UNIVERSIDAD DE LAS FUERZAS ARMADAS

NOMBRE: Renata Alejandra Salazar Calva
FECHA: 14 / 11 / 2024
NRC: 13627
MATERIA: Programación Orientada a Objetos

CONSULTA: (1)

¿Qué es el paradigma de la programación orientado a objetos?

Es un modelo o estilo de programación que ofrece instrucciones sobre como manejarlo y se fundamenta en el concepto de clases y objetos. Este método se utiliza para organizar componentes sencillos y reutilizables de piezas de código (clases), con el objetivo de generar instancias únicas de objetos. El objetivo del paradigma de POO es abandonar el enfoque en la lógica pura de los programas y empezar a pensar en objetos, construyendo así la base de este paradigma. Esto resulta beneficioso en sistemas grandes, ya que en vez de centrarse en funciones, se consideran las relaciones o interacciones entre los componentes

- PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS:

- LA ENCAPSULACIÓN → Esta almacena toda la información relevante de un objeto en su interior y únicamente revela la información seleccionada al exterior. Esta propiedad garantiza que los datos de un objeto se mantengan ocultos para el exterior, recopilando en una categoría los atributos que poseen un acceso privado y los conductos o procedimientos que poseen un acceso público.

- LA ABSTRACCIÓN → En la programación Orientado a Objetos, los programas tienden a ser de gran tamaño y los objetos interactuán mucho entre ellos. Así, la abstracción simplifica la conservación de un código de gran tamaño en el que pueden aparecer diversas modificaciones con el transcurso del tiempo. Por lo tanto, la abstracción se fundamenta en emplear elementos sencillos para simbolizar la complejidad. los Objetos y clases simbolizan el código soporte, escondiendo los detalles, escondiendo los detalles intrincados al usuario.

- LA HERENCIA → La herencia establece vínculos jerárquicos entre clases, permitiendo que atributos y procedimientos compartidos sean reutilizados. Las clases principales amplían atributos y conductos básicas, se pueden generar clases principales y añadiendo atributos y conductas adicionales. es uno de los elementos fundamentales de la Programación Orientado a Objetos

- EL POLIMORFISMO → Este se basa en la creación de objetos con comportamientos similares, lo que facilita el procesamiento de objetos de diferentes formas se refiere a la habilidad de mostrar lo mismo interfaz para diferentes formas subyacentes o tipos de información. Mediante el uso de la herencia, los objetos tienen la capacidad de anular las conductas Principales compartidas, generando comportamiento secundarios particulares. El polimorfismo posibilita que el mismo procedimiento lleve a cabo dos comportamientos diferentes: la anulación de método y la sobrecarga de este.

Qué es una clase, un objeto, un atributo y un método?

CLASE → es una plantilla o un modelo que define las propiedades y comportamientos comunes de un conjunto de objetos, por lo tanto una clase específica tus datos (atributos) y que funciones (métodos) tendrán los objetos que se crean a partir de esa.

EJEMPLO: La clase "Empleado", se define como atributos a "nombre" y "salario" y métodos como "está trabajando".

OBJETO → es una instancia de una clase, representa una entidad concreta que tiene un estado (valores de atributos) y un comportamiento (métodos) esto quiere decir si la clase es el plano, el objeto es el edificio construido a partir de ese plano.

EJEMPLO: La clase "vehículo" especifica podría ser un modelo "corolla" la marca "Toyota" y el año especifico "2021".

ATRIBUTO → es una variable dentro de una clase que define las propiedades o características de los objetos creados a partir de esa clase. Los atributos representan el estado de un objeto.

EJEMPLO La clase "Mascota", los atributos "nombre", "tipo" y "edad"

MÉTODO → es una función definida dentro de una clase que describe los comportamientos o acciones que los objetos de esa clase pueden realizar. Los métodos pueden manipular los atributos de la clase y definir cómo interactúan los objetos.

EJEMPLO En el objeto "Persona" los métodos son "saludar", "Hola, mi nombre es".

Qué es un sistema de control de versionamiento y para que sirve?

