

## Práctica 2

# Bibliotecas

Con esta práctica se pretende que el alumno:

- empiece a estructurar las aplicaciones usando el concepto de biblioteca e
- implemente una biblioteca usando los paquetes de Ada.

### 2.1 Requisitos

1. Entorno de programación en Ada.
2. Programación a pequeña escala en lenguaje Ada.
3. Implementación de paquetes y de tipos privados (la teoría se desarrolla en la L02 «Programación a gran escala» de la asignatura).

### 2.2 Introducción teórica

En esta práctica vamos a escribir una biblioteca para trabajar con números fraccionarios. Estos números se construyen como parejas de números enteros:  $\mathbb{Q} = \mathbb{Z} \times \mathbb{Z} = \{(n, d); n \in \mathbb{Z} \text{ y } d \in \mathbb{Z}\}$ . Los elementos de la pareja se llaman numerador y denominador respectivamente, y se suelen escribir de la forma  $n/d$ .

Las operaciones aritméticas con fracciones se definen de la forma siguiente:

- Suma de fracciones  $[+]$ :

$$\frac{a}{b} + \frac{c}{d} = \frac{a \times d + c \times b}{b \times d}$$

- Elemento opuesto  $[- \text{ unario}]$ :

$$-\frac{a}{b} = \frac{-a}{b}$$

- Resta de fracciones  $[-]$ :

$$\frac{a}{b} - \frac{c}{d} = \frac{a}{b} + \left(-\frac{c}{d}\right)$$

- Producto de fracciones  $[\times]$ :

$$\frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d}$$

- Elemento inverso  $[()^{-1}]$ :

$$\left(\frac{a}{b}\right)^{-1} = \frac{b}{a}$$

- División de fracciones[÷]:

$$\frac{a}{b} \div \frac{c}{d} = \frac{a}{b} \times \left(\frac{c}{d}\right)^{-1}$$

Como vemos, la operaciones suma, elemento opuesto, producto y elemento inverso se definen a partir de operaciones con números enteros. Se dice que son operaciones primitivas, en oposición a la resta y la división que se definen a partir de las operaciones anteriores. A la hora de implementar la biblioteca de fracciones, se puede optar por implementar todas las operaciones como primitivas.

El conjunto  $\mathbb{Q}$  se divide en clases de equivalencia con el siguiente criterio:

$$\frac{a}{b} = \frac{c}{d} \Leftrightarrow a \times d = b \times c$$

Estas clases de equivalencia son la forma de definir la igualdad entre fracciones.

Una fracción se llama reducible cuando se cumple:

$$\frac{a}{b} = \frac{n \times c}{n \times d}, \quad n \in \mathbb{N}, \quad n \neq 1, \quad c \in \mathbb{Z} \text{ y } d \in \mathbb{Z}$$

Toda clase de equivalencia tiene una única fracción irreducible. Como todas las fracciones que pertenecen a una misma clase de equivalencia representan al mismo número racional, se suele elegir como representante de esa clase a su fracción irreducible.

Si  $\frac{a}{b}$  y  $\frac{c}{d}$  pertenecen a la misma clase de equivalencia y  $\frac{c}{d}$  es irreducible, entonces se cumple que:

$$\frac{a}{b} = \frac{\text{mcd}(a,b) \times c}{\text{mcd}(a,b) \times d}$$

donde  $\text{mcd}(a,b)$  es el *máximo común divisor* de  $a$  y  $b$ . Esto se puede ver más claro con el ejemplo siguiente:

$\frac{10}{4} = \frac{5}{2}$  donde  $\frac{5}{2}$  es irreducible; como  $\text{mcd}(10,4) = 2$ , se cumple que:

$$\frac{10}{4} = \frac{2 \times 5}{2 \times 2}$$

## 2.3 Tareas a realizar

Escribir un paquete de nombre `Fracciones` que se ajuste a la especificación siguiente:

```

1 package Fracciones is
2     type fraccion_t is private;
3
4     function "+" (X, Y: fraccion_t) return fraccion_t;
5     function "-" (X: fraccion_t) return fraccion_t;
6     function "-" (X, Y: fraccion_t) return fraccion_t;
7     function "*" (X, Y: fraccion_t) return fraccion_t;
8     function "/" (X, Y: fraccion_t) return fraccion_t;
9     function "=" (X, Y: fraccion_t) return Boolean;
10
11     -- Operaciones de entrada/salida con la consola
12     procedure Leer (F: out fraccion_t);
13     procedure Escribir (F: fraccion_t);
14
15     -- Constructor de numeros fraccionarios a partir de numeros enteros
16     function "/" (X, Y: Integer) return fraccion_t;
17
18     -- Operaciones para obtener las partes de una fraccion
19     function Numerador (F: fraccion_t) return Integer;
```

```

20     function Denominador(F:fraccion_t) return Positive;
21 private
22     type fraccion_t is record
23         Num: Integer;
24         Den: Positive;
25     end record;
26 end Fracciones;

