MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Utilización de objetos

11

ÍNDICE

OBJETIVOS

Crear y utilizar objetos.

Diferenciar entre paso por valor y paso por referencia.

Utilizar atributos y métodos de los objetos.

Utilizar métodos estáticos.

Comprender el almacenamiento en memoria de los objetos.

Diferenciar entre objeto y variable.

/ 1. Introducción y contextualización práctica

En este tema hablaremos sobre los objetos, sus características y cómo interactúan entre ellos.

Vamos a profundizar levemente en los métodos, viendo con detenimiento en qué consiste el paso por valor y el paso por referencia.

Por último, veremos cómo podremos obtener los datos de los objetos por teclado introduciéndolos el usuario, y cómo podremos mostrar correctamente nuestros objetos a los usuarios.

Como se puede observar, todos los temas que estamos estudiando están muy relacionados y en todos ellos aparecen conceptos ya estudiados, pero en cada unidad buscamos profundizar en ellos y relacionarlos.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Fig. 1. Una buena base lo facilita todo

Audio Intro. "Las instrucciones de control"

https://bit.ly/302Yt1j

/ 2. Características de los objetos

Ya sabemos que los objetos en PDO **son representaciones de entidades del mundo real en nuestros programas**, por lo cual, el procedimiento que teníamos para saber si algo puede ser candidato a objeto, era analizar lo siguiente:

- ¿Tiene características propias?
- ¿Puede realizar acciones propias?

Una vez recordado qué es un objeto, nos debemos preguntar lo siguiente, ¿qué es lo que hace único a un objeto?

Los objetos poseen **3 características que los distinguen**:

- **Identidad**. Esta propiedad nos va a permitir distinguir un objeto de otro del mismo tipo, de forma que, aunque dos objetos del mismo tipo tengan los mismos valores en sus atributos, serán distintos entre sí, es decir, serán independientes los unos de los otros. En Java esto se implementa mediante direcciones de memoria.
- **Estado**. Esta propiedad viene definida por todos los atributos que tiene un objeto y los valores de los mismos.
- **Comportamiento**. Esta propiedad viene definida por todas las acciones que puede realizar un objeto; viene definida por los métodos del objeto, desde los constructores hasta los métodos definidos por el programador.

Fig. 2. Un coche puede ser un objeto

2.1. Propiedades y métodos de los objetos

Una vez vistas las características de los objetos, vamos entrar un poco más en profundidad en ellas, y las relacionaremos con propiedades anteriormente estudiadas.

El estado **es la propiedad que define los atributos de los objetos, es decir, se compone de todas las variables u objetos que componen un objeto, las denominadas variables miembro o variables de objeto**. Estas pueden ser de los tipos primitivos (int, double, char, boolean...) o pueden ser objetos de otras clases.

Por ejemplo, si tenemos una clase Coche, su estado estará compuesto por matrícula, marca, modelo, kilómetros recorridos, etc.

El comportamiento **es la propiedad que define las acciones que un objeto puede llevar a cabo.** Estas acciones se refieren a los métodos que se han implementado en la clase que instancia el objeto. Estos serán los constructores (por defecto, con parámetros y de copia), los métodos get, los métodos set, el método toString y todos aquellos métodos[1] que cree el programador.

Podemos considerar que un objeto engloba tanto propiedades como métodos en un único ser.

Como ya sabemos, los datos están encapsulados dentro del objeto. Esto se conseguía declarándolos con un ámbito privado, y la única forma de modificarlos es mediante sus métodos.

Fig. 3. Privacidad

1. Siempre que nos refiramos a métodos se incluirán tanto las funciones como los procedimientos.

2.2. Interacción entre objetos

Ya hemos visto que los programas en Java están compuestos íntegramente de objetos, ya sean de clases internas, como puede ser la clase String, o de clases creadas por el programador.

El siguiente paso es saber cómo los objetos interactúan entre ellos para hacer que nuestros programas funcionen correctamente.

Los objetos **se comunican entre ellos mediante sus métodos**, con lo que podremos programar la interacción entre ellos haciendo que sea lo compleja que necesitemos.