Son un tipo de software que ayuda a hacer un seguimiento de los cambios realizados en el código a lo largo del tiempo. A medida que un desarrollador edita el código, el sistema de control de versiones toma una instantánea de los archivos. Después guarda esa instantánea de forma permanente para que se pueda recuperar más adelante si es necesario.
Esto sirve para:

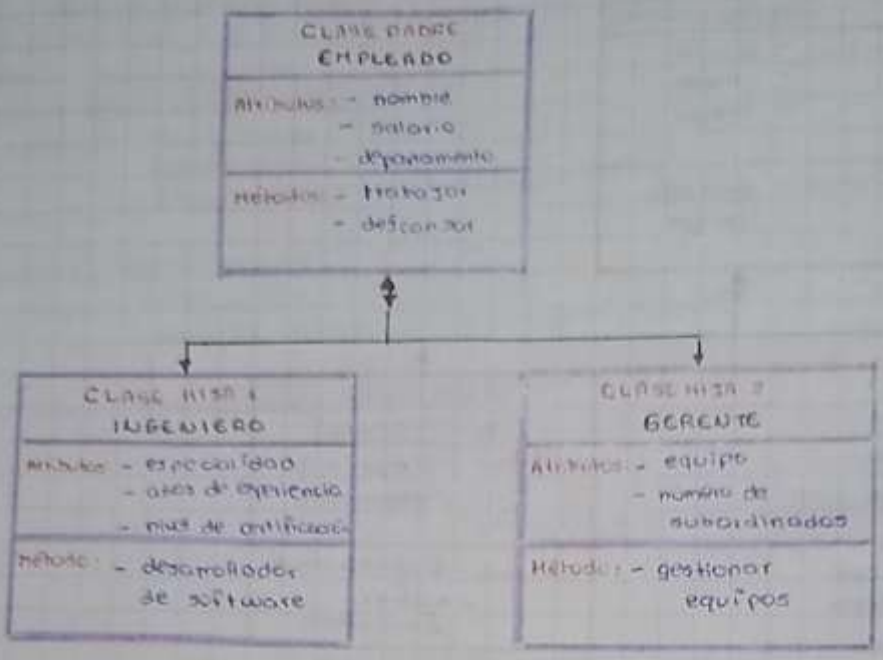
- Crear flujos de trabajo: evitar el caos a todos los usuarios que usan su propio proceso de desarrollo con herramientas diferentes e incompatibles. Los sistemas de control de versiones proporcionan permisos y cumplimiento de procesos, por lo que todos permanecen en sintonía.

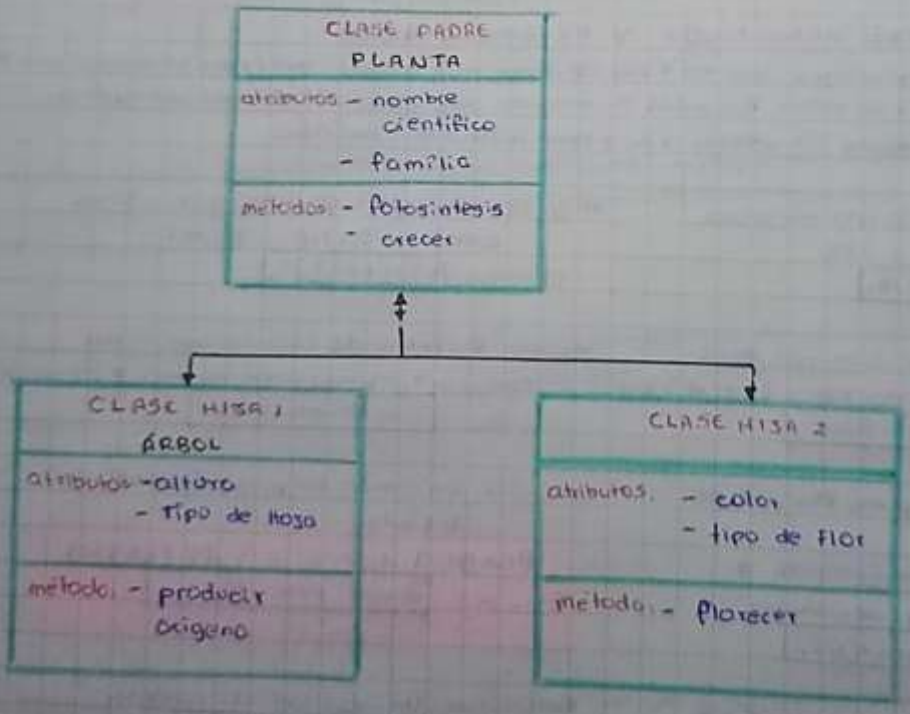
- Codificar: el control de versiones sincroniza las versiones y garantiza que los cambios no entren en conflicto con los cambios de otros usuarios. El equipo se basa en el control de versiones para ayudar a resolver y evitar conflictos, incluso cuando los usuarios realizan cambios al mismo tiempo.

