TEMA 5 POO INTRODUCCIÓN

Eduardo Martín-Sonseca

Contenido

[1. METODOS 2](#_Toc128253179)

[1.1. METODOS SENTENCIA RETURN 2](#_Toc128253180)

[1.2. METODOS – LLAMADA 2](#_Toc128253181)

[1.3. METODOS – PASO DE PARAMETROS 2](#_Toc128253182)

[2. POO – INTRODUCCIÓN 3](#_Toc128253183)

[3. OBJETOS 3](#_Toc128253184)

[3.1. OBJETOS CREACIÓN 4](#_Toc128253185)

[3.2. OBJETOS USO 4](#_Toc128253186)

[3.3. OBJETOS DESTRUCCIÓN 4](#_Toc128253187)

[4. BENEFICIOS DE LA POO 4](#_Toc128253188)

[5. PROPIEDADES DE LA POO 4](#_Toc128253189)

[6. CLASES 5](#_Toc128253190)

[6.1. CLASES – CREACIÓN 5](#_Toc128253191)

[6.2. CLASES – ENCAPSULAMIENTO 5](#_Toc128253192)

[7. REFERNENCIA THIS 5](#_Toc128253193)

[8. LA CLASE OBJECT 5](#_Toc128253194)

[9. CONSTRUCTORES 6](#_Toc128253195)

[9.1. CONSTRUCTORES POR DEFECTO 6](#_Toc128253196)

[9.2. CONSTRUCTORES ALTERNATIVOS 6](#_Toc128253197)

[9.3. CONSTRUCTORES SOBRECARGADOS 6](#_Toc128253198)

[10. MIEMBROS ESTATICOS 6](#_Toc128253199)

# METODOS

Un método es un conjunto de instrucciones que realizan una tarea concreta, y se agrupan bajo un identificador, para ejecutar esas instrucciones, hay que **llamar** al método, la utilidad principal es la **reutilización de código** debido a que un método puede ser utilizado varias veces desde un mismo programa y también mejora la **legibilidad del código,** la visibilidad es alguno de los modificadores de acceso, el método puede ser o no estático, el tipo de retorno es el tipo de dado a devolver, (es de tipo void si no devuelve nada), la lista de parámetros se indica de la siguiente forma :

**tipo1 param1, tipo2 parm2, ...**

## METODOS SENTENCIA RETURN

Sirve para devolver un valor, el valor puede ser literal o una variable, el valor retornado tiene que ser el mismo en la definición del método, se pueden escribir varias sentencias **return** en un mismo método y en distintas posiciones

**public int suma(int a, int b) {  
int resultado = a + b;  
return resultado;**

## METODOS – LLAMADA

Para utilizar un método, hay que llamarle, escribiendo en el código fuente el nombre del método seguido de tantos parámetros como hay en el método

**public static void main(String[] args) {  
saludo(Hola);  
}**

**public static void saludo(String msg) {  
System.out.println(msg);  
}**

## METODOS – PASO DE PARAMETROS

* **Parámetros formales**: Son aquellos que están definidos en la declaración del método
* **Parámetros actuales** Son los pasados en la llamada al método

Cuando se llama al método, los parámetros actuales se copian en los formales

* **Paso de parámetros por valor** se pasa una copia del valor de los actuales a los formales
  + La información esta duplicada en memoria
  + Las modificaciones no afectan al método actual

**public void metodo(int i){  
System.out.println("En el método - Antes i=" + i);  
i++;  
System.out.println("En el método - Después i=" + i)  
}  
public static void main(String[] args) {  
Pruebas app=new Pruebas();  
int n=0;  
System.out.println("En el main - Antes n=" + n);  
app.metodo(n);  
System.out.p)rintln("En el main - Después n=" + n);**

**}**

* **Paso de parámetros por referencia** se pasa una copia de la dirección de memoria del objeto
  + Las modificaciones afectan al parámetro actual

**public void metodo(MiClase obj){  
System.out.println("En el método - Antes a=" + obj.a);  
obj.a++;  
System.out.println("En el método - Después a=" + obj.a)  
}  
public static void main(String[] args) {  
Pruebas app=new Pruebas();  
MiClase mC=new MiClase();  
mC.a=0;  
System.out.println("En el main - Antes a=" + mC.a);  
app.metodo(mC);  
System.out.p)rintln("En el main - Después a=" + mC.a);**

# POO – INTRODUCCIÓN

**El modelo procedimental** está en torno a procesos

**El modelo orientado a objetos** esta orientado a los datos los cuales son considerados más importantes que los procesos, hace más sencillo resolver problemas al dividirlo en objetos, debido a que cada objeto funciona de forma independiente

# OBJETOS

Son creados a partir de una clase (un objeto es una clase en ejecución), y una clase es el tipo de datos del objeto, sus características son :

* **Identidad**: Cada objeto es único y diferente
* **Estado**: Son los valores de los atributos en un momento determinado
* **Comportamiento** Son los métodos del objeto

El ciclo de vida de un objeto es

* Creación
* Uso
* Destrucción

# OBJETOS CREACIÓN

La declaración de un objeto consiste en la declaración de una variable del tipo de la clase objeto

La instanciación es la creación del objeto (ponerlo en memoria), para instanciarlo, se necesita un **constructor** de la clase mediante el operador **new**, si usamos un objeto sin antes haber lo instanciado se lanza la excepción **NullPointerException**

# OBJETOS USO

Una vez creado el objeto, se puede utilizar:

* Accediendo a sus variables
* Llamando a los métodos

Para referirse, se utiliza el operador.

