RESUMEN TEORICO POO

RESUMEN - TEORÍA POO

a tu wacha

MÉTODO → Una operación concreta de una determinada clase.

MECANISMOS (MOMC):

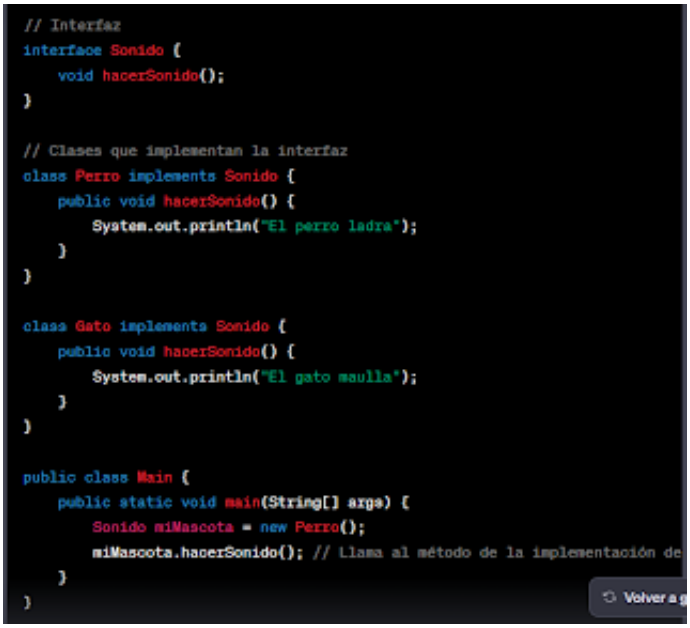
1. Mensajes
2. Métodos
3. Objetos
4. Clases

CARACTERÍSTICAS (HAPE):

1. Herencia
2. Abstracción
3. Polimorfismo
4. Encapsulamiento

POLIMORFISMO → Es la capacidad de los distintos objetos (pertenecientes a distintas clases) de reaccionar a un mismo mensaje, cada uno a su manera.

Ej: heredar un método de las clase padre a las clases hijas y que estas lo implementen de distintas maneras, tmb con interfaces.



```java
// Interfaz
interface Sonido {
    void hacerSonido();
}

// Clases que implementan la interfaz
class Perro implements Sonido {
    public void hacerSonido() {
        System.out.println("El perro ladra");
    }
}

class Gato implements Sonido {
    public void hacerSonido() {
        System.out.println("El gato maulla");
    }
}

public class Main {
    public static void main(String[] args) {
        Sonido miMascota = new Perro();
        miMascota.hacerSonido(); // Llama al método de la implementación de
    }
}
```

CLASE → Una descripción de un conjunto de objetos similares. No se puede hablar de “generalizacion” de objetos, sino es una estructura de los objetos.

OBJETOS → Son ejemplares de una clase.

HERENCIA → Es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.

ABSTRACCIÓN → Es la generalización conceptual de los atributos y métodos de un determinado conjunto de objetos.

Hace referencia a quitar las propiedades y acciones de un objeto para dejar solo aquellas que sean necesarias para la resolución del problema en cuestión.

Consiste en abstraer los métodos y los datos comunes a un conjunto de objetos y almacenarlos en una clase.

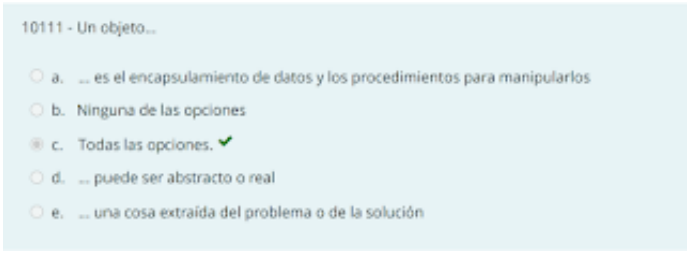
TIPOS DE VISIBILIDAD → Private, public, protected

