TECNICATURAUNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

PROGRAMACIÓN II

Trabajo Práctico 7: Herencia y Polimorfismo en Java

OBJETIVO GENERAL

Comprender y aplicar los conceptos de herencia y polimorfismo en la Programación Orientada a Objetos, reconociendo su importancia para la reutilización de código, la creación de jerarquías de clases y el diseño flexible de soluciones en Java.

MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Herencia	Uso de `extends` para crear jerarquías entre clases, aprovechando el principio is-a.
Modificadores de acceso	Uso de private, protected y public para controlar visibilidad.
Constructores y super	Invocación al constructor de la superclase con super(...) para inicializar atributos.
Upcasting	Generalización de objetos al tipo de la superclase.
Instanceof	Comprobación del tipo real de los objetos antes de hacer conversiones seguras.
Downcasting	Especialización de objetos desde una clase general a una más específica.
Clases abstractas	Uso de abstract para definir estructuras base que deben ser completadas por subclases.
Métodos abstractos	Declaración de comportamientos que deben implementarse en las clases derivadas.
Polimorfismo	Uso de la sobrescritura de métodos (@Override) y llamada dinámica de métodos.
Herencia	Uso de `extends` para crear jerarquías entre clases, aprovechando el principio is-a.

Caso Práctico

Desarrollar las siguientes Katas en Java aplicando herencia y polimorfismo. Se recomienda repetir cada kata para afianzar el concepto.

1. Vehículos y herencia básica

- Clase base: Vehículo con atributos marca, modelo y método **mostrarInfo()**
- Subclase: Auto con atributo adicional **cantidadPuertas**, sobrescribe **mostrarInfo()**
- Tarea: Instanciar un auto y mostrar su información completa.

2. Figuras geométricas y métodos abstractos

- Clase abstracta: Figura con método **calcularArea()** y atributo nombre
- Subclases: **Círculo y Rectángulo** implementan el cálculo del área
- Tarea: Crear un array de figuras y mostrar el área de cada una usando polimorfismo.

3. Empleados y polimorfismo

- Clase abstracta: Empleado con método **calcularSueldo()**
- Subclases: **EmpleadoPlanta, EmpleadoTemporal**
- Tarea: Crear lista de empleados, invocar **calcularSueldo()** polimórficamente, usar instanceof para clasificar

4. Animales y comportamiento sobrescrito

- Clase: Animal con método **hacerSonido() y describirAnimal()**
- Subclases: Perro, Gato, Vaca sobrescriben **hacerSonido()** con **@Override**
- Tarea: Crear lista de animales y mostrar sus sonidos con polimorfismo

CONCLUSIONES ESPERADAS

- Comprender el mecanismo de herencia y sus beneficios para la reutilización de código.
- Aplicar polimorfismo para lograr flexibilidad en el diseño de programas.
- Inicializar objetos correctamente usando **super** en constructores.
- Controlar el acceso a atributos y métodos con modificadores adecuados.
- Identificar y aplicar **upcasting, downcasting** y **instanceof** correctamente.
- Utilizar clases y métodos abstractos como base de jerarquías lógicas.
- Aplicar principios de diseño orientado a objetos en la implementación en Java.

Consideraciones:

1. Vehículos y herencia básica

Se creará una clase base Vehiculo con los atributos marca y modelo, y un método mostrarInfo(). Luego, se definirá una subclase Auto que hereda de Vehiculo usando extends, añadiendo el atributo cantidadPuertas. El constructor de Auto llamará al constructor de Vehiculo usando super() para inicializar los atributos heredados. Finalmente, Auto sobrescribirá el método mostrarInfo() para incluir la nueva información, pero reutilizando el método del padre. La clase Main instanciará un Auto y llamará a su método.

2. Figuras geométricas y métodos abstractos

Se definirá una clase abstracta Figura que tendrá un atributo nombre y un método abstracto calcularArea(). Luego, se crearán dos subclases concretas, Circulo y Rectangulo, que heredarán de Figura. Ambas clases implementarán obligatoriamente el método calcularArea() con sus fórmulas matemáticas específicas. En la clase Main, se creará un array de tipo Figura (upcasting) que almacenará instancias de Circulo y Rectangulo. Finalmente, se recorrerá el array y se llamará al método calcularArea() de cada objeto, demostrando el polimorfismo.

3. Empleados y polimorfismo

Se creará una clase abstracta Empleado con un método abstracto calcularSueldo(). Se definirán dos subclases: EmpleadoPlanta (con un sueldo fijo y un bono) y EmpleadoTemporal (con un sueldo basado en horas trabajadas y tarifa por hora). Ambas implementarán calcularSueldo(). En la clase Main, se creará una ArrayList de tipo Empleado y se agregarán instancias de ambas subclases. Se recorrerá la lista invocando calcularSueldo() de forma polimórfica. Adicionalmente, dentro del bucle, se usará el operador instanceof para identificar y clasificar el tipo real de cada objeto Empleado.

4. Animales y comportamiento sobrescrito

Se creará una clase base Animal con un método hacerSonido() que imprimirá un sonido genérico. Luego, se crearán tres subclases: Perro, Gato y Vaca. Cada una de estas clases heredará de Animal y sobrescribirá (@Override) el método hacerSonido() para proporcionar su sonido característico ("Guau", "Miau", "Muu"). En la clase Main, se creará una lista de Animal y se agregarán instancias de las tres subclases. Finalmente, se iterará sobre la lista y se llamará al método hacerSonido() de cada animal, demostrando cómo se invoca el método de la subclase correcta gracias al polimorfismo.

Link a repositorio y código: Link

Alumno: Federico Iacono