

# Tema 1

## Introducción al lenguaje de programación Ada



# Objetivos

1. Comenzar a familiarizarse con la **programación** en lenguaje Ada
  - ▣ Conocer la estructura general de un programa
  - ▣ Conocer su sintaxis básica
2. Manejar el IDE GnatStudio para la construcción de proyectos Ada, compilación y ejecución de programas
3. Introducción a la concurrencia en Ada: Tareas



# Índice

1. **Origen y características generales**
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Historia de Ada

- Iniciativa del DoD (Departamento de Defensa de los EEUU)
- Lenguaje para el desarrollo de sistemas de tiempo real, empotrados, de control y de misión crítica.
- 4 versiones normalizadas
  - Ada 83 (ISO 8652:1987)
  - Ada 95 (ISO 8652:1995)
  - Ada 2005 (ISO 8652:1995/Adm 1:2007)
  - Ada 2012 (ISO 8652:2012)



[https://es.wikipedia.org/wiki/Ada\\_\(lenguaje\\_de\\_programación\)](https://es.wikipedia.org/wiki/Ada_(lenguaje_de_programación))

# Características de Ada

- Diseñado específicamente para sistemas de tiempo real
  - Procesamiento concurrente
  - Bibliotecas y sentencias para el manejo del tiempo
  - Fiabilidad y seguridad (por ejemplo, manejo de excepciones)
  - Acceso al hardware e interrupciones
- Sintaxis descendiente de Pascal
  - Estructura en bloques
  - Fuertemente tipado
  - Legibilidad
- Pensado para construir sistemas grandes y cambiantes
  - Paquetes y unidades genéricas
  - Biblioteca jerárquica
  - Interfaces normalizadas con otros lenguajes (C, Fortran)



## 1. Origen y características generales

# Ada en la Industria



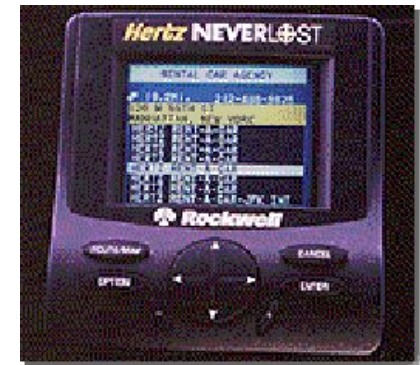
**Boeing 777**



**TGV. Tren de alta velocidad**



**Metro de París, Londres y Nueva York**



**GPS**



**Resonancia Magnética Nuclear**



**Intelsat**

<https://www.sigada.org/education/pages/success.html>



# Índice

1. Origen y características generales
2. **IDE: GnatStudio**
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Software utilizado en la asignatura

- Máquina virtual con VirtualBox
  - Ubuntu 20.04
  - IDE GnatStudio (compilador GNAT 2020)
  - GtkAda 3.24.20

Descargar desde el siguiente enlace:

[https://drive.google.com/file/d/14u\\_RY2ez4BleHD2b2E5ClfOECtHq4gv5/view?usp=sharing](https://drive.google.com/file/d/14u_RY2ez4BleHD2b2E5ClfOECtHq4gv5/view?usp=sharing)

.zip (8,21 Gb)

.vdi (20,46 Gb)

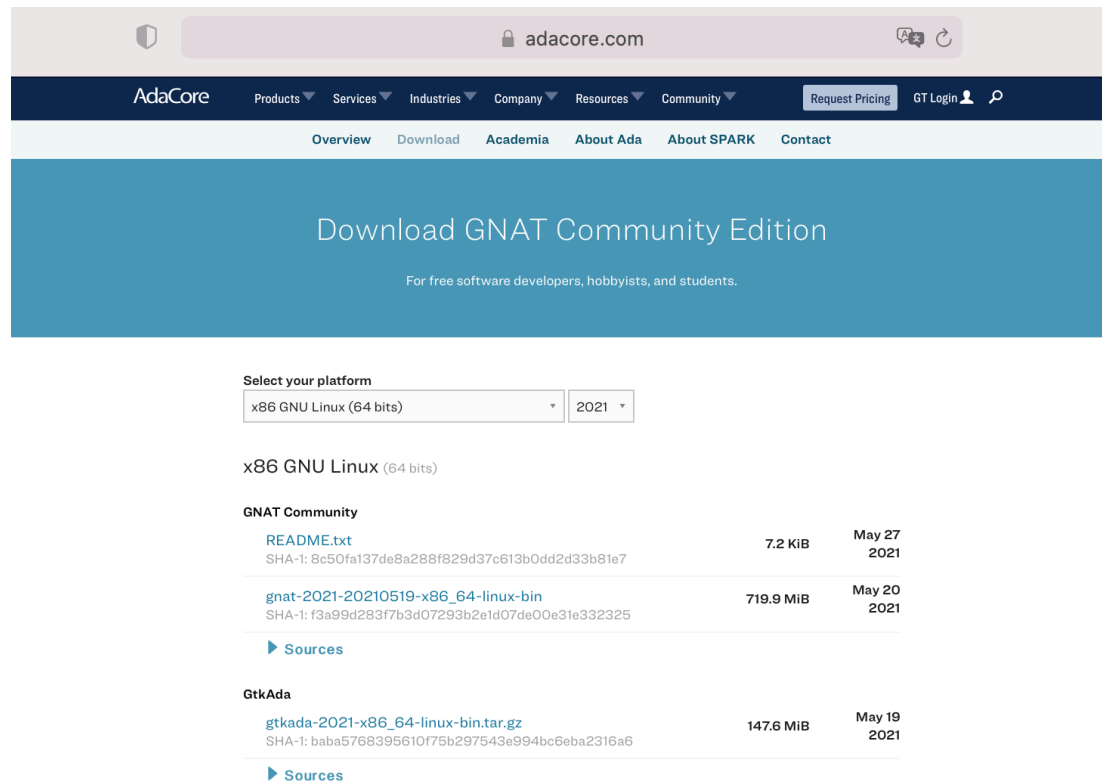




## 2. IDE: GnatStudio

# Software instalado en la Máquina Virtual

<https://www.adacore.com/download/more>



The screenshot shows the AdaCore website's download page for GNAT Community Edition. The page has a dark blue header with the AdaCore logo and navigation links. Below the header, there's a light blue section with the title "Download GNAT Community Edition" and a subtitle "For free software developers, hobbyists, and students." Underneath, there's a "Select your platform" section with two dropdown menus: "x86 GNU Linux (64 bits)" and "2021". Below this, there's a table of download links for the selected platform and version.

