PROGRAMACIÓN (PRG). EUI. Curso 2001/2002

# *Tema 2. CONCEPTOS BÁSICOS*

DSIC.- F. Marqués y N. Prieto

## Índice General



## 1 Estructura elemental de un programa

Un programa es una secuencia de instrucciones, escritas en un cierto lenguaje de programación. El programa más sencillo en Java se implementa como una clase que tiene un único **bloque** principal llamado *main* que contiene, entre llaves, la secuencia de instrucciones a realizar. Los detalles sintácticos que aparecen en el esquema se estudiarán en temas posteriores.

```
class NombrePrograma
{
  public static void main (String args[])
  {
  ...
```

```
  }
}
```

Un programa cuyo efecto sea presentar una frase de saludo en pantalla, por ejemplo *"Hola a todos"*, sería el siguiente:

```
class Hola
{
   public static void main (String args[])
   {
      System.out.println ("Hola a todos");
   }
}
```

El bloque principal de este programa consta de una única instrucción que escribe en pantalla la secuencia de caracteres comprendidos entre las dobles comillas, *parámetro real*. Esta instrucción es la llamada a una operación o método (*println*) de la clase que representa la salida estándard (*System.out*).

## 2 Entrada y salida elemental

Aunque en un tema posterior se estudiará los detalles sobre cómo se realiza la entrada de datos y la salida de resultados en Java, por el momento se efectuará utilizando una *interfaz* desarrollada por los profesores de la asignatura denominada `nsIO`. Esta interfaz está configurada como un paquete ("`package`") de Java, y está disponible en la página web de la asignatura.

Esta interfaz permite utilizar **ficheros de texto** para realizar la entrada y la salida de una manera muy sencilla, para ello define dos clases:

- La clase **output** que permite preparar un fichero para escritura, en particular la pantalla (*output*), escribir valores numéricos, lógicos, caracteres, y secuencias o cadenas de caracteres (*write* y *writeln*), y cerrar el fichero cuando ya no se vaya a escribir más (*close*).
- La clase **input** que permite preparar un fichero para lectura, en particular desde el teclado (*input*), leer un caracter (*read*), un entero (*readint*), un real (*readdouble*), y otros valores numéricos, así como leer una palabra (*readword*), una línea (*readline*), cerrar un fichero cuando termina la lectura (*close*), y otras.

Para poder utilizar este paquete se escribirá al comienzo del fichero una instrucción especial para poder hacer uso (importar) las clases y operaciones definidas en dicho paquete. De la forma siguiente:

```
// instrucción para poder utilizar el paquete de entrada/salida nsIO
import nsIO.*;
```

El programa anterior utilizando esta interfaz quedaría:

```
import nsIO.*;
class Hola {
   public static void main (String args[]) {
      output fsal = new output();
      fsal.writeln ("Hola a todos");
      fsal.close();
   }
}
```

A continuación se va a presentar un ejemplo de un programa en el que se van mostrando ciertos mensajes al usuario y para pasar de uno a otro, el programa espera a que el usuario pulse alguna tecla:

```
import nsIO.*;
class Adivinanza {
   public static void main (String args[]) {
      output fsal = new output();
      input fent  = new input();
      int resultado;

      fsal.writeln ("Piensa un número...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 5...");
      fent.readln();
      fsal.writeln ("Súmale 6...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 4...");
      fent.readln();
      fsal.writeln ("Súmale 9...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 5...");
      fent.readln();
      fsal.writeln ("Escribe el resultado...");
      ...
      ...
      fsal.close();
      fent.close();
   }
}
```

# 3 Variables y asignación

En el programa *Adivinanza* se ha llegado al punto en el que el usuario debe introducir el resultado obtenido, y el programa debe "descubrir" cuál era el número que había pensado el usuario. Para ello, el programa debe de leer el número que el usuario introduce por teclado y necesita guardar este resultado final en algún "lugar" para poder ser capaz de recuperarlo y calcular el número original.

El computador mantiene los datos en dispositivos de memoria que están organizados en posiciones o celdas de memoria. Una posición de memoria, o varias consecutivas pueden almacenar un dato de valor simple, como por ejemplo un entero o un caracter. En un lenguaje de programación estas posiciones de memoria se manipulan mediante lo que se denominan **variables**. La descripción de una variable en un programa se llama **declaración de variable** y contiene el identificador de la variable y el tipo al que pertenece.

El **tipo** al que pertenece una variable:

- define el conjunto de valores que son susceptibles de ser almacenados en dicha variable
- y las operaciones que se pueden realizar con ella

La declaración de variables en Java se realiza especificando un tipo de datos simple seguido de una lista de identificadores válidos, separados por comas y finalizando con *;*. Se pueden inicializar las variables en el momento de su declaración, para ello se utiliza el símbolo de asignación =.

La **asignación** se utiliza para dar valores a las variables o reemplazar los valores que ya tienen por otros nuevos. Su sintaxis es:

```
Identificador = Expresión;
```

Una expresión está formada por operandos, que pueden ser constantes, variables o subexpresiones, y operadores, y tiene un tipo asignado. En una instrucción de asignación, los tipos de la variable y la expresión deben de ser compatibles.

Ejemplos de declaraciones de variables:

```
int var1, var2;
char ch1, ch2='u';
float f1=2.0; f2=3.0+f1;
```

Para almacenar el valor del resultado que introduce el usuario en el programa *Adivinanza*, se utilizará una variable que se puede llamar, por ejemplo *resultado*, que debe de ser de tipo entero. En Java se declarará escribiendo:

```
int resultado;
```

El siguiente paso que debe de realizar el programa es el de "adivinar" el número que ha pensado el usuario. Esto se puede hacer en dos pasos: restar 165 y dividir por 100. Por ejemplo si el resultado es 199865, el número pensado es 1997. Para hacer esto el programa utiliza una *expresión aritmética* (resultado-165)/100.

El programa *Adivinanza* completo quedaría:

```
import nsIO.*;
class Adivinanza {
   public static void main (String args[]) {
      output fsal = new output();
      input fent  = new input();
      int resultado;

      fsal.writeln ("Piensa un número...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 5...");
      fent.readln();
      fsal.writeln ("Súmale 6...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 4...");
      fent.readln();
      fsal.writeln ("Súmale 9...");
      fent.readln();
      fsal.writeln ("Multiplícalo por 5...");
      fent.readln();
      fsal.writeln ("Escribe el resultado...");
      resultado=fent.readint();
      fsal.write("El numero que habias pensado es: ");
      fsal.writeln((resultado-165)/100);
      fsal.close();
      fent.close();
   }
}
```

Obsérvese en este ejemplo cómo hay operaciones o métodos que devuelven un valor (*readint*) y otros que no (*writeln*), y que la forma de utilizarlos es diferente.

## 4 Algunas cuestiones sintácticas

Las palabras que se usan en un programa son *identificadores* y *palabras reservadas*:

**Los identificadores** son nombres que el programador da a variables, métodos y clases. Por ejemplo la palabra *Hola* en los primeros programas es un identificador, en este caso un identificador de clase. Un identificador en Java debe comenzar por

una letra y a continuación cualquier combinación de letras, números, el carácter subrayado _, y el signo de dólar $. Son identificadores válidos *fent*, *fsal*, *resultado*, *Adivinanza*, *Pi*, *x2*, ...

**Las palabras reservadas** tienen un significado fijo en el lenguaje de programación y no se pueden usar como identificadores. En la tabla siguiente se muestran las palabras reservadas de Java; a lo largo del curso se utilizarán muchas de ellas.

| | | | | |
|---|---|---|---|---|
| abstract | default | if | private | throw |
| boolean | do | implements | protected | throws |
| break | double | import | public | transient |
| byte | else | instanceof | return | try |
| case | extends | int | short | void |
| catch | final | interface | static | volatile |
| char | finally | long | super | while |
| class | float | native | switch | |
| const | for | new | synchronized | |
| continue | goto | package | this | |

La documentación necesaria para explicar las diferentes partes que componen un programa se puede incluir en el mismo a través de *comentarios*. Hay distintas maneras diferentes de poner comentarios en Java, se pueden utilizar los símbolos /* para abrir comentario y */ para cerrarlo. También se puede utilizar //, que hace que el compilador interprete todo lo que resta de la línea como comentario.

Por ejemplo:

```
class Hola {
   public static void main (String args[]) {
      /* Ahora va el saludo */
      System.out.println ("Hola a todos");
   }
}
```

O bien:

```
class Hola {
   public static void main (String args[]) {
      // Ahora va el saludo
      System.out.println ("Hola a todos");
   }
}
```

En Java existen caracteres especiales que tiene un papel de *separadores*. Son los siguientes:

**Paréntesis ()** : Contiene listas de parámetros en la definición y llamada a un método

**Llaves {}** : Sirven para englobar bloques de código y para valores iniciales de vectores

**Corchetes [ ]**: Sirven para la declaración de vectores y para hacer referencias a elementos de los mismos

**Punto y coma ;** : Separador de instrucciones

**Coma ,** : Separador de identificadores del mismo tipo en una declaración de variables, separación de argumentos en definición de métodos

**Punto .** : Separador de nombres de paquetes de subpaquetes y clases y para referenciar elementos de un objeto

# 5 Algunos Tipos elementales

## 5.1 Tipos numéricos

En Java se definen los tipos numéricos siguientes:

| NOMBRE | TAMAÑO EN BITS |
|---|---|
| byte | 8 |
| short | 16 |
| int | 32 |
| long | 64 |
| float | 32 |
| double | 64 |

Los cuatro primeros permiten respresentar números enteros de diferentes tamaños; los dos últimos permiten representar números reales.

Por ejemplo, la declaración

```
int suma=1;
```

introduce una variable entera cuyo identificador es *suma* y se le asigna el valor 1.

La declaración

```
int i;
```

introduce la variable entera $i$ sin inicializar a ningún valor concreto.

La instrucción de asignación

```
suma=suma+2;
```

sustituye el actual valor de *suma* con un valor que es el que resulta de evaluar la expresión *suma+2*.

## 5.2 Operadores aritméticos

Los operandos en una expresión pueden ser constantes, variables y subexpresiones (posiblemente incluídas entre paréntesis). En la tabla siguiente se presentan los operadores aritméticos y los correspondientes operadores reducidos.

| OPERADOR | NOMBRE OPERACIÓN | OPERADOR | NOMBRE OPERACIÓN |
|---|---|---|---|
| + | Suma | += | Suma y asignación |
| − | Resta | − = | Resta y asignación |
| ∗ | Multiplicación | ∗ = | Multiplicación y asignación |
| / | División | /= | División y asignación |
| % | Módulo | %= | Módulo y asignación |
| ++ | Incremento en 1 | −− | Decremento en 1 |

Por ejemplo, 217 segundos son 217/60 minutos y 217%60 segundos

Los operadores reducidos son equivalentes a una operación y una asignación donde uno de los operandos es el destino del resultado. Por ejemplo la instrucción `a+=b`, es equivalente a `a=a+b`.

Los operadores incremento y decremento producen la variación correspondiente en una unidad de la variable a la que afectan. Por ejemplo `a++` es equivalente a `a=a+1`

En Java se aplican las **reglas de precedencia de operadores** usuales: los paréntesis preceden a la división y a la multiplicación, que se ejecutan antes que la suma y la resta. Los grupos de precedencia para los operadores aritméticos son los siguientes:

```
grupo 0: ( )
grupo 1: ++  --  +(unario)  -(unario)
grupo 2: *  /  %
grupo 3: +   -
```

Dentro del mismo grupo la precedencia se establece de izquierda a derecha.

## 5.3 Tipos char y String

El **tipo char** permite representar caractéres como letras, números y caractéres especiales. Java utiliza la codificación *Unicode* de caracteres. *Unicode* define un conjunto de caracteres internacional codificados a 16 bits [1].

La declaración:

```
char c='s',d;
```

introduce dos variables de tipo caracter c y d. El valor inicial de la primera es la letra *s*.

El tratamiento de las cadenas de caracteres en Java es especial y se realiza a través de la clase **String**. La declaración:

[1] los primeros 256 carácteres de la tabla *Unicode* son los correspondientes en la tabla *ASCII* (código de 8 bits)

```
String nombre,palabra,linea;
```

introduce tres variables que pueden almacenar secuencias de caracteres. Aunque a lo largo del curso se introducirá más información sobre esta clase, algunas cosas que se pueden realizar ya son las siguientes:

Supongamos que se introduce por teclado la siguiente frase:

```
Hola mundo
```

Si se ejecuta la siguiente instrucción de lectura:

```
nombre=fent.word();
```

el contenido de la variable `nombre` sería `Hola`.

