UD02 – UTILIZACIÓN DE OBJETOS

Contenido

1 Objetos

Java es un lenguaje orientado a objetos. No abordaremos la programación orientada a objetos hasta más avanzado el curso, pero resulta conveniente ir avanzando algunos conceptos que nos ayudarán a comprender mejor como funcionarán los programas que vamos a ir desarrollando.

La programación orientada a objetos (OOP) es una metodología de lenguaje de programación en la que los programas se organizan en torno a datos u objetos que tratan de modelar entidades del mundo real. Un objeto es una abstracción de cualquier cosa que puedas pensar o ver en este instante, un perro, una silla, una pelota, un misil teledirigido, un enemigo en un videojuego, etc.

Las bases fundamentales de la OOP, compartidas por los diferentes lenguajes orientados a objetos, son las siguientes:

- Todo es un objeto, con una identidad propia.
- Un programa es un conjunto de objetos que interactúan entre ellos.
- Un objeto puede estar formado por otros objetos más simples.
- Cada objeto pertenece a un tipo concreto: una clase.
- Objetos del mismo tipo tienen un comportamiento idéntico.

Objeto: entidad *existente en la memoria del ordenador* que tiene unas propiedades, atributos o datos sobre sí mismo almacenados por el objeto a las que nos referimos de manera genérica

como **estado**, unas operaciones disponibles específicas o métodos que define su **comportamiento** y una **identidad**, que los distinguen de cualquier otro objeto.

Clase: abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener.

Decimos que un objeto es una instancia de una clase.

Vamos a tratar de visualizar estos conceptos con un ejemplo.

```
// Ejemplo objetos tipo String
public class Cadenas {

	public static void main(String[] args) {
		// Declaramos variables de tipo String
		String hola = "Hola,";
		String adios = "Adiós,";
		String mundo = new String("mundo!");
		String mensaje = new String(hola.toUpperCase() + mundo.toUpperCase());

		// Mostramos por pantalla estos textos en mayúsculas
		System.out.println(mensaje);
		System.out.println(adios.toUpperCase() + mundo.adios.toUpperCase());

		//Ejemplos de instanciación
		String ejemplo = "Ejemplo";
		String ejemplo1 = "Ejemplo";
		String ejemplo2 = new String("Ejemplo");
		String ejemplo3 = new String(ejemplo);
		String ejemploDoble = ejemplo + " doble";

		//¿Cuando son dos objetos iguales?
		System.out.println("Operador ==");
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo1 + "\" :" + (ejemplo
== ejemplo1));
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo2 + "\" :" + (ejemplo
== ejemplo2));
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo3 + "\" :" + (ejemplo
== ejemplo3));

		//Utilización del método equals
		System.out.println("Método equals de la clase String");
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo1 + "\" :" +
(ejemplo.equals(ejemplo1)));
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo2 + "\" :" +
(ejemplo.equals(ejemplo2)));
		System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo3 + "\" :" +
(ejemplo.equals(ejemplo3)));
	}

}
```

Ya hemos visto el tipo de dato *String*, en realidad *String* es una clase definida en Java que nos permite representar texto. Las clases en Java son consideradas como tipos de datos compuestos y por lo tanto podemos declarar variables de estos tipos. Esto es lo que ocurre en el programa, declaramos la variable hola de tipo *String* y le damos el valor “Hola,”. Dicho de otro modo, declaramos el objeto hola de la clase *String* con el valor “Hola,”.

Cada una de las variables de tipo *String* que hemos definido son objetos diferentes con diferentes estados (valores) pero con el mismo comportamiento definido por la clase *String*,

así a todas las variables podemos pedirles que nos den la cadena en mayúsculas o minúsculas, por ejemplo.

1.1 Características de los objetos

Una instancia es un elemento tangible (ocupa memoria durante la ejecución del programa) generado a partir de una definición de clase. Todos los objetos empleados en un programa han de pertenecer a una clase determinada.

Para saber más...

