipado)
- Un conjunto de operaciones que se pueden aplicar al tipo



Declaración de variables básicas

En Ada las variables se **declaran** como "nombre_de_variable : tipo":

```
x : Integer;                -- una variable de tipo integer llamada x
a, b, c : Integer;          -- Tres variables de tipo integer
d, e, f : Integer := 0;     -- Podemos inicializar las variables
M: Float:=5.5;              -- Declaramos una variable de tipo Float
```

Tipos derivados y subtipos

- Un **tipo derivado** es construido a partir de otro existente, por lo que hereda sus características, pero tiene un nombre distinto y puede tener un rango de valores distinto
- Un **subtipo** permite asignarle un nombre y rango nuevos a un tipo ya existente, pero **sin crear un tipo nuevo**

Ejemplo : Tipos derivados y subtipos

```
-- Tipo derivado de Float
type masa1_t is new Float range 0.0..10_000.0;
-- Subtipo (el tipo es Float)
Subtype masa2_t is Float range 0.0..12_000.0;

X: Float:=5_000; -- Variable de tipo Float
Y: masa1_t;      -- Variable de tipo masa1_t
Z: masa2_t;      -- Variable de tipo masa2_t (de tipo Float)

Y:=X;                -- ??????????
Z:=X;                -- ??????????
Z:=Y;                -- ??????????
Z:=20_000.0;         -- ??????????
```

Ejemplo : Tipos derivados y subtipos

```
-- Tipo derivado de Float
type masa1_t is new Float range 0.0..10_000.0;
-- Subtipo (el tipo es Float)
Subtype masa2_t is Float range 0.0..12_000.0;

X: Float:=5_000; -- Variable de tipo Float
Y: masa1_t;      -- Variable de tipo masa1_t
Z: masa2_t;      -- Variable de tipo masa2_t (de tipo Float)

Y:=X;                -- Ilegal X e Y no son del mismo tipo
Z:=X;                -- Legal
Z:=Y;                -- Ilegal son tipos distintos
Z:=20_000.0;         -- Ilegal, fuera de rango
```

Ada permite conversión de tipos **explícita**

```
I: Integer := 20
X: Float;
X:=I                -- Ilegal
X:=Float(I);        -- Legal
```

Tipos derivados y subtipos ¿Por qué?

- A primera vista puede parecer redundante o exagerado
- La razón es incrementar la seguridad y fiabilidad del código y de los sistemas que comanda

El incidente de la Mars Climate Orbiter

- La Mars Climate Orbiter (MCO) fue una sonda de la NASA lanzada desde Cabo Cañaveral el 11 de diciembre de 1998 por un cohete Delta II 7425 y llegó a Marte el 23 de septiembre de 1999, después de un viaje de 9 meses y medio
- La Mars Climate Orbiter se destruyó debido a un error de navegación, consistente en que el equipo de control en la Tierra hacía uso del Sistema Anglosajón de Unidades para calcular los parámetros de inserción y envió los datos a la nave, que realizaba los cálculos con el sistema métrico decimal

El incidente de la Mars Climate Orbiter [1]

- Si los datos hubieran sido fuertemente tipados existirían los tipos Empuje_Nm2 y Empuje_Libras y al ser distintos no se podrían asignar entre ellos

Remember the Mars Climate Orbiter incident from 1999?

[1] Más en: `https://www.youtube.com/watch?reload=9&v=urcQAKKAAlO`

Ejemplo : Tipos enteros (Integer, Natural, Positive)

Podemos crear un tipo derivado limitando el rango (**range**):

```
-- Tipo dias_t definido por el usuario
-- Es un Positivo con rango 1 a 31
type dias_t is new Positive range 1..31;
```

Podemos omitir el tipo base si el rango implícitamente lo fija:

```
-- Tipo dias_t definido por el usuario
-- Es un Positivo con rango 1 a 31
type dias_t is range 1..31;
```

Ejercicio 1.1: Puntuación

Haz un programa de nombre puntuación que pida por teclado el número de canastas de 3, de 2 y personales y calcule los puntos totales del jugador, mostrándolos por pantalla

- Elige de forma adecuada los tipos para las variables
- ¿Qué ocurre si introducimos un valor negativo por teclado? ¿Cómo se puede solucionar?