Si, por lo contrario, se desea leer toda la línea, se haría:

```
linea=fent.readline();
```

el contenido de la variable `linea` sería `Hola mundo`.

## 5.4 Constantes

Al igual que las variables, los valores constantes tienen un también un tipo de dato asignado.

**Enteras** Por defecto la base es decimal aunque también pueden representarse en octal y hexadecimal. Los valores octales van precedidos por un 0 y los hexadecimales por 0X o 0x. Por ejemplo, el número 27 se puede escribir como el literal entero 27, el octal 033 y el hexadecimal 0x1B. Las constantes enteras tienen por defecto el tipo `int`.

**Reales** Las constantes reales pueden expresarse en coma fija, por ejemplo 2.0, 3.1419 o .667. También pueden expresarse en notación científica, por ejemplo 6.022E23, 314159e-05. Por defecto generan un tipo `double`.

**Caracteres** La forma de expresar una constante de tipo *char* es incluyéndola entre comillas simples. Los caracteres no visibles se pueden introducir mediante una secuencia de escape, por ejemplo `'\n'` para el caracter nueva línea.

**Cadenas** Las constantes de tipo cadena se expresan entre comillas dobles, por ejemplo "Esto es una cadena".

## 5.5 Conversión de tipos

Cuando se realiza una instrucción de asignación:

```
Identificador = Expresion;
```

tanto la variable como la expresión deben de ser del mismo tipo o de tipos compatibles. Una expresión puede asignarse a una variable siempre que sea de un tipo compatible, generalmente "menor", que el tipo de la variable. La secuencia siguiente define la compatibilidad real entre tipos numéricos. Un valor perteneciente a cualquiera de los tipos que aparecen en la lista puede asignarse a otra variable perteneciente a alguno de los tipos situados en la lista a su derecha:

$$\text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double}$$

Otras formas de conversión de tipos se pueden realizar explícitamente a través de lo que se llama *casting* ( (tipo) expresion ), y también utilizando funciones adecuadas de ciertos paquetes estándard.

## 5.6 Algunas funciones predefinidas

La clase *Math* define métodos o funciones para calcular funciones trigonométricas, redondear valores reales, y otros cálculos.

Para poder referenciar dichas operaciones se debe escribir:

Math.NombreMétodo(..);

Algunas constantes y métodos útiles son:

1. Las constantes *E* y *PI*. Math.E=2.7182818284590452354, Math.PI=3.14.159265358979323846.
2. Las funciones de redondeo, con x de tipo *double*:
   - *ceil(x)* : devuelve el número entero más pequeño que es mayor o igual a x. Ejemplo: double x=879.327; double ceilx=Math.ceil(x); // ceilx=880
   - *floor(x)* : devuelve el número entero más grande que es menor o igual a x. Ejemplo: double x=879.327; double floorx=Math.floor(x); // floorx=879
   - *round(x)* : convierte el real x al entero más próximo. Ejemplo: int redondeo=Math.round(6243.45) // 6243; int redondeo=Math.round(6243.65); // 6244
3. Funciones trigonométricas:
   - *sin(x)* : calcula el seno del ángulo (en radianes) x
   - *cos(x)* : calcula el coseno del ángulo (en radianes) x
   - *asin(x)* : calcula el arco seno del ángulo x (x entre -1 y 1)
   - *acos(x)* : calcula el arco coseno del ángulo x (x entre -1 y 1)
   - *atan(x)* : calcula el arco tangente del ángulo x
4. Otras funciones:
   - *abs(x)* : calcula el valor absoluto de x (entero o real)

- *exp(x)* : calcula e elevado a x (x es real)
- *log(x)* : calcula el logaritmo natural de x (x real y no negativo)
- *max(x,y)* : compara los números x e y (enteros o reales) y devuelve el mayor
- *min(x,y)* : compara los números x e y (enteros o reales) y devuelve el menor
- *pow(x,y)* : calcula x elevado a y. No está definida si x es negativo o 0 e y no es entero, ni tampoco si x=0 e y es negativo o 0
- *random()* : genera un número (pseudo)aleatorio entre 0.0 y 1.0
- *sqrt(x)* : calcula la raix cuadrada positiva de x (x no negativo)

# 6 Bloques de instrucciones

Como se ha visto en un programa en Java las instrucciones aparecen consecutivamente, utilizándose el símbolo `;` como terminador de cualquiera de ellas. A dicho tipo de composición de instrucciones se le denomina *composición secuencial*, y es característico de la mayoría de los lenguajes de programación.

