MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Introducción a la orientación a objetos

8

ÍNDICE

OBJETIVOS

Conocer los principios básicos de la programación orientada a objetos.

Distinguir los conceptos de clase y objeto.

Conocer el concepto de encapsulado de información.

Conocer el concepto de visibilidad en las clases y los diferentes tipos de visibilidad.

Diferenciar entre variable y objeto.

Conocer el método de paso de mensajes entre clases.

/ 1. Introducción y contextualización práctica

En este tema, hablaremos sobre la programación dirigida a objetos o programación orientada a objetos y de sus grandes ventajas a la hora de crear código.

Vamos a hablar sobre los conceptos básicos que nos proporciona tanto a nivel de protección de información, como a nivel de encapsulamiento y abstracción de los datos. Los conceptos de clase y objetos van a ser fundamentales.

Por último, veremos cómo implementar la programación dirigida a objetos en Java mediante la creación de clases y objetos de las mismas.

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

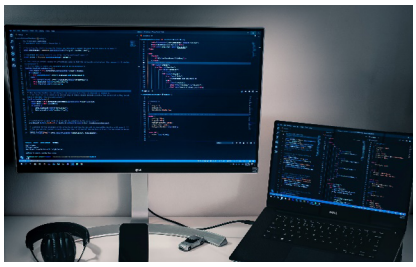

Fig. 1. Listos para aprender

Audio Intro. La programación dirigida a objetos

https://bit.ly/38RPEex

/ 2. Programación orientada a objetos

Cuando surgieron los primeros **lenguajes de programación** se utilizaba el **paradigma de la programación estructurada**, el cual permitía realizar cualquier programa mediante las estructuras básicas, que son:

- **Las secuencias** (instrucciones simples).
- **Las instrucciones condicionales.**
- **Las instrucciones repetitivas o bucles.**

Con el paso del tiempo este paradigma se quedó pequeño, ya que los programas cada vez eran mayores en cuanto a líneas de código y complejidad, con lo que su creación, entendimiento y mantenimiento se hacían cada vez más difíciles.

Todo esto terminó haciendo que se originara un nuevo paradigma en el mundo de la programación, **la programación dirigida a objetos.**

La programación dirigida a objetos es un nuevo enfoque que usa objetos para poder representar elementos del mundo real, tal como pueden ser una persona, un animal, un coche o una motocicleta.

Podemos destacar que mientras que la programación estructurada se centraba en el conjunto de acciones que había que realizar, la programación dirigida a objetos se va a centrar en la relación que existe entre los datos que necesitamos, y las acciones que podrán realizar, surgiendo así un nuevo concepto, la abstracción de información.

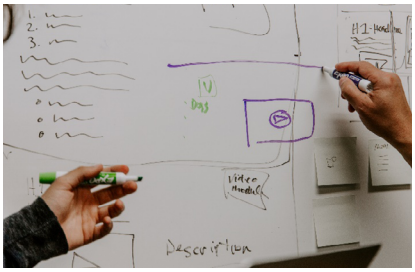

Fig. 2. Relación entre datos y acciones

2.1. Principios básicos de la programación orientada a objetos

La programación orientada a objetos nos va a proporcionar los siguientes principios:

- **Abstracción**: Con la abstracción la programación dirigida a objetos consigue que cada objeto utilizado funcione como un único agente, teniendo sus propias características y funciones, independientemente de cómo esté construido.
- **Encapsulamiento**: El encapsulamiento proporciona cohesión en los datos que forman un objeto, ya que esos datos estarán encapsulados dentro del objeto, aunque no los veamos.
- **Modularización**: Mediante la modularización podemos dividir un problema en otros problemas más pequeños, esto es uno de los aspectos fundamentales que proporciona la programación dirigida a objetos.
- **Herencia**: El concepto de herencia se basa en las relaciones existentes entre las clases, pudiendo éstas formar una relación jerárquica, donde haya clases que heredan comportamiento de otras.
- **Polimorfismo**: El polimorfismo proporciona a un objeto la posibilidad de tener diferentes comportamientos que están asociados a diferentes objetos. Este concepto va ligado al de herencia.
- **Ocultación de la información**: Este principio se basa en que cada objeto ha de estar aislado del exterior, es decir, desde fuera no se puede conocer cómo está compuesto un objeto por dentro.

