PROGRAMACION II.
Modulo 7: Herencia y polimorfismo.

Trabajo práctico 7: Herencia y polimorfismo.

Alumno: LEPKA AGUSTIN
Comisión: 13

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.

Clase MainEj1:

```
/*
PROGRAMACION II
MODULO 7: HERENCIA Y POLIMORFISMO
ALUMNO: LEPKA AGUSTIN
COMISION: 13
*/
package Ejercicio1;

public class MainEj1 {

    public static void main(String[] args) {
       //creamos un objeto Auto utilizando upcasting (tratado como
Vehiculo)
        Vehiculo miAuto = new Auto("Toyota", "Corolla", 4);

        //llamamos al método que mostrará info
        miAuto.mostrarInfo();
    }

}
```

Clase Vehiculo:

```
package Ejercicio1;

//Clase base Vehiculo
public class Vehiculo {

    //atributos visibles solo en la clase o clases hijas
    protected String marca;
    protected String modelo;

    //constructor que inicializa atributos del vehículo
    public Vehiculo(String marca, String modelo) {
        this.marca = marca;   //Guarda la marca recibida
        this.modelo = modelo; //Guarda el modelo recibido
    }

    //metodo para mostrar información del vehículo
    public void mostrarInfo() {
        System.out.println("Marca: " + marca + ", Modelo: " + modelo);
    }
}
```

Clase Auto:

```
package Ejercicio1;

//clase Auto hereda de Vehiculo usando extends
public class Auto extends Vehiculo {

    //atributo propio de Auto
    private int cantidadPuertas;

    //constructor que recibe variables y llama al constructor padre
con super(...)
    public Auto(String marca, String modelo, int cantidadPuertas) {
        super(marca, modelo);            //Llama al constructor de
Vehiculo
        this.cantidadPuertas = cantidadPuertas; //Guarda cantidad de
puertas
    }

    //se sobreescribe el método mostrarInfo para agregar más datos
    @Override
    public void mostrarInfo() {
        super.mostrarInfo(); //se llama al método mostrarInfo de
Vehiculo
        System.out.println("Puertas: " + cantidadPuertas);
    }
}
```

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.

Clase MainEj2:

```
/*
PROGRAMACION II
MODULO 7: HERENCIA Y POLIMORFISMO
ALUMNO: LEPKA AGUSTIN
COMISION: 13
*/
package Ejercicio2;


public class MainEj2 {

    public static void main(String[] args) {

        //Creamos un arreglo de tipo Figura que almacena hijos
        Figura[] figuras = {
            new Circulo(4), //Circulo con radio 4
            new Rectangulo(5, 10)//Rectángulo 5x10
        };

        //Recorremos e invocamos métodos de polimorfismo
        for (Figura f : figuras) {
            f.mostrar(); //cada figura usa su propia versión de
calcularArea
```

```
            }


    }
}
```

Clase Figura:

```
package Ejercicio2;
//Clase abstracta base para figuras
public abstract class Figura {

    //Nombre de la figura accesible en clases hijas
    protected String nombre;

    //Constructor para asignar nombre
    public Figura(String nombre) {
        this.nombre = nombre;
    }

    //Método abstracto: las clases hijas deben implementarlo
    public abstract double calcularArea();

    //Método común para mostrar datos
    public void mostrar() {
        System.out.println(nombre + " - Area: " + calcularArea());
    }
}
```

Clase Circulo:

```
package Ejercicio2;

//Clase Circulo que hereda de Figura
public class Circulo extends Figura {

    //Atributo específico del círculo
    private double radio;

    //Constructor del círculo
    public Circulo(double radio) {
        super("Circulo"); //Enviamos nombre a la clase madre
        this.radio = radio;
    }

    //Implementación del cálculo de área
    @Override
    public double calcularArea() {
        return Math.PI * radio * radio;
    }
}
```

Clase Rectangulo:

```
package Ejercicio2;

//Clase Rectangulo que hereda de Figura
public class Rectangulo extends Figura {
```

```
    //Atributos exclusivos del rectángulo
    private double ancho;
    private double alto;

    //Constructor del rectángulo
    public Rectangulo(double ancho, double alto) {
        super("Rectangulo"); //Se envía el nombre a Figura
        this.ancho = ancho;
        this.alto = alto;
    }

    //Implementamos el cálculo del área
    @Override
    public double calcularArea() {
        return ancho * alto;
    }
}
```

