

## 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.

## RESOLUCION

### 1. Vehículos y herencia básica

#### A. Estructura de Archivos:

Vehiculo.java

Auto.java

Main\_E1.java

#### B. Código

Vehiculo.java

```
package CASO1;
```

```
public class Vehiculo {  
    protected String marca;  
    protected String modelo;  
  
    public Vehiculo(String marca, String modelo) {  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
  
    public void mostrarInfo() {  
        System.out.println("Modelo: " + modelo + ", marca: " + marca);  
    }  
}
```

Auto.java

```
package CASO1;
```

```
public class Auto extends Vehiculo {  
    private int cantidadDePuertas;  
  
    public Auto(int cantidadDePuertas, String marca, String modelo) {  
        super(marca, modelo);  
        this.cantidadDePuertas = cantidadDePuertas;  
    }  
  
    @Override  
    public void mostrarInfo() {  
        System.out.println("Modelo: " + this.modelo + ", marca: " + this.marca + ", cantidad de puertas: "  
+ cantidadDePuertas);  
    }  
}
```

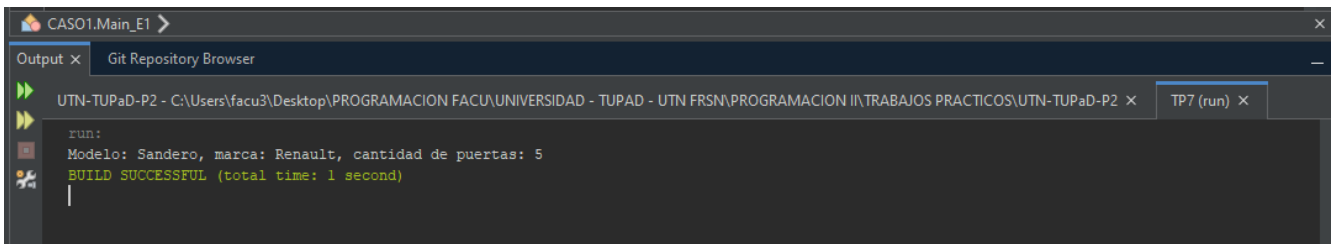
Main\_E1.java

```
package CASO1;

public class Main_E1 {
    public static void main(String[] args) {
        // Instanciamos un auto
        Auto a = new Auto(5, "Renault", "Sander");

        // Llamamos al metodo para mostrar su informacion
        a.mostrarInfo();
    }
}
```

### C. Resultados output



```
CASO1.Main_E1
Output x Git Repository Browser
UTN-TUPaD-P2 - C:\Users\facu3\Desktop\PROGRAMACION FACU\UNIVERSIDAD - TUPAD - UTN FRSN\PROGRAMACION II\TRABAJOS PRACTICOS\UTN-TUPaD-P2 x TP7 (run) x
run:
Modelo: Sander, marca: Renault, cantidad de puertas: 5
BUILD SUCCESSFUL (total time: 1 second)
```

## 2. Figuras geométricas y métodos abstractos

### A. Estructura de Archivos:

Figura.java

Circulo.java

Rectangulo.java

Main\_E2.java

### B. Código

Figura.java

```
package CASO2;

public abstract class Figura {
    protected String nombre;

    public Figura(String nombre) {
        this.nombre = nombre;
    }

    // Método abstracto, debe ser implementado por las subclases
    public abstract void calcularArea();
}
```

Circulo.java

```
package CASO2;

public class Circulo extends Figura {
    private double radio;
    private static final double PI = 3.14; // Usaremos un valor fijo simple

    public Circulo(double radio, String nombre) {
        super(nombre);
        this.radio = radio;
    }

    @Override
    public void calcularArea() {
        double area = PI * radio * radio; // Fórmula:  $PI * radio^2$ 
        System.out.println("El area del circulo " + nombre + " es: " + area);
    }
}
```

Rectangulo.java

```
package CASO2;

public class Rectangulo extends Figura {
    private double base;
    private double altura;

    public Rectangulo(double base, double altura, String nombre) {
        super(nombre);
        this.base = base;
        this.altura = altura;
    }

    @Override
    public void calcularArea() {
        double area = base * altura; // Fórmula:  $base * altura$ 
        System.out.println("El area del rectangulo " + nombre + " es: " + area);
    }
}
```