Un puntero es una variable que contiene la dirección de memoria de un dato o de otra variable. Esto quiere decir, que el puntero apunta al espacio físico donde está el dato o la variable. Un puntero puede apuntar a un objeto de cualquier tipo. En lenguaje C es posible que el programador manipule por programa punteros, este mecanismo resulta ser muy potente al poder acceder directamente a elementos físicos del ordenador, pero difícil de controlar y poco seguro en caso de errores en la programación si se accede o cambia un dato en una dirección de memoria que no es la que realmente era deseada. En Java no hay posibilidad de acceso a punteros directamente por parte del programador por lo que se considera el desarrollo más seguro en cuanto a gestión de memoria.

Aunque el término a veces se emplea de una forma imprecisa, un objeto es una *instancia* de una clase predefinida en Java o declarada por el usuario y *referenciada* por una variable que almacena su dirección de memoria. Cuando se dice que Java *no tiene punteros* simplemente se indica que Java no tiene punteros que el programador pueda *manipular*, ya que todas las referencias a objeto son de hecho punteros en la representación interna.

Las 3 características principales de un objeto son:

- **Identidad**. Se refiere a la cualidad que identifica de manera unívoca a un objeto. Un objeto ocupa una dirección en memoria. Para Java dos objetos serán diferentes si están guardados en distintas direcciones de memoria.
- **Estado**. Se corresponde con el valor actual que tienen los atributos del objeto.
- **Comportamiento**. Se refiere a las operaciones disponibles específicas o métodos. Un objeto responde al comportamiento definido por las operaciones de la clase a la que pertenece. Esto es a los métodos definidos en la clase a la que pertenece.

1.2 Utilización de objetos

Veamos en detalle la utilización de objetos en el programa anterior

Para poder utilizar un objeto hay que declararlo, inicializarlo e instanciarlo. La declaración de un objeto es igual a la declaración de una variable de tipo primitivo:

```
identificadorTipo identificadorVariable;
```

En el programa encontramos la declaración del objeto de tipo *String mundo*.

```
String mundo = new String("mundo!");
```

La inicialización se lleva a cabo con el operador de asignación (**=**)

1.2.1 Instanciación de objetos

La manera más habitual de instanciar un objeto es mediante la palabra reservada ***new*** seguida de nombre de la clase y los parámetros de inicialización (si los tiene) entre paréntesis.

Así en nuestro ejemplo:

```
String mundo = new String("mundo!");
```

Encontramos que la misma línea de código incluye declaración, inicialización e instanciación:

Declaración:

```
String mundo = new String("mundo!");
```

Inicialización:

```
String mundo = new String("mundo!");
```

Instaciación:

```
String mundo = new String("mundo!");
```

De manera que pasamos como parámetro el literal *String* con el que queremos dar valor a nuestra nueva variable.

El tipo *String* es un tipo especial ya que, sin ser un tipo primitivo, está incluido en el sistema y es uno de los más usados al representar texto. Por ello suele tener en los lenguajes un tratamiento especial. Hemos visto que tenemos literales para este tipo de dato y además podemos inicializar una variable de tipo *String* asignando directamente a una variable un literal sin necesidad de utilizar el mecanismo general de instanciación que acabamos de ver.

```
String hola = "Hola,";
```

También que podemos crear nuevas variables de tipo *String* a partir de cualquier expresión de tipo *String*, las siguientes fórmulas son válidas:

```
String hola = "Hola,";
String hola1 = hola;
String hola2 = new String(hola);
String hola3 = hola + hola1;
```

En la primera se instancia una variable de tipo *String* utilizando un literal, en la segunda se le asigna un objeto ya existente, en la tercera se crea una nueva instancia que contiene el mismo texto que una variable ya existente y en la cuarta se crea una instancia con el valor resultante de la expresión que concatena dos variables ya existentes.

