Herencia

La herencia en Java es un mecanismo mediante el cual un objeto adquiere todas las propiedades y comportamientos de una clase padre. Es un componente esencial de la programación orientada a objetos (POO).

Java admite la herencia simple, lo que significa que una clase puede heredar solo de una clase padre. Esto ayuda a evitar la complejidad de la herencia múltiple. Sin embargo, se puede lograr una forma de herencia múltiple a través del uso de interfaces.

Aplicando herencia

Para crear una clase padre o superclase en Java, simplemente declaramos la clase como lo haríamos normalmente. La clase que debemos modificar es la clase hija o subclase.

Es importante tener en cuenta que las propiedades de la clase padre, en lugar de ser privadas (private), **deben declararse como protegidas (protected)**. Esto permite que las propiedades solo puedan ser utilizadas en la clase padre y en las clases hijas.

Palabra clave extends

En las subclases, debemos utilizar la palabra clave "extends" para indicar que la clase debe heredar todos los atributos y métodos de otra clase.

```
public class Estudiante extends Persona {
    private String grado;
}
```

Ahora, la clase Estudiante hereda las propiedades de nombre y edad de la clase Persona.

Palabra clave super

La palabra clave "**super**" se utiliza para hacer referencia al constructor y métodos de la clase Padre, de la misma manera que se usa "**this**".

```java
// Superclase o clase padre
public class Persona {
    protected String nombre;
    protected int edad;

    public Persona() {
    }

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

```java
// Subclase o clase hija
public class Estudiante extends Persona {

    private String grado;

    public Estudiante() {
        super();
        this.edad = 14;
        this.nombre = "Elias";
        this.grado = "segundo";
    }

    public Estudiante(String nombre, int edad, String grado) {
        super(nombre, edad);
        this.grado = grado;
    }
}
```

En el ejemplo, podemos acceder a las propiedades de la clase padre porque están declaradas con el modificador de acceso "protected".

Sobrescribir métodos

Los métodos de las superclases pueden ser sobrescritos por las subclases utilizando la anotación "@Override". De esta manera, se modifica el comportamiento del método para adaptarlo a las particularidades de la clase hija.

```java
public class Persona {
    public void hablar() {
        System.out.println("Soy una persona");
    }
}
```

```java
public class Estudiante extends Persona {
    @Override
    public void hablar() {
        System.out.println("Soy un estudiante");
    }
}
```

Si un objeto Estudiante invoca al método "hablar()", se imprimirá en la consola "Soy un estudiante". Si no se sobrescribe el método "hablar()", se imprimirá "Soy una persona.

La clase Object

En Java, todos los objetos heredan de una clase base común, la clase Object. Esta herencia es implícita, lo que significa que no es necesario utilizar "extends Object" para que se produzca.

La clase Object tiene varios métodos que pueden ser sobrescritos en las clases derivadas para proporcionar un comportamiento específico. Algunos de estos métodos incluyen equals(), hashCode(), toString(), clone(), finalize(), getClass(), notify(), notifyAll(), y wait().

- **equals()**: Este método se utiliza para comparar dos objetos y determinar si son iguales. Puede ser sobrescrito en clases personalizadas para realizar una comparación basada en los atributos de los objetos en lugar de la igualdad de referencia.
- **hashCode()**: Devuelve un código hash único para un objeto, utilizado en estructuras de datos como las tablas hash para indexar y buscar objetos de manera eficiente. Se recomienda sobrescribir este método si se sobrescribe equals() para garantizar coherencia.
- **toString()**: Devuelve una representación en forma de cadena de texto del objeto, útil para imprimir información legible sobre el objeto, generalmente con fines de depuración o visualización.

- **clone()**: Se utiliza para crear y devolver una copia superficial del objeto. Este método ha caído en desuso y se recomienda implementar métodos de copia propios para copiar objetos.
- **finalize()**: Se llama antes de que el recolector de basura elimine un objeto, permitiendo la limpieza de recursos. Este método está obsoleto desde Java 9 y su uso no se recomienda.
- **getClass()**: Se utiliza para obtener la clase de tiempo de ejecución del objeto y no puede ser sobrescrito. Es útil para realizar operaciones de reflexión.

La importancia de equals() hashCode() y toString()

Sobrescribir los métodos equals() y hashCode() es crucial cuando se crean objetos personalizados, ya que permiten comparaciones basadas en los valores de los atributos en lugar de en la referencia en memoria.

El método toString() permite devolver una representación de cadena de texto de los atributos del objeto para su impresión en pantalla.

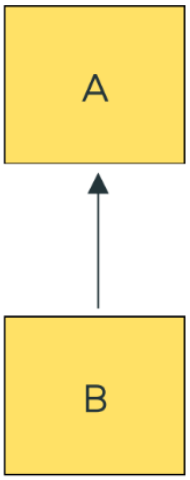
Polimorfismo

El polimorfismo se refiere a la capacidad de un método u objeto de tomar múltiples formas.

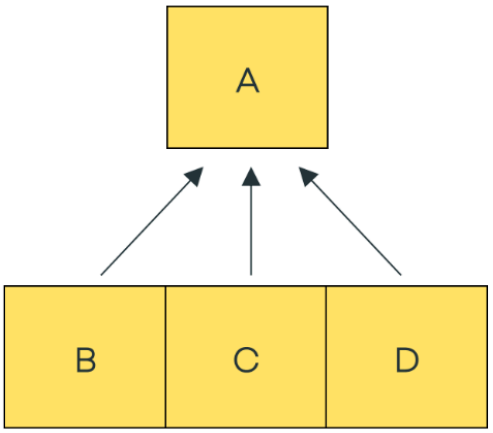
- **Polimorfismo de métodos:** Se produce mediante la sobrecarga y sobrescritura de métodos en una clase, lo que permite que un método tenga múltiples implementaciones con el mismo nombre.
- **Polimorfismo de objetos:** Se da cuando una superclase se utiliza para referirse a un objeto de una subclase (se verá con el uso de instanceof) o cuando una clase implementa interfaces.

Tipos de herencia

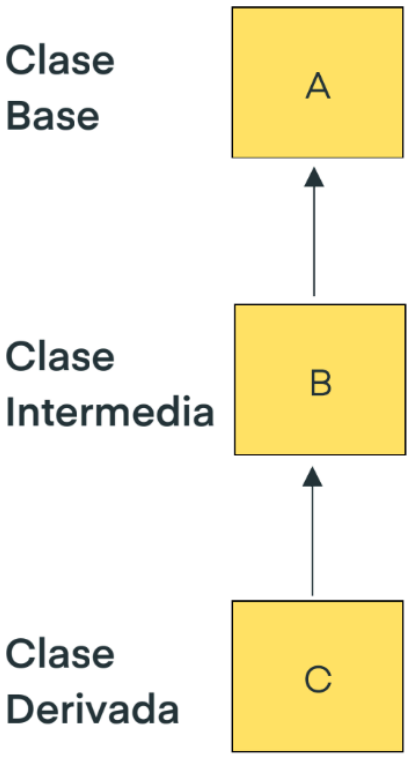
- **Herencia única:** Una subclase hereda de una sola superclase.



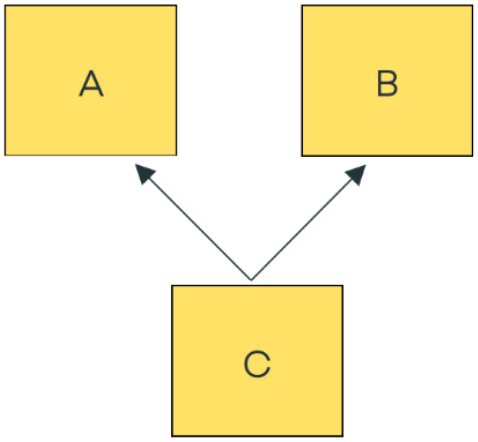
- **Herencia jerárquica:** Una superclase es la base para más de una subclase.



- **Herencia multinivel:** Una clase derivada hereda de una clase base y a su vez actúa como clase base para otra clase.



- **Herencia múltiple:** No es compatible en Java con clases, pero se puede lograr a través de interfaces



Operador Instanceof

Este operador se utiliza para probar si un objeto es una instancia de una clase, subclase o clase que implementa una interfaz particular.

```java
class Animal {}
class Perro extends Animal {}

public class Instanceof {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Animal perro = new Perro();

        System.out.println(perro instanceof Animal);  // Devuelve true
        System.out.println(perro instanceof Perro);  // Devuelve true
        System.out.println(animal instanceof Perro);  // Devuelve false
    }
}
```

Pattern matching

El Pattern Matching, o coincidencia de patrones, es un mecanismo común en programación que permite verificar si un valor coincide con un patrón específico y ejecutar cierto código en función de esa coincidencia. En las últimas versiones de Java, se ha estado incorporando el concepto de Pattern Matching en el lenguaje, mejorando significativamente la simplicidad y la seguridad de ciertos patrones de programación comunes.

Pattern Matching para instanceof (Introducido en Java 16)

Esta característica reduce la necesidad de una conversión explícita después de una operación exitosa de instanceof.

```java
public class Main {
    public static void main(String[] args) {
        Object obj = "Hola mundo";
        if (obj instanceof String s) {
            // Puedes declarar la variable directamente al usar instanceof.
            System.out.println(s.toLowerCase());
        }
    }
}
```

Pattern Matching en switch expression (En revisión en Java 20):

Proporciona mayor flexibilidad y poder expresivo en las declaraciones switch al permitir patrones complejos con case.

```java
class Animal {}
class Perro extends Animal {}

public class Main {
```

```java
    public static void main(String[] args) {
        Object[] objects = { "Hola", 10, 20.0, true, new Animal(), new Perro()
};

        for (Object obj : objects) {
            switch (obj) {
                case String s -> System.out.println("String: " + s);
                case Integer i -> System.out.println("Integer: " + i);
                case Double d -> System.out.println("Double: " + d);
                case Boolean b -> System.out.println("Boolean: " + b);
                case Perro p -> System.out.println("Es un perro");
                case Animal a -> System.out.println("Es un animal");
                default -> System.out.println("Tipo desconocido");
            }
        }
    }
}
```