por Nacho Iborra

Entornos de Desarrollo
Bloque 2
Tema 1: Estructura básica de un programa en Java. I/O. Tipos básicos de datos. Operadores.

2.1.1. Introducción a Java

Java es un lenguaje de programación orientado a objetos creado por Sun Microsystems a mediados de los 90. Su propósito principal es implementar programas capaces funcionar en distintas plataformas, como en diferentes sistemas operativos, e incluso en algunos electrodomésticos (frigoríficos, lavadoras...) Actualmente el lenguaje lo mantiene Oracle.

Como hemos visto en temas anteriores. Java compila el código fuente mediante un lenguaje intermedio que se puede ejecutar en una **Máquina virtual de Java**. De este forma los programas de Java son multiplataforma ya que sólo se necesita instalar esta máquina virtual en nuestro sistema operativo.

2.1.1.1. Software necesario

Para desarrollar aplicaciones Java se necesita instalar el kit de desarrollo de Java *Java JDK*. Puedes descargarlo en la web de Oracle Java. Además se necesita un IDE para crear los proyectos Java. Algunos posibles son:

- Un IDE simple y ligero como **Geany**, configurable para trabajar con JDK y usar el compilador correspondiente.
- Un IDE específico como **Eclipse** o **NetBeans**, que están asociados a JDK y nos permiten crear proyectos más complejos con varios fuentes, librerías etc.

En los primeros temas de este bloque (hasta que aprendamos algo sobre clases), usaremos Geany en vez de Eclipse o NetBeans, porque resulta más sencillo crear programas Java con solo un fuente. Más adelante, utilizaremos Eclipse o NetBeans para trabajar con proyectos con varias clases y fuentes.

2.1.2. Primeros pasos con Java

2.1.2.1. Nuestro primer programa en Java

Vamos a comenzar con Java creando un progrma sencillo que escriba por pantalla "Hello".

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

La estructura de este programa es muy similar al equivalente en C#: siempre necesitaremos definir una clase, aunque solo vayamos a utilizar la función `main`. Esta función `main` se escribe **en minúsculas** en Java.

Además, cada clase pública debe tener el mismo nombre que el fuente en la que está contenida, por tanto el ejemplo anterior se guardará como `MyClass.java` (los fuentes Java tienen la extensión `.java`).

2.1.2.2. Compilar y ejecutar un programa

Para compilar en Geany deberemos usar el compilador interno `javac` que creará un fichero llamado `MyClass.clas` en la carpeta donde está el fuente. Este será el fichero compilado que se ejecutará en la máquina virtual de Java. Por tanto, cuando ejecutemos el programa, Geany usará la herramienta `java` para ejecutarlo y sacar el mensaje "Hello" por la pantalla.

2.1.3. Variables y tipos de datos básicos

Para trabajar con tipos de datos básicos como enteros, coma flotante, caracteres, booleanos y cadenas, tenemos en Java los siguiente tipos de datos:

- `byte`, `short`, `int`, `long` para números enteros
- `float` y `double` para números reales
- `char` para caracteres
- `boolean` para booleanos. Sus valores serán `true` o `false`.
- `String` para cadenas. (este tipo comienza en mayúsculas en Java)

No hay modificadores de signo, como hay en C++ o C#. El rango de datos de estos tipos son más o menos los mismos que en otros lenguajes.

2.1.3.1. Declaración de variables

Las variables en Java se declaran como en otros lenguajes como C#: se especifica el tipo de dato y el nombre. También se pueden declarar más de una variable del mismo tipo en una línea separadas por comas así como asignarles valores iniciales en su declaración:

```
int number = 0, result = 1;
char symbol;
String name = "Pepe";
```

Las variables se pueden declarar en cualquier parte del código, pero es una buena práctica agrupar las declaraciones al principio de la función donde se usan. (en el `main` en estos primeros ejemplos).

2.1.3.2. Valores por defecto

Si no se le asigna ningún valor a una variable, cogerá un valor por defecto en función del tipo que sea. Para datos numéricos (enteros o reales), el valor seá `0`. Para los caracteres, el valor por defecto será el código `'\u0000'`. Para las cadenas será `null` y para los booleanos `false`.