ENCAPSULAMIENTO → es uno de los cuatro conceptos fundamentales de la programación orientada a objetos (junto con la

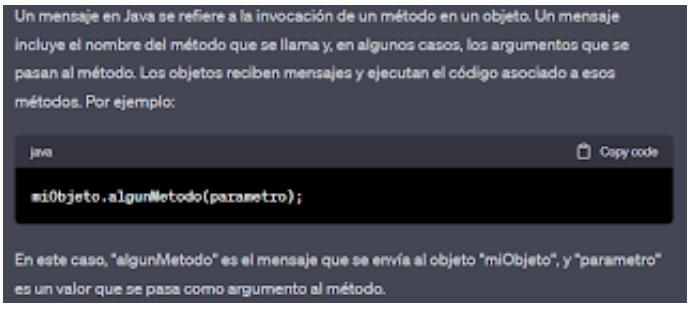
abstracción, la herencia y el polimorfismo) y se refiere a la práctica de ocultar los detalles internos de un objeto y exponer solo la interfaz pública a otros objetos. El objetivo principal del encapsulamiento es proteger los datos internos de una clase y controlar el acceso a ellos a través de métodos públicos.

En Java, por ejemplo, el uso de modificadores de acceso como private: visible solo en la misma clase, protected visible desde dentro de la misma clase, subclases y clases del mismo paquete, public: visible desde cualquier lugar permite implementar el encapsulamiento. Los campos se declaran como privados, y los métodos públicos (getters y setters) se utilizan para interactuar con esos campos.

DE MEMORIA PABLO PUTO



MENSAJES → La forma de comunicarse e interactuar que tienen los objetos



Que es la programación orientada a objetos?

Un enfoque que se basa en la creación y manipulación de objetos

Cuales son las ventajas del uso de la POO versus la descomposición

algorítmica? → Reusabilidad y Adaptación al cambio

Los objetos de una misma clase se distinguen entre si → por sus atributos

Las VARIABLES puede ser de 4 tipos:

De instancias o objetos: Definidas dentro de un objeto

De clases: Son las " static ".

Locales: son las creadas dentro de un metodo, como un contador

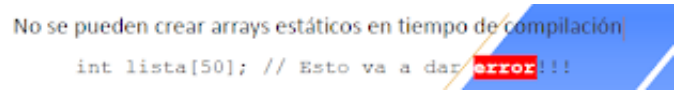
Parametros: las que son pasadas como parametro dentro de un metodo.

Pueden ser: tipo primitivo de datos, vectores, clases.

TIPOS DE DATO:

Tipos Datos	Tamaño (bits)	Valor inicial	Rango
byte	8	0	-2^8 a 2^8-1 -128 a 127
short	16	0	-2^{16} a $2^{16}-1$ -32768 a 32767
int	32	0	-2^{32} a $2^{32}-1$ 2.147.483.648 a 2.147.483.64
long	64	0	-2^{64} a $2^{64}-1$ $-9*10^{18}$ a $9*10^{18}$
float	32	0.0F	Simple precisión $-3.4*10^{38}$ a $3.4*10^{38}$
double	64	0.0D	Doble precisión $-1.79*10^{308}$ a $1.79*10^{308}$
char	16	\0	ASCII Unicode
boolean	8	false	

VECTORES: Clase arrayList con sus propios metodos como size, length, add, remove. Es un tipo de dato por ser un objeto de una clase.



CONSTANTES LOCALES → NO existen en java

Existen constantes de instancias anteponiendo la clausula final.

final int CONST_INT= 10;

CASTING → Cast o casting es la conversión de un tipo de dato a otro y se utiliza para asegurarse de que un valor se almacene o se interprete correctamente

Cast Implícito: es automático

La jerarquía en las conversiones de menor a mayor es:

byte → short →int → long → float → double

Cast Explícito: forzado por el programador, indicando el nuevo tipo entre paréntesis.