Fig. 3. Un objeto es como una caja de la que no conocemos lo que hay en su interior

/ 3. Caso práctico 1: "Utilizar la programación dirigida a objetos o no, esa es la cuestión"

Planteamiento. Pilar y José están realizando un programa en el que tienen que representar los alumnos de un instituto para poder hacer gestiones con algunos de sus datos: darlos de alta, baja, hacer consultas, etc.

Observan que a cada alumno lo representa una serie de datos y no son pocos, y en ese momento entran en duda, ¿sería mejor crear variables para representar esos datos, o deberíamos utilizar la programación dirigida a objetos?

Nudo. ¿Qué piensas sobre ello? ¿Cuál de las dos formas crees que será la mejor para resolver este problema, la programación dirigida a objetos o la declaración de variables individuales?

Desenlace. Cuando necesitemos representar un dato del mundo real, en este caso un alumno, lo más fácil es utilizar la programación orientada a objetos, ya que, gracias a sus principios, como la encapsulación y protección de datos, harán que el problema se simplifique muchísimo en comparación con usar una variable independiente para representar cada una de las propiedades de los alumnos. Pensemos que, si a un alumno lo representan 10 datos diferentes, ¿cuántos datos necesitaremos para un instituto de 1000 alumnos que es lo normal que se puede dar?

Si usamos la programación dirigida a objetos con la creación de una única clase tendríamos el trabajo resuelto.

Fig. 4. Piensa bien como diseñar los datos

3.1. Ventajas e inconvenientes de la programación dirigida a objetos

La programación orientada a objetos tiene tanto ventajas como inconvenientes, aunque hay que destacar, que son muchas más ventajas que inconvenientes.

La programación dirigida a objetos presenta las siguientes ventajas:

- **Reusabilidad de código**: Una vez tengamos el diseño de una clase y verificado que ésta funciona correctamente, si necesitamos volver a usarla en otro proyecto podremos hacerlo sin problema.
- **Mantenibilidad**: Una vez que hemos aprendido a abstraer la información para transformarla en una clase, los programas que utilizan la PDO[1] son muchísimo más fáciles de leer, entender y mantener.
- **Modificabilidad**: Al igual que pasa con la mantenibilidad, una vez hemos aprendido a abstraer la información, los programas son mucho más fáciles de modificar en caso de que sea preciso.
- **Fiabilidad**: Al estar la PDO basada en la técnica de divide y vencerás, consistente en ir descomponiendo un problema en problemas más pequeños para resolverlo, podremos encontrar de una forma mucho más fácil posibles errores.

La programación dirigida a objetos presenta los siguientes inconvenientes:

- Hay un cambio en la forma de pensar para trabajar con ella, que posee cierta dificultad.
- La ejecución de estos programas es algo más lenta.

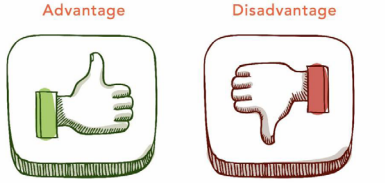

Fig. 5. Ventajas vs desventajas

1. Programación Dirigida a Objetos

3.2. Recolector de basura

El recolector de basura, o también llamado en inglés **garbage collector**, es una pieza clave del lenguaje Java. Es una grandísima ventaja frente a otros lenguajes que no lo poseen, como, por ejemplo, C++.

El recolector de basura es el **encargado de eliminar**, de forma automática, todos aquellos **objetos** que ya **no** son **referenciados por ninguna variable** y que se quedan "a la deriva" en la memoria del ordenador.

Su funcionamiento es el siguiente. Cada vez que instanciamos un objeto éste se va a crear en un **espacio de memoria llamado HEAP**, especialmente creado para dicho fin. Cuando hemos terminado de trabajar con dicho objeto se queda sin una variable que lo referencie, pero no es eliminado del HEAP. Cada cierto tiempo, antes de empezar a ejecutar un programa Java notaremos que se tarda un poquito más de lo normal en ejecutar, esto nos indicará que se está ejecutando el recolector de basura, y es en ese instante, el momento que se localizará del HEAP, todos los objetos no referenciados, tras lo cual, se procederá a eliminarlos de la memoria uno a uno.