Cuando un objeto utiliza uno de sus métodos para comunicarse con otro decimos que el primer objeto le envía un mensaje al segundo y que el segundo objeto recibe un mensaje del primero.

Los mensajes también se conocen como peticiones.

En PDO, definimos un protocolo con los mensajes que pueden enviar o recibir un objeto. Dicho protocolo podrá ser modificado agregando o eliminando métodos de las clases.

Sabiendo esto, podemos decir que el comportamiento de cualquier programa que desarrollemos es:

- Creación de objetos.
- Comunicación de los mismos mediante el envío y recepción de mensajes.
- Eliminación de los objetos que ya no sean necesarios. El encargado de esto es el recolector de basura, no el programador ni el usuario.

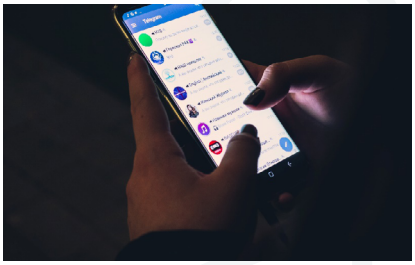

Fig. 4. Envío de mensajes

2.3. Almacenamiento en memoria

Cuando ejecutamos un programa, nos apoyamos en la memoria RAM del ordenador, es decir, todos los objetos y variables que usemos estarán alojados en memoria durante la ejecución del mismo.

Todo este control lo realiza el sistema operativo, es decir, que **es totalmente transparente para el usuario**.

Durante la ejecución del programa se van a usar diferentes zonas de memoria, cada una de las cuales posee un cometido diferente. Estas son:

- **La pila de llamadas**. En la pila de llamadas se guardarán todos los datos necesarios para la ejecución de los métodos de los objetos. Una pila es una estructura LIFO (del inglés, Last In, First Out), en la que el último elemento que entró es el primero en salir. Cada vez que se llame a un método se insertará una entrada en la pila.
- **El área de datos (HEAP)**. Esta es la zona de memoria donde se van a almacenar todas las variables, tanto primitivas como objetos. Cada vez que ejecutemos un new estaremos almacenando el objeto en HEAP.
- **El área de datos estáticos**. En esta zona de memoria se van a almacenar las variables globales del programa y las variables estáticas.
- **El área del código**. Este es el área de memoria en el que se van a almacenar las instrucciones que el programa debe ejecutar. Mientras más líneas de código tenga el programa más ocupará esta zona de memoria.

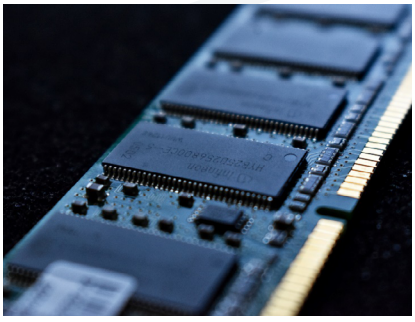

Fig. 5. Memoria RAM

2.4. Declaración e instanciación de objetos

Ya hemos visto en unidades anteriores que los objetos se crean como si fuesen variables. La sintaxis para la creación de un objeto es la siguiente:

```
NombreClase nombreobjeto;
```

Una vez hemos creado el objeto, aún no hemos terminado el proceso, ya que nos falta la parte más importante, su instanciación.

Cuando instanciamos un objeto estamos haciendo que el sistema operativo reserve memoria en HEAP para poder almacenarlo, y, por lo tanto, que el objeto tenga un valor válido.

La instanciación de los objetos se hace mediante la sentencia new y su sintaxis es:

```
nombreobjeto = new NombreClase( [parámetros] );
```

La instanciación de los objetos **es algo obligatorio en PDO, ya que si no los instanciamos no se habrá reservado memoria para que puedan tener un valor**, por lo que serán objetos nulos.

Los objetos nulos son muy peligrosos, ya que si intentamos hacer cualquier acción con uno de ellos el programa fallará automáticamente y se cerrará.

Cuando obtengamos un error por un objeto nulo, NetBeans nos mostrará que hemos sufrido un error de puntero nulo, o NullPointerException. La forma de tratar con este tipo de errores lo veremos en las siguientes unidades.