TIPOS DE OPERADORES

Matemáticos
De Asignación
Incremento/Decremento
De Comparación
Lógicos
De Bits
De Bits y de Asignación

BIBLIOTECAS DE CLASE → Por que usarlas / ventajas / caracteristicas

JAVA ofrece un conjunto de clases disponibles y garantizadas para ser usadas en cualquier ambiente JAVA.

Reutilización de código : contienen código preescrito y probado. Permite reutilizar funcionalidades comunes sin reescribir código, ahorro de tiempo y esfuerzo.

Funcionalidad amplia : manipulación de archivos, manejo de E/S, procesamiento de cadenas, matemáticas, redes, GUI, bases de datos, etc.

Consistencia : Están diseñadas siguiendo estándares y prácticas recomendadas, lo que garantiza la consistencia y la calidad en el código.

Documentación : Suelen ir acompañadas de documentación detallada que describe cómo usar sus componentes, sus parámetros y su funcionalidad. Esto facilita la comprensión y el uso de la biblioteca.

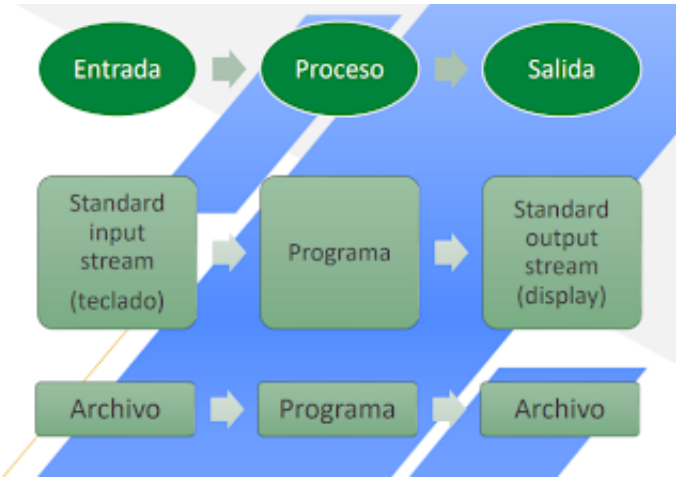
Amplia comunidad de usuarios : Generalmente tienen una comunidad activa de usuarios y desarrolladores que comparten conocimientos y experiencias, lo que facilita la resolución de problemas y el aprendizaje.

Portabilidad : Las bibliotecas de clases suelen estar disponibles en múltiples plataformas y sistemas

operativos, lo que permite desarrollar software portátil que funcione en diferentes entornos.

DATOS DE ENTRADA Y DE SALIDA

Stream: flujo secuencial de datos. : conjunto de facilidades asociadas a los procedimientos de conversión de objetos en secuencias de caracteres y viceversa.



FUNCIONES: Tipo de acceso y visibilidad → Public, private, protected, package (default)

PARAMETROS DINAMICOS → Se usa cuando se desconoce el numero de variables a pasar.

Ejemplo:

```
int calculo (int a, int b, int ... numero){
    int suma = 0;
    for(int i = 0; i < numero.length; i++)
        { suma += numero[i]; }
return (a*b)+suma; }
```

```
public static void main (String[] args)
```

public: puede ser llamada desde cualquier otro método, incluyendo el intérprete de JAVA.

static: es un método de la propia clase y no de la instancia.

void: el tipo de dato que retorna la función. La función main no retorna nada.

String[] args: Los argumentos pasados al programa se encuentran en un array de String.

Sobrecarga de funciones (overloading) → Cuando existen múltiples definiciones para una misma función. Java reconoce que definición utilizar en base a la cantidad y tipos de sus argumentos.

MAIN

A diferencia de C, JAVA no almacena en la primera posición del array (args [0]) el nombre del programa ejecutado, sino que guarda el primer argumento informado.

CLASES (1):

Una clase es un tipo de dato definido por el programador que contiene datos y funciones, llamados miembros de la clase.

• Los datos miembros definen el estado (atributos y valores) de cada objeto.

• Las funciones miembros definen el comportamiento de cada objeto (métodos).

