Polimorfismo | Clase abstracta | Interface

Polimorfismo

El polimorfismo permite que un objeto de una clase se comporte como si fuera de otra clase dentro de una jerarquía de herencia. Hay dos tipos principales de polimorfismo en Java:

1. **Polimorfismo en tiempo de compilación (Sobrecarga de métodos)**: Ocurre cuando dos o más métodos en la misma clase tienen el mismo nombre pero diferentes parámetros.
2. **Polimorfismo en tiempo de ejecución (Sobrescritura de métodos)**: Ocurre cuando un método en una subclase tiene el mismo nombre y parámetros que un método en su superclase.

Clase abstracta

Es una clase que no puede ser instanciada directamente. Se utiliza como una clase base para otras clases y puede contener métodos abstractos y no abstractos. Un método abstracto es un método que se declara sin una implementación, es decir, sin un cuerpo, y debe ser implementado por las subclases concretas.

Interface

Es una referencia abstracta que especifica un conjunto de métodos que una clase debe implementar. Las interfaces proporcionan una forma de lograr la abstracción total, ya que no pueden tener implementación de métodos (hasta Java 8, donde se introdujeron métodos por defecto y métodos estáticos). Las interfaces se utilizan para especificar lo que debe hacer una clase, sin dictar cómo debe hacerlo.

Diferencias entre una Clase Abstracta y una Interfaz

- **Herencia múltiple**: Una clase puede implementar múltiples interfaces, pero solo puede heredar de una clase abstracta.
- **Implementación de métodos**: Una clase abstracta puede tener métodos con implementación completa, mientras que una interfaz (hasta Java 8) solo puede tener métodos abstractos. A partir de Java 8, una interfaz puede tener métodos por defecto y métodos estáticos.
- **Variables**: Las variables en una interfaz son implícitamente `public`, `static` y `final`, mientras que las variables en una clase abstracta pueden tener cualquier modificador de acceso y no son necesariamente finales o estáticas.

Ejemplo de código

A continuación se muestra la clase abstracta Animal cuyos atributos son nombre, edad, y grupo; como atributos de tiene el comportamiento de comer y dormir.

```
public abstract class Animal{
    private String nombre;
    private int edad;
    private String grupo;

    public Animal(String nombre, int edad, String grupo){
        this.nombre = nombre;
        this.edad = edad;
        this.grupo = grupo;
    }

    public String getNombre(){
        return this.nombre;
```

```
    }

    public int getEdad(){
        return this.edad;
    }

    public void comer(){
        System.out.println("Soy un animal y voy a comer");
    }
    public void dormir(){
        System.out.println("Soy un animal y voy a dormir");
    }
}
```

El código de abajo muestra la clase Leon, esta hereda de la clase Animal.

```
public class Leon extends Animal{
    public Leon(String nombre, int edad){
        super(nombre,edad,"felino");
    }

    @Override
    public void comer(){
        System.out.println("Soy un " + this.getClass().getSimpleName() + ", me llamo " +
this.getNombre() + " y voy a comer gacelas");
    }

    @Override
    public void dormir(){
        System.out.println("Soy un " + this.getClass().getSimpleName() + ", me llamo " +
this.getNombre() + " y me voy a dormir");
    }

    @Override
    public String toString(){
        return "Animal: León |" + " Nombre: " + this.getNombre();
    }
}
```

El código de abajo muestra la clase Perro, en donde se hereda de Animal y se implementa la interfaz Mascota, pues no todos los animales tienen un comportamiento amigable.

```
public class Perro extends Animal implements Mascota{

    private String raza;

    public Perro(String nombre, int edad, String raza){
        super(nombre,edad,"canino");
        this.raza = raza;
    }
```

```
    @Override
    public void comer(){
        System.out.println("Soy un " + this.getClass().getSimpleName() + ", me llamo " +
this.getNombre() + " y voy a comer croquetas");
    }

    @Override
    public void dormir(){
        System.out.println("Soy un " + this.getClass().getSimpleName() + ", me llamo " +
this.getNombre() + " y me voy a dormir");
    }

    public void jugar(){
        System.out.println("Soy " + this.getNombre() + ", vamos a jugar amigo!!!");
    }

    public void pedirMimos(){
        System.out.println("Dame amor amigo!!!");
    }

    @Override
    public String toString(){
        return "Animal: Perro |" + " Nombre: " + this.getNombre();
    }
}
```

La interfaz Mascota cuenta con los comportamientos jugar y pedir mimos.

```
public interface Mascota{
    void jugar();
    void pedirMimos();
}
```

En el programa principal se instancian dos animales dentro de un arreglo; un león llamado Simba y un perro llamado Huesos. Posteriormente se itera el arreglo y se muestra la información de ambos animales.

Cuando llega la hora de la comida se alimentan a los animales a través del método comer, aqui es donde sucede el polimorfismo pues la respuesta que se obtiene al invocar el método comer depende del tipo de animal sobre el que se trabaje.

Posteriormente se pretende jugar con los animales, para esto se emplea el método definido en la interfaz Mascota, de manera que no se pueda jugar con el león.

Finalmente se duermen los animales de nuestro zoológico empleando los métodos propios de su respectiva clase.

```
public class Zoo{
    public static void main(String[] args){
        Animal[] animales ={
                new Leon("Simba",6),
                new Perro("Huesos",4,"chihuahua")
        };
        System.out.println("----Nuestro zoo cuenta con los siguientes animales:");
        for(Animal animal:animales){
```

```
            System.out.println(animal.toString());
        }

        System.out.println("----Es hora de alimentar a los animales");
        for(Animal animal:animales){
            animal.comer();
        }

        System.out.println("----Durante la comida voy a acariciar a los animales");
        for(Animal animal:animales){
            if(animal instanceof Mascota){
                ((Mascota) animal).jugar();
            }
        }

        System.out.println("----Es hora de que los animales duerman");
        for(Animal animal:animales){
            animal.dormir();
        }
    }
}
```

Al ejecutar nuestro programa se obtiene:

```
----Nuestro zoo cuenta con los siguientes animales:
Animal: León | Nombre: Simba
Animal: Perro | Nombre: Huesos
----Es hora de alimentar a los animales
Soy un Leon, me llamo Simba y voy a comer gacelas
Soy un Perro, me llamo Huesos y voy a comer croquetas
----Durante la comida voy a acariciar a los animales
Soy Huesos, vamos a jugar amigo!!!
----Es hora de que los animales duerman
Soy un Leon, me llamo Simba y me voy a dormir
Soy un Perro, me llamo Huesos y me voy a dormir
```