Ejemplo : Tipos reales (Coma fija y flotante)

Podemos fijar el número de cifras significativas (**digits**):

```
-- Digits 10 dice que tendra hasta 10 cifras
-- (a ambos lados de la coma)
-- Range fija el intervalo a [-1.0, 1.0]
-- Al combinarlos hace que sean 9 decimales
-- y una cifra antes de la coma
type real_t is new Float digits 10 range -1.0..1.0;
```

Podemos fijar los saltos entre valores consecutivos (**delta**)

```
-- Tipo en coma fija
-- Delta (Diferencia entre valores consecutivos) de 0.125
-- Range en el intervalo [0.0, 12.0]
type masa_t is delta 0.125 range 0.0..12.0;
```

Ejercicio 1.2: Área de un rectángulo

Haz un programa de nombre área que pida por teclado el alto y el ancho de un rectángulo y calcule su área, mostrándolo por pantalla

- Elige de forma adecuada los tipos para las variables
- ¿Qué ocurre si introducimos un tipo no válido por teclado? ¿Cómo se puede solucionar?

Tipos enumerados

Permiten definir un conjunto de valores

- Tipos enumerados predefinidos

```
Boolean    -- Tipo predefinido. Valores TRUE y FALSE
Character  -- Tipo predefinido. Valores: caracteres ASCII
```

- Tipos enumerados definidos por el usuario

```
-- meses_t es de tipo Enumeration
type meses_t is (enero, febrero, marzo, abril, mayo, junio);
```

Ejercicio 1.3: Fecha de nacimiento

Haz un programa de nombre nacimiento que pida por teclado la fecha de nacimiento y la muestre por pantalla

- Declara de forma adecuada los tipos para el día, mes y año
- ¿Qué ocurre si introducimos un tipo no válido por teclado? ¿Cómo se puede solucionar?

Tipo array

Es una colección de elementos de un mismo tipo base

- Ejemplos de declaración de tipos array

```
type vector_t is array (1..10) of Float;
type array_vector_t is array (1..5) of vector_t;
type matriz_t is array (1..10, 1..10) of Float;

V: vector_t;       -- Declaramos una variable tipo vector_t

V2: array(1..10) of Float; -- Esto se puede hacer
     -- pero no es recomendado ya que crea un tipo sin
     -- nombrarlo
```

- Podemos acceder a sus elementos de 3 maneras

```
V(1) := 10.23;          -- Acceso por indice
V := (1.0, 23.7, 2.4,...,2.4); -- Notacion posicional
V := (1=>1.0, 2=>23.7, others => 2,4); -- Notacion nombrada
```

Tipo array

Es una colección de elementos de un mismo tipo base

- Ejemplos de declaración de tipos array

```
type vector_t is array (1..10) of Float;
type array_vector_t is array (1..5) of vector_t;
type matriz_t is array (1..10, 1..10) of Float;
```

- Ejemplos de manejo de variables de tipo array

```
V: vector_t;        -- Declaramos una variable tipo vector_t

AV: array_vector_t;
AV(1) := (others=>0.0);
AV(2) := V;

M: matriz_t;
M(2,3) := 3,7;
M(2) := V;          -- Error
M(2)(3) := 3,7;     -- Error

AV := (others=> (others=>0.0)); -- Inicializacion a cero
```

Ejercicio 1.4: Manejo de arrays

Haz un programa que cree un array de enteros con 5 elementos, inicializado a 3, 23, 4, 67 y 332

- Mostrar por pantalla la suma de sus elementos
- Mostrar por pantalla la suma de los elementos impares

Tipo array **no restringido**

Los tipos "array no restringido" son aquellos en los que el rango de los índices **no se establece al definir el tipo**, sino que se concreta posteriormente.