Recuerda que siempre que aparezca un error NullPointerException se deberá a que no le hemos aplicado new a un objeto.

```
public class JavaApplication2 {

    public static void main(String[] args) {

    }

}
```

Fig. 6. Creando un objeto

Audio 1. "Consecuencias de iguala un objeto a null"
https://bit.ly/2DwZqqU

2.5. Utilización de métodos

Ya sabemos que una de las partes de las clases son los métodos, y cuáles son los tipos de métodos que podemos crear.

La forma de nombrar los métodos según el estándar CamelCase será:

- Si el método se compone solo de una palabra irá en minúscula.
- Si el método se compone de varias palabras la primera irá en minúscula, mientras que en las demás, la primera letra será mayúscula.

Para poder llamar a un método de un objeto vamos a utilizar el operador punto (.) con la siguiente sintaxis:

```
[variable = ] nombreobjeto.nombreMetodo( [parámetros] )
```

Es importante recordar que para llamar a un método de un objeto antes tendremos que haberlo instanciado para que no sea nulo.

También podremos crear métodos en la clase principal de nuestro proyecto, a los cuales, para llamarlos, no será necesario crear ningún objeto, sino que podremos llamarlos directamente por su nombre:

```
[variable =] nombreMetodo( [parámetros] )
```

La única peculiaridad es que tendremos que indicar que son métodos estáticos, utilizando para ello la palabra reservada static.

/ 3. Caso práctico 1: "Rapidez o buen diseño"

Planteamiento. Pilar y José están realizando un ejercicio en el que tienen que crear clases que representen lo necesario para la gestión de una clínica veterinaria. Hablando sobre la resolución del ejercicio José le comenta a Pilar que sería mejor y más rápido guardar toda la información que necesitan en una única clase, ya que así con la creación de un objeto podrán resolver el problema de forma rápida y fácil.

Nudo. ¿Qué piensas sobre ello? ¿Crees que es apropiado guardar toda la información que se necesita en una clase? ¿Así se resolverá el problema más rápido?

Desenlace. Cuando vayamos a realizar un programa que utilice orientación a objetos, veremos que seguramente nos tendremos que parar un momento para pensar cómo diseñamos la solución. Todo buen diseño se basa en la creación de clases que representen fielmente las entidades del mundo real, por tanto, tendremos varias clases en cualquier proyecto que necesitemos. Meter en una única clase toda la información que necesitamos es una mala práctica, ya que no tendremos la claridad que ofrecen las clases bien diseñadas ni la abstracción de la información que éstas proveen.

Como conclusión, vale la pena dedicarle algo de tiempo a pensar el diseño de la solución para conseguir un código mucho más sencillo de hacer y comprender.

Fig. 7. El camino más rápido no lleva a la mejor solución

/ 4. Paso por valor vs paso por referencia

A la hora de invocar los métodos de los objetos podremos pasarles parámetros o no, según hayamos diseñado el método en la clase correspondiente.

Como ya adelantamos en temas anteriores, **los lenguajes de programación suelen usar dos técnicas a la hora de pasar los parámetros a los métodos: el paso por valor y el paso por referencia**.

Cuando se utiliza el paso por valor en los parámetros de un método, estamos haciendo una copia del valor original en el parámetro, por lo que, si realizamos cualquier modificación de ese parámetro dentro del método esta no tendrá efecto en la variable original.

Cuando se utiliza el paso por referencia en los parámetros de un método, en este caso, lo que estamos haciendo es pasar una referencia de la variable original en el parámetro del método, por lo que, si realizamos cualquier modificación de ese parámetro dentro del método, se hará también en la variable original.

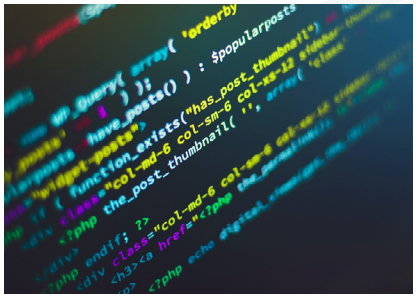

Fig. 8. Programando en diferentes lenguajes

El que se modifique la variable original al modificar el valor del parámetro en el paso por referencia, se debe a que la referencia que se pasa, es la dirección de memoria de la variable, con lo que tendríamos dos variables que apuntan a la misma dirección de memoria, es decir, tenemos dos variables pero en realidad es una.