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

Clase MainEj3:

```
/*
PROGRAMACION II
MODULO 7: HERENCIA Y POLIMORFISMO
ALUMNO: LEPKA AGUSTIN
COMISION: 13
*/
package Ejercicio3;
import java.util.ArrayList;
public class MainEj3 {


    public static void main(String[] args) {
        //Lista polimórfica con Empleado
        ArrayList<Empleado> empleados = new ArrayList<>();

        //Agregamos distintos tipos de empleado
        empleados.add(new EmpleadoPlanta("Ana", 500000));
        empleados.add(new EmpleadoTemporal("Luis", 20, 15000));

        //Recorremos la lista y usamos polimorfismo
        for (Empleado e : empleados) {
            System.out.println(e.nombre + " cobra $" +
e.calcularSueldo());

            //instanceof para identificar tipo real
            if (e instanceof EmpleadoTemporal) {
                EmpleadoTemporal temp = (EmpleadoTemporal) e;
//downcasting
                System.out.println("Días trabajados: " +
temp.getDiasTrabajados());
            }
    }
```

```
}

}
```

Clase Empleado:

```
package Ejercicio3;

//clase abstracta Empleado
public abstract class Empleado {

    //Nombre del empleado
    protected String nombre;

    //Constructor
    public Empleado(String nombre) {
        this.nombre = nombre;
    }

    //metodo abstracto: cada tipo de empleado define su sueldo
    public abstract double calcularSueldo();
}
```

Clase EmpleadoPlanta:

```
package Ejercicio3;

//Empleado de planta que tiene sueldo fijo
public class EmpleadoPlanta extends Empleado {

    private double sueldoBase;

    public EmpleadoPlanta(String nombre, double sueldoBase) {
        super(nombre); // enviamos nombre a la clase padre
        this.sueldoBase = sueldoBase;
    }

    //Implementamos sueldo fijo
    @Override
    public double calcularSueldo() {
        return sueldoBase;
    }
}
```

Clase EmpleadoTemporal:

```
package Ejercicio3;

//Empleado temporal que cobra por día trabajado
public class EmpleadoTemporal extends Empleado {

    private int diasTrabajados;
    private double pagoPorDia;

    public EmpleadoTemporal(String nombre, int diasTrabajados, double
pagoPorDia) {
        super(nombre);
        this.diasTrabajados = diasTrabajados;
        this.pagoPorDia = pagoPorDia;
    }
```

```
    @Override
    public double calcularSueldo() {
        return diasTrabajados * pagoPorDia;
    }

    //metodo exclusivo para acceder días
    public int getDiasTrabajados() {
        return diasTrabajados;
    }
}
```

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

Clase MainEj4:

```
/*
PROGRAMACION II
MODULO 7: HERENCIA Y POLIMORFISMO
ALUMNO: LEPKA AGUSTIN
COMISION: 13
*/

package Ejercicio4;

public class MainEj4 {

    public static void main(String[] args) {
        //Array de animales (con polimorfismo)
        Animal[] animales = { new Perro(), new Gato(), new Vaca() };

        //Llamamos métodos
        for (Animal a : animales) {
            a.describirAnimal();//método heredado
            a.hacerSonido();//método sobrescrito según el animal
            System.out.println("----");
        }
    }

}
```

Clase Animal:

```
package Ejercicio4;

//Clase base Animal
public class Animal {

    //metodo general de sonido
    public void hacerSonido() {
        System.out.println("Sonido genérico");
    }

    //describe el animal
```

```
    public void describirAnimal() {
        System.out.println("Soy un animal");
    }
}
```

Clase Perro:

```
package Ejercicio4;

public class Perro extends Animal {

    //Sobrescribimos comportamiento
    @Override
    public void hacerSonido() {
        System.out.println("Guau!");
    }
}
```

Clase Vaca:

```
package Ejercicio4;

public class Vaca extends Animal {

    @Override
    public void hacerSonido() {
        System.out.println("Muu!");
    }
}
```

Clase Gato:

```
package Ejercicio4;

public class Gato extends Animal {

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