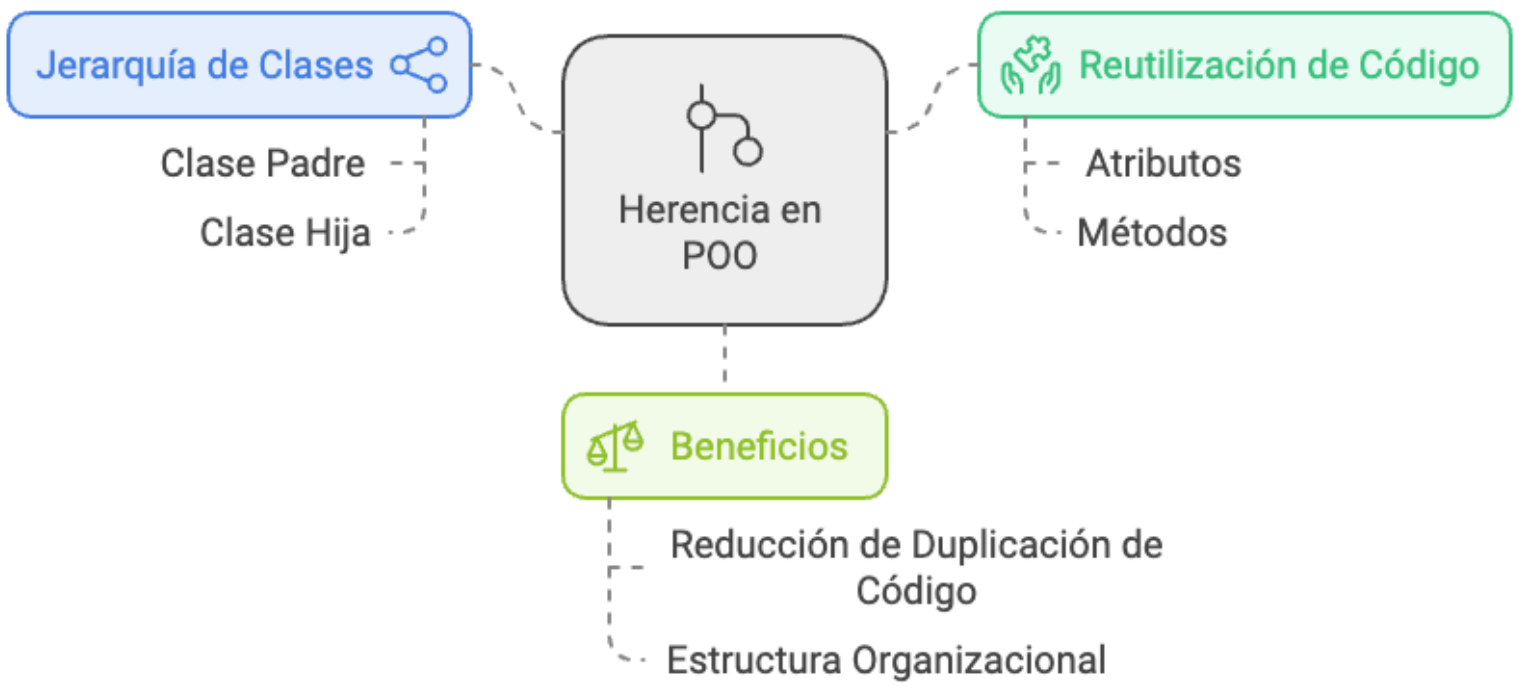
Concepto de Herencia

La herencia es uno de los principales conceptos de la programación orientada a objetos (POO). Permite que una clase herede los atributos y métodos de otra clase, llamada clase padre o superclase. La clase que hereda se conoce como clase hija o subclase. La herencia facilita la reutilización del código y establece una relación jerárquica entre las clases.



Aplicando la Herencia.

Veamos ahora como podemos empezar a aplicar el concepto de la Herencia.

Crear una clase padre (superclase)

Comencemos creando una clase padre básica. Imaginemos que estamos construyendo una aplicación de gestión de empleados y queremos tener una clase base llamada "Empleado" de la cual otras clases específicas de empleados heredarán sus atributos y métodos.

```
public class Empleado {
    private String nombre;
    private String apellido;
    private int edad;

    // Constructor
    public Empleado(String nombre, String apellido, int edad) {
            this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }

    // Getters y setters
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
```

```
    return apellido;
}

public void setApellido(String apellido) {
    this.apellido = apellido;
}

public int getEdad() {
    return edad;
}

public void setEdad(int edad) {
    this.edad = edad;
}

// Método para obtener detalles del empleado
public void obtenerDetalles() {
    System.out.println("Nombre: " + nombre);
    System.out.println("Apellido: " + apellido);
    System.out.println("Edad: " + edad);
  }
}
```

En este ejemplo, la clase "Empleado" tiene tres atributos: nombre, apellido y edad. También tiene un constructor para inicializar estos atributos y métodos para obtener y establecer los valores de los atributos, así como un método para imprimir los detalles del empleado.