1.2.2 Utilización de métodos

Una vez que tenemos un objeto creado, es decir, una vez que en nuestro código hay una declaración, una inicialización y una instanciación de un objeto ya podemos utilizar ese objeto haciendo referencia a su identificador. Por ejemplo, la siguiente en la siguiente sentencia mostraría por consola el texto guardado en la variable *hola* convertido en mayúsculas:

```
System.out.println(hola.toUpperCase());
```

En este caso estamos llamando al método *toUpperCase()* de la clase *String*. Dicho de otro modo, estamos enviando un mensaje al objeto hola para que ejecute el método *toUpperCase()*, entonces el objeto, que implementa el comportamiento de la clase *String*, devolverá el resultado de transformar a mayúsculas los caracteres de la cadena que contiene.

Ojo, no modifica el texto que contiene el objeto, sino que crea una nueva cadena de caracteres y la devuelve como resultado de la llamada al objeto. Después, como ya hemos visto, esta cadena de caracteres que es devuelta se muestra por consola.

De manera similar el objeto *System.out* recibe el mensaje de ejecutar el método *println* y en respuesta muestra el texto que recibe por la consola.

Los métodos disponibles en un objeto determinan su comportamiento y ejecutarán las acciones correspondientes. El término método se utiliza de manera general en programación orientada a objetos, también podemos referirnos a ellos como funciones o procedimientos.

1.2.3 Parámetros y valores devueltos

En el ejemplo que acabamos de ver se hacían llamadas a dos métodos. El primero al método *toUpperCase* de la clase *String*, en este caso no se pasa ningún valor como parámetro de entrada y sí que devuelve un valor como resultado de la ejecución. En el segundo caso, el método *println* sí que recibe un valor como parámetro de entrada, el texto que debe imprimir, y en cambio no devuelve ningún valor de retorno. Su ejecución en cambio imprime por consola el texto recibido.

De manera general los métodos podrán recibir parámetros o no y podrán devolver o no un valor de retorno como resultado de la ejecución. Si se devuelve un valor de retorno hablamos de funciones y en caso contrario de procedimientos. De manera general, en OOP, se utiliza el término método que engloba ambos conceptos.

1.2.4 Utilización de propiedades

Las propiedades son atributos de los objetos, estos atributos pueden estar disponibles para ser usados por el programa. Siguiendo con la misma sentencia de ejemplo, out es un atributo de la clase System que representa la salida estándar del sistema; la consola. Podemos acceder a está haciendo referencia a su identificador y a través de este a los métodos que posee.

```
System.out.println(hola.toUpperCase());
```

Este es un ejemplo también de objeto dentro de un objeto; el objeto *out* está incluido como propiedad en el objeto *System*.

Se dice que las propiedades definen el estado del objeto.

1.2.5 Utilización de métodos estáticos

Podemos observar también una diferencia entre el objeto *System* y los objetos hola o mundo de la clase *String*. Para utilizar los objetos de la clase *String* debíamos declarar, inicializar e instanciar el objeto primero, pero podemos utilizar las propiedades de la clase *System* y llamar a los métodos de sus objetos directamente, sin hacer ninguna instanciación.

System es una clase que no se puede instanciar y provee varios atributos o propiedades que proporcionan funcionalidades útiles para el programador referentes al entorno. A los métodos o propiedades que son accesibles directamente a través de la clase sin instanciarla se denominan métodos estáticos. Decimos entonces que *System* sólo tiene parte estática, veremos más adelante que se pueden definir comportamientos estáticos (de clase) y de instancia para la misma clase.

Estos conceptos son conceptos avanzados que irán viendo a lo largo del curso y por ahora no entraremos en más detalles. Por ahora es suficiente con ir acostumbrándonos a su uso y su

terminología; si un método es accesible directamente desde la clase se denomina estático o de clase, en otro caso se denomina método de instancia.

1.2.6 Destrucción de objetos y liberación de memoria

Cuando no necesitamos utilizar un objeto en nuestro código porque ya ha realizado su función podemos hacer una asignación del identificador a *null*, es decir que no apunte a ninguna dirección de memoria. Cuando esto ocurre la memoria podrá ser liberada.

Esto no ocurre de manera automática, sino que hay un proceso en background en la máquina virtual de Java llamado *GarbageCollector* que se encarga de liberar la memoria de los objetos que ya no se usan, tanto porque hayamos hecho una asignación a *null* como porque el ámbito de definición de la variable haya terminado su ejecución.