Main\_E2.java

```
package CASO2;

import java.util.ArrayList;

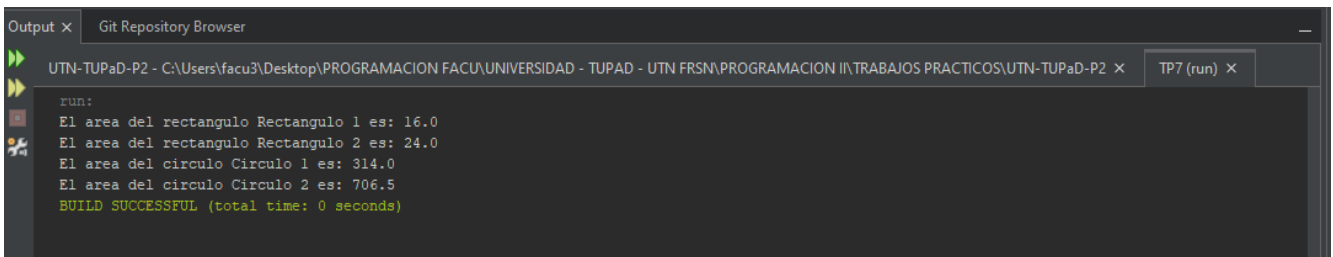
public class Main_E2 {
    public static void main(String[] args) {
        // Creamos un array list de figuras
        ArrayList<Figura> figuras = new ArrayList<>();

        // Creamos y añadimos figuras al array
    }
}
```

```
figuras.add(new Rectangulo(4.0, 4.0, "Rectangulo 1"));
figuras.add(new Rectangulo(6.0, 4.0, "Rectangulo 2"));
figuras.add(new Circulo(10.0, "Circulo 1"));
figuras.add(new Circulo(15.0, "Circulo 2"));

// Recorremos el array y llamamos a calcularArea() usando polimorfismo
for (Figura f : figuras) {
    f.calcularArea();
}
}
```

### C. Resultados output



```
Output x Git Repository Browser
UTN-TUPaD-P2 - C:\Users\facu3\Desktop\PROGRAMACION FACU\UNIVERSIDAD - TUPAD - UTN FRSN\PROGRAMACION II\TRABAJOS PRACTICOS\UTN-TUPaD-P2 x TP7 (run) x
run:
El area del rectangulo Rectangulo 1 es: 16.0
El area del rectangulo Rectangulo 2 es: 24.0
El area del circulo Circulo 1 es: 314.0
El area del circulo Circulo 2 es: 706.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 3. Empleados y polimorfismo

### A. Estructura de Archivos:

Empleado.java

EmpleadoPlanta.java

EmpleadoTemporal.java

Main\_E3.java

### B. Código

Empleado.java

package CASO3;

public abstract class Empleado {

// El método debe recibir un objeto del tipo GENÉRICO Empleado.

public double calcularSueldo(Empleado e) { // <-- ¡AQUÍ ESTÁ LA CLAVE!

if (e instanceof EmpleadoPlanta) {

return 900000.0;

} else if (e instanceof EmpleadoTemporal) {

return 850000.0;

} else {

return 0.0; // Caso por defecto

}

```
}  
}
```

EmpleadoPlanta.java

```
package CASO3;
```

```
public class EmpleadoPlanta extends Empleado {  
}
```

EmpleadoTemporal.java

```
package CASO3;
```

```
public class EmpleadoTemporal extends Empleado {  
}
```

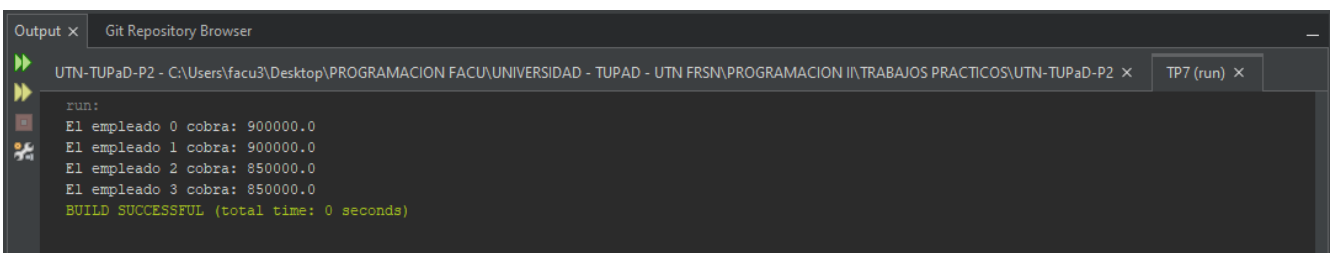
Main\_E3.java