Crear una clase hija (subclase)

Ahora crearemos una clase hija llamada "Desarrollador" que herede de la clase "Empleado". Esta clase hija tendrá atributos y métodos adicionales específicos para los desarrolladores.

```
public class Desarrollador extends Empleado {
    private String lenguajePrincipal;

    // Constructor
    public Desarrollador(String nombre, String apellido, int edad, String lenguajePrincipal) {
        super(nombre, apellido, edad);
        this.lenguajePrincipal = lenguajePrincipal;
    }

// Getter y setter para el lenguaje principal
public String getLenguajePrincipal() {
    return lenguajePrincipal;
}

public void setLenguajePrincipal(String lenguajePrincipal) {
    this.lenguajePrincipal = lenguajePrincipal;
}

// Método para obtener detalles del desarrollador
@Override
public void obtenerDetalles() {
    super.obtenerDetalles();
    System.out.println("Lenguaje principal: " + lenguajePrincipal);
}
```

```
}
```

La clase "Desarrollador" extiende la clase "Empleado" mediante la declaración `extends Empleado`. Además de los atributos heredados, tiene un atributo adicional llamado "lenguajePrincipal". En el constructor de la clase "Desarrollador", utilizamos la palabra clave "super" para llamar al constructor de la clase padre y establecer los atributos heredados. También tenemos getters y setters para el atributo "lenguajePrincipal". Además, hemos sobrescrito el método "obtenerDetalles()" de la clase padre utilizando la anotación `@Override`, para añadir la información específica del desarrollador.

Usar las clases en un programa

Ahora podemos usar nuestras clases "Empleado" y "Desarrollador" en un programa principal para demostrar la herencia en acción. Aquí hay un ejemplo:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de la clase Empleado
        Empleado empleado = new Empleado("John", "Doe", 30);
        empleado.obtenerDetalles();

        System.out.println("----------------");

        // Crear un objeto de la clase Desarrollador
        Desarrollador desarrollador = new Desarrollador("Jane", "Smith", 25, "Java");
        desarrollador.obtenerDetalles();
    }
}
```

En este programa de ejemplo, creamos un objeto de la clase "Empleado" y llamamos al método "obtenerDetalles()" para mostrar los detalles del empleado. Luego, creamos un objeto de la clase "Desarrollador" y también llamamos al método "obtenerDetalles()". Como la clase "Desarrollador" hereda de la clase "Empleado", podemos acceder a los atributos y métodos de ambas clases.

El resultado de ejecutar este programa será:

```
Nombre: John
Apellido: Doe
Edad: 30
----------------
Nombre: Jane
Apellido: Smith
Edad: 25
Lenguaje principal: Java
```

Como puedes ver, la clase "Desarrollador" hereda los atributos y métodos de la clase "Empleado" y añade su propio atributo y método específico.

Beneficios de la Herencia.

La herencia ofrece varios beneficios en la programación orientada a objetos. Aquí están algunos de los beneficios clave:

1. Reutilización de código: La herencia permite la reutilización de código al permitir que las clases hijas hereden atributos y métodos de la clase padre. Esto evita la duplicación de código y fomenta la eficiencia y el mantenimiento del código.
2. Extensibilidad: La herencia permite extender y ampliar el comportamiento de una clase existente. Al crear una nueva clase hija, se pueden agregar nuevos atributos y métodos adicionales sin modificar la clase padre original.
3. Polimorfismo: La herencia es fundamental para lograr el polimorfismo, que es la capacidad de objetos de diferentes clases relacionadas por herencia para responder de manera diferente al mismo mensaje o llamada de método. Esto permite tratar a los objetos de diferentes clases de manera uniforme y brinda flexibilidad en el diseño y la implementación de sistemas.

4. Organización y jerarquía: La herencia establece una relación jerárquica entre las clases. Puede crear una jerarquía de clases donde las clases más específicas (subclases) heredan características comunes de las clases más generales (superclases). Esto facilita la organización y estructuración del código, lo que resulta en un diseño más claro y comprensible.
5. Abstracción y encapsulación: La herencia ayuda a lograr los principios de abstracción y encapsulación. Permite definir una clase base abstracta que encapsula el comportamiento común y los atributos compartidos por varias clases hijas. Esto promueve una mayor modularidad y facilita la comprensión y el mantenimiento del código.

Estos son solo algunos de los beneficios de la herencia en la programación orientada a objetos. Sin embargo, es importante usar la herencia de manera adecuada y consciente para evitar complicaciones y problemas de diseño.

Instructor: Cristian David Henao Hoyos