1.3 Operador "es igual" aplicado a objetos

Vimos en el tema anterior que podíamos usar el operador **==** para verificar si dos variables son iguales. Para los tipos de datos primitivos esto significa que tienen el mismo valor. En cambio, para los objetos, **el operador == devuelve true si los dos objetos que estamos comparando son el mismo, es decir apuntan a la misma dirección de memoria**.

Observa el resultado de las siguientes líneas de código

```
//Ejemplos de instanciación
String ejemplo = "Ejemplo";
String ejemplo1 = "Ejemplo";
String ejemplo2 = new String("Ejemplo");
String ejemplo3 = new String(ejemplo);
String ejemploDoble = ejemplo + " doble";

//¿Cuando son dos objetos iguales?
System.out.println("Operador ==");

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo1 + "\" :" +
(ejemplo == ejemplo1));

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo2 + "\" :" +
(ejemplo == ejemplo2));

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo3 + "\" :" +
(ejemplo == ejemplo3));
```

Salida:

```
Operador ==
"Ejemplo" es igual a "Ejemplo" :true
"Ejemplo" es igual a "Ejemplo" :false
"Ejemplo" es igual a "Ejemplo" :false
```

En este caso ocurre que la variable ejemplo se inicializa con una *String* que se instancia internamente a través de su literal, la variable ejemplo1 utiliza la misma instancia creada en la línea anterior. En cambio, las variables ejemplo2 y ejemplo3 se inicializan creando sus propias instancias (*new*) y tomando como valores para su texto el texto de las instancias recibidas como parámetros, es decir copian el texto que reciben a su propia dirección de memoria, pero no usan la dirección de memoria que reciben para sí mismas.

Este comportamiento puede estar alejado de lo que esperamos, pero tenemos que acostumbrarnos a él y es general en el manejo de objetos.

¿Cómo validar entonces si dos cadenas de texto contienen el mismo texto? Para esto utilizaremos el método *equals* de la clase *String*. Veámoslo en el ejemplo considerando los mismos objetos que en el ejemplo anterior:

```
//Utilización del método equals
System.out.println("Método equals de la clase String");

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo1 + "\" :" +
(ejemplo.equals(ejemplo1)));

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo2 + "\" :" +
(ejemplo.equals(ejemplo2)));

System.out.println("\"" + ejemplo + "\" es igual a \"" + ejemplo3 + "\" :" +
(ejemplo.equals(ejemplo3)));
```

En este caso la salida será:

```
Método equals de la clase String
"Ejemplo" es igual a "Ejemplo" :true
"Ejemplo" es igual a "Ejemplo" :true
"Ejemplo" es igual a "Ejemplo" :true
```

1.4 Librerías de objetos

Vimos en la unidad anterior que cuando desarrollamos código en Java podemos utilizamos la palabra clave *Package* para agrupar hacer que la clase que construimos pertenezca a un determinado paquete. Veremos más adelante como usar este mecanismo y las ventajas que ofrece, por ahora nos quedamos con que es un mecanismo que nos permite agrupar clases que tienen características comunes.

Cuando queremos utilizar las clases contenidas en un paquete podemos hacer lo importándolas primero, para ello utilizamos la palabra reservada *import*:

```
import packageName.ClassName;
```

Para importar una clase de un paquete.

```
import packageName.*;
```

Para importar todas las clases de un paquete.

Por ejemplo puede resultar conveniente importar la clase *java.lang.Math* si necesitamos realizar operaciones matemáticas. Iremos viendo a lo largo del curso estas y otras clases y paquetes que forman parte de Java.

2 Entrada simple de datos en Java

Vimos en la unidad anterior cómo utilizar la salida estándar por consola en Java, es momento ahora de estudiar como introducir datos al programa por teclado.

Para leer datos de teclado podemos usar un objeto de tipo Scanner disponible en la biblioteca *java.util*, una de las presentes por defecto en Java.