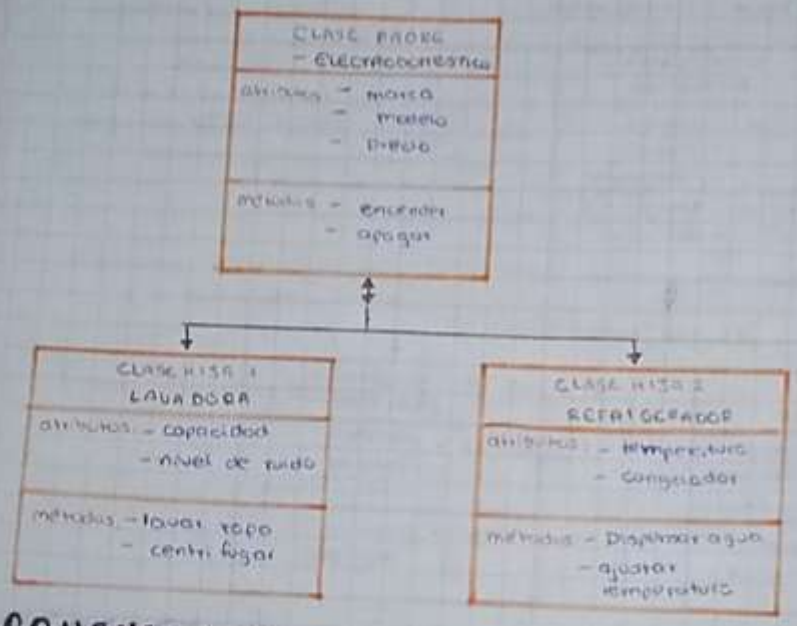
- Mantener un historial: El control de versiones mantiene un historial de los cambios a medida que el equipo guarda nuevas versiones del código. Los miembros del equipo pueden revisar el historial para averiguar la persona que realizó los cambios, porque los hizo y en qué momento.

- Automatización de tareas: Las características de automatización del control de versiones ahorran tiempo y generan resultados coherentes. La automatización de pruebas, el análisis de código y la implementación cuando se guardan nuevas versiones en el control de versiones con solo tres ejemplos.

Hacer 3 UML 2 clases hijas y 1 clase padre







CONSULTA 2:

TIPOS DE DATOS PRIMITIVOS Y REFERENCIADOS

- Los tipos de datos primitivos son los tipos de datos más básicos que están integrados en el lenguaje y no están basados en ningún otro clase. Estos tipos almacenan directamente los valores y no tienen métodos asociados.

1) BYTE: Un entero de 8 bits con signo.
Rango: -128 a 127
EJEMPLO: `byte edad = 25;`

2) SHORT: Un entero de 16 bits con signo
Rango: -32,768 a 32,767
EJEMPLO: `short año=2023;`

3) INT: Un entero de 32 bits con signo
Rango: -2,147,483,648 a 2,147,483,647
EJEMPLO: `int salario = 50000;`

4) LONG: Un entero de 64 bits con signo
Rango: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
EJEMPLO: `long distancia = 8L;`

5) FLOAT: Un número de punto flotante de 32 bits.
Precisión: Aproximadamente 7 dígitos decimales
EJEMPLO: `float altura: 5,8f;`

6) DOUBLE: Un número de punto flotante de 64 bits
RANGO: Aproximadamente 15 dígitos decimales
EJEMPLO: `double pi = 3.5;`

7) CHAR: Un único carácter Unicode de 16 bits.
Rango: '\u0000' a '\uffff' (0, 535)
EJEMPLO: `char inicial = 'A';`

8) BOOLEAN: Un valor que solo puede ser `true` o `false`
EJEMPLO: `boolean esMayor = true;`

- Los tipos referenciados son más complejos y están basados en clases. En lugar de almacenar directamente los valores, estos tipos almacenan referencias (direcciones de memoria) a los objetos que contienen los valores.

TIPOS DE REFERENCIADOS INCLUYEN:

1) CLASES: `String`, `Integer`, `ArrayList`, etc.
   Ejemplo: `String nombre = "Juan";`
   `Integer edad = 25;`
   `ArrayList<String> lista = new ArrayList<>();`

2) ARRAYS: Colecciones del mismo tipo
   Ejemplo: `int[] numeros = {1, 2, 3, 4, 5};`
   `String[] nombres = {"Ana", "Carlos", "Luis"};`

3) INTERFACES como `Runnable`, `Serializable` etc.
   EJEMPLO: `Runnable tarea = new Runnable() {`
   `@Override`
   `public void run() {`
   `System.out.println("Tarea en ejecución");`
   `}`
   `};`

4) ENUMERACIONES (Enums): Tipos especiales que representan un conjunto de constantes
   EJEMPLO: `public enum Dia {`
   `LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO`
   `}`

TIPOS DE DATOS ESTÁTICOS

Los datos estáticos generalmente se refieren a variables y métodos que se declaran con la palabra clave `static`. En Java, estos datos están asociados a la clase en sí, en lugar de a instancias específicas de la clase.

