

PROGRAMACIÓN II

Trabajo Práctico 7: Herencia y Polimorfismo en Java

Estudiante: Roqué, Gabriel Osvaldo

Matrícula: 101636

Link de GitHub:

<https://github.com/Ozzetas/Programacion2.git>

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.

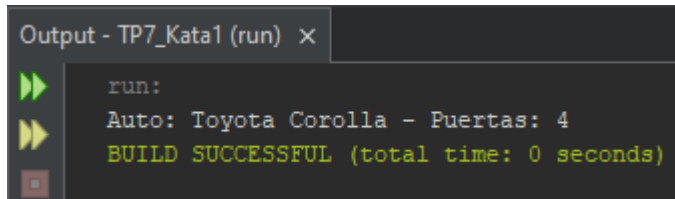
Polimorfismo	Uso de la sobrescritura de métodos (@Override) y llamada dinámica de métodos.
Herencia	Uso de `extends` para crear jerarquías entre clases, aprovechando el principio is-a.

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

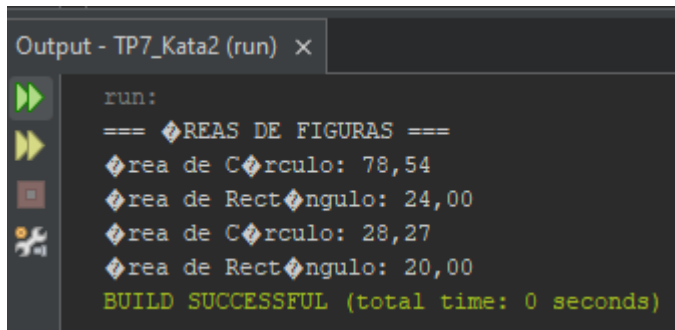
- Clase base: Vehículo con atributos marca, modelo y método **mostrarInfo()**
- Subclase: Auto con atributo adicional **cantidadPuertas**, sobrescribe **mostrarInfo()**
- Tarea: Instanciar un auto y mostrar su información completa.



```
Output - TP7_Kata1 (run) x
run:
Auto: Toyota Corolla - Puertas: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.



```
Output - TP7_Kata2 (run) x
run:
=== REAS DE FIGURAS ===
rea de Círculo: 78,54
rea de Rectángulo: 24,00
rea de Círculo: 28,27
rea de Rectángulo: 20,00
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

```
Output - TP7_Kata3 (run) x
run:
=== NOMINA DE EMPLEADOS ===
Empleado: Juan Pérez (E001) - Sueldo: $62500,00
? Tipo: Planta Permanente
Empleado: Ana Gómez (E002) - Sueldo: $36000,00
? Tipo: Temporal
Empleado: Luis Martínez (E003) - Sueldo: $49500,00
? Tipo: Planta Permanente
Empleado: María López (E004) - Sueldo: $40000,00
? Tipo: Temporal
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

```
Output - TP7_Kata4 (run) x
run:
=== SONIDOS DE ANIMALES ===
Animal: Rex (Perro)
Rex dice: Guau guau!

Animal: Luna (Gato)
Luna dice: Miau miau!

Animal: Mimosa (Vaca)
Mimosa dice: Muuu!

Animal: Max (Perro)
Max dice: Guau guau!

Animal: Nieve (Gato)
Nieve dice: Miau miau!

BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