Adicionalmente, algunas partes de un programa en Java consisten en un grupo de declaraciones seguidas de instrucciones, estando todo englobado entre llaves. Dicha organización recibe en Java el nombre de *bloque*. Un *bloque* es, por lo tanto, una parte de un programa en Java englobada entre llaves y formada por declaraciones iniciales y una secuencia de instrucciones (posterior a las declaraciones).

El siguiente es un ejemplo de un bloque inscrito en un método `main` ya visto:

```
{
    output fsal = new output();
    fsal.writeln ("Hola a todos");
    fsal.close();
}
```

Además del lenguaje Java existen otros lenguajes con características similares en cuanto a su organización utilizando *bloques*. Estos lenguajes reciben el nombre de *lenguajes orientados a bloques*.

Una de las ventajas del uso de los bloques radica en el hecho de que allí donde el lenguaje permite el uso de una instrucción simple (por ej., una asignación seguida de ";") es posible también utilizar un bloque (pudiéndose así escribir una secuencia de instrucciones). Esta característica será muy utilizada más adelante para agrupar instrucciones.

Los bloques pueden declararse o *anidarse* unos dentro de otros, de forma que se habla de *bloques externos* (porque contienen a otros) o *internos* (cuando están contenidos dentro de otros). Las declaraciones efectuadas al comienzo de un bloque tienen sólo validez en el interior del mismo (incluso en bloques internos), fuera del bloque no surten efecto. Dentro de un bloque se pueden utilizar tanto los identificadores definidos en el mismo, como en cualquier otro bloque externo que lo comprenda. En distintos bloques

anidados no se pueden definir identificadores con el mismo nombre, en caso de hacerse así se generará un error durante la compilación. El siguiente es un ejemplo de bloques anidados:

```
{ int dia=10, mes=12, anyo=2000;
  {
   double temperatura;
   dia = 12;
   temperatura = 36.8;
   System.out.println(dia);
   System.out.println(temperatura);
  }
  System.out.println(dia);
  // la var. temperatura no se puede referenciar aquí
}
```

Se denomina *ámbito* de una variable a la parte del programa en la que dicha variable es conocida y puede ser utilizada. Una variable se dice que es *local* en el bloque en el que se define y *global* para para los bloques internos a éste. Las siguientes reglas están relacionadas con el concepto de bloque y el uso de variables:

- Todas las variables definidas en el mismo bloque deben tener nombres diferentes.
- Una variable definida en un bloque es conocida desde esta definición hasta el final del bloque. Como caso particular, una variable definida en un bloque es conocida en todos los bloques internos a éste.
- Las variables se deben definir al comienzo del bloque más interno en el que se utilizan.

# 7 Ejercicios

1. Escribir un programa en Java que pregunte *Cómo te llamas?*, lea el nombre que introduces por teclado, por ejemplo Albert (acabando con NL) y escriba en pantalla *Hola Albert*.
2. Escribir un programa en Java que pregunte tu nombre, dirección y teléfono y escriba en pantalla una ficha.
3. Escribir un programa en Java que transforme en euros la cantidad que se introduce como dato en pesetas (1 euro son 166.386 pesetas).
4. Escribir un programa en Java que transforme una temperatura en grados Fahrenheit a grados Celsius (1$^o$C=33.8$^o$F).
5. Escribir un programa en Java para calcular la superficie y el volumen de una esfera a partir del valor del radio (supóngase que es un valor positivo).

6. Escribir un programa en Java para calcular el volumen de una esfera a partir del valor de su superficie (supóngase que es un valor positivo).

7. Una empresa de transporte por carretera ha adquirido vehículos nuevos que viajan más rápido que los antiguos. Les gustaría conocer cómo afectará ésto a la duración de los viajes. Supóngase que la reducción media que se consigue del tiempo total de viaje es del 15%. Escribir un programa en Java que lea el horario de salida y llegada antiguo, calcule el nuevo horario de llegada y muestre en pantalla el nuevo tiempo de viaje y la nueva hora de llegada.