**Tutorial Interactivo: Herencia y Polimorfismo en Java**

En este tutorial, aprenderás los conceptos de **herencia** y **polimorfismo** en Java, dos principios clave de la programación orientada a objetos. Cada sección incluye ejemplos, preguntas y tareas prácticas que te ayudarán a aplicar los conceptos aprendidos.

**1. Concepto de Herencia**

**Definición:**

La **herencia** es un mecanismo en Java que permite a una clase heredar las propiedades (atributos) y comportamientos (métodos) de otra clase. La clase que hereda se llama **subclase** o **clase derivada**, y la clase de la que hereda se llama **superclase** o **clase base**.

**Sintaxis de Herencia:**

java

Copiar código

class SuperClase {

// Atributos y métodos de la superclase

}

class SubClase extends SuperClase {

// Atributos y métodos adicionales de la subclase

}

**Ejemplo:**

java

Copiar código

// Clase base

class Animal {

String nombre;

public void hacerSonido() {

System.out.println("El animal hace un sonido");

}

}

// Clase derivada

class Perro extends Animal {

public void hacerSonido() {

System.out.println("El perro ladra");

}

}

En este ejemplo:

* La clase Perro hereda los atributos y métodos de la clase Animal. Sin embargo, sobrescribe el método hacerSonido() con su propio comportamiento.

**Tarea 1:**

Crea una clase base llamada **Vehiculo** con el atributo **velocidadMaxima** y el método **mover()**. Luego, crea una clase derivada llamada **Coche** que sobrescriba el método **mover()** para mostrar "El coche se mueve".

**Pregunta 1:**

¿Cuál es la diferencia entre una superclase y una subclase? ¿Qué se hereda y qué no?

**2. Concepto de Polimorfismo**

**Definición:**

El **polimorfismo** permite que una misma operación o método pueda comportarse de diferentes formas según el objeto que lo llame. En Java, el polimorfismo se puede lograr mediante **sobrescritura de métodos** y el uso de **referencias de superclase**.

**Ejemplo de Polimorfismo:**

java

Copiar código

class Animal {

public void hacerSonido() {

System.out.println("El animal hace un sonido");

}

}

class Perro extends Animal {

public void hacerSonido() {

System.out.println("El perro ladra");

}

}

class Gato extends Animal {

public void hacerSonido() {

System.out.println("El gato maúlla");

}

}

public class Main {

public static void main(String[] args) {

Animal miAnimal = new Perro(); // Polimorfismo

miAnimal.hacerSonido(); // El perro ladra

}

}

En este ejemplo:

* La variable **miAnimal** es de tipo **Animal**, pero puede hacer referencia a un objeto de tipo **Perro**. El método **hacerSonido()** invoca la versión de la clase Perro debido al polimorfismo.

**Tarea 2:**

Crea una clase **Ave** que herede de **Animal** y sobrescriba el método **hacerSonido()** para mostrar "El ave canta". Luego, en el método principal, crea un objeto de tipo **Animal** que haga referencia a un objeto de tipo **Ave** y ejecuta el método **hacerSonido()**.

**Pregunta 2:**

¿Cuál es la diferencia entre sobrescritura y sobrecarga de métodos? ¿Cómo se relacionan estos conceptos con el polimorfismo?

**3. Herencia Múltiple en Java (a través de interfaces)**

**Definición:**

Java no permite la herencia múltiple directa entre clases (una clase no puede heredar de más de una clase), pero este problema se resuelve mediante **interfaces**. Una clase puede implementar múltiples interfaces.

**Ejemplo:**

java

Copiar código

interface Volador {

void volar();

}

interface Nadador {

void nadar();

}

class Pato implements Volador, Nadador {

public void volar() {

System.out.println("El pato vuela");

}

public void nadar() {

System.out.println("El pato nada");

}

}

En este ejemplo:

* La clase Pato implementa dos interfaces, **Volador** y **Nadador**, lo que permite que un pato tenga ambas habilidades.

**Tarea 3:**

Define dos interfaces: **Conducible** con el método **conducir()** y **Navegable** con el método **navegar()**. Luego, crea una clase **Amfibia** que implemente ambas interfaces.

**Pregunta 3:**

¿Por qué Java no permite la herencia múltiple directa? ¿Cómo solucionan las interfaces este problema?

**4. Clases Abstractas vs Interfaces**

**Definición:**

* Una **clase abstracta** es una clase que no puede ser instanciada directamente. Puede contener métodos abstractos (sin implementación) y métodos concretos.
* Una **interfaz** es un conjunto de métodos abstractos que una clase puede implementar. Todas las variables en una interfaz son públicas, estáticas y finales.

**Ejemplo de Clase Abstracta:**

java

Copiar código

abstract class Vehiculo {

abstract void mover(); // Método abstracto

}

class Bicicleta extends Vehiculo {

public void mover() {

System.out.println("La bicicleta se mueve");

}

}

**Ejemplo de Interfaz:**

java

Copiar código

interface Volador {

void volar();

}

**Tarea 4:**

Crea una clase abstracta **InstrumentoMusical** con el método abstracto **tocar()**. Luego, crea una clase concreta **Guitarra** que extienda de InstrumentoMusical y defina el método tocar().

