Programación Orientada a Objetos en Java

PROFESOR JUAN LUIS HERENCIA GUERRA

Extensión de clases

JERARQUIZACIÓN

Definición de Extensión de Clases

- La herencia en Java permite a una clase (subclase) heredar campos y métodos de otra clase (superclase). Esto facilita la reutilización del código y establece una relación jerárquica entre las clases.

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}

class Perro extends Animal {
    void hacerSonido() {
        System.out.println("El perro ladra");
    }
}

public class Main {
    public static void main(String[] args) {
        Perro perro = new Perro();
        perro.hacerSonido();  // Output: El perro ladra
    }
}
```

Transitividad

- La transitividad en herencia significa que, si la clase B hereda de la clase A, y la clase C hereda de la clase B, entonces la clase C también hereda de la clase A.

```java
class A {
    void metodoA() {
        System.out.println("Método A");
    }
}

class B extends A {
    void metodoB() {
        System.out.println("Método B");
    }
}
```

```java
class C extends B {
    void metodoC() {
        System.out.println("Método C");
    }
}

public class Main {
    public static void main(String[] args) {
        C obj = new C();
        obj.metodoA();  // Output: Método A
        obj.metodoB();  // Output: Método B
        obj.metodoC();  // Output: Método C
    }
}
```

Diseño: Generalización y Especificación

- La generalización es el proceso de extraer características comunes de varias clases y crear una clase base. La especificación es la creación de clases derivadas que implementan o extienden la funcionalidad de la clase base.

```java
class Vehiculo {
    void mover() {
        System.out.println("El vehículo se mueve");
    }
}

class Coche extends Vehiculo {
    void abrirPuertas() {
        System.out.println("El coche abre las puertas");
    }
}

class Bicicleta extends Vehiculo {
    void pedalear() {
        System.out.println("La bicicleta se pedalea");
    }
}
```

Implementación

- La implementación se refiere a escribir el código que permitirá la realización comportamiento requerido por la clase y el método.

```
class Calculadora {
    int sumar(int a, int b) {
        return a + b;
    }
}
```

Herencia y Constructores

- Los constructores no se heredan, pero pueden ser invocados usando la palabra clave super para llamar al constructor de la superclase.

```java
class Animal {
    String nombre;

    Animal(String nombre) {
        this.nombre = nombre;
    }
}

class Perro extends Animal {
    Perro(String nombre) {
        super(nombre);
    }
}

public class Main {
    public static void main(String[] args) {
        Perro perro = new Perro("Fido");
        System.out.println(perro.nombre);
    }
}
```

Redefinición

- La redefinición (overriding) permite a una subclase proporcionar una implementación específica de un método que ya está definido en su superclase.

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}

class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El gato maúlla");
    }
}
```

Modificador final

- El modificador final puede ser utilizado para prevenir que una clase sea extendida, un método sea sobrescrito, o que una variable cambie su valor una vez asignado.

```
final class Vehiculo {
    final void mover() {
        System.out.println("El vehículo se mueve");
    }
}
```

Clases Abstractas

- Las clases abstractas no pueden ser instanciadas y pueden contener métodos abstractos que deben ser implementados por las subclases.

```java
abstract class Animal {
    abstract void hacerSonido();
}

class Vaca extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("La vaca muge");
    }
}
```

Operador instanceof

- El operador instanceof se utiliza para comprobar si un objeto es una instancia de una clase específica o de una subclase.

```
public class Main {
    public static void main(String[] args) {
        Animal animal = new Gato();
        if (animal instanceof Gato) {
            System.out.println("El animal es un gato");
        }
    }
}
```

Polimorfismo

- El polimorfismo permite a un objeto tomar muchas formas. Un método puede comportarse de diferentes maneras según el objeto que lo invoque.

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra");
    }
}
```

```java
class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El gato maúlla");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Perro();
        miAnimal.hacerSonido();

        miAnimal = new Gato();
        miAnimal.hacerSonido();
    }
}
```

Casting de Objetos

- El casting de objetos permite convertir un objeto de una clase a otro tipo, siempre y cuando exista una relación de herencia entre las clases.

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}

class Perro extends Animal {
    void hacerSonido() {
        System.out.println("El perro ladra");
    }

    void ladrar() {
        System.out.println("Guau guau");
    }
}
```

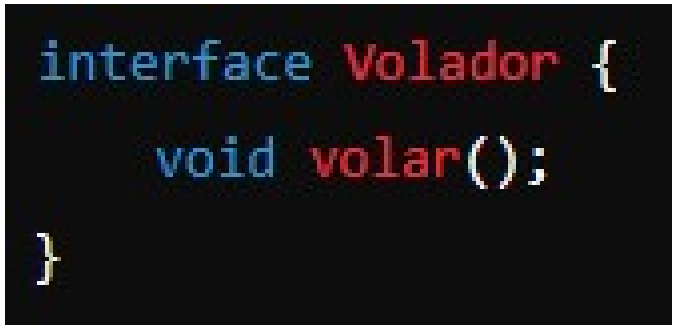
```java
public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Perro();
        ((Perro) miAnimal).ladrar();
    }
}
```

Clases abstractas e interfaces

JERARQUIZACIÓN

Interfaz

- Una interfaz en Java es una colección de métodos abstractos y constantes. Las interfaces especifican qué debe hacer una clase, pero no cómo lo hace. Las clases que implementan una interfaz deben proporcionar la implementación de los métodos definidos en la interfaz.



Herencia

- En Java, una clase puede heredar de otra clase (herencia simple), pero puede implementar múltiples interfaces. Una clase abstracta también puede extender otra clase y puede implementar interfaces.

```java
abstract class Animal {
    abstract void hacerSonido();
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra");
    }
}
```

```java
class Pajaro extends Animal implements Volador {
    @Override
    void hacerSonido() {
        System.out.println("El pájaro canta");
    }

    @Override
    public void volar() {
        System.out.println("El pájaro vuela");
    }
}
```

Datos

- Las interfaces no pueden tener variables de instancia, pero pueden tener constantes. Las clases abstractas pueden tener tanto variables de instancia como métodos abstractos y concretos.

```
interface Constantes {
    int MAX_VELOCIDAD = 120;  // Constante
}

abstract class Vehiculo {
    int velocidad;  // Variable de instancia

    abstract void mover();
}
```

Implementación

- Las clases que implementan una interfaz deben proporcionar implementaciones para todos los métodos de la interfaz. Las clases abstractas pueden proporcionar implementaciones parciales y dejar algunos métodos como abstractos.

```java
interface Volador {
    void volar();
}

abstract class Ave implements Volador {
    void comer() {
        System.out.println("El ave come");
    }
}

class Aguila extends Ave {
    @Override
    public void volar() {
        System.out.println("El águila vuela alto");
    }
}
```

Polimorfismo

- El polimorfismo permite tratar objetos de diferentes clases de manera uniforme a través de una interfaz común o una clase base común. Esto es especialmente útil cuando se usa con clases abstractas e interfaces.

```java
interface Volador {
    void volar();
}

class Avion implements Volador {
    @Override
    public void volar() {
        System.out.println("El avión vuela en el cielo");
    }
}
```

```java
class Pajaro implements Volador {
    @Override
    public void volar() {
        System.out.println("El pájaro vuela en el aire");
    }
}

public class Main {
    public static void main(String[] args) {
        Volador[] voladores = {new Avion(), new Pajaro()};

        for (Volador volador : voladores) {
            volador.volar();
        }
    }
}
```

Casting de Objetos

- El casting de objetos permite convertir un objeto de una clase a otro tipo, siempre que haya una relación de herencia o implementación de interfaz. Esto es útil para trabajar con interfaces y clases abstractas.

```java
interface Volador {
    void volar();
}

class Avion implements Volador {
    @Override
    public void volar() {
        System.out.println("El avión vuela en el cielo");
    }

    void despegar() {
        System.out.println("El avión despega");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Volador volador = new Avion();
        volador.volar();

        if (volador instanceof Avion) {
            ((Avion) volador).despegar();
        }
    }
}
```

Casting de Objetos

- El casting de objetos permite convertir un objeto de una clase a otro tipo, siempre que haya una relación de herencia o implementación de interfaz. Esto es útil para trabajar con interfaces y clases abstractas.

```java
interface Volador {
    void volar();
}

class Avion implements Volador {
    @Override
    public void volar() {
        System.out.println("El avión vuela en el cielo");
    }

    void despegar() {
        System.out.println("El avión despega");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Volador volador = new Avion();
        volador.volar();

        if (volador instanceof Avion) {
            ((Avion) volador).despegar();
        }
    }
}
```