15. Parámetros o argumentos

Objetivos:
a) Profundizar en el concepto de parámetro de una clase e indicar su mecanismo de funcionamiento.
b) Interpretar el código fuente de una aplicación Java donde aparecen parámetros de distintos tipos
c) Construir una aplicación Java sencilla, convenientemente especificada, que declare métodos con diferentes tipos de parámetros

Los parámetros o argumentos son una forma de intercambiar información con el método. Pueden servir para introducir datos para ejecutar el método (entrada) o para obtener o modificar datos tras su ejecución (salida).

15.1. Declaración de parámetros

Los parámetros se declaran en la cabecera de la declaración de los métodos. Al declararse el parámetro, se indica el tipo de dato y el identificador correspondiente. Los parámetros o argumentos de un constructor o de un método pueden ser de cualquier tipo, ya sean tipos primitivos o referencias de objetos (en este caso debe indicarse el identificador de la clase correspondiente). Ejemplos de declaraciones de cabeceras de métodos:

```
public double devuelve()            // Sin parametros
    { . . . return . . . ;}
public void asigna(double x)        // Un parametro, x de tipo double
    {  . . .  }
public int elMayor(int a, int b)    // Dos parametros, a y b de tipo int
    {  . . .  }
public static double sumatorio (double [] v)   // Un parametro, v, array real
    {  . . .  }
public boolean caducado (Fecha fechaLimite)   // Un parámetro de la clase Fecha
    {  . . .  }
```

El identificador del parámetro se emplea sólo dentro del método para hacer referencia al argumento correspondiente y puede coincidir con el de un atributo de la misma clase. En tal caso, se dice que *oculta* a la variable miembro. Esta técnica suele emplearse en los constructores para inicializar una instancia. Por ejemplo en la clase `Circulo`:

```
public class Circulo {
    int x, y, radio;
    public circulo(int x, int y, int radio) {
        . . .
    }
}
```

La clase `Circulo` tiene tres atributos o variables miembro `x`, `y` y `radio` y un constructor con tres argumentos con los mismos identificadores que facilitan los valores iniciales a los atributos respectivamente. Los identificadores de los parámetros ocultan a las variables miembro dentro del cuerpo del constructor, de forma que `x`, `y` y `radio` dentro del constructor hacen referencia a los parámetros y no a las variables miembro. Para acceder a las variables miembro, es necesario emplear la palabra reservada `this` que referencia a la instancia que está siendo inicializada por el constructor.

```
public class Circulo {
    int x, y, radio;
    public Circulo(int x, int y, int radio) {
        this.x = x;           // this.x hace referencia al atributo x
                              // x hace referencia al parametro x
        this.y = y;           // this.y hace referencia al atributo y
                              // y hace referencia al parametro y
        this.radio = radio;   // this.radio hace referencia al atributo radio
                              // radio hace referencia al parametro radio
    }
}
```

Normalmente dentro del cuerpo del método de una clase puede hacerse referencia directa a las variables miembro de las instancias. Salvo en el caso del ejemplo anterior en el que las variables miembro están ocultas. Por otro lado, los parámetros de un mismo método no puede compartir el mismo identificador (no pueden coincidir) con el de una variable local. El siguiente código genera un error de compilación:

```
public void asigna(double x) {       // Un parametro, x de tipo double
    double x;                        // Error de compilacion
     . . .  }
```

15.2. Uso alternativo de this como referencia al parámetro implícito

En la clase `CuentaBancaria` se implementa un método `transferencia` que transfiere el saldo de una cuenta `origen` (parámetro formal explícito) a otra cuenta (parámetro implícito):

```
/**
 *  Declaracion de la clase CuentaBancaria
 *  Ejemplo de declaracion de variables
 *  metodos estaticos y uso de this
 *  @author A. Garcia-Beltran
 *  Ultima revision: abril, 2003
 */
public class CuentaBancaria {
    // Atributos o variables miembro
    private double saldo;
    public static int totalCuentas=0;
    // Metodos
    public CuentaBancaria( ) {
        this(0.0);
    }
    public CuentaBancaria( double ingreso ) {
        saldo = ingreso;
        incCuentas();
    }
    public double saldo() {
        return saldo;
    }
    public static void incCuentas () {
        totalCuentas++;
    }
    // Transfiere todo el dinero de la cuenta origen a la actual
    public void transferencia( CuentaBancaria origen ) {
        saldo += origen.saldo;
        origen.saldo=0;
    }
}
```

Ejemplo de programa que emplea la clase `CuentaBancaria` con una llamada al método `transferencia`:

```
/**
 *  Ejemplo de uso de la clase CuentaBancaria
 *  @author A. Garcia-Beltran
 *  Ultima revision: abril, 2003
 */
public class PruebaCuentaBancaria {
    public static void main (String [] args) {
        System.out.println("Total cuentas: " + CuentaBancaria.totalCuentas);
        CuentaBancaria c1;
        c1 = new CuentaBancaria(17.5);
        System.out.println("Nueva cuenta con: " + c1.saldo() + " euros");
        System.out.println("Total cuentas: " + CuentaBancaria.totalCuentas);
        CuentaBancaria c2;
        c2 = new CuentaBancaria(20.0);
        System.out.println("Nueva cuenta con: " + c2.saldo() + " euros");
        System.out.println("Total cuentas: " + CuentaBancaria.totalCuentas);
        System.out.println("Transferencia de cuenta 2 a cuenta 1");
        c1.transferencia(c2);
        System.out.println("Cuenta 1 con: " + c1.saldo() + " euros");
        System.out.println("Cuenta 2 con: " + c2.saldo() + " euros");
    }
}
```

La ejecución del código anterior origina la siguiente salida por pantalla:

```
$>java PruebaCuentaBancaria↵

Total cuentas: 0
Nueva cuenta con: 17.5 euros
Total cuentas: 1
Nueva cuenta con: 20.0 euros
Total cuentas: 2
Transferencia de cuenta 2 a cuenta 1
Cuenta 1 con: 37.5 euros
Cuenta 2 con: 0.0 euros
```

En principio, el programa funciona correctamente, pero el código del método `transferencia` puede dar lugar a que el saldo de las cuentas *origen* y *destino* se anule si ambas son la misma cuenta (la llamada al método sería `c1.transferenciaPeligrosa(c1);`). Se propone la siguiente modificación del código para que considere este caso (en el que no se realiza ninguna operación con ningún saldo).

```
public void transferenciaSegura( CuentaBancaria origen ) {
    if ( this == origen )   // Test de verificacion:  ¿destino igual a origen?
         return;
    saldo += origen.saldo;
    origen.saldo=0;
}
```

15.3. Paso por valor y paso por referencia

En Java todos los parámetros de los métodos se pasan *por valor*. Cuando se realiza la llamada a un método, los parámetros formales (parámetros indicados en la declaración) reservan un espacio en memoria independiente y reciben los valores de los parámetros reales (parámetros indicados en la llamada al método). ¿Qué consecuencias tiene el paso por valor de los parámetros?

- Cuando el argumento es de tipo primitivo, el paso por valor significa que durante la ejecución del método se reserva un nuevo espacio para el parámetro formal y no se puede modificar el parámetro real durante la ejecución del método.
- Cuando el argumento es de tipo referencia (por ejemplo, un *array*, un objeto,...) el paso por valor significa que no puede modificarse la referencia pero se pueden realizar llamadas a los métodos del objeto y modificar el valor asignado a las variables miembro accesibles del objeto durante la ejecución del método.

Por ejemplo, en el siguiente programa el método `cambiar` utiliza un parámetro de tipo primitivo (`a`, un valor numérico entero) y un parámetro de tipo referencia (`b`, un *array* de valores enteros):

```
/**
 *  Ejemplo de uso de parametros de distintos tipos
 *  @author A. Garcia-Beltran
 *  Ultima revision: abril, 2004
 */
public class Parametros {
    public static void main (String [] args ) {
       int n;
       int [] v = new int[2];
       n=10;
       v[0]=20;
       v[1]=30;
       System.out.println("Antes:   " + n + " " + v[0] +" "+ v[1]);
       cambiar(n, v);
       System.out.println("Despues: " + n + " " + v[0] +" "+ v[1]);
       }
    public static void cambiar (int a, int [] b) {
       a = 50;
       b[0] = 60;
       b[1] = 70;
       System.out.println("Dentro:  " + a + " " + b[0] +" "+ b[1]);
       }
}
```

La ejecución del código anterior origina la siguiente salida por pantalla:

```
$>java Parametros↵

Antes:   10 20 30
Dentro:  50 60 70
Despues: 10 60 70
```

Esto es a menudo una fuente de errores: el programador escribe un método que trata de modificar el valor de uno de sus parámetros y el método no funciona como esperaba.

15.4. Variables locales

En el cuerpo de un método pueden declararse variables adicionales que sólo pueden emplearse dentro del propio método. Estas variables *locales* pueden ser de un tipo primitivo o de un tipo referencia. Su existencia (reserva de espacio en memoria) sólo dura mientras se está ejecutando la llamada al método.

```
public static double sumatorio (double [] v) {
    int k;                          // Variable local entera
    double x=0;                     // Variable local real
```

```
for (int k=0; k<v.length; k++)
     x += v[k];
return x;
}
```

Una vez finalizada la ejecución del método, las variables locales k y x dejan de existir.