- Ejemplos de declaración de tipos array no restringido

```
-- Declaracion del tipo array no restringido
-- Hay que declarar el tipo del indice
type array_t is array (integer range <>) of float;
```

- Posteriormente, **en la parte declarativa del código** podemos declarar variables de ese tipo con el rango de los índices a nuestra elección

```
L: array_t(1..10);
L2: array_t(2..30);
```

Tipo registro

Los tipos registro son colecciones de elementos del mismo o diferentes tipos

- Ejemplos de declaración de un tipo registro

```
-- Definimos un par de tipos para usarlos en el registro
type dias_t is range 1..31
type meses_t is (enero, febrero, marzo, abril, mayo, junio);

-- Definimos un tipo registro con los tipos anteriores
type fecha_t is record
    Dia : dias_t;
    Mes : meses_t;
    Year : Natural;
end record;
```

Tipo registro (cont)

Los tipos registro son colecciones de elementos del mismo o diferentes tipos

- Para acceder a los elementos de un registro se utiliza el operador "."

```
Hoy : fecha_t;

Hoy.Dia := 31; Hoy.mes := enero; Hoy.Year := 2018;
Hoy :=(Dia =>31, Mes => enero, Year=> 2018);
```

Ejercicio 1.5: Registros

Repite los ejercicios 1.1 y 1.3 utilizando un tipo registro

Tipo puntero [2]

Los punteros en Ada funcionan de una forma similar a otros lenguajes de programación y se denominan “accesos”

- Los punteros se implementan con el tipo **access** en Ada
- **Automáticamente** son inicializados con el valor null

```
-- Definicion del tipo access
type pInteger_t is access integer; -- Tipo puntero a integer

-- Declaracion de variables del tipo
Pint1, Pint2 : pInteger_t;          -- Punteros a integer

-- Asignacion dinamica
Pint1 := new integer;    -- Reservamos memoria para un integer
```

[2] http://www.gedlc.ulpgc.es/docencia/mp_i/GuiaAda/ada13.html

Tipo puntero (cont)

```
-- Asignacion dinamica e inicializacion
Pint2 := new integer'(35);  -- Reservamos memoria para un integer
                            -- y le asignamos el valor 35
i:integer;
i := Pint2.all; -- Se asigna a i el valor de la variable
                -- referenciada por Pint2
```

Tipo puntero (array dinámico)

Asignación dinámica de memoria

```
-- Creamos un tipo array no restringido
type array_t is array (integer range <>)  of integer;
-- Creamos un tipo puntero al tipo anterior
type pArray_t is access array_t;

-- Declaramos una variable de tipo puntero a array_t
v : pArray_t;

-- Dinamicamente reservamos memoria del tamano deseado
v := new array_t(1..5); --Reservamos memoria para un array de 5 elem
```

Tipos

Estructuras de control

Atributos

Subprogramas

Excepciones

If then else

```
if A >=B then
    Mayor:=A;
else
    Mayor:=B;
end if;
```

Shortcuts

EXP1 **and then** EXP2 → Evalua EXP2 sólo si EXP1=True
EXP1 **or else** EXP2 → Evalua EXP2 sólo si EXP1=False

```
if A/=0.0 and then B/A > 1.0 then
    Sentencia;
end if;
```

Selección múltiple

```
if Expr1 then
    Instr1;
elsif Expr2 then
    Intr2;
else
    Instr3;
end if;
```

Switch case

```
case Mes is
    when enero | mayo => Instr1;
    when junio..septiembre => Instr2;
    when others=> null;
end case;
```

Bucle simple

```
loop
    Instr1;
    exit when condicion;
end loop;
```

Bucle for

```
-- Podemos recorrer un rango de forma ascendente
for I in 1..10 loop
    Leer_Numero(A(I));
end loop;
-- Podemos recorrer un rango de forma descendente
for I in reverse 1..10 loop
    Leer_Numero(A(I));
end loop;
```

Bucle while

```
I:=1;
while I<= 10 loop
    Leer_Numero(A(I));
    I:=I+1;
end loop;
```