Sin embargo, no es una buena práctica dejar que los valores por defecto en nuestros programas sean asignados por el compilador, es mejor asignar estos valores nosotros mismos.

2.1.3.3. Algunas conversiones útiles

Existen algunas conversiones de tipo que son muy usuales en las aplicaciones Java:

- Si se quiere convertir entre tipos numéricos, solo es necesario realizar un cast. Por ejemplo si se quiere convertir de `float` a `int`:

```
float pi = 3.1416;
int piInteger = (int)pi;
```

- Si se quiere convertir de cadena a entero o float o double, hay algunos métodos útiles en las clases `Integer`, `Float` o `Double`:

```
int number = Integer.parseInt("42");
float floatNumber = Float.parseFloat("3.1416");
double doubleNumber = Double.parseDouble("3.141592654");
```

- Si necesitamos convertir de numérico a cadena resulta bastante sencillo ya que bastaría con concatenar una cadena vacía `""` con el correspondiente valor numérico.

```
int number = 23;
String numberString = "" + number;
```

2.1.4. Operaciones y operadores

Java tiene más o menos el mismo conjunto de operadores que podemos encontrar en muchos otros lenguajes de programación:

- **Operadores aritméticos**: `+`, `-`, `*`, `/`, `%` (módulo), `++` (incremento), `--` (decremento)
- **Asignaciones**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- **Comparaciones**: `>`, `>=`, `<`, `<=`, `==`, `!=`

> **NOTA:** la comparación `==` NO FUNCIONA con cadenas. Se debe usar el método `equals` (`string1.equals(string2)`)

- **Lógicos**: `&&` (AND), `||` (OR), `!` (NOT)

2.1.5. Entrada / Salida Básica

Veamos como mostrar información por la pantalla y pedir datos al usuario.

2.1.5.1. Entrada básica: Scanner

Para que el usuario introduzca datos, la forma más fácil es mediante el objeto `Scanner`. Se necesitará importar `java.util.Scanner` para poder usarlo. Debemos crear un elemento Scanner y llamar a sus métodos para leer datos introducidos por el usuario. Algunos de estos métodos son: `nextLine` (para leer una línea de texto hasta que se pulse Enter) y `nextInt` (para leer explícitamente un entero).

```
import java.util.Scanner;
...
public class ClassName
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                int number = sc.nextInt();
                String text = sc.nextLine();
                sc.close();
        }
```

Hay otros muchos otros métodos, como `nextFloat`, `nextBoolean`... que son muy similares a `nextInt`, y nos permiten leer datos específicos, en vez de leer texto y convertirlo posteriormente al tipo correspondiente (como hacíamos con `Console.ReadLine` en C#). Estos datos pueden ser introducidos separados por espacios o por saltos de línea (Intro).

```
int number1, number2;
// These two numbers can be entered either separated by whitespaces
// or by new lines
number1 = sc.nextInt();
number2 = sc.nextInt();
```

Especial cuidado con la combinación de tipos de datos

Supongamos que se quiere leer esta información por teclado:

```
23 43
Hello world
```

Se podría pensar en usar el método `nextInt` dos veces, y después `nextLine` para leer la cadena del final, pero esto NO sería correcto. Cuando se usa `nextInt` para leer valores enteros, no se lee el fin de línea que hay tras el número 43, por tanto, al usar el método `nextLine` una vez, lo que se lee es este fin de línea y no la segunda línea de texto. La forma correcta de hacerlo sería:

```
int number1 = sc.nextInt();
int number2 = sc.nextInt();
sc.nextLine(); // Read and ignore end of first line
String text = sc.nextLine();
```

2.1.5.2. Salida básica: System.out.print(ln)

Para mostrar datos por pantalla se pueden usar las instrucciones `System.out.print` o `System.out.println` (dependiendo de si se quiere un salto de línea o no). Se pueden concatenar varios valores usando el operador (+):

```
int result = 12;
System.out.println("The result is " + result);
```

2.1.5.3. Salida formateada

Aparte de la instrucción tradicional `System.out.println` para mostrar por pantalla, se pueden usar otras opciones para darle a esta salida un formato concreto. Para ello, se usa `System.out.printf`. Esta instrucción se comporta de forma similar al `printf` original de C. Tiene un número variable de parámetros y en el primero debe ir la cadena a sacar por pantalla, y dentro de esta cadena se pueden

incluir una serie de caracteres especiales que determinarán el tipo de dato por el que se sustiuirán en la salida por pantalla. Por ejemplo:

```
System.out.printf("The number is %d", number);
```

En esta instrucción el símbolo `%d` será reemplazado por la variable `number`, y esta variable debe ser de tipo entero. (que es lo que significa `%d`).