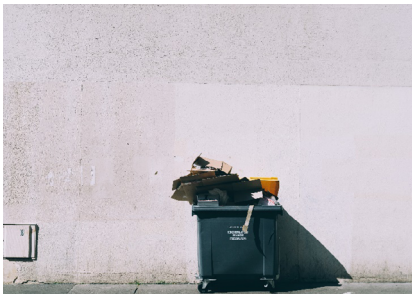

Fig. 6. Recolección de basura

El recolector de basura puede utilizar varias técnicas de borrado de objetos no referenciados, como por ejemplo **el borrado paralelo o el borrado G1.**

Incluso podremos lanzarlo desde una terminal cuando se desee, para que elimine los objetos no referenciados en el momento que se decida.

3.3. Protección de datos en la programación orientada a objetos

Un punto muy importante en la programación dirigida a objetos es el de **protección de datos.**

Este punto hace alusión directa a los conceptos de encapsulación y ocultación de la información.

Si recordamos el hecho de **encapsular información**, quiere decir que los datos que componen un objeto no serán visibles desde fuera, teniendo que usar métodos concretos para poder visualizarlos y modificarlos. Esto nos proporciona un nivel de protección de datos bastante alto, ya que desde fuera no se podrá acceder a la forma en la que se ha diseñado el objeto, evitando así posibles problemas futuros.

Fig. 7. Prohibido el paso

El principio de ocultación de información nos dice que podemos aplicar a las variables de nuestros objetos diferentes modificadores en cuanto a la visibilidad en el exterior se refiere. Básicamente, podremos hacer que sean visibles o no, teniendo así el poder de decidir qué partes de nuestros objetos dejamos ver desde el exterior.

Otra forma de protección de datos sería la de **organizarlos en paquetes**, como veremos en el siguiente punto.

3.4. Paquetes

En programación, cuando hacemos alusión a un **paquete**, estamos hablando sobre un **contenedor de clases** (u otros tipos de ficheros, como por ejemplo imágenes) que nos permitirá agruparlas según su funcionamiento.

Al final, lo que vamos a generar cuando creamos un paquete es una carpeta, es decir, podemos considerar que los paquetes son carpetas que nos van a ayudar a organizar nuestras clases según su comportamiento.

Los paquetes los podremos utilizar sea cual sea el lenguaje de programación que usemos, ya sean de escritorio, como C++, Java o Python, lenguajes de apps, como Android o Swift, o lenguajes de programación web, como PHP.

El uso de paquetes nos proporciona las siguientes ventajas:

- **Agrupamiento de clases** con características comunes.
- **Reutilización de código.**
- **Mayor seguridad** al poder crear niveles de acceso a las clases.

Cuando necesitemos utilizar una clase que esté en un paquete diferente debemos de importarla, haciendo uso de la instrucción import. Podremos importar tantas clases que estén en paquetes diferentes como necesitemos.

Un ejemplo de importación de una clase de otro paquete es cuando utilizamos el tipo de dato Scanner, utilizado para leer información por teclado, de ahí que tengamos que importar la clase para que funcione.

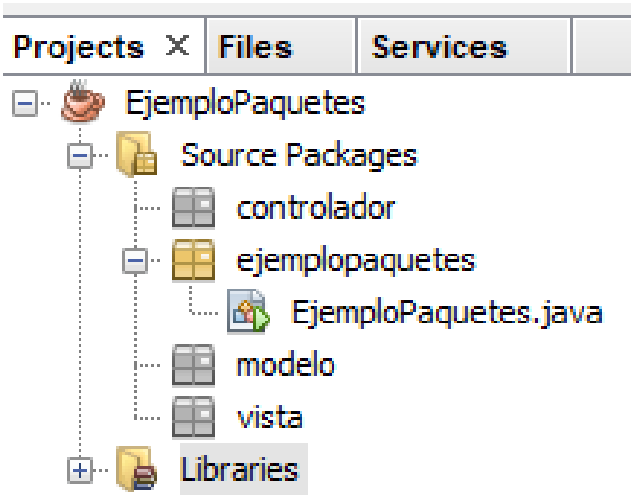

Fig. 8. Ejemplo de proyecto con Paquetes.

