**Resumen JAVA POO**

**Paradigma de Programación:**

Por lo que un paradigma de programación se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas.

**POO:**

La **Programación Orientada a Objetos (POO)** es un paradigma de programación, es decir, un modelo o un estilo de programación que se basa en el concepto de **clases y objetos**. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de código (clases) para crear instancias individuales de objetos.

La programación orientada a objetos se enfoca en los objetos, sus atributos y las interacciones

que se producen entre ellos para diseñar programas.

Un programa orientado a objetos es, esencialmente, un **conjunto de objetos** que se crean

**interaccione entre sí** y dejan de existir cuando ya no son útiles durante la ejecución de un

programa.

Sigue el principio de desarrollo de *software* utilizado por muchos programadores **DRY**

***(Don’t Repeat Yourself)****,* para evitar duplicar el código y crear de esta manera programas

eficientes. Además, evita el acceso no deseado a los datos o la exposición de código propietario mediante la encapsulación y la abstracción.

**¿Qué es un Objeto?:**

Un objeto en la programación es la forma que tenemos de llevar a código, la representación más fidedigna posible de un objeto de la vida real. Un objeto tendrá características (atributos) que son comunes a todos ellos, pero pueden variar entre sí.

**ESTADO Y COMPORTAMIENTO:**

El estado de un objeto es una **lista de variables conocidas como sus atributos**, cuyos valores

representan el estado que caracteriza al objeto.

El comportamiento es una **lista de métodos, procedimientos, funciones u operaciones** que un

objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como

instancias.

**¿QUÉ SON LOS ATRIBUTOS?**

Los atributos son **características comunes** a todos los objetos.

El estado o información de un objeto se almacena en atributos. Los atributos pueden ser de tipos primitivos de Java o del tipo de otros objetos.

**¿QUÉ SON LOS CONSTRUCTORES?**

Los constructores con métodos propios del objeto que nos permiten CREARLO. A la creación de un objeto se le denomina INSTANCIACIÓN.

Toda clase debe definir un método especial denominado constructor para instanciar los objetos de la clase. Este método tiene el mismo nombre de la clase.

El método constructor se ejecuta cada vez que se instancia un objeto de la clase. Este método se utiliza para **inicializar** los atributos del objeto que se instancia. Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra this. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase. La instanciación de un objeto consiste en asignar un espacio de memoria al que se hace referencia con el nombre del objeto. Los identificadores de los objetos permiten acceder a los valores almacenados en cada objeto.

Cada clase tiene al menos un constructor. Si no se escribe un constructor, el lenguaje de

programación Java le provee uno por defecto. Este constructor no posee argumentos y tiene un cuerpo vacío.

**ABSTRACCIÓN Y ENCAPSULAMIENTO**

**ABSTRACCIÓN:**

La abstracción encarada desde el punto de vista de la programación orientada a objetos es el mecanismo por el cual se proveen los límites conceptuales de los objetos y se expresan sus características esenciales, dejando de lado sus características no esenciales. Si un objeto tiene más características de las necesarias los mismos resultan difíciles de usar, modificar, construir y comprender. En el análisis hay que concentrarse en ¿Qué hace? Y no en ¿Cómo lo hace?

**ENCAPSULAMIENTO:**

La encapsulación o encapsulamiento significa reunir en una cierta estructura a todos los

elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad, y es el proceso de agrupamiento de datos y operaciones relacionadas bajo una misma unidad de programación, lo que permite aumentar la cohesión de los componentes del sistema. El encapsulamiento **oculta lo que hace un objeto** de lo que hacen otros objetos y del mundo exterior por lo que se denomina también ocultación de datos.

Ocultar el estado interno y hacer que toda interacción sea a través de los métodos del objeto es un mecanismo conocido como encapsulación de datos.

**MODIFICADORES DE ACCESO**

**Public:** Este modificador permite a **acceder a los elementos desde cualquier clase**, independientemente de que esta pertenezca o no al paquete en que se encuentra el

elemento.

**Private:** Es el modificador más restrictivo y especifica que los elementos que lo utilizan

**sólo pueden ser accedidos desde la clase en la que se encuentran**. Este modificador sólo

puede utilizarse sobre los atributos de una clase y sobre interfaces y clases internas, no

sobre clases o interfaces de primer nivel, dado que esto no tendría sentido.

**Protected:** Este modificador indica que los elementos **sólo pueden ser accedidos desde**

**su mismo paquete y desde cualquier clase que extienda la clase** en que se encuentra,

independientemente de si esta se encuentra en el mismo paquete o no. Este modificador,

como private, no tiene sentido a nivel de clases o interfaces no internas.

**MÉTODOS PROPIOS**

Los métodos son funciones que determinan el comportamiento de los objetos. Un objeto se

comporta de una u otra forma dependiendo de los métodos de la clase a la que pertenece. Todos los objetos de una misma clase tienen los mismos métodos y el mismo comportamiento.

Existen tres tipos de métodos: métodos de consulta, métodos modificadores y operaciones. Los métodos de consulta sirven para extraer información de los objetos, los métodos modificadores sirven para modificar el valor de los atributos del objeto y las operaciones definen el comportamiento de un objeto.

**GETTER & SETTER**

Para acceder a los atributos de un objeto se definen los **métodos get y set**. Los métodos get se

utilizan para **consultar el estado de un objeto** y los métodos set para **modificar su estado**. Un

método get se declara public y a continuación se indica el tipo de dato que devuelve. Es un

método de consulta. La lista de parámetros de un método get queda vacía. En el cuerpo del

método se utiliza return para devolver el valor correspondiente al atributo que se quiere devolver, y al cual se hace referencia como **this.nombreAtributo.**

Por otra parte, un método set se declara public y devuelve void. La lista de parámetros de un

método set incluye el tipo y el valor a modificar. Es un método modificador. El cuerpo de un

método set asigna al atributo del objeto el parámetro de la declaración.

**¿CÓMO UTILIZAMOS ESTOS MÉTODOS?**

Un método se puede invocar dentro o fuera de la clase donde se ha declarado. Si el método se

invoca dentro de la clase, basta con indicar su nombre**. Si el método se invoca fuera de la clase** entonces se debe indicar el nombre del objeto y el nombre del método. Cuando se invoca a un método ocurre lo siguiente:

• En la línea de código del programa donde se invoca al método se calculan los valores de

los argumentos.

• Los parámetros se inicializan con los valores de los argumentos.

• Se ejecuta el bloque código del método hasta que se alcanza return o se llega al final del

bloque.

• Si el método devuelve un valor, se sustituye la invocación por el valor devuelto.

• La ejecución del programa continúa en la siguiente instrucción donde se invocó el

método.

**ATRIBUTOS Y MÉTODOS ESTÁTICOS**

Un atributo o un método de una clase se puede modificar con la palabra reservada static para

indicar que este atributo o método no pertenece a las instancias de la clase si no a la propia clase.

Se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es

única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria), es decir que, si se poseen múltiples instancias de una clase, cada una de ellas no tendrán una copia propia de este atributo, si no que todas estas instancias compartirán una misma copia del atributo. A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto).

En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static. Cuando usamos

”static final” se dice que creamos una constante de clase, un atributo común a todos los objetos de esa clase.

**ATRIBUTOS FINALES**

En este contexto indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. La palabra reservada **final** determina que un atributo no puede ser sobrescrito o redefinido, es decir, no funcionará como una variable “tradicional”, sino como una constante. Toda constante declarada con *final* ha de ser inicializada en el mismo momento de declararla. El modificador *final* también se usa como palabra clave en otro contexto: una clase *final* es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.

**CLASE SERVICIO**

**La clase servicio (service) o control**, va a ser una clase auxiliar que nos va a ayudar con el manejo de las clases y los objetos de esas clases, pero para poder explicar esto, primero vamos a tener que ver **los patrones generales de software GRASP**. Aunque se considera que más que **patrones** propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

**GRASP** es el acrónimo de General **Responsibility Assignment Software Patterns**. Una de las

cosas más complicadas en Orientación a Objeto consiste en elegir las clases adecuadas decidir

como estas clases deben interactuar.

Dentro de los patrones GRASP, vamos a utilizar el patrón experto. El GRASP de experto en

información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo,

que la responsabilidad de la creación de un objeto o la implementación de un método debe

recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo

obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada

(disminución del acoplamiento).

**Problema**

¿Cuál es el principio general para asignar responsabilidades a los objetos?

**Solución**

Asignar una responsabilidad al experto en información.

**Beneficios**

Se mantiene el encapsulamiento, los objetos utilizan su propia información para llevar a cabo sus tareas. Se distribuye el comportamiento entre las clases que contienen la información requerida.

Son más fáciles de entender y mantener.

Esta clase del patrón experto va a ser la clase servicio.

Es una clase común y corriente pero que se va a encargar de crear los objetos y va a tener todos los métodos necesarios para la utilización de ese objeto. Supongamos que necesitamos un método que le sume un valor x a un atributo del objeto, ese método estará en la clase control. Siempre se crea una clase control, por cada clase que tengamos, si tenemos las clases Persona y Sueldo, crearemos una clase control para Persona y otra para Sueldo. La idea es que una clase servicio, se encargue de solo una clase.

**CLASES DE UTILIDAD PARTE 2**

Recordemos que las clases de utilidad son clases dentro del API de Java que son muy utilizadas

en el desarrollo de aplicaciones.

Entre las clases de utilidad de Java más utilizadas y conocidas están las siguientes: Arrays, String, Integer, Math, Date, Calendar y GregorianCalendar. En la guía anterior vimos solo las clases Math y String.