En Java el paso por valor se hace con los parámetros que sean de tipo de variable primitiva, y el paso por referencia se hace con los parámetros que sean objetos.

Vídeo 1. "Paso por valor y por referencia"
https://bit.ly/2BX12dl

/ 5. Métodos estáticos

Ya vimos que podíamos declarar variables de clase (las variables estáticas) y que éstas eran variables que se compartían por todos los objetos que se crearan de esa clase.

De igual forma, tenemos los métodos estáticos. **Estos pertenecen a la clase, no al objeto, de ahí que también se les llame métodos de clase.**

La sintaxis de los métodos estáticos es la siguiente:

static [valor devuelto / void] nombreMetodo ([parámetros])

Como observamos, podemos crear tanto funciones estáticas, como procedimientos, pudiendo éstos recibir o no parámetros, es decir, cualquier tipo de método puede ser estático.

Estos métodos sólo podrán acceder a variables estáticas y sólo podrán llamar a otros métodos estáticos.

Al ser estos métodos de clase, no necesitaremos crear ningún objeto para poder llamarlo, ya que los métodos estáticos son métodos de clase, y basta con utilizar el operador punto sobre el nombre de la clase:

```
public class JavaApplication2 {

    public static void main(String[] args) {

    }

}
```

Fig. 9. El método estático más usado en el main

[variable =] NombreClase.nombreMetodo([parámetros])

Vídeo 2. "Uso de métodos estáticos y de objeto"
https://bit.ly/2Ok9cz1

5.1. Bloques estáticos

En las clases de Java tenemos lo que se conoce como un bloque estático, cuya sintaxis es la siguiente:

```
static {

// código

}
```

Estos bloques se ejecutarán solo la primera vez que la clase se carga en memoria, sin importar la cantidad de objetos instanciados de ella que tengamos.

Realizarán la función de "constructor" para las variables estáticas, es decir, éstas se han de inicializar aquí, ya que a los constructores no se les puede dar un valor.

Estos bloques se ejecutan antes que los constructores.

Si tenemos variables estáticas en una clase, **es conveniente inicializarlas usando un bloque estático, no en la declaración de la variable ni en un método estático**.

Si tenemos más de un bloque estático en la misma clase, **se ejecutarán en el orden en el que están escritos**.

/ 6. Caso práctico 2: "Divide y vencerás"

Planteamiento: Pilar y José están realizando una práctica en la que sorprendentemente no hay clases. Ésta consiste en resolver un problema bastante complicado que les está dando más de un dolor de cabeza.

Pilar propone resolver el problema en el método main, porque no hay clases, dice, pero José recuerda que el profesor les insistió bastante en que deben resolver los problemas modularizando, es decir, mediante la creación de métodos.

Nudo: ¿Qué piensas al respecto? ¿Crees que porque no haya clases implicadas en la resolución de un problema se debe resolver en el main? ¿Tiene José razón y deberían modularizar el problema?

Desenlace: Cuando estamos resolviendo un problema en programación, independientemente del lenguaje que estemos usando, es conveniente modularizar lo máximo posible. Con la modularización vamos a conseguir dividir un problema relativamente complicado de resolver en subproblemas más pequeños y, por ende, más fáciles de resolver.

Fig. 10. División del problema

Hay autores que llevan esta práctica al extremo y recomiendan tener en el main una única línea de código, esto puede ser contraproducente, ya que en cierto modo va a dificultar la comprensión del programa, siendo lo más adecuado, en main tener la lectura de las variables que vayan a hacer falta para resolver el problema, las llamadas a los métodos de los objetos, y la impresión de resultados.

/ 7. Entrada/salida de objetos

7.1. Entrada

Ya hemos visto en unidades anteriores cómo podemos leer por teclado los valores de variables de tipo primitivo. Para ello utilizábamos variables de tipo Scanner, aunque ya estamos preparados para afirmar, que lo que usamos en realidad son objetos de la clase Scanner.