```
package CASO3;
```

```
import java.util.ArrayList;
```

```
public class Main_E3 {  
    public static void main(String[] args) {  
        // Inicializamos un array de empleados  
        ArrayList<Empleado> empleados = new ArrayList<>();  
  
        // Creamos empleados y los agregamos al array (Upcasting implícito)  
        empleados.add(new EmpleadoPlanta());  
        empleados.add(new EmpleadoPlanta());  
        empleados.add(new EmpleadoTemporal());  
        empleados.add(new EmpleadoTemporal());  
  
        int i = 0;  
        // Recorremos el array y llamamos al metodo para calcular sueldo  
        for (Empleado e : empleados) {  
            // Se invoca el método del padre, que internamente usa instanceof  
            System.out.println("El empleado " + i + " cobra: " + e.calcularSueldo(e));  
            i++;  
        }  
    }  
}
```

### C. Resultados output



```
Output x Git Repository Browser  
UTN-TUPaD-P2 - C:\Users\facu3\Desktop\PROGRAMACION FACU\UNIVERSIDAD - TUPAD - UTN FRSN\PROGRAMACION II\TRABAJOS PRACTICOS\UTN-TUPaD-P2 x TP7 (run) x  
run:  
El empleado 0 cobra: 900000.0  
El empleado 1 cobra: 900000.0  
El empleado 2 cobra: 850000.0  
El empleado 3 cobra: 850000.0  
BUILD SUCCESSFUL (total time: 0 seconds)
```

#### 4. Animales y comportamiento sobrescrito

##### A. Estructura de Archivos:

Animal.java

Perro.java

Gato.java

Vaca.java

Main\_E4.java

##### B. Código

Animal.java

```
package CASO4;
```

```
public class Animal {  
    // Método por defecto  
    public void hacerSonido() {  
        System.out.println("Sonido de animal genérico...");  
    }  
  
    // Método adicional  
    public void describirAnimal() {  
        // Lógica de descripción  
    }  
}
```

Perro.java

```
package CASO4;
```

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

Gato.java

```
package CASO4;
```

```
public class Gato extends Animal {  
    @Override  
    public void hacerSonido() {  
        System.out.println("Miau!!");  
    }  
}
```



```
}  
}
```

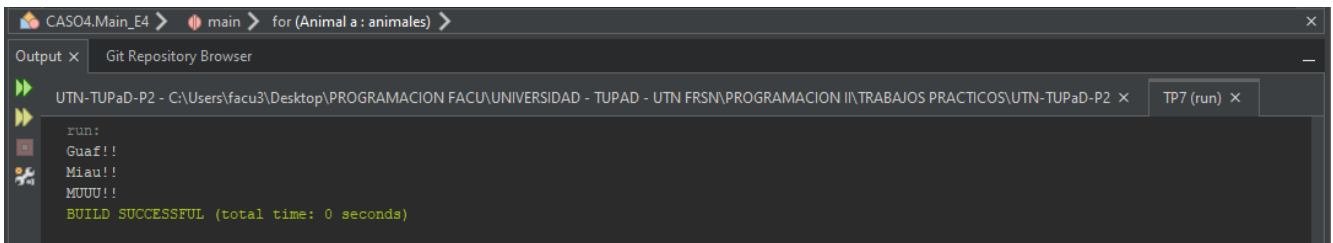
Vaca.java

```
package CASO4;  
  
public class Vaca extends Animal {  
    @Override  
    public void hacerSonido() {  
        System.out.println("MUUU!!");  
    }  
}
```

Main\_E4.java

```
package CASO4;  
  
import java.util.ArrayList;  
  
public class Main_E4 {  
    public static void main(String[] args) {  
        // Inicializamos array de animales  
        ArrayList<Animal> animales = new ArrayList<>();  
  
        // Creamos y agregamos animales al array  
        animales.add(new Perro());  
        animales.add(new Gato());  
        animales.add(new Vaca());  
  
        // Recorremos el array y llamamos al metodo hacerSonido() (Polimorfismo)  
        for (Animal a : animales) {  
            a.hacerSonido();  
        }  
    }  
}
```

### C. Resultados output



Link de repo: <https://github.com/Dario-Cabrera/UTN-TUPaD-P2.git>

Alumno: Cabrera Dario Ezequiel

DNI: 41375492