Vídeo 1. Creación de paquetes
https://bit.ly/2OkWmQW

3.5. Visibilidad de los miembros de una clase

Dentro del encapsulamiento y ocultación de información en las clases, podremos definir varios niveles de protección los cuales podremos usar según la que nos convenga más en cada momento.

Existen cuatro niveles principales de protección diferentes para los distintos elementos de una clase:

- **Estándar**: Es el estado predeterminado.
- **Público**: Este método permite que los miembros de la clase puedan ser accedidos desde el exterior de la propia clase y desde cualquier parte del programa.
- **Protegido**: Los miembros de la clase solo serán accesibles desde la clase y las clases que heredan (a cualquier nivel).
- **Privado**: Los miembros de la clase solo serán accesibles desde la propia clase, no desde fuera.

El uso de paquetes es altamente recomendado debido a la facilidad que nos proporciona para leer y organizar el código de nuestros proyectos.

Como veremos más adelante, una técnica muy utilizada en programación se basa en el trabajo con paquetes para organizar el código, esta es el Modelo Vista Controlador.

Modificador	Public	Protected	Private	Defecto
Acceso desde la propia clase	SI	SI	SI	SI
Acceso desde otras clases del mismo paquete	SI	SI	NO	SI
Acceso desde una subclase en el mismo paquete	SI	SI	NO	SI
Acceso desde subclases en otros paquetes	SI	SI	NO	NO
Acceso desde otras clases en otros paquetes	SI	NO	NO	NO

Tabla 1. Resumen visibilidad

/ 4. Clases y objetos

Una clase es la representación de algo del mundo real en un programa informático. Para detectar cuando necesitamos crear una clase nos tenemos que preguntar lo siguiente: ¿Esto existe en el mundo real? ¿Tiene propiedades? ¿Puede realizar acciones?

Un ejemplo muy claro de esto es una persona, que en respuesta a las preguntas tenemos:

- **¿Esto existe en el mundo real?**: Sí, las personas existen el mundo real.
- **¿Tiene propiedades?**: Sí, una persona tiene un DNI, un nombre, unos apellidos, una edad...
- **¿Puede realizar acciones?**: Sí, una persona puede saludar, puede andar...

Si cumple con las tres preguntas, es un caso claro de clase, si por el contrario, no cumple con alguna de ellas, probablemente no sea un candidato a poder ser representado mediante una clase.

Un objeto, sin embargo, es una variable de una clase que hemos creado. Los objetos son instancias de las clases, y podrán realizar todas las acciones que se han definido. Estarán compuestos en su interior de las variables con las que hemos creado la clase.

Cabe destacar que los objetos serán independientes unos de otros, es decir, tendrán la misma composición en su interior, pero el hecho de manipular un atributo de un objeto solo hará que se modifique en dicho objeto, no en los demás.

Como resumen final, podemos decir que una clase es el molde para crear los objetos.

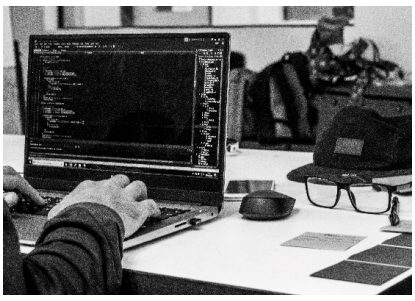

Fig. 9. Creando clases

Audio 1. Diferencia entre variables y objetos
https://bit.ly/38Z9swQ

4.1. Ciclo de vida de un objeto

Todos los programas que creemos en Java van a partir de una clase, "la clase principal", que es la encargada de ejecutar el método principal que lanzará el programa, este es el método main, que podemos ver cuando creamos un proyecto nuevo en NetBeans.