Deberemos primero importar la clase de la biblioteca

```
import java.util.Scanner;
```

Entonces, tenemos que asignarle un identificador e instanciarla. En este caso es necesario pasar por parámetro el objecto *System.in* que representa la entrada estándar del sistema.

```
Scanner sc = new Scanner(System.in);
```

A partir de aquí ya podemos usarla dentro de nuestro código fuente.

La clase *Scanner* una utilidad de java que facilita el procesamiento de la información de una entrada de datos obteniendo a partir del texto recibido la información ya formateada a los tipos de datos primitivos y a *String*.

El siguiente ejemplo muestra cómo introducir por teclado dos números enteros y hacer una operación con ellos.

```
// Esta línea hace que la biblioteca esté disponible.
import java.util.Scanner ;

// Un programa que lee un entero y lo muestra por consola.
public  class Sumar {
  public  static  void main (String [ ] args)  {
    // inicializa la biblioteca.
    Scanner sc = new Scanner(System.in);
    // Se pone un mensaje de bienvenida.
    System.out.println("Vamos a sumar dos números enteros");
    // Se lee un valor entero por teclado.
    // Se espera que se pulse la tecla de retorno.
    System.out.print ("Escribe un número y pulsa la tecla de retorno:");
    int i = sc.nextInt();
    sc.nextLine();
    // Se vuelve a hacer ...
    System.out.print("Volver a hacerlo:");
    int j = sc.nextInt() ;
    sc.nextLine() ;
    // Hacemos la operación.
    int resultado = i + j;
    // Imprime el resultado por consola!
    // Se usa el operador suma entre una cadena de texto y un entero
    System.out.println("La suma de los dos valores es" + resultado +  ".");
  }
}
```

Una vez el objeto *sc* está inicializado (en este caso, sólo se hace una sola vez), el programa está preparado para leer datos de diferentes tipos desde el teclado. Por el momento, basta con ver cómo se pueden leer datos de uno en uno, usando diferentes líneas de texto para entrar cada dato individual. Para hacerlo, hay que alternar las instrucciones de lectura enumeradas en la tabla siguiente, según el tipo de dato que se quiere leer desde el teclado, y la instrucción `sc.nextLine();`, tal como se ve en el ejemplo anterior. Recuerda que *sc* es el identificador que damos al objeto *Scanner*, podríamos elegir cualquier otro siempre que cumplamos las reglas para dar nombre a los identificadores de variables que vimos en la unidad anterior. Se recomienda siempre seguir las reglas de estilo de Java.

Los métodos disponibles en la clase Scanner permiten obtener como resultado la evaluación del dato introducido por el teclado convertido al tipo de dato solicitado. Se encarga, pues, de

transformar el texto escrito, una cadena de texto, en un dato del tipo correspondiente. Normalmente, todo dato leído será asignado a alguna variable previamente declarada.

Usando este procedimiento, si en una misma línea se introduce texto con más de un dato, sólo se lee el primero de todos. El resto se ignora.

Tabla. Métodos de la clase Scanner para la lectura de datos por teclado

Instrucción	Tipo de dato leído
nextByte()	byte
nextShort()	short
nextInt()	int
nextLong()	long
nextFloat()	float
nextDouble()	double
nextBoolean()	boolean
next()	String

Este mecanismo no permite leer caracteres individuales. Recuerda, sin embargo, que, desde el punto de vista del tipo de dato, una cadena de texto que contiene un único carácter **no** es lo mismo que un carácter individual. Otro aspecto importante es que la forma en que reconoce los valores reales, si interpreta los decimales separados con un punto o una coma (por ejemplo, 9.5 o 9,5), depende de la configuración local del sistema. En la configuración para español se separan con una coma.

Si haces pruebas con estas instrucciones podrás comprobar que se requiere que los datos introducidos tengan un formato correcto, de otra manera se producirá un error y el programa se detendrá inmediatamente mostrando un mensaje de error por consola. A lo largo del curso veremos de qué manera se pueden controlar estas situaciones, por ahora deberemos prestar atención a la hora de introducir la información.

El método ***nextLine()*** también nos permitiría recoger todos los datos presentes en una línea y volcarlos sobre una variable de tipo *String*.