Los métodos que utilizamos para leer los diferentes tipos de datos son:

- **Datos de tipo entero**: nextInt()
- **Datos de tipo real**: nextDouble()
- **Datos de tipo cadena**: nextLine()
- **Datos de tipo booleano**: nextBoolean()

Pero ¿qué ocurre con los objetos que creemos de una clase que hemos creado nosotros mismos?

Fig. 11. Diseñando la solución de un problema

No existen métodos específicos para leer objetos de clases creadas por el programador, es decir, si creamos una clase Motocicleta, no existe en la clase Scanner un método 'nextMotocicleta', ni nada por el estilo.

En estos casos, lo que se tendría que hacer es leer uno a uno los datos de los que está compuesto el objeto que queremos instanciar, para que, una vez leídos, podamos crear el objeto con el constructor con parámetros, o bien utilizar sus métodos set para darles el valor que queramos.

7.2. Salida de objetos

Al igual que pasa con la entrada de datos, ya hemos visto en unidades anteriores cómo mostrar el valor de las variables de tipo primitivo por pantalla.

Para esto utilizamos la orden **System.out.println()**, pasándole como parámetro una cadena con todo lo que queramos mostrar por pantalla.

Pero como nos pasaba con la entrada... ¿qué ocurre con los objetos de una clase que hemos creado nosotros?

Si mostramos mediante la orden System.out.println(), un objeto de una clase creada por nosotros, veremos que no aparece su valor, sino algo extraño. Este algo extraño es la dirección de memoria a la que hace referencia el objeto.

Si queremos mostrar la información de las variables de nuestro objeto tenemos dos posibles formas de hacerlo:

- **La primera es mostrando en System.out.println() los valores devueltos por los métodos get del objeto**, siempre que estén implementados.
- La segunda forma, y más cómoda, **es escribiendo el método toString de la clase**.

Una vez lo tengamos implementado **podremos mostrar directamente el objeto en un System.out.println()**, por lo que en lugar de aparecer la dirección de memoria del mismo, aparecerá su información tal y como la tenemos formateada en el método toString.

/ 8. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos visto con un poco de más detenimiento las características de los objetos, cómo interactúan entre ellos, cómo se almacenan en memoria, cómo podemos declararlos y cómo podemos utilizar sus métodos.

También hemos aprendido que hay dos posibles formas de pasar los parámetros a los métodos, por valor y por referencia, y qué implica cada una de las dos formas.

Hemos estudiado también qué son los métodos y bloques estáticos, y su importancia si tenemos variables estáticas o de clase.

Por último, hemos visto cómo podemos leer por teclado y mostrar por pantalla la información que contienen nuestros objetos de forma correcta.

Resolución del caso práctico de la unidad

Cuando hemos profundizado un poco ya en nuestras investigaciones sobre la programación dirigida a objetos, nos damos cuenta de que las clases están por todas partes, en cualquier programa que se precie.

Concretamente en Java, el lenguaje que estamos aprendiendo, las clases tienen un peso importantísimo.

Java es un lenguaje de programación orientada a objetos puro, esto quiere decir que todo lo que hacemos en nuestros programas está basado en la orientación a objetos, es decir, en las clases, por tanto, no existe otro elemento que las pueda reemplazar.

Tal es este peso que el método "main" que llevamos utilizando desde el principio, se encuentra en una clase, la clase principal, la cual será la que lanzará el programa al ejecutarse, así que desde un principio estábamos usando la programación dirigida a objetos sin saberlo.

/ 9. Bibliografía

Zonas de memoria durante la ejecución de programas. (2010, abril 13). Recuperado 20 de mayo de 2020, de https://parasitovirtual.wordpress.com/2010/04/13/zonas-de-memoria/

Parámetros por referencia en Java. (2015, septiembre 9). Recuperado 20 de mayo de 2020, de http://lineadecodigo.com/java/parametros-por-referencia-en-java/

G. (2018, marzo 19). https://guru99.es/. Recuperado 20 de mayo de 2020, de https://guru99.es/java-static-variable-methods/