**Pregunta 4:**

¿Cuál es la diferencia clave entre una clase abstracta y una interfaz? ¿Cuándo usarías una clase abstracta en lugar de una interfaz y viceversa?

**5. Uso de super para Referenciar la Superclase**

**Definición:**

La palabra clave **super** se utiliza en Java para referenciar la superclase desde una subclase. Se puede usar para invocar el constructor de la superclase o para acceder a métodos y atributos que han sido sobrescritos.

**Ejemplo:**

java

Copiar código

class Animal {

public void hacerSonido() {

System.out.println("El animal hace un sonido");

}

}

class Perro extends Animal {

public void hacerSonido() {

super.hacerSonido(); // Invoca el método de la superclase

System.out.println("El perro ladra");

}

}

En este ejemplo:

* **super.hacerSonido()** invoca el método hacerSonido() de la superclase Animal, y luego la subclase Perro añade su propio comportamiento.

**Tarea 5:**

Crea una clase base **Persona** con un método **hablar()**. Luego, crea una subclase **Estudiante** que sobrescriba el método **hablar()**, pero que también invoque el método hablar() de la clase Persona usando super.

**Pregunta 5:**

¿Qué utilidad tiene la palabra clave super en el contexto de la herencia? ¿Cuándo sería útil invocar métodos o constructores de la superclase?

**6. Resumen de Herencia y Polimorfismo**

| **Concepto** | **Descripción** |
| --- | --- |
| **Herencia** | Permite que una clase derive atributos y métodos de otra clase. |
| **Polimorfismo** | Un mismo método puede comportarse de manera diferente según el objeto que lo invoque. |
| **Sobrescritura** | Modificar el comportamiento de un método de la superclase en una subclase. |
| **Interfaces** | Conjunto de métodos abstractos que una clase puede implementar, útil para herencia múltiple. |
| **Clases Abstractas** | Clases que no pueden ser instanciadas directamente, pero pueden tener métodos concretos. |
| **super** | Permite invocar métodos y constructores de la superclase desde la subclase. |

**Ejercicio Final**

**Objetivo:**

Crea un sistema de clases para una **jerarquía de animales** donde tengas:

1. Una clase abstracta **Animal** con el método abstracto **comer()**.
2. Subclases **Perro**, **Gato**, y **Pajaro** que sobrescriban el método comer().
3. Crea una interfaz **Volador** e implementa esta interfaz en **Pajaro**.
4. Usa el polimorfismo para crear una lista de animales y ejecutar el método comer() en cada uno.

**Pasos:**

1. Declara la clase abstracta Animal y sus subclases.
2. Implementa la interfaz Volador en Pajaro.
3. Usa polimorfismo para crear una lista de diferentes animales y ejecuta su método comer().

**Preguntas y Respuestas**

**Pregunta 1**: ¿Cuál es la diferencia entre una superclase y una subclase? ¿Qué se hereda y qué no?  
**Respuesta**: La superclase es la clase base de la que se heredan atributos y métodos, mientras que la subclase es la clase derivada. No se heredan los constructores ni los métodos privados.

**Pregunta 2**: ¿Cuál es la diferencia entre sobrescritura y sobrecarga de métodos? ¿Cómo se relacionan estos conceptos con el polimorfismo?  
**Respuesta**: Sobrescritura es redefinir un método de la superclase en la subclase, mientras que sobrecarga es definir múltiples métodos con el mismo nombre pero diferentes parámetros. La sobrescritura está directamente relacionada con el polimorfismo.

**Pregunta 3**: ¿Por qué Java no permite la herencia múltiple directa? ¿Cómo solucionan las interfaces este problema?  
**Respuesta**: Java no permite herencia múltiple para evitar ambigüedades y conflictos entre métodos heredados. Las interfaces permiten la herencia múltiple porque sólo definen métodos, sin implementar comportamiento.

**Pregunta 4**: ¿Cuál es la diferencia clave entre una clase abstracta y una interfaz? ¿Cuándo usarías una clase abstracta en lugar de una interfaz y viceversa?  
**Respuesta**: Las clases abstractas pueden tener métodos concretos y abstractos, mientras que las interfaces sólo definen métodos abstractos (hasta Java 8). Usarías una clase abstracta cuando deseas compartir código entre clases relacionadas, y una interfaz cuando deseas implementar comportamiento en clases no relacionadas.

**Pregunta 5**: ¿Qué utilidad tiene la palabra clave super en el contexto de la herencia? ¿Cuándo sería útil invocar métodos o constructores de la superclase?  
**Respuesta**: super permite invocar métodos y constructores de la superclase, lo que es útil cuando se desea reutilizar el comportamiento de la clase base en una subclase, o cuando se necesita inicializar atributos de la clase base.

**Conclusión**

En este tutorial has aprendido sobre **herencia** y **polimorfismo**, dos pilares fundamentales de la programación orientada a objetos en Java. Practicar estos conceptos te ayudará a desarrollar aplicaciones más flexibles, reutilizables y mantenibles.