GNAT Community		
<a href="#">README.txt</a> SHA-1: 8c50fa137de8a288f829d37c613b0dd2d33b81e7	7.2 KiB	May 27 2021
<a href="#">gnat-2021-20210519-x86_64-linux-bin</a> SHA-1: f3a99d283f7b3d07293b2e1d07de00e31e332325	719.9 MiB	May 20 2021
<a href="#">Sources</a>		

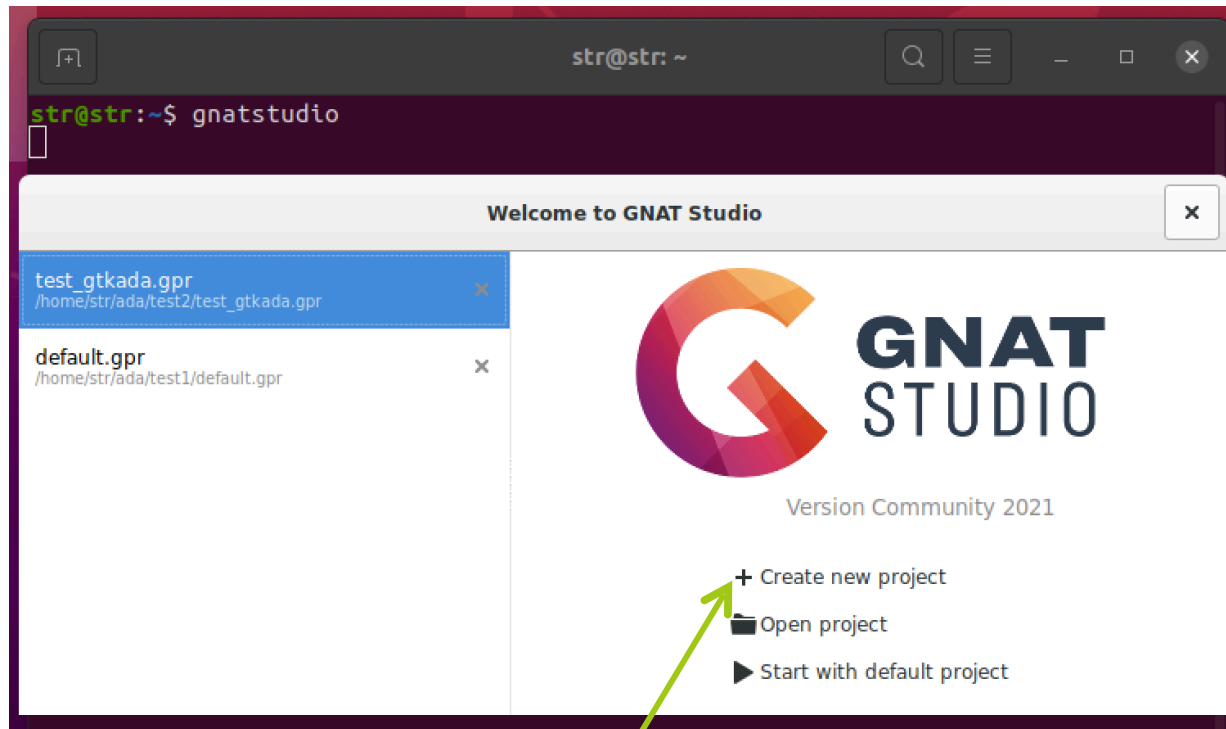
GtkAda		
<a href="#">gtkada-2021-x86_64-linux-bin.tar.gz</a> SHA-1: baba5768395610f75b297543e994bc6eba2316a6	147.6 MiB	May 19 2021
<a href="#">Sources</a>		



## 2. IDE: GnatStudio

# Arrancar el IDE

1. Desde línea de comandos arrancamos el IDE (tecleamos **gnatstudio**)

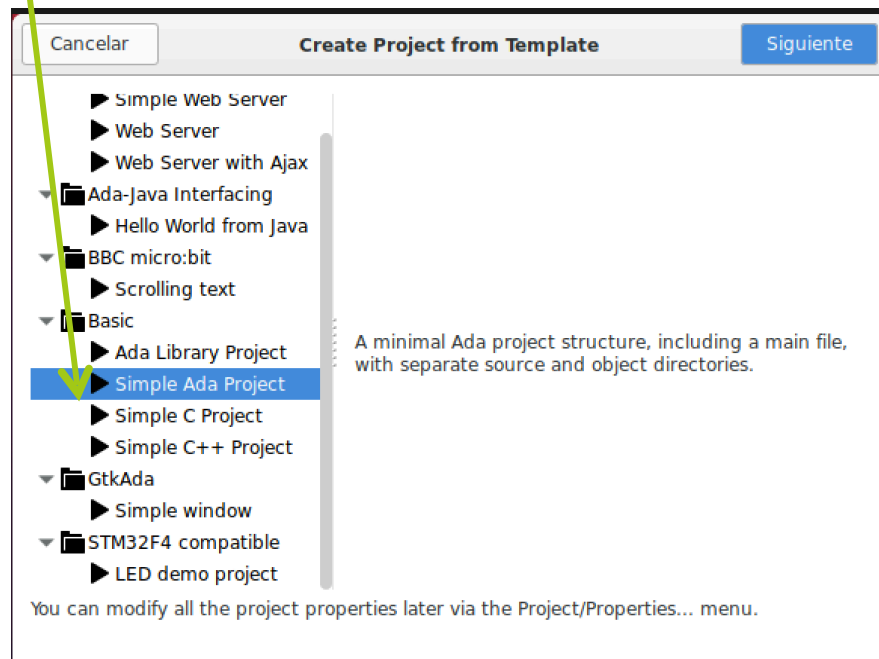


2. Elegimos crear un nuevo proyecto

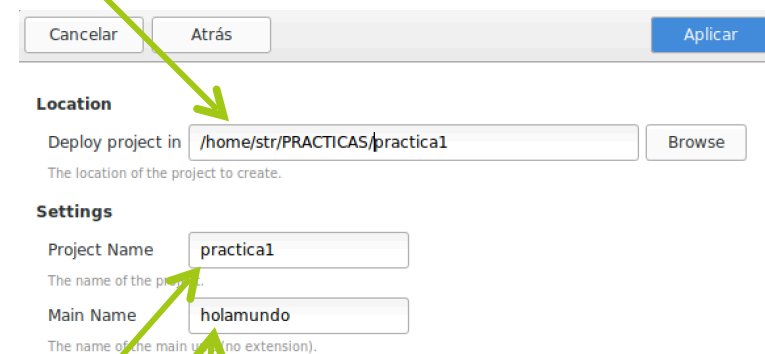


# Crear un proyecto Ada

3. Elegimos la opción de crear un proyecto básico de Ada



4. Indicar la ruta donde se guardará el proyecto

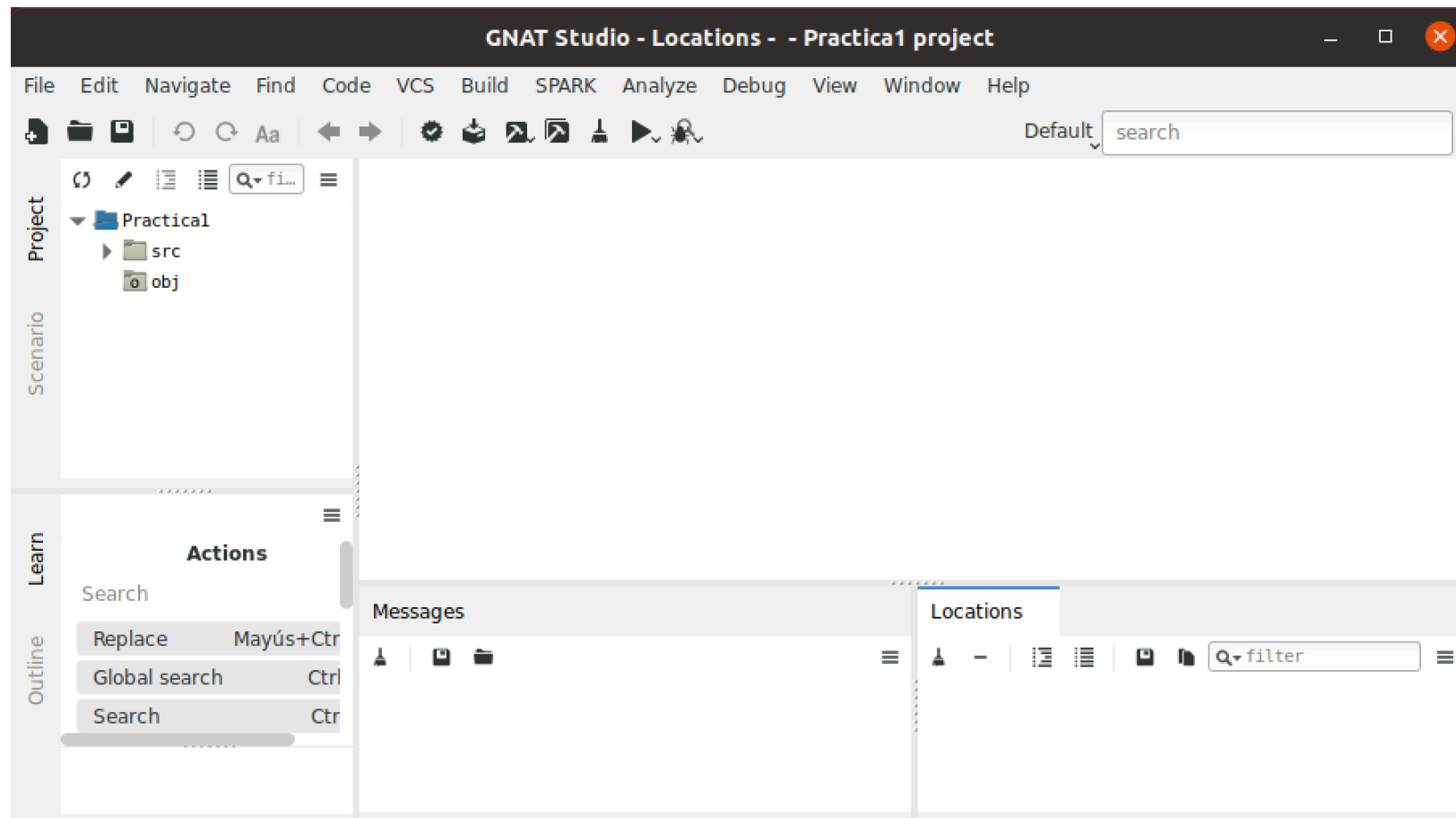


5. Le damos nombre al proyecto y al fichero que contendrá el procedimiento principal (*main*)



## 2. IDE: GnatStudio

# Ventana principal de un proyecto



# Propiedades del proyecto

Desde el menú contextual del nombre del proyecto (botón derecho del ratón), seleccionar Project->Properties

gtkada

/src  
/obj

Programas principales

**Properties for Practica1**

**General**

**Name & Location**

Name:

Path:

☒ Paths should be relative paths

Apply changes to:  
☒ Show as hierarchy

**Project**

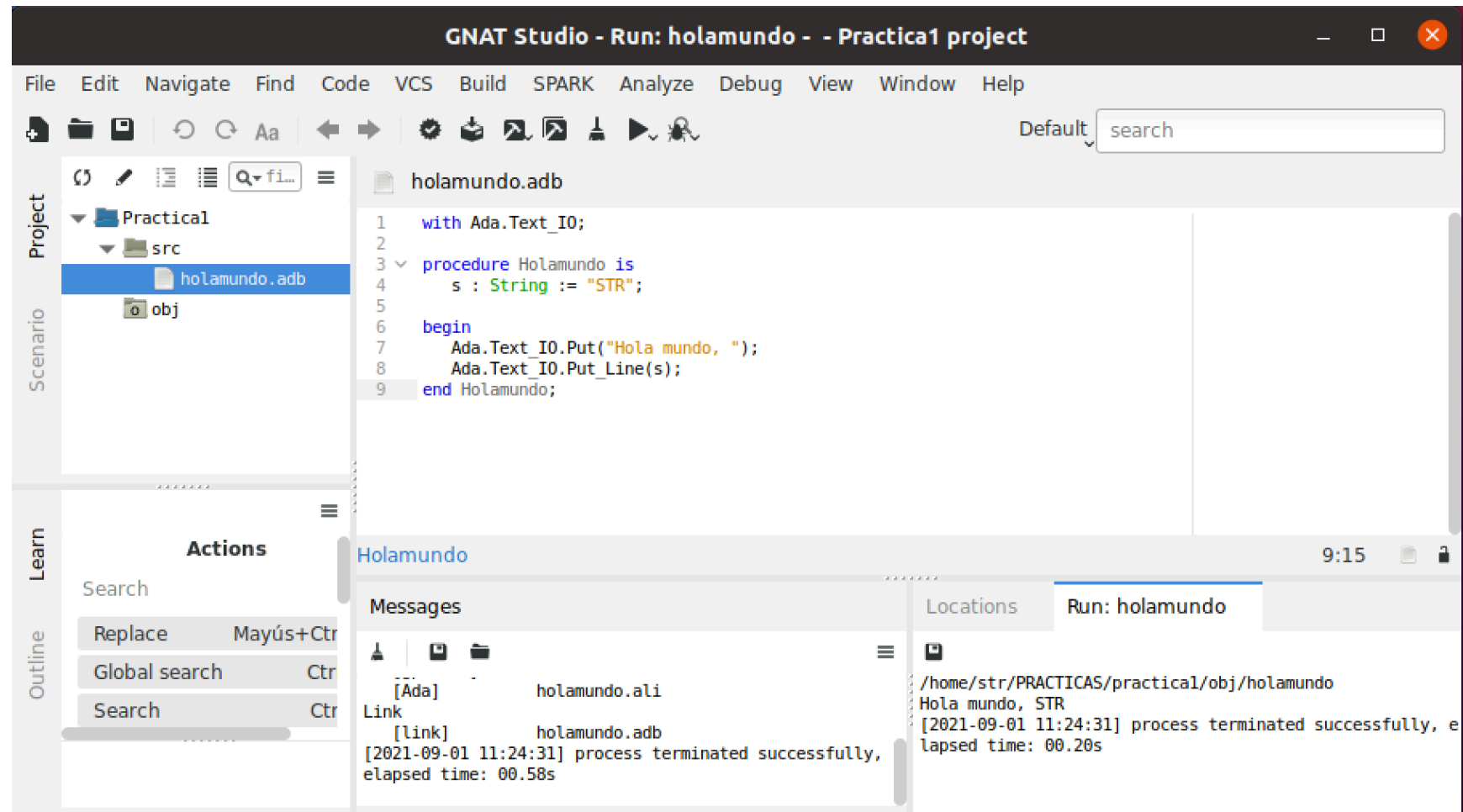
☒ Practica1

**Scenario**

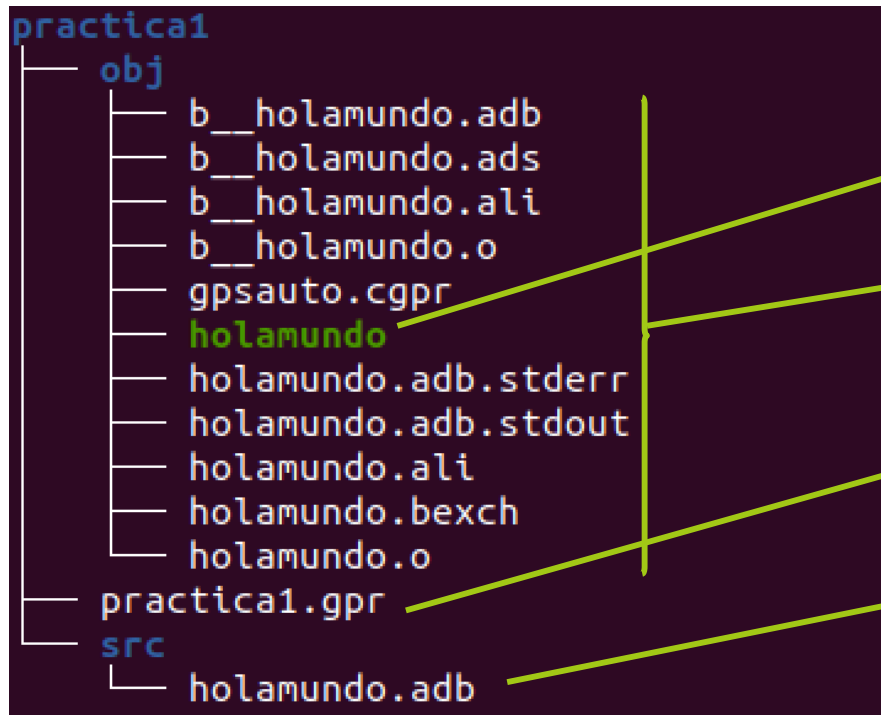


## 2. IDE: GnatStudio

# Crear un programa



# Estructura de ficheros de un proyecto



Ejecutable

Compilación y enlazado, e información del IDE

Información del proyecto

Código Fuente

```
practica1.gpr  
  
1  project Practical is  
2      for Source_Dirs use ("src");  
3      for Object_Dir use "obj";  
4      for Main use ("holamundo.adb");  
5  end Practical;  
|
```



# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. **Fuentes de Información**
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas





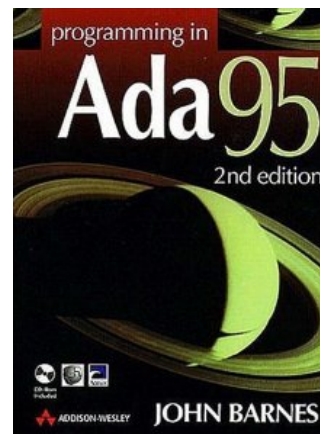
# Bibliografía recomendada de Ada

## ▣ Ada para propósito general

### **Programming in Ada95**

John Barnes

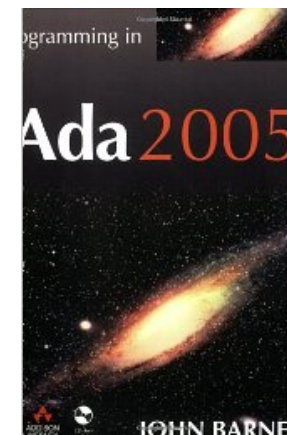
Addison Wesley. 2001



### **Programming in Ada 2005**

John Barnes

Addison Wesley. 2006

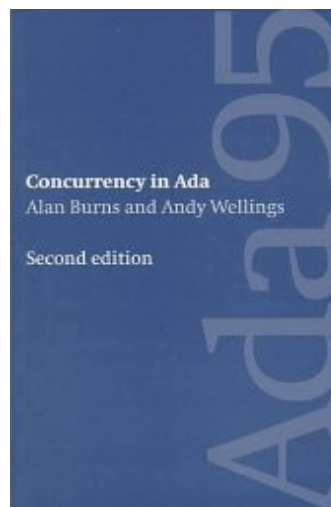


## ▣ Ada para STR

### **Concurrency in Ada**

Alan Burns and Andy Wellings

Cambridge University Press. 1998



### **Concurrent and Real-Time Programming in Ada**

Alan Burns and Andy Wellings

Cambridge University Press. 2007



## Manuales de referencia de Ada

### Ada 2012 Reference Manual

- Es el estándar del lenguaje
- Se puede acceder desde el IDE : Help -> GNAT
  - `file:///usr/gnat/share/doc/gnat/html/arm12.html`



## Información On-line

[https://en.wikibooks.org/wiki/Ada\\_Programming/All\\_Chapters](https://en.wikibooks.org/wiki/Ada_Programming/All_Chapters)

<http://www.adapower.com/index.php?Command=Learn>



# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. **Estructura de un programa**
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Unidades de Programa

- ▣ Una **unidad de programa** es:
  - ▣ un procedimiento
  - ▣ una función
  - ▣ un paquete
  - ▣ una unidad genérica
  - ▣ una tarea
  - ▣ una unidad protegida



# Programa en Ada

- Un **programa** Ada está formado por una o más unidades de programa :
  - Un **procedimiento principal** que se ejecuta inicialmente
  - El resto de unidades de programa se pueden encapsular en **paquetes**
    - Paquetes predefinidos
    - Paquetes escritos por el programador



# Mi primer programa en Ada

```
with Ada.Text_IO;
```

```
procedure Hola_Mundo is  
begin
```

```
    Ada.Text_IO.Put_Line("Hola mundo");
```

```
end Hola_Mundo;
```



# Especificación y Cuerpo

- Cada unidad de programa (excepto la del programa principal) se divide en dos partes:
  - una **especificación**, que define su interfaz con el “mundo exterior” (**\*.ads**)
  - un **cuerpo**, que contiene los detalles de implementación (**\*.adb**)





# Unidades de Compilación

- Compilación por **separado**
- Una **unidad de compilación** es:
  - un procedimiento o una función
  - un paquete
  - una unidad genérica
- Se pueden **anidar**
- En un fichero **sólo** puede haber **una unidad** de compilación



# Unidades de Librería

- Unidades de compilación **independientes** que se pueden usar en distintos programas
- La cláusula **with** proporciona visibilidad a las unidades de librería
- Acceso a sus componentes mediante:
  - la **notación de punto**
  - el uso de la cláusula **use**
- El paquete **Standard** (tipos y operadores predefinidos) que no es necesario incluirlo con la cláusula with



# Procedimientos y Funciones

- Los **parámetros** pueden tener tres modos:
  - **in** : valores de entrada a los procedimientos y funciones (modo por defecto)
  - **out** : valores de salida (sólo procedimientos)
  - **in out** : valores de entrada/salida (sólo procedimientos)
- **Sobrecarga** (*Overloading*)
- **Recursión**



# Procedimientos

```
procedure nombre_procedimiento ( parámetros ); -- especificación del procedimiento
```

```
procedure nombre_procedimiento (parámetros ) is – cuerpo del procedimiento  
  -- declaración de variables locales  
begin  
  -- sentencias  
exception  
  -- manejador de excepciones  
end nombre_procedimiento;
```



# Funciones

```
function nombre_función ( parámetros ) return tipo_función; -- especificación  
-- de la función
```

```
function nombre_función (parámetros ) return tipo_función is -- cuerpo  
-- de la función  
  
  -- declaración de variables locales  
  
  begin  
    -- sentencias  
  
    return ...  
  
  exception  
    -- manejador de excepciones  
  
  end nombre_función;
```



# Invocación de procedimientos y funciones

```
procedure Incrementar (valor : in out Integer; incr : in Integer := 1) is  
begin  
    valor := valor + incr;  
end Incrementar;
```

- La **llamada a procedimientos** es una instrucción simple, que puede formar parte de cualquier secuencia de instrucciones

```
Incrementar(x,2);                -- asociación de parámetros por posición  
Incrementar(incr => 2; valor => x); -- asociación de parámetros por nombre  
Incrementar(x);                  -- incr => 1 (valor por defecto)
```

- La **llamada a una función** puede formar parte de cualquier sentencia en donde se pueda utilizar el valor que devuelve



# Paquetes

- Mecanismo de **encapsulación** y **ocultación** de información

```
package nombre_paquete is -- especificación del paquete  
    -- declaraciones públicas  
private  
    -- declaraciones privadas  
end nombre_paquete;
```

```
package body nombre_paquete is -- cuerpo del paquete  
    -- implementación  
begin  
    -- sentencias de inicialización  
end nombre_paquete;
```



# Ejemplo de *Package* en Ada

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

package PKG_fichero is
  procedure Crear_Fichero;
  procedure Abrir_Fichero;
  procedure Cerrar_Fichero;
  procedure Escribir_Fichero (cadena:String);

  private
    nomb_fich: String:= "salida.txt";
    fich: File_Type;

end PKG_fichero;
```

```
package body PKG_fichero is

  procedure Crear_Fichero is
  begin
    Create(fich, Append_File, nomb_fich);
  end Crear_Fichero;

  procedure Abrir_Fichero is
  begin
    Open(fich, Append_File, nomb_fich);
  end Abrir_Fichero;

  procedure Cerrar_Fichero is
  begin
    Close(fich);
  end Cerrar_Fichero;

  procedure Escribir(cadena:String) is
  begin
    Put_Line(fich,cadena);
  end Escribir_Fichero;

end PKG_fichero;
```





# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. **Sintaxis**
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Identificadores

- No se distingue entre mayúsculas y minúsculas
- De longitud arbitraria
- El primer carácter deber ser una letra
- El resto de caracteres pueden ser letras, dígitos o el carácter subrayado '\_' (éste último no terminal ni dos seguidos)

### Identificadores legales

nombre\_fichero  
Nombre\_Fichero  
num\_1  
n1

### Identificadores ilegales

nombre\_\_fichero  
Nombre-Fichero  
1num  
num\_



# Literales

### Numéricos

10

10.0

1E3

5.0E-1

### Carácter (entre comillas simples)

'a'

'1'

'+'

### Cadenas (entre comillas dobles)

“esto es un ejemplo de cadena”

### null

- sentencia nula
- valor específico para punteros



# Comentarios del código fuente

- Comienzan con dos guiones “- -”
- Terminan con el fin de línea
- Pueden aparecer en cualquier punto del programa

*-- Este comentario ocupa una línea completa*

*x := x + 1; -- este comentario está después de una sentencia*

*x := x + -- este comentario está entre una sentencia que ocupa dos líneas*  
*1;*



# Operadores

=    /=    <    <=    >    >=

+    -

\*    /    mod

\*\*    abs

and    or    not    xor    and then    or else

in

&



# Sentencias simples

### Asignación

*nombre\_de\_variable := expresion;*

### Control de flujo de ejecución

**exit;**     - - *salida incondicional de un bucle*

**exit when** *condicion;*

**return;**   - - *terminación de un subprograma*

**raise** *nombre\_excepcion;*



# Sentencias de Selección



```
if condicion then  
    sentencias  
end if;
```

```
if condicion then  
    sentencias;  
elsif condicion then  
    sentencias;  
else  
    sentencias;  
end if;
```



```
case expresion is  
    when valor_expresion => sentencias;  
    when valor_expresion => sentencias;  
    when others => sentencias;  
end case;
```



# Bucles

### □ **for**

- La variable de control:
  - No se declara
  - Es local al bucle
  - No se puede modificar

```
for i in 1..10 loop  
  sentencias;  
end loop;
```

```
for i in reverse 1..10 loop  
  sentencias;  
end loop;
```

### □ **while**

```
i := 0;  
while i < 10 loop  
  sentencias;  
  i := i + 1;  
end loop;
```

### □ **loop**

```
loop  
  sentencias;  
  exit when condicion;  
end loop;
```





# Bloques

- Ada es un lenguaje estructurado en bloques

**declare** - - *opcional*

- - ***parte declarativa***

**begin**

- - ***sentencias ejecutables***

**exception** - - *opcional*

- - ***manejador de excepciones***

**end;**



# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. **Excepciones**
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Concepto de Excepción

- Una **excepción** es la ocurrencia de una situación inesperada
- Excepciones **predefinidas** :
  - Constraint\_error
  - Program\_error
  - Storage\_error
  - Tasking\_error
  - Las relacionadas con la entrada/salida
  - etc.
- Excepciones definidas por el **usuario**:

Error1, Error2 : **exception**; -- la declaración de excepciones está sujeta a las reglas  
-- generales de declaración



# Manejo de Excepciones

- ▣ Las excepciones predefinidas se lanzan **automáticamente**
- ▣ Las excepciones definidas por el usuario se lanzan utilizando la sentencia **raise**
- ▣ El manejador de excepciones se coloca al final de cualquier bloque **begin .. End**

**begin**

...

**exception**

**when** Error1 | Error2 => *instrucciones a ejecutar si ocurre la excepción Error1 o la Error2;*

**when** Error3 => *instrucciones a ejecutar si ocurre la excepción Error3;*

**end;**

- ▣ Si la excepción no es tratada en el manejador del mismo bloque, se **propaga** al nivel superior

- ▣ Si no se encuentra ningún manejador, se **aborta** la ejecución del programa



## 6. Excepciones

# Ejemplo de Excepciones

```
with Ada.Exceptions; use Ada.Exceptions;
package body PKG_fichero is
  procedure Abrir_Fichero is
  begin
    Open(fich, Append_File, nomb_fich);
  exception
    when Name_Error => Create(fich, Append_File, nomb_fich);
    when event: others => Put_Line("ERROR en Abrir_Fichero: " & exception_name(exception_identity(event)));
  end Abrir_Fichero;

  procedure Cerrar_Fichero is
  begin
    Close(fich);
  exception
    when event: others => Put_Line("ERROR en Cerrar_Fichero: " & exception_name(exception_identity(event)));
  end Cerrar_Fichero;

  procedure Escribir_Fichero(cadena:String) is
  begin
    Put_Line(fich,cadena);
  exception
    when event: others => Put_Line("ERROR en Escribir_Fichero: " & exception_name(exception_identity(event)));
  end Escribir_Fichero;
end PKG_fichero;
```



# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. **Tipos de datos**
8. Unidades genéricas
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Ada es Fuertemente Tipado

- No se permiten asignaciones entre variables de tipos distintos

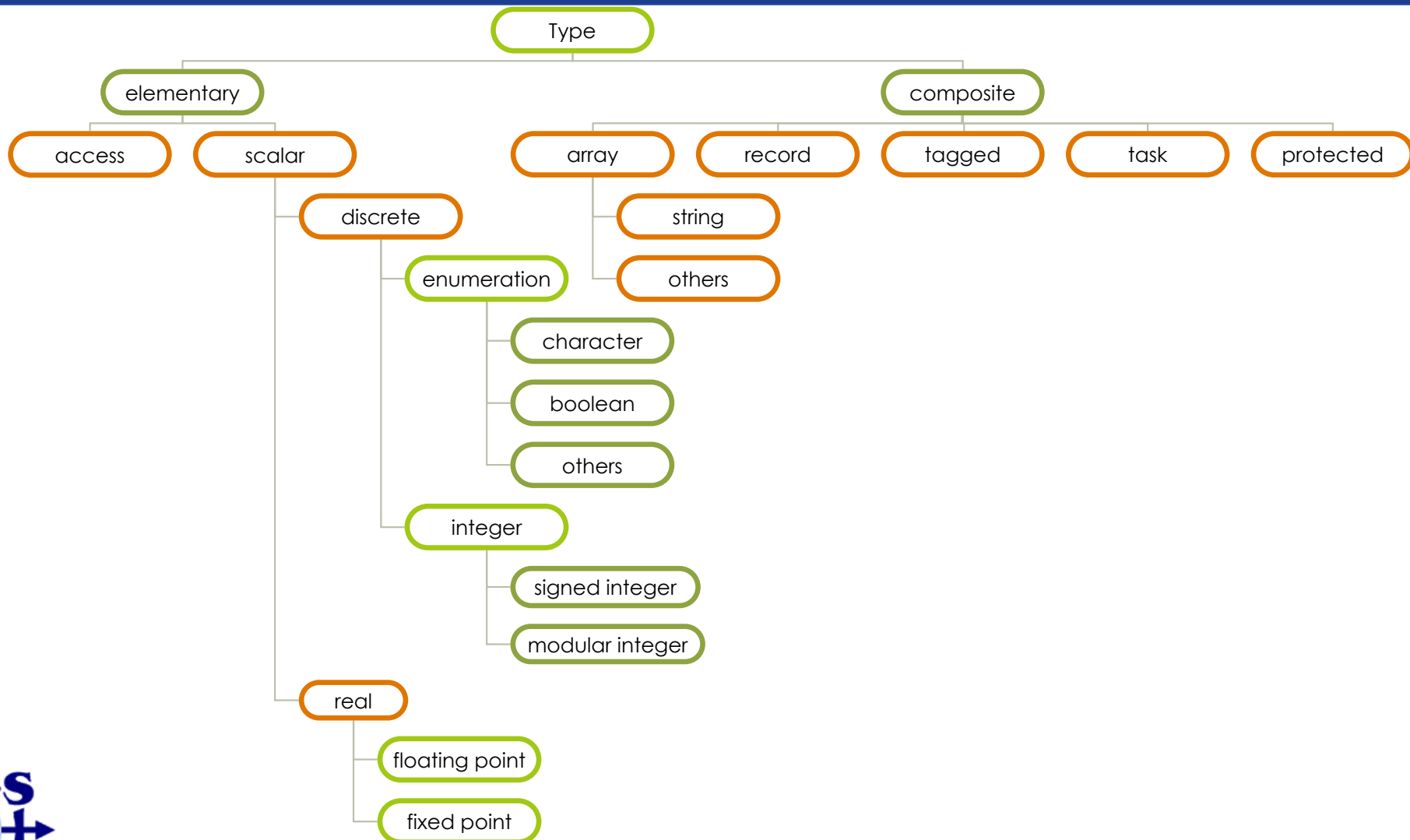
```
c: character := '1';  
i: integer := 1;  
n: natural;  
i:= c; -- ilegal, error de compilación  
n:= i-5; -- error de ejecución (exception constraint error)
```

- Se pueden realizar **conversiones explícitas** (con precaución)

```
float(38); -- valor 38.0  
integer(4.4 + 5.3); -- valor 10
```



# Jerarquía de Tipos de Datos





# Tipos de Datos Discretos

### □ Enumerables

□ **Boolean** -- (true, false)

□ **Character**

□ Definido por el usuario:

□ **type** TColor **is** (Rojo, Verde, Azul);

### □ Enteros

□ **Integer**

□ se puede utilizar '\_' para separar dígitos: 100\_000 = 100000

□ **Modulares:** **type** TByte **is mod** 256;

□ Aritmética modular (por ejemplo,  $255+1=0$ )



# Tipos de Datos Reales

- ▣ Coma flotante
  - ▣ Precisión relativa - - depende del compilador
  - ▣ **Float**
  - ▣ Definido por el usuario:
    - ▣ **type** Intervalo **is digits 5 range** -1.0...1.0;
- ▣ Coma fija
  - ▣ Precisión absoluta
  - ▣ Ordinarios:
    - ▣ **type** Voltage **is delta** 0.125 **range** 0.0..5.25;
  - ▣ Decimales
    - ▣ **type** Euros **is delta** 0.01 **digits** 15;



# Subtipos

- Subconjunto de valores de un tipo, definido por una restricción
  - subtype** Negative **is Integer range** Integer'First..-1;
  - subtype** Uppercase **is Character range** 'A'..'Z';
  - subtype** Probability **is float range** 0.0..1.0;
- Subtipos predefinidos:
  - **Natural** -- **subtype** Natural is Integer range 0..Integer'Last
  - **Positive** -- **subtype** Positive is Integer range 1..Integer'Last
- Las operaciones con distintos subtipos de un mismo tipo base están permitidas



# Tipos Compuestos

### ■ Arrays

■ **String** -- **type** String **is array** (Positive **range** <>) **of** Character

```
cad1: String(1..80);      -- si no se inicializa, se debe especificar un rango
cad2 : String := "Hola";  -- entre dobles comillas
```

### ■ Definido por el usuario:

```
type Voltages is array (1..50) of Voltage;
type Matrix is array (1..10,1..10) of Float;
```

```
v : Voltages;
m : Matrix;
V(2) := 0.250;
M(1,7) := 10.0;
```

### ■ Registros

```
type T_Coordenada is
record
  x : Integer;
  y : Integer
end record;
```

```
coord : T_Coordenada;
coord.x := 5; -- acceso a campos mediante notación punto '.'
coord.y := -7;
```



# Arrays “ilimitados”

- Se pueden declarar arrays sin especificar el número de sus elementos  
`type T_Vector is array (Integer range <>) of Float;`
- No se pueden declarar variables de un array no restringido  
`v : T_Vector; -- ilegal, error de compilación`
- Dos posibles usos son:
  - **Tipos base** para subtipos de vectores "limitados"  
`subtype Vector_A is T_Vector(1..10);`  
`subtype Vector_B is T_Vector(1..1000);`
  - **Parámetros** de subprogramas  
`function Maximo (v: in T_Vector) return Float;`



### Tipo Access (Punteros)

- Se utiliza como puntero a estructuras de datos, procedimientos y funciones, u otros tipos access

```
Type Ptr_Int is access Integer;
```

```
I: Integer;
```

```
IP : Ptr_Int := new Integer; -- crear un objeto dinámico
```

```
IP.all := 42; -- actualizar el contenido de IP
```



# Atributos de Tipos Escalares

- ❑ **T'First**
- ❑ **T'Last**
- ❑ **T'Range**
- ❑ **T'Succ(valor)**
- ❑ **T'Pred(valor)**
- ❑ **T'Image(valor)**
- ❑ **T'Width**
- ❑ **T'Value(cadena)**



# Tipos privados

- Visible en la especificación de un paquete, pero su implementación está oculta dentro del mismo
- Son una forma de implementar tipos abstractos de datos en Ada

```
package Complex_Arithmetic is
  type Complex is private;
  function "+" (X,Y: Complex) return Complex;
  function "-" (X,Y: Complex) return Complex;
  function "*" (X,Y: Complex) return Complex;
  function "/" (X,Y: Complex) return Complex;
  function Comp (A,B: Float) return Complex;
  function Real_Part (X: Complex) return Float;
  function Imag_Part (X: Complex) return Float;
private
  type Complex is
    record
      Real_Part: Float;
      Imag_Part: Float;
    end record;
end Complex_Arithmetic;
```





# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. **Unidades genéricas**
9. Librerías predefinidas
10. Concurrencia en Ada: Tareas



# Concepto de unidad genérica

- Es un componente **reusable** de software
- Permite definir plantillas de componentes en los que se dejan indefinidos algunos aspectos (**parámetros genéricos**)
  - Tipos de datos, objetos, operaciones, etc.
- Los componentes concretos se crean a partir de la plantilla **instanciando** los parámetros genéricos
- En Ada podemos tener:
  - Subprogramas genéricos
  - Paquetes genéricos



# Procedimientos y funciones genéricas

### **generic**

-- *parámetros genéricos*

**function** nombre\_función (parámetros) **return** tipo\_funcion;

### **generic**

-- *parámetros genéricos*

**procedure** nombre\_procedimiento (parámetros);

-- Especificación

### **generic**

**type** Tdato **is private**;

**procedure** Intercambiar (d1: **in out** Tdato; d2: **in out** Tdato);

-- Instanciación

Procedure Intercambiar\_Entero is new Intercambiar(Tdato=>Integer);

Procedure Intercambiar\_Caracter is new Intercambiar(Tdato => Character);



# Paquetes genéricos

### generic

```
type TData is private; -- parámetros genéricos
package Store is
  type TBuffer is limited private;
  procedure Give (d: TData; b: in out TBuffer);
  procedure Take (d: out TData; b: in out TBuffer);

  private -- declaraciones privadas
    size : constant := 80;

    type TVector is array (1..size) of TData;
    subtype TLon is integer range 0..size;

    type TBuffer is
      record
        vector: TVector;
        lon   : TLon := 0;
      end record;
end Store;

package body Store is
  -- implementación
end Store;
```

### -- Utilización

```
type TDate is
  record
    Day: Integer range 1..31;
    Month: Integer range 1..12;
    Year: Integer range 1066..2066;
  end record;

package Date_Store is new Store(TData => TDate);
package Int_Store is new Store(TData => Integer);

BufferI : Int_Store.TBuffer;
BufferD : Date_Store.TBuffer;

i : integer;
date : TDate

...

Int_Store.Give(i, BufferI);
Date_Store.Take(date, BufferD);
```



# Paquetes genéricos predefinidos para Entrada/Salida

- Utilizados para I/O de datos definidos por el usuario
- En el paquete `Ada.Text_IO` se definen los siguientes subpaquetes genéricos:
  - `Integer_IO`
  - `Float_IO`
  - `Fixed_IO`
  - `Modular_IO`
  - `Decimal_IO`
  - `Enumeration_IO`

```
type Tmi_Integer is new Integer range 0..100;  
package pkg_Tmi_Integer is new  
  Ada.Text_IO.Integer_IO(Tmi_Integer);  
  ...  
  Num : Tmi_Integer;  
  ...  
  Pkg_Tmi_Integer.get(num);  
  Pkg_Tmi_Integer.put(num);
```



# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. **Librerías predefinidas**
10. Concurrencia en Ada: Tareas



# Biblioteca estándar

- Paquetes predefinidos:
  - Operaciones con caracteres y cadenas
    - **Ada.Characters**, **Ada.Strings**, etc.
  - Cálculo numérico
    - **Ada.Numerics**, **Ada.Numerics.Generic\_Elementary\_Functions**, etc.
    - También números complejos, vectores y matrices
  - Entrada/Salida
    - **Ada.Text\_IO**, **Ada.Integer\_Text\_IO**, **Ada.Float\_Text\_IO**, **Gnat.io**, etc.
  - Otros...



# Anexos de Ada

- Los anexos
  - definen paquetes de biblioteca
  - no añaden sintaxis ni vocabulario
- Las versiones normalizadas de Ada definen:
  - Un núcleo para todas las implementaciones (core language)
  - Unos anexos específicos para
    - programación de sistemas
    - sistemas de tiempo real
    - sistemas distribuidos
    - cálculo numérico
    - fiabilidad y seguridad
    - Otros...





# Índice

1. Origen y características generales
2. IDE: GnatStudio
3. Fuentes de Información
4. Estructura de un programa
5. Sintaxis
6. Excepciones
7. Tipos de datos
8. Unidades genéricas
9. Librerías predefinidas
10. **Concurrencia en Ada: Tareas**



# Declaración de Tareas

- Las tareas se declaran dentro de cualquier parte declarativa de :
  - un subprograma
  - un bloque
  - un paquete
  - el cuerpo de otra tarea
- Están formadas por
  - Una **especificación** (interfaz con el exterior)
  - Un **cuerpo** (comportamiento de la tarea)

## Tipo tarea definido por usuario

```
task type Tipo_A;  
task type Tipo_B;  
  
A : Tipo_A;  
B : Tipo_B;
```

## Tipo anónimo

```
task A;  
task B;
```



# Tipo Task definido por usuario

## Especificación

```
task type T_tarea (X : tipo_parametro) is  
  -- declaración de interfaz con otras tareas  
private  -- opcional  
end T_Tarea;
```

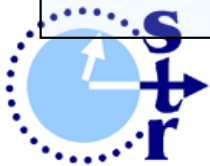
```
task type T_tarea;  
  -- si no tiene interfaz con  
  -- otras tareas
```

## Cuerpo

```
task body T_Tarea is  
  -- declaraciones locales  
begin  
  -- implementación del comportamiento  
  -- de la tarea  
end T_Tarea;
```

**Parámetros** de inicialización (**discriminantes**)

- ▣ de tipos discretos
- ▣ de tipo puntero (access)



# Task anónima

## Especificación

```
Task nombre_tarea is  
  -- declaración de interfaz con otras tareas  
private  -- opcional  
end nombre_tarea;
```

```
Task nombre_tarea;  
  -- si no tiene interfaz con  
  -- otras tareas
```

## Cuerpo

```
task body nombre_tarea is  
  -- declaraciones locales  
begin  
  -- implementación del comportamiento de la tarea  
End nombre_tarea;
```



### Fases de una Tarea

**declare**

...

**A: T\_tarea;**

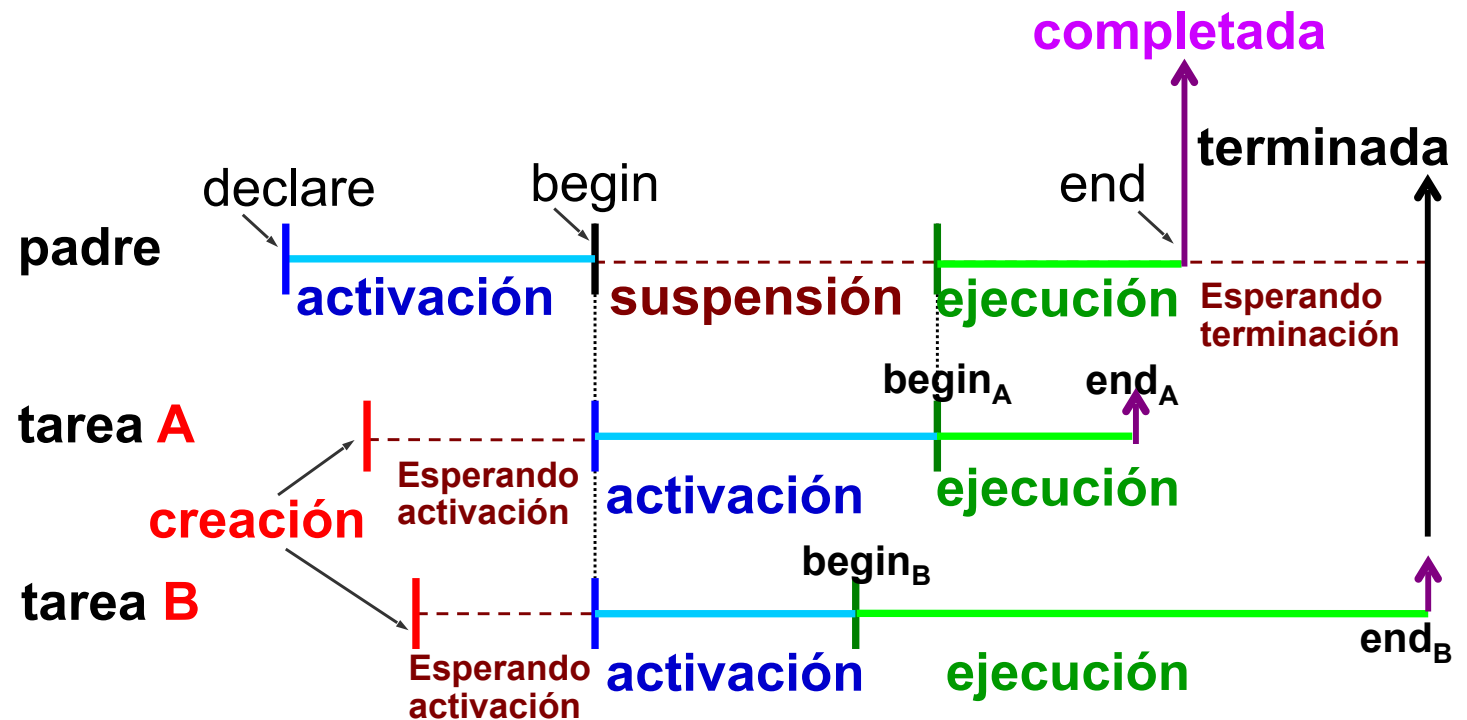
**B: T\_tarea;**

...

**begin**

...

**end;**



### Creación de tareas

```
Procedure Ejemplo is
  task type T_tarea;
  type Ptr_T_tarea is access T_Tarea;

  task body T_tarea is
  begin
    ...
  end;

  Tarea_E : T_tarea;  -- CREACIÓN DE TAREA ESTÁTICA
  Tarea_D : Ptr_T_Tarea; -- declaración de puntero a tipo tarea
Begin
  ...
  Tarea_D := new T_Tarea;  -- CREACIÓN DE TAREA DINÁMICA
end Ejemplo;
```



### Activación de tareas

- La activación de una tarea es la elaboración de su **parte declarativa**
- **Tareas estáticas**
  - Se activan justo antes de empezar la ejecución de la unidad
- **Tareas dinámicas**
  - Se activan al ejecutarse el operador *new*



# Ejemplo de creación y activación

```
Procedure Ejemplo is
  task type T_tarea;
  type Ptr_T_tarea is access T_Tarea;

  task body T_tarea is
  begin
    ...
  end;

  R : T_tarea;  -- CREACIÓN de la tarea R
  P : Ptr_T_Tarea;
  Q : Ptr_T_Tarea := new T_Tarea; -- CREACIÓN Y ACTIVACIÓN
                                   -- de la tarea Q.all

Begin  -- ACTIVACIÓN de la tarea R
  ...
  P := new T_Tarea;  -- CREACIÓN Y ACTIVACIÓN de P.all
  Q := new T_Tarea;  -- CREACIÓN Y ACTIVACIÓN de otra tarea Q.all
                     -- La primera tarea Q sigue ejecutándose,
                     -- pero pasa a ser anónima

end Ejemplo;
```





### Terminación de tareas

- ❑ Ejecución completada
- ❑ Suicidio
- ❑ Aborto
- ❑ Excepción no manejada
- ❑ Nunca
- ❑ Innecesaria



# Manejo de excepciones en tareas

- Como norma general, las excepciones **no se propagan** a la tarea padre
- **Sólo si** se produce durante la activación de la tarea hijo, se propaga **Tasking\_Error** al padre



## Bibliografía Recomendada

Concurrency in Ada (**2nd edition**)

Alan Burns and Andy Wellings

Cambridge University Press (1998)

 Capítulos 2 y 4 (excepto Apartado 2.5)



# Bibliografía Complementaria

Programming in Ada95 (**2nd edition**)

John Barnes

Addison Wesley (2001)

 Capítulo 18 (Apartados 18.1; 18.7)