Todas las instancias de objetos tienen un ciclo de vida predeterminado, el cual se cumplirá desde que los creamos hasta que no los necesitamos y los destruya el recolector de basura para liberar la memoria utilizada, y así pueda ser reutilizada por otro objeto.

El ciclo de vida de un objeto puede ser similar al ciclo de vida de un ser vivo, ya que estos "nacen", "son utilizados" y "mueren".

Para poder crear un objeto útil que pueda cumplir con su ciclo de vida, las clases han de cumplir con ciertos requisitos:

1. Han de tener, al menos, un constructor, con el cual podremos crear los objetos.

2. Han de tener métodos, con los cuales podremos utilizar los objetos para lo que necesitemos.

Lo que ocurre en los estados del ciclo de vida es:

- **Creación:** Se hace una reserva de memoria y se inicializan sus atributos.
- **Utilización:** Se usan los métodos y atributos del objeto.
- **Destrucción:** Se elimina la referencia del objeto y más tarde el recolector de basura lo elimina y libera la memoria que ocupaban sus atributos.

Fig. 10. Ciclo de vida de un objeto

4.2. Creación de clases en Java

Una vez que tenemos claro que existen elementos que los podemos representar mediante una clase, es necesario conocer que, para ello, debemos utilizar la palabra reservada class para crear dicha clase en Java.

Los nombres de las clases se rigen por las mismas normas de los identificadores, salvo con la peculiaridad de que los nombres de las clases empezarán siempre por una letra mayúscula, para así poder distinguirlas de una variable u objeto.

Esto se debe a la notación CamelCase, utilizada por los programadores a la hora de nombrar los elementos, como son las variables, clases, métodos...

Las clases se componen de dos partes:

- **Atributos:** Serán los atributos de la propia clase, es decir, representarán las propiedades de la entidad del mundo real que vamos a representar. Podrá tener tantos como se necesite.
- **Métodos:** Estos representan las acciones que puede realizar la entidad del mundo real que vamos a representar. Igualmente, se podrá tener tantos como se necesite.

Un ejemplo de clase puede ser el de la siguiente figura.

```
public class Persona
{
    String DNI, nombre, apellidos;
    int edad;

    public void saludar()
    {
        System.out.println("Hola soy " + nombre);
    }
}
```

Fig. 11. Clase Persona

4.3. Creación de objetos en Java

Una vez que tenemos creada nuestra clase, el siguiente paso será la instanciación de objetos.

Para la instanciación de objetos de una clase debemos seguir los siguientes pasos:

- Declaración del objeto.
- Instanciación del objeto.

Cuando creamos el objeto, estamos creando una "variable" del tipo de la clase a la que pertenezca dicho objeto. Por ejemplo, si tenemos una clase 'Coche', podremos crear una "variable" del tipo 'Coche', igual que haríamos con una variable del tipo int.

Cuando instanciamos el objeto, estamos reservando en memoria el espacio suficiente para almacenar todas las variables de las que se compone el objeto y creando la referencia del objeto a su posición en memoria. Esto lo conseguiremos con el operador new.

Un ejemplo de creación de un objeto de la clase Coche puede ser:

Coche coche1 = new Coche();

En este caso, estamos creando un objeto llamado coche1 que instancia la clase Coche y a su vez estamos reservando memoria para dicho objeto al utilizar el operador new.

Si no utilizásemos el operador new tendríamos un objeto nulo, el cual, en cuanto intentemos utilizarlo nos daría un error. Concretamente sería el conocido NullPointerException, el cual, podremos subsanarlo instanciando de forma correcta el objeto que produzca el fallo.

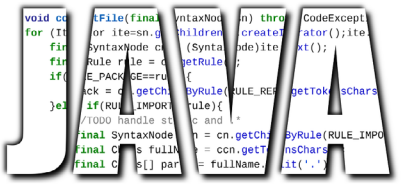

Fig. 12. La creación de objetos y clases en Java es fundamental.

Vídeo 2. Creación de una clase y un objeto
https://bit.ly/2C4lRTW

/ 5. Caso práctico 2: "¿Qué visibilidad usar?"