Tipos

Estructuras de control

Atributos

Subprogramas

Excepciones

Atributos de los tipos y los objetos[3]

Los atributos son operaciones predefinidas que se pueden aplicar a tipos, objetos y otras entidades. Tienen la siguiente sintaxis:

- Entidad'Atributo
- Tipo'Atributo(Entidad)

Atributos aplicables a **tipos**

- **Image:** Se usa con la forma X'Image(Y), siendo X cualquier tipo discreto e Y una variable de ese tipo. Devuelve una cadena (String) con la representación del valor de la variable Y.
- **First:** Indica el mínimo valor que puede tomar una variable de un tipo discreto, sea entero o enumeración. Se usa con la forma T'First, siendo T un tipo discreto (entero o enumeración).
- **Last:** Indica el máximo valor que puede tomar una variable de un tipo discreto, sea entero o enumeración.
- **Pos:** Indica la posición ocupada por un determinado valor en un tipo enumeración. Por ejemplo: TColor'Pos(ROJO). La primera posición se considera 0.

[3] `https://es.wikibooks.org/wiki/Programación_en_Ada/Atributos`

Atributos aplicables a **tipos** (cont.)

- **Pred:** TDía'Pred(VIERNES) indica el anterior valor a VIERNES que toma el tipo TDía, si no existe, se eleva la excepción Constraint_Error.
- **Succ:** TColor'Succ(ROJO) indica el siguiente valor a ROJO que toma el tipo TColor, si no existe, se eleva la excepción Constraint_Error.
- **Digits:** Representa el número de decimales de la mantisa de un tipo flotante.
- **Ceiling:** Se usa con la forma X'Ceiling(Y), siendo X cualquier tipo flotante e Y una variable de ese tipo. Devuelve el "techo" de Y, es decir el valor entero más pequeño que es mayor o igual que Y.
- **Rounding:** Se usa con la forma X'Rounding(Y), siendo X cualquier tipo flotante e Y una variable de ese tipo. Devuelve el valor de Y redondeado al entero más cercano. Si Y está exactamente entre dos valores enteros, toma el valor del mayor entero (p.e, Float'rounding(1.5)=2.0).
- **Truncation:** Se usa con la forma X'Truncation(Y), siendo X cualquier tipo flotante e Y una variable de ese tipo. Devuelve el valor truncado de Y a un valor entero.
- **Val:** Indica el valor que ocupa una determinada posición en un tipo enumeración. Por ejemplo: COLOR'Val(1).
- **Size:** Indica el mínimo espacio en bits en que se pueden almacenar objetos de este tipo. Técnicamente se define como lo que ocuparía un componente de un registro de este tipo cuando el registro está empaquetado (pragma Pack).

Atributos aplicables a **objetos**

- **Valid:** si tiene una representación válida para su tipo. Útil cuando se obtienen valores desde el «mundo exterior» mediante Unchecked_Conversion u otro mecanismo.
- **First, Last:** aplicados a arrays dan el primer y el último índices del array.
- **Range:** Vector'Range indica el rango que ocupa la variable Vector, es decir, equivale a Vector'First..Vector'Last. En el caso de más de una dimensión, el valor Matriz'Range(1) indica el rango de la primera dimensión.

✎ Ejemplo : Comprobar valores leídos en un socket

Nuestro programa puede leer valores de control por un socket

```
Resultado : Natural;
...
-- Leemos un valor por un socket y lo ponemos en Natural
...
if not Resultado'Valid then
    -- 'Resultado' esta fuera del rango, con valor negativo
    Result := Natural'First;
end if
```

Ejemplo de atributos sobre **tipos**

```
Integer'First   -- Primer valor entero
Integer'Last    -- Ultimo valor entero
Float'Digits    -- Numero digitos significativos de un Float
```

Ejemplo de atributos sobre **tipos**

```
M : array (1..2, 0..5) of integer;
M'First(1);   -- 1
M'First;      -- 1
M'First(2);   -- 0
M'Last(1);    -- 2
M'Last(2);    -- 5
M'Length(1)   -- 2
M'Length(2); -- 6
M'Range(1);   -- 1..2
M'Range(2);   -- 0..5
```

Ejercicio 1.6: Atributos

Repite el ejercicio 1.4 utilizando estructuras de control y atributos ¿qué ventajas tiene hacer el ejercicio usando atributos?