- VARIABLES ESTÁTICAS: son compartidas por todas las instancias de una clase. Esto quiere decir que todas las instancias de la clase acceden a la misma copia de la variable.

```
EJEMPLO: public class Contador {
             public static int cuenta = 0;

             public Contador() {
                 cuenta++;
             }
         }
```

- Métodos estáticos: pertenecen a la clase en lugar de a las instancias de la clase. Esto quiere estos métodos solo pueden acceder a variables estáticas y llamar a otros métodos estáticos.

```
EJEMPLO: public class Utilidades {
             public static int sumar(int a, int b) {
                 return a+b;
             }
         }
```

- CARACTERISTICAS

- Almacenamiento Común: Una variable se almacena en una área de memoria común y es accesible para todo tipo de clases.
- Inicialización: Las variables estáticas se inicializan una vez, cuando la clase se carga por primera vez.
- Uso sin instancia: Tanto las variables como los métodos estáticos se pueden usar sin necesidad de crear una instancia de la clase.
- Acceso directo: Se accede a variables y métodos estáticos utilizando el nombre de la clase, seguido del operador (.).

- USOS COMUNES DE DATOS ESTÁTICOS

- Contadores: Para contar el número de instancias creadas en la clase de una clase.
- Constantes: Para definir valores constantes que son comunes a todas las instancias de la clase.
- Métodos Utilitarios: Para crear métodos que realizan las operaciones comunes y no dependen de los estados de los objetos.

TIPOS DE DATOS DINÁMICOS

Son aquellos cuya estructura o tamaño pueden cambiar durante la ejecución del programa. Los datos dinámicos son más flexibles y pueden adaptarse a la necesidad del programa en tiempo de ejecución.

- TIPOS

1) Listas Enlazadas (Linked Lists): Es una colección de nodos donde cada nodo contiene un dato y una referencia o enlace al siguiente nodo de la secuencia. Las listas enlazadas permiten la inserción y eliminación eficiente de elementos.

Uso: manejo de datos dinámicos donde el tamaño de la colección puede cambiar frecuentemente

EJEMPLO:
```
import java.util.LinkedList;
LinkedList<String> listaEnlazada = new LinkedList<>();
listaEnlazada.add("A");
listaEnlazada.add("B");
listaEnlazada.remove("A");
System.out.println(listaEnlazada); // Muestra: [A, B]
```

2) Listas (List): son estructuras de datos que pueden crecer o reducir en la ejecución de el programa su tamaño también permiten almacenar elementos en un orden secuencial.

Uso: Almacenamiento y manipulación de colecciones de datos con tamaño variable.

Ejemplo:
```
import java.util.ArrayList;

ArrayList<Integer> miLista = new ArrayList<>();
miLista.add(1);
miLista.add(2);
miLista.add(3);
miLista.add(0);
System.out.println(miLista); // Muestra: [1,2,3]
```

3. Diccionarios (dictionary) o Maps (Maps). Un diccionario es una estructura de datos que almacena pares clave-valor. Permite el acceso rápido a los valores a través de sus claves.

Uso: almacenamiento de datos asociados con claves únicas.

EJEMPLO: import java.util.HashMap;

```
public class EjemplosMap {
    public static void main(String[] args) {
        HashMap<String, String> mapa= new HashMap<>();
        mapa.put("Nombre", "Carlos");
        mapa.put("Edad", "28");
        System.out.println(mapa);
    }
}
```

4. Conjuntos (Set) es una colección de elementos únicos y sin orden específico estos permiten operaciones matemáticas como la unión, la intersección y la diferencia.

Uso: Almacenamiento de colecciones de datos únicos y realización de operaciones de conjuntos.

EJEMPLO: import java.util.HashSet;

```
HashSet<Integer> conjunto = new HashSet<>();
conjunto.add(1);
conjunto.add(2);
conjunto.add(3);
conjunto.remove(2);
System.out.println(conjunto);
```

5. Pilas (Stack) y Colas (Queue).

- PILAS (Stack): Siguen el principio LIFO (Último en entrar, primero en salir).

EJEMPLO: import java.util.Stack;

```
Stack<Integer> pila = new Stack<>();
pila.push(1);
pila.push(2);
pila.push(3);
pila.pop();
System.out.println(pila); // Muestra [1]
```

- COLAS (Queue): Siguen el principio FIFO (Primero en entrar, primero en salir)

EJEMPLO:
```
import java.util.LinkedList;
import java.util.Queue;

Queue<Integer> cola = new LinkedList<>();
cola.add(1);
cola.add(2);
cola.poll();
System.out.println(cola); // Muestra [2]
```