Restrictores de una clase: public,private,protected, packkage

Constructor de un objeto

Un constructor es una función miembro especial de la clase, que es llamada al instanciar un objeto.

Su objetivo es crear e inicializar un objeto de forma tal que contengan valores válidos.

Si no se declara un constructor, JAVA crea uno por default sin argumentos e inicializando todos sus datos miembros con valores por defecto acorde al tipo de dato.

El constructor, secuencial y recursivamente, reserva memoria para cada uno de los datos miembros.

Posee el mismo nombre que la clase a la que pertenece.

No retorna ningún valor y no puede ser declarado estático.

Se pueden tener constructores múltiples o alternativos con el mismo nombre y diferentes argumentos , para inicializar el estado de un objeto de distintas formas (sobrecarga de funciones).

Herencia

Mecanismo que permite definir clases derivadas a partir de otras clases ya existentes.

La clase derivada o clase hija o subclase hereda todas las propiedades de la clase existente, que recibe el nombre de clase base o clase padre o super clase

Puede ser, a su vez, una clase base.

Hereda todos los miembros de la clase base.

Puede acceder a los miembros public y protected de la clase (o clases) bases como si fueran miembros de la misma.

No tiene acceso a los miembros private de la clase base.

Añade a los miembros heredados sus propios miembros.

Los miembros heredados por una clase derivada pueden, a su vez, ser heredados por más clases derivadas de ella.

El mensaje se envía desde una subclase y se transmite a las super clases través de

la jerarquía definida hasta encontrar una definición del método.

OVERRIDING

es la forma de crear un método en la subclase que tenga la misma identificación de un método en una superclase

Este nuevo método oculta el método de la superclase.

super → llama a un metodo definido en una superclase

this → hace referencia a los miembros de la instancia actual

DESTRUCCIÓN DE OBJETOS

Garbage Collector → forma automática de liberar memoria.

El Garbage Collector en tiempo de ejecución rastrea cada objeto creado, detecta aquellos que ya no son referenciados y libera el objeto en caso de ser necesario.

Ventajas

detecta objetos no utilizados

reutilización de la memoria

no necesita explícita liberación (destructores)

Desventaja

incrementa el tiempo de ejecución

Método de finalización → finalize()

Java interpreta que tiene que limpiar la memoria porque el objeto ya no es alcanzable por ninguna referencia y está listo para ser eliminado de la memoria.

Se utilizan para:

optimizar la remoción de un objeto
eliminar las referencias a otro objeto
liberar recursos externos

Cláusula *final*

Impide la redefinición de métodos, la extensión completa de clases y el cambio de valores de los atributos.

Una clase final es aquella de la cual no se puede heredar.

Clases y métodos abstractos

Un método abstracto es un método declarado en una clase para el cual esa clase no proporciona la implementación (no tiene código).

Están disponibles en las clases derivadas, las cuales deben definir al mismo

Una clase abstracta es aquella que tiene al menos un método abstracto .

Una clase que extiende a una clase abstracta debe implementar los métodos

abstractos , o bien, volverlos a declarar como abstractos, con lo que ella misma

se convierte también en clase abstracta.

Pero una clase abstracta no se puede instanciar (no se pueden crear objetos).

No tiene completa su implementación y algo abstracto no puede materializarse

¿Para qué sirve entonces? Para implementar el Polimorfismo de herencia

Miembros Static

Un dato miembro de una clase declarado static implica que sólo existirá una copia de ese miembro , la cual es compartida por todos los objetos de la clase

También conocida como Variable de Clase, existe aunque no existan objetos de esa clase

La inicialización de un miembro estático es igual que la de cualquier dato miembro de la clase, anteponiendo la palabra reservada static

Debe referenciarse sobre el nombre de la clase directamente, aunque JAVA permite hacerlo desde un objeto.

Paquetes

Permite agrupar clases e interfaces.