Planteamiento: Pilar y José siguen realizando el ejercicio de los alumnos del instituto. Ya tienen creada una clase 'Alumno' con todas las variables necesarias para su representación, pero necesitan darles un método de visibilidad.

En su investigación han aprendido que existen diferentes métodos de visibilidad en la programación orientada a objetos y por ello no se saben decidir sobre cuál de ellos sería mejor utilizar.

Nudo: ¿Qué piensas al respecto? ¿Cuál de los métodos de visibilidad existentes en la programación dirigida a objetos crees que es el más adecuado?

Desenlace: Cuando usamos programación dirigida a objetos tenemos varios métodos de visibilidad en las clases, como son public, private y protected. Siempre que utilicemos la abstracción que nos proporciona la programación dirigida a objetos debemos de seguir la regla que nos dice que todos los atributos miembros de las clases (las variables de dentro de la clase) deben ser privadas, para poder así cumplir con la encapsulación de información, y que todos los métodos deben ser públicos, para así poder ser usados desde fuera por los objetos. Por último, recordar que las clases deben ser públicas en sí mismas.

Fig. 13. Un objeto con los atributos públicos es como una caja abierta, se ve lo que hay dentro

/ 6. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos visto el concepto de programación dirigida a objetos, algo básico y muy necesario para cualquiera que intente aprender a programar en cualquier lenguaje hoy en día.

La programación dirigida a objetos, o PDO, tiene sus ventajas e inconvenientes, aunque como hemos visto, son más las ventajas que los inconvenientes haciendo que el hecho de programar sea mucho más sencillo.

También hemos comprobado que existen distintos niveles de protección de datos a la hora de crear los objetos y que, deberemos utilizar unos u otros según lo que estemos realizando.

Por último, hemos aprendido cómo podemos crear una clase en Java haciendo uso de la palabra reservada class, y cómo crear correctamente un objeto, definiendo una variable del tipo de la clase que queramos, e instanciándolo con el operador new para que se cree en memoria de forma correcta.

Resolución del caso práctico de la unidad

Hoy en día, cuando aprendemos nuestro primer lenguaje de programación es importante que éste sea orientado a objetos, la razón es muy simple. La gran mayoría de lenguajes de programación que se usan en la actualidad (todavía se usan lenguajes de programación antiguos que no son orientados a objetos, como el lenguaje C) soportan la orientación a objetos.

La orientación a objetos puede parecer un poco extraña al principio, pero cuando la comprendemos somos conscientes de su gran poder y las enormes ventajas que proporciona.

Además, asentando las bases de la programación dirigida a objetos en nuestro primer lenguaje de programación, tendremos las cosas muchísimo más fáciles cuando queramos aprender otro lenguaje nuevo, ya que probablemente éste también sea orientado a objetos.

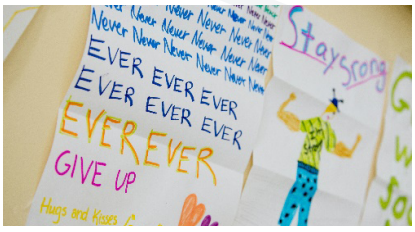

Fig. 14. No es fácil aprender a programar Truco: no rendirse

/ 7. Bibliografía

Programación orientada a objetos. (2018, marzo 18). Recuperado 17 de mayo de 2020, de https://unsplash.com/photos/26MJGnCM0Wc

Programación estructurada. (2020, abril 23). Recuperado 17 de mayo de 2020, de https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada

Programación orientada a objetos. (2020, marzo 12). Recuperado 17 de mayo de 2020, de https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

Roldán, A. (s. f.). Ventajas de la Programación Orientada a Objetos. Recuperado 17 de mayo de 2020, de https://www.ciberaula.com/cursos/java/ventajas_poo.php

P. (2020, febrero 15). El recolector de basura de Java, que hace y como funciona en cada versión. Recuperado 17 de mayo de 2020, de https://picodotdev.github.io/blog-bitix/2020/02/el-recolector-de-basura-de-java-que-hace-y-como-funciona-en-cada-version/