Existen otros símbolos para representar diferentes tipos de datos. Aquí puedes encontrar algunos de ellos:

- `%d` para tipos enteros (`long`, `int`)
- `%f` para números reales (`float` y `double`)
- `%n` para representarl una nueva línea (similar a `\n`, pero independiente de la plataforma). En este caso, no necesitamos añadir ningún parámetro.

Se pueden reemplazar tantos símbolos como se quiera dentro de una cadena de salida, siempre añadiendo el número correspondiente de parámetros en la instrucción `printf`. Por ejemplo:

```
System.out.printf("The average of %d and %d is %f",
number1, number2, average);
```

Además a los símbolos primarios `%d` y `%f` se les puede añadir otra información entre el '%' y la letra para especificar información de formato.

Especificar dígitos enteros

Por ejemplo, si se quiere sacar por pantalla un entero con un número de dígitos determinado, podemos hacer así:

```
System.out.printf("The number is %05d", number);
```

donde `05` significa que el entero va a tener, al menos, 5 dígitos, y si no tuviera suficientes dígitos, entoces se deberá rellenar con ceros. La salida de esta instrucción si el número fuera `33` sería `The number is 00033`. Si no se pone el `0`, entoces el número se rellenará con espacios en blanco. Por tanto esta instrucción:

```
System.out.printf("The number is %10d", number);
```

si el número fuera `33`, produciría la siguiente salida: `The number is         33`.

Especificar dígitos decimales

De la misma forma que se formatean números enteros, se pueden formatear números reales. Se usa el mismo patrón anterior para la parte entera:

```
System.out.printf("The number is %3f", number);
```

Pero, además, se puede especificar el número de decimales añadiendo un punto y el número de decimales deseados, de la siguiente manera:

```
System.out.printf("The number is %3.3f", number);
```

De esta forma, si el número es `3.14159`, la salida será: `The number is   3.142`.

2.1.6. Declaración de constantes

Las constantes se declaran en Java declarando el dato como `final` y `static` (aprenderemos el significado de estas palabras más adelante). Normalmente estas constantes se colocan al principio de la clase. Se pueden declarar `public`, `protected` o `private`, según qué clases vayan a acceder a estos valores (aprenderemos también más sobre el uso de estos modificadores).

```
class MyClass
{
        public static final int MAX_USERS = 10;
    ...
}
```

Ejercicios propuestos:

2.1.6.1. Crea un programa llamado *FormattedDate* con una clase con el mismo nombre dentro. El programa pedirá al usuario que introduzca el día, el mes y el año de su nacimiento (todos los valores son enteros). Entonces, se sacará por pantalla la fecha de nacimiento con el siguiente formato *d/m/a*. Por ejemplo, si el usuario teclea día = 7, mes =11, año =1990, el programa sacará *7/11/1990*.

2.1.6.2. Crea un programa llamado *GramOunceConverter* que trasforme de gramos a libras. El programa pedirá al usuario que introduzca un peso en gramos (un número entero), y se mostrará el correspondiente peso en libras (número real). 1 libra = 28,3495 gramos.

2.1.6.3. Crea un programa llamado *NumbersStrings*. Este programa debe pedir al usuario que introduzca 4 números que guardará en 4 variables `String`. Entonces, el programa concatenará el

primer par de números formando un número entero. Y concatenará el segundo par de números para formar otro entero. Después de esto se sumarán los dos números resultantes y se sacará el resultado de la suma. Por ejemplo si se introduce 23,11,45 y 122, entonces el programa creará el 2311 y el 45112. Y la suma resultará 47423.

2.1.6.4. Crea un programa llamado *CircleArea* que defina una constante float llamada `PI` con el valor `3.14159`. El programa pedirá al usuario que introduzca el radio de un círculo y se sacará por pantalla el área de dicho círculo (`PI` * radio *radio). Este área se mostrará por pantalla con 2 decimales.