Ejercicios Programación Orientada a Objetos (POO)

1. Crear una clase Persona que contenga dos variables: nombre y edad, luego un método para mostrar el nombre y edad de varios objetos de la clase Persona.

```
package poo;

import java.util.Scanner;

public class Ejercicio1 {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		Persona persona1 = new Persona();

		System.out.println("Introduce tú nombre: ");

	  persona1.nombre = sc.next();

	  System.out.println("Introduce tú edad: ");

	  persona1.edad = sc.nextInt();

	  persona1.saludar();

	  Persona persona2 = new Persona();

	  System.out.println("Introduce tú nombre: ");

	  persona2.nombre = sc.next();

	  System.out.println("Introduce tú edad: ");

	  persona2.edad = sc.nextInt();

	  persona2.saludar();

	}

}
class Persona {
    String nombre;
    int edad;

    void saludar() {
        System.out.println("Hola, soy " + nombre + " y tengo " + edad
+ " años.");
    }
}
```

2. Crear dos clases que a su vez hereden de una tercera clase y compartan un mismo método, es decir, que se sobrescriba el método en las tres clases.

```
package poo;

public class Ejercicio2 {
```

```
      public static void main(String[] args) {
            Animal animal1 = new Animal();
        animal1.hacerSonido();

        Perro perro1 = new Perro();
        perro1.hacerSonido();

        Gato gato1 = new Gato();
        gato1.hacerSonido();

      }

}
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}
class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}
class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El gato maulla.");
    }
}
```

3. Crear una clase con su constructor, las variables encapsuladas y varios métodos.

```
package poo;

public class Ejercicio3 {

      public static void main(String[] args) {
            Libro libro1 = new Libro("Harry Potter", "J.K. Rowling");
        System.out.println("Título: " + libro1.getTitulo());
        System.out.println("Autor: " + libro1.getAutor());

      }

}
class Libro {
    private String titulo;
    private String autor;

    public Libro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }

    public String getTitulo() {
        return titulo;
    }

    public String getAutor() {
```

```
        return autor;
    }
}
```

4. Supongamos que tienes una clase llamada “Persona” con los atributos “nombre” y “edad”, implementa los métodos getter y setter para estos atributos.

```
Spackage poo;

public class Ejercicio4 {

      public static void main(String[] args) {
            Persona persona = new Persona();

        // Usando el setter para asignar valