Programación orientada a objetos Individual - POO - Unidad 2
Actividad: Taller sobre Polimorfismo
Tutor: JOHN CARLOS ARRIETA ARRIETA DOCENTE

Taller: Polimorfismo en Java

Objetivo del Taller:

En este taller, los estudiantes aprenderán el concepto de polimorfismo en Java, cómo permite a las clases derivadas especializar el comportamiento de la clase base, y cómo utilizar referencias de la clase base para invocar métodos en tiempo de ejecución. Se proporcionarán explicaciones claras, ejemplos prácticos, y ejercicios para reforzar estos conceptos.

Temario:

1. **Contextualización y Definición del Polimorfismo**
   - ¿Qué es el polimorfismo?
   - Tipos de polimorfismo en Java: estático y dinámico.
2. **Objetivos del Uso del Polimorfismo**
   - Reutilización de código y especialización de comportamientos.
   - Permitir la interacción uniforme entre diferentes tipos de objetos.
3. **Cuándo Usar y Cuándo No Usar el Polimorfismo**
   - Casos recomendados y no recomendados.
4. **Sintaxis y Reglas del Polimorfismo**
   - Uso de referencias de la clase base y métodos sobrescritos.
5. **Ejemplos de Uso Correcto**
   - Ejemplos prácticos que muestran la implementación y uso del polimorfismo.
6. **Ejemplos de Uso Incorrecto con Errores de Compilación**
   - Ejemplos que violan las reglas del polimorfismo y no compilan.
7. **Ejemplos de Uso Incorrecto sin Generar Error de Compilación**
   - Ejemplos que son malas prácticas pero que no generan errores.
8. **Ejercicios Propuestos**
   - Ejercicios para poner en práctica el uso correcto e incorrecto del polimorfismo.

---

1. Contextualización y Definición del Polimorfismo

¿Qué es el Polimorfismo?

El polimorfismo es un principio fundamental de la programación orientada a objetos que permite a un objeto de una clase derivada ser tratado como un objeto de su clase base. El polimorfismo permite que un mismo método tenga diferentes comportamientos según el objeto en el que se invoque.

Tipos de Polimorfismo en Java:

1. **Polimorfismo Estático:** También conocido como sobrecarga de métodos. Se determina en tiempo de compilación y permite definir múltiples métodos con el mismo nombre pero con diferentes firmas.
2. **Polimorfismo Dinámico:** También conocido como polimorfismo en tiempo de ejecución o sobrescritura de métodos. Permite que las referencias de la clase base apunten a objetos de la clase derivada y ejecuten el método sobrescrito en tiempo de ejecución.

---

2. Objetivos del Uso del Polimorfismo

Objetivos Clave:

1. **Reutilización y Extensibilidad:** Permitir que las clases derivadas especialicen el comportamiento heredado de la clase base.
2. **Interacción Uniforme entre Objetos:** Permitir que las referencias de la clase base inviquen métodos en objetos de las clases derivadas, facilitando el diseño modular y flexible.

Ejemplo Simple de Polimorfismo Dinámico:

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

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

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

// Clase de prueba
public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Perro();
        miAnimal.hacerSonido(); // Muestra: El perro ladra.

        miAnimal = new Gato();
        miAnimal.hacerSonido(); // Muestra: El gato maúlla.
    }
}
```

---

3. Cuándo Usar y Cuándo No Usar el Polimorfismo

Cuándo Usar el Polimorfismo:

- Cuando se desea que las clases derivadas proporcionen su propia implementación de métodos heredados de la clase base.
- Cuando se necesita permitir que un mismo método tenga diferentes comportamientos según el objeto que lo invoque.
- Cuando se necesita un diseño flexible y extensible.

Cuándo No Usar el Polimorfismo:

- Cuando no se requiere que las clases derivadas modifiquen el comportamiento de la clase base.
- Cuando el diseño de la aplicación no requiere la utilización de jerarquías de clases o herencia.

---

4. Sintaxis y Reglas del Polimorfismo

Uso de Referencias de la Clase Base:

En polimorfismo dinámico, se utiliza una referencia de la clase base para apuntar a un objeto de la clase derivada. Los métodos sobrescritos en la clase derivada se ejecutarán en función del objeto referenciado en tiempo de ejecución.

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

public class Coche extends Vehiculo {
    @Override
    public void mover() {
        System.out.println("El coche se está moviendo.");
    }
}

public class Bicicleta extends Vehiculo {
    @Override
    public void mover() {
        System.out.println("La bicicleta se está moviendo.");
    }
}
```

---

5. Ejemplos de Uso Correcto

Ejemplo Correcto 1: Uso de Polimorfismo para Definir Comportamientos Específicos

```java
// Clase base
public class Empleado {
    public void trabajar() {
        System.out.println("El empleado está trabajando.");
    }
}

// Clases derivadas
public class Gerente extends Empleado {
    @Override
    public void trabajar() {
        System.out.println("El gerente está gestionando el equipo.");
    }
}

public class Ingeniero extends Empleado {
    @Override
    public void trabajar() {
        System.out.println("El ingeniero está diseñando.");
    }
}

// Clase de prueba
public class Main {
    public static void main(String[] args) {
        Empleado empleado1 = new Gerente();
        Empleado empleado2 = new Ingeniero();

        empleado1.trabajar(); // Muestra: El gerente está gestionando el equipo.
        empleado2.trabajar(); // Muestra: El ingeniero está diseñando.
    }
}
```

Explicación: La referencia de la clase base `Empleado` apunta a objetos de diferentes clases derivadas (`Gerente` e `Ingeniero`). Esto permite invocar métodos de manera polimórfica.

---

6. Ejemplos de Uso Incorrecto con Errores de Compilación

Ejemplo Incorrecto 1: Intentar Invocar un Método que No Existe en la Clase Base

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

// Clase derivada
public class Perro extends Animal {
    public void correr() {
        System.out.println("El perro está corriendo.");
    }
}

// Clase de prueba (incorrecta)
public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Perro();
        miAnimal.correr(); // Error de compilación: El método correr() no existe en Animal
    }
}
```

Explicación: Aunque `miAnimal` apunta a un objeto de tipo `Perro`, solo se pueden invocar los métodos que existen en la clase base `Animal`.

---

7. Ejemplos de Uso Incorrecto sin Generar Error de Compilación

Ejemplo Incorrecto 2: Uso Innecesario de Polimorfismo

```java
// Clase base
public class Vehiculo {
    public void mover() {
        System.out.println("El vehículo se está moviendo.");
    }
}

// Clase derivada
public class Moto extends Vehiculo {
    @Override
    public void mover() {
        super.mover(); // Uso innecesario si no se modifica el comportamiento
    }
}
```

Explicación: Aunque el código compila, llamar a `super.mover()` desde `Moto` es innecesario si no se modifica el comportamiento de `mover` en la clase derivada.

---

8. Ejercicios Propuestos

Ejercicio 1: Clase `Figura` y Clases Derivadas `Circulo` y `Rectangulo`

1. Define una clase `Figura` con un método `calcularArea` que imprima un mensaje general.
2. Define dos clases derivadas (`Circulo` y `Rectangulo`) que sobrescriban `calcularArea` para mostrar sus respectivas áreas.
3. Crea una clase de prueba que demuestre el uso del polimorfismo al calcular las áreas de diferentes figuras.

Ejercicio 2: Clase `Persona` y Clases Derivadas `Estudiante` y `Profesor`

1. Define una clase `Persona` con un método `presentarse` que imprima un mensaje general.
2. Define dos clases derivadas (`Estudiante` y `Profesor`) que sobrescriban `presentarse` para mostrar mensajes específicos.
3. Crea una clase de prueba donde instancies objetos de `Persona`, `Estudiante` y `Profesor`, y uses polimorfismo para invocar `presentarse`.

Ejercicio 3: Uso Incorrecto de Polimorfismo

1. Intenta invocar un método que exista solo en una clase derivada utilizando una referencia de la clase base y observa el error de compilación.
2. Crea una clase derivada que sobrescriba un método pero que no modifique realmente el comportamiento del método de la clase base, y discute por qué esto es una mala práctica.

Ejercicio 4: Clase `Vehículo` con Clases Derivadas `Coche` y `Bicicleta`

1. Define una clase `Vehículo` con un método `mover` que imprima un mensaje general.
2. Define dos clases derivadas (`Coche` y `Bicicleta`) que sobrescriban `mover` para mostrar mensajes específicos.
3. Crea una clase de prueba donde se utilice una referencia de `Vehículo` para invocar los métodos de los objetos `Coche` y `Bicicleta`.

Conclusión del Taller:

En este taller, los estudiantes han aprendido el concepto de polimorfismo en Java, sus tipos y cómo permite la especialización del comportamiento heredado. El polimorfismo es una herramienta poderosa para crear diseños flexibles y escalables en la programación orientada a objetos.

Puntos Clave a Recordar:

- El polimorfismo permite que un mismo método tenga diferentes comportamientos según el objeto que lo invoque.
- Existen dos tipos de polimorfismo en Java: polimorfismo estático (sobrecarga de métodos) y polimorfismo dinámico (sobrescritura de métodos).
- Las referencias de la clase base pueden apuntar a objetos de clases derivadas, permitiendo la invocación de métodos sobrescritos.
- Es importante respetar las reglas del polimorfismo para evitar errores y diseñar un código claro y mantenible.