Tipos

Estructuras de control

Atributos

Subprogramas

Excepciones

Definición

Son las unidades básicas de ejecución en Ada y definen código reentrante (que puede ser interrumpido y retomado). Son las funciones y los procedimientos

1.- Procedimientos

- No retornan ningún valor
- Sus parámetros pueden ser **in**, **out** o **in out**
- Por omisión son **in**

Ejemplo: Procedimientos

```
with Ada.Numerics.Elementary_Functions;
use Ada.Numerics.Elementary_Functions;
-- Interfaz
procedure resolver(A,B,C:Float; R1,R2:out Float; ok:out Boolean) is
Z: Float;          -- variable local
begin              -- Cuerpo
    Z:= B*B - 4.0*A*C;
    if Z<0.0 or A = 0.0 then
        Ok:=false;
    else
        Ok := true;
        R1:= (-B+sqrt(z))/(2.0*A);
        R2:= (-B-sqrt(z))/(2.0*A);
    end if;
end resolver;
```

Cuestiones

Las variables A,B y C ¿son in/out o in out? ¿para qué crees que se añade el paquete Numerics.Elementary_Functions?

2.- Funciones

- Devuelven un resultado
- Sus parámetros sólo pueden ser **in**

Ejemplo: Funciones

```
function minimo (X,Y:Float) return Float is
begin
    if X>Y then
        return Y;
    else
        return X;
    end if;
end minimo;
```

Ejercicio 1.7

Crear un programa de nombre raiz_minima que calcule las raíces de un polinomio de segundo grado pasado por teclado y muestre por pantalla la menor de las dos

- Los coeficientes del polinomio serán de tipo Float
- Por comodidad, cuando pruebes tu programa hazlo con coeficientes enteros para que sepas el resultado de las raíces facilmente

Tipos

Estructuras de control

Atributos

Subprogramas

Excepciones

Excepciones [4]
Programación a pequeña escala

Definición

- En Ada, cuando se produce algún error durante la ejecución de un programa, se eleva una excepción.
- Dicha excepción puede provocar la terminación abrupta del programa, pero se puede controlar y realizar las acciones pertinentes.

Manejo de las excepciones

- Cuando se espere que pueda presentarse alguna excepción en parte del código del programa, se puede escribir un manejador de excepciones en las construcciones que lo permitan (bloques o cuerpos de subprogramas, paquetes o tareas), aunque siempre está el recurso de incluir un bloque en cualquier lugar del código

[4] `https://es.wikibooks.org/wiki/Programación_en_Ada/Excepciones`

Ejemplo

Dentro de cualquier bloque podemos escribir un manejador de excepciones con **exception**

```
begin
   -- ...
exception
     when Constraint_Error =>
          Put ("Error de rango.");
     when Program_Error | Tasking_Error =>
          Put ("Error de flujo.");
     when others =>
          Put ("Otro error.");
end;
```

* En el momento en el que se produzca la elevación de Constraint_Error durante la ejecución de la secuencia de sentencias entre begin y exception, el flujo de control se interrumpe y se transfiere a la secuencia de sentencias que siguen a la palabra reservada => del manejador correspondiente.

Excepciones predefinidas

En Ada, dentro del paquete Standard, existen unas excepciones predefinidas

- **Constraint_Error:** cuando se intenta violar una restricción impuesta en una declaración, tal como indexar más allá de los límites de un array o asignar a una variable un valor fuera del rango de su subtipo
- **Program_Error:** se produce cuando se intenta violar la estructura de control, como cuando una función termina sin devolver un valor
- **Storage_Error:** es elevada cuando se requiere más memoria de la disponible.
- **Tasking_Error:** cuando hay errores en la comunicación y manejo de tareas.
- **Name_Error:** se produce cuando se intenta abrir un fichero que no existe.

Ejemplo

En la función Tomorrow si se produce la excepción Constraint_Error la función devolverá el primer día

```
function Tomorrow (Hoy: TDia) return TDia is
begin
    return TDia'Succ(Hoy);
exception
    when Constraint_Error =>
        return TDia'First;
end Tomorrow;
```

Control y propagación de las excepciones

- Nótese que **no se puede devolver nunca el control a la unidad donde se elevó la excepción.** Cuando se termina la secuencia de sentencias del manejador, termina también la ejecución de dicha unidad.
- Si no se controla una excepción, ésta se **propaga dinámicamente por las sucesivas unidades invocantes** hasta que se maneje en otra o directamente termina la ejecución del programa proporcionando un mensaje con la excepción provocada por pantalla.

Ejercicio 1.8

Repite el ejercicio 1.1 y captura la excepción que se produce cuando introduces un valor no válido. ¿Qué ocurre cuando capturas la excepción? ¿crees que es el comportamiento deseado?

Ejercicio 1.9

Repite el ejercicio 1.8 de forma que puedas volver a introducir el valor erróneo

Ejemplo control y propagación de excepciones

Programar un procedimiento principal que utilice la función Tomorrow del ejemplo anterior y que también capture la excepción Constraint_Error. Prueba a capturar la excepción dentro de la función, a ignorarla dentro de la función y a elevarla dentro de la función. Comenta los resultados.

```
type TDia is (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado,
    Domingo);
```

Ejemplo control y propagación de excepciones

Hay una restricción que hay que tener en cuenta con respecto a los manejadores. **Excepciones elevadas en la parte declarativa no son manejadas por los manejadores de ese bloque.**[5]

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

procedure Be_Careful is
    function Dangerous return Integer is
    begin
        raise Constraint_Error;
        return 42;
    end Dangerous;

    -- A : Integer := Dangerous; -- La excepcion se eleva aqui
    A : Integer;
begin
    -- declare
    -- A : Integer := Dangerous; -- La excepcion se eleva aqui
    begin
        A := Dangerous;              -- La excepcion se eleva aqui
        Put_Line(Integer'Image(A));
    exception
        when Constraint_Error=>Put_Line("error !");
    end;
end Be_Careful;
```

[5] https://learn.adacore.com/courses/intro-to-ada/chapters/exceptions.html

Excepciones definidas por el usuario

- Es normal encontrar situaciones en las que el error no se encuentra entre las excepciones predefinidas
- Se pueden declarar excepciones propias utilizando el tipo **exception**

```
nombre_error: exception;
```

- Para lanzar nuestra excepción usamos **raise**

```
ErrorPilaEnteros: exception;      -- Declaramos la excepcion

function Quitar return Integer is
begin
    if Cima = 0 then
        raise ErrorPilaEnteros;  -- Se eleva la excepcion.
   end if;
        Cima := Cima - 1;
   return Pila(Cima+1);
 end Quitar;
```

Excepciones definidas por el usuario (cont)

- Cuando recogemos una excepción podemos elegir manejarla, no hacer nada o continuar propagándola con un nuevo raise de la misma excepción o de otra que queramos
- Ada proporciona información sobre una determinada excepción haciendo uso del paquete predefinido **Ada.Exceptions** y tras obtener la ocurrencia de la excepción mediante esta notación:

```
nombre_error: exception;
```

- Para lanzar nuestra excepción usamos **raise**

```
-- Ocurrencia es un nombre arbitrario que le damos
-- para acceder a la excepcion
when Ocurrencia : ErrorPilaEnteros =>
   Put_Line (Ada.Exceptions.Exception_Information (Ocurrencia));
```

Ejercicio 1.10

Modifica el ejercicio 1.9 para mostrar la información de la excepción

Grado en Ingeniería en Computadores

Sistemas en Tiempo Real

L2. - Programación a pequeña escala

2 de febrero del 2022