```

**Fichero 2.1:** Especificación del paquete Fracciones (fracciones.ads)

Para probar este paquete se utilizará el programa siguiente:

```

1  with Ada.Text_Io, Fracciones;
2  with Ada.Exceptions;
3  use Ada.Text_Io, Fracciones;
4
5  procedure Prueba is
6      package Integer_Es is new Integer_Io (Integer);
7      use Integer_es;
8      Practica_no_Apta: exception;
9      A: fraccion_t := 2/3;
10     B: fraccion_t := (-9)/18;
11     P: fraccion_t := 0/5;
12
13 begin
14
15     if Numerador (B) /= -1 or
16        Denominador (B) /= 2 then
17         raise Constraint_Error;
18     end if;
19     Put("El valor de A = ");
20     Escribir(A);
21     Put(" El valor de B = ");
22     Escribir(B);
23     Put(" El valor de P = ");
24     Escribir(P);
25     Put_Line(" ");
26     if B /= 1/(-2) then raise Practica_no_Apta;
27     end if;
28     Put("A+B = ");
29     Escribir(A+B);
30     if A+B /= 1/(6) then raise Practica_no_Apta;
31     end if;
32     Put("-A = ");
33     Escribir(-A);
34     if -A /= -2/3 then raise Practica_no_Apta;
35     end if;
36     Put("A-B = ");
37     Escribir(A-B);
38     if A-B /= 7/6 then raise Practica_no_Apta; end if;
39     Put("A*B = ");
40     Escribir(A*B);
41     if A*B /= -1/3 then raise Practica_no_Apta; end if;
42     Put("A/B = ");
43     Escribir(A/B);

```

```

44  if A/B /= -4/3 then raise Practica_no_Apta; end if;
45  Put("A-B = ");
46  Escribir(A-B);
47  if A-B /= 7/6 then raise Practica_no_Apta; end if;
48  Put("A-A = ");
49  Escribir(A-A);
50  if A-A /= 0/3 then raise Practica_no_Apta; end if;
51
52  Put("A/P = ");
53  Escribir(A/P);
54  P:=A/P;
55  P:=1/1;
56  for I in 1..10 loop
57      P:=P*B;
58  end loop;
59  if P/= 1/1024 then raise Practica_no_Apta; end if;
60  Put_Line ("Practica apta");
61 exception
62  when Ocurrencia : Practica_no_Apta =>
63      Put_line ("Practica no apta.");
64      Put (Ada.Exceptions.Exception_Information (Ocurrencia));
65  when Ocurrencia : Constraint_Error =>
66      Put_Line ("Practica no apta:");
67      Put_Line ("Las fracciones tienen que representarse mediante");
68      Put_Line ("fracciones irreducibles");
69      Put_Line ("Es necesario reducir las fracciones");
70      Put_Line (Ada.Exceptions.Exception_Information (Ocurrencia));
71 end Prueba;

```

**Programa 2.2:** Prueba del paquete Fracciones (pruebaFORSLIDESONLY.adb)

## 2.4 Tips & Tricks (Trucos y consejos)

- Evita copiar y pegar directamente desde el PDF, ya que esto puede generar espacios innecesarios que, aunque no sean visibles de inmediato, pueden causar errores en el código.
- Como recordatorio, aquí tienes un ejemplo de cómo se realiza el overloading de operadores en Ada.

```

1  with Ada.Text_IO; -- we're using the Ada.Text_IO package to write to the console
2
3  procedure overloading_operators is
4
5      -- Define a new type
6      type My_Integer is new Integer;
7
8      -- Create an instance of the My_Integer_IO package to write to the console.
9      -- Without this package, we wouldn't be able to use 'Image to convert the
10     -- our new type to a string.
11     package My_Integer_IO is new Ada.Text_IO.Integer_IO(My_Integer);
12
13     -- Overload the + operator for our new type
14     function "+" (Left, Right : My_Integer) return My_Integer is
15     begin

```

```
16     return My_Integer(Integer(Left) + Integer(Right));
17 end "+";
18
19 -- Declare some variables
20 A, B, C : My_Integer;
21 begin
22     A := 5;
23     B := 10;
24     C := A + B; -- Uses our overloaded + operator
25
26     -- Output the result
27     Ada.Text_IO.Put_Line("Result: " & C'Image);
28 end overloading_operators;
```

**Programa 2.3:** Overloading Operators in Ada