Visibilidad	Public	Protected	Private	Package
Propia Clase	*Si*	*Si*	*Si*	*Si*
Otra Clase del Paquete	*Si*	*Si*	*No*	*Si*
Otra Clase Fuera del Paquete	*Si*	*No*	*No*	*No*
Clase Derivada dentro del Paquete	*Si*	*Si*	*No*	*Si*
Clase Derivada fuera del Paquete	*Si*	*Si*	*No*	*No*

Relaciones de clases

Asociacion → Composición y agregación. Clases compuestas de otras clases. Rombo blanco y negro.

COMPOSICIÓN → La clase principal compone un objeto nuevo

de la clase relacionada (clase secundaria o asociada). En general, el objeto miembro de la clase relacionada no tiene sentido si no existe la clase principal.

AGREGACIÓN → Se le pasa el objeto ya creado por parámetro. Una clase es independiente de la otra.

toString() → Método de la clase object. Devuelve una representación en string del objeto. Mejora la visualización del contenido.

Interfaces

→ Expresiones puras de diseño.

→ Son conceptualizaciones no implementadas que sirven de guía para definir un

determinado concepto y su comportamiento , pero sin desarrollar un mecanismo de

solución.

VENTAJAS:

Obligar a que ciertas clases utilicen los mismos métodos

(nombres y parámetros).

Es un mecanismo para abstraer métodos a un nivel superior.
Establecer relaciones entre clases que no estén relacionadas
Implementar herencia múltiple.
Organizar la programación.

En JAVA NO esta soportada la herencia múltiple, por eso se usan las interfaces.

Diferencias entre una clase y una interfaz:

Se emplea la palabra reservada “ interface ” en lugar de class
Algunos de sus métodos pueden no estar definidos, solamente declarados (métodos abstractos).
Puede contener métodos implementados, llamados métodos default o métodos de extensión virtual (desde JAVA 8).
Sus atributos deben ser tratados como public , static y final
Las interfaces sólo admiten los modificadores de acceso public y package.
Los métodos son implícitamente public y abstract. • Los atributos son implícitamente public, final y static.

Más características:

Las interfaces carecen de funcionalidad dado que sus métodos, generalmente, no están definidos. Necesitan

algún mecanismo que sí lo haga: las clases.

Las interfaces deben ser implementadas a través de clases, las cuales son las encargadas de proporcionar la definición de sus métodos.

Si una clase no sobrescribe alguno de sus métodos abstractos, automáticamente, esta clase se convierte en abstracta.
Una interfaz define un tipo de conducta de la clase que la implementa: asume las constantes de la interfaz y codifica sus métodos.
Para indicar que una clase implementa una interfaz, se agrega la palabra reservada “ implements ” en la definición de la clase.
Una clase no puede tener más que una clase antecesora (no hay herencia múltiple entre clases
Una clase puede implementar más de una interfaz

Una interfaz no puede implementar otra interfaz , aunque sí extenderla (extends)
Una interfaz puede heredar de más de una interfaz antecesora
Entre interfaces, está admitida tanto el uso de herencia simple como de herencia múltiple.
La interfaz derivada incluye todas las constantes y declaraciones de métodos de la o las interfaces bases.

INTERFACES VS CLASES ABSTRACTAS

Una clase abstracta puede incluir métodos implementados y no implementados o

abstractos y miembros constantes y otros no constantes. Las interfaces contienen una lista de métodos que pueden o no estar implementados e incluir la declaración de algunos miembros constantes.

Una clase no puede heredar de dos clases abstractas, pero sí puede heredar de una clase abstracta e implementar una interface, o implementar dos o más interfaces. Las interfaces tienen una jerarquía propia, permitiéndose la herencia múltiple.

Las interfaces proporcionan un mecanismo de encapsulación de los protocolos de los métodos sin forzar el uso de la herencia, asociando clases sin relación alguna.

Es decir → La idea clave aquí es que las interfaces permiten que diferentes clases proporcionen su propia implementación de estos métodos sin necesidad de heredar de una clase común. Esto significa que clases que no tienen una relación de herencia directa pueden compartir comportamientos comunes definidos por la interfaz.