# OBJETOS DESTRUCCIÓN

Los destructores de objetos se utilizan para **liberar recursos** y cerrar flujos abiertos

No reciben parámetros

No esta permitida la sobrecarga, en Java, no hay destructores

# BENEFICIOS DE LA POO

* **Modularidad** El código fuente de un objeto puede mantenerse y reescribirse sin que haya que reprogramar el código de otros objetos de la aplicación
* **Reutilización de código** Se puede utilizar clases y objetos de terceras personas, no tenemos que conocer detalles, sino solamente la interfaz
* **Facilidad de testeo y reprogramación** Saber reemplazar el objeto problemática por otro similar o reprogramarlo
* **Ocultación de información** Se ocultan detalles de implementación

# PROPIEDADES DE LA POO

* **Abstracción** Abstrae las características del objeto y crea clases con atributos y métodos
* **Encapsulamiento:** Ejecutar los métodos de un objeto sin tener que saber cómo funciona
* **Ocultación de la información:** La zona privada de los objetos, es utilizada por métodos de la propia clase
* **Herencia**: Una clase (subclase) puede heredar propiedades de la otra clase(superclase)
* **Polimorfismo** El mismo método funciona de formas diferentes según el objeto que le llame

# CLASES

Una clase es la **definición** de un tipo de objeto, la plantilla define las propiedades y metodos a todos los objets de esa clase

Una clase es un concepto **abstracto** que no ocupa memoria, un objeto es un concepto **real**, cada objeto tiene su espacio

## CLASES – CREACIÓN

* **Definir atributos** los datos miembros de esa clase, pueden ser públicos (accesibles desde otra clase) o privados (sólo accesibles por código de su propia clase)
* **Definir métodos:** Son las acciones que puede realizar la clase
* **Definir código de inicialización**: Crear un método constructor que inicialice los atributos

## CLASES – ENCAPSULAMIENTO

Para controlar el acceso a los miembros de la clase, se utiliza modificadores de acceso

* **Public**: se puede acceder a cualquier método desde fuera de la clase
* **Private** : Se puede acceder por los métodos de la misma clase
* **Protected** : Se puede acceder por los métodos de la misma clase, o de clases derivada como métodos de otras clases que se encuentran en el mismo paquete

Tabla

Descripción generada automáticamente

# REFERNENCIA THIS

Referencia al objeto que se esta utilizando (el que ejecuta el método), resuelve ambigüedades

# LA CLASE OBJECT

Cualquier clase es subclase de Object, los que vamos a utilizar son

* **toString()** : devuelve la representación **textual** del objeto,
* **equals()** : Sirve para comparar objetos
* **Compateto()** : Es una alternativa a equals
* **getClass()** Devuelve la clase del objeto en tiempo de ejecución

# CONSTRUCTORES

Un constructor es un método no estático que se ejecuta de forma automática cuando se crea el objeto de la clase, sirve para **inicializar** los datos de la clase

* método no estático
* Tiene el mismo nombre de la clase
* Puede tener 0 o más argumentos
* No devuelve ningún valor
* Debe ser publico

## CONSTRUCTORES POR DEFECTO

Un constructor por defecto no tiene parámetros, el valor inicial a los datos de la clase es asignando valores por defecto, (el constructor por defecto se crea cuando no existes otros constructores, e inicializa todos los datos a cero, en caso de boolean a false

**public class Punto {  
private double x;  
private double y;  
public Punto() {  
x=0.0;  
y=0.0;  
}  
}**

## CONSTRUCTORES ALTERNATIVOS

Un constructor alternativo es aquel que tiene parámetros

**public class Punto {  
private double x;  
private double y;  
public Punto(double \_x, double \_y) {  
x=\_x;  
y=\_y;**

## CONSTRUCTORES SOBRECARGADOS

Un constructor sobrecargado es definir en la misma clase mas de un constructor cada uno de ellos con distinto número de parámetros, son bastante buenos debido a que proporcionan varias opciones para la creación de objetos

## MIEMBROS ESTATICOS

Son atributos y miembros que pueden ser accedido a ellos sin crear un objeto, se accede desde la clase

# MIEMBROS ESTATICOS ATRIBUTOS ESTATICOS

Un atributo static indica que el atributo no pertenece a las instancias de la clase, si no a la propia clase , es decir cada una de ellas no tendrán una copia de ese atributo, si no que todas comparten una misma copia del atributo, a los atributos estáticos se les llama **atributos de clase**

Los atributos tienen que ser PRIVADOS

GET para devolver el valor

SET para modificar

El encapsulamiento es la **ocultación de información**

El método constructor es NO ESTATICO, no tiene un valor de retorno, (Una clase por defecto tiene un constructor)

y tiene que ser público es un método que se ejecuta automáticamente para poner un estado inicial al objeto

Toda la clase object tiene los siguientes métodos:

* **toString**: Devuelve en forma de texto los valores de los atributos (no devuelve parámetros (El **@Override**, es un método que se está redefiniendo a otro método)
* **. equals ()**: Compara los atributos en memoria
* **getClass**: Sirve para saber cuál es la clase del objeto

COMPARACIONES ENTRE OBJETOS

* . toequals:
* . compareTo:

EXCEPCIONES

Una excepción es una condición anomala que ocurre mediante la ejecución del programa, deben ser manejadas por el programación, a proporcionar un código llamado **manejador de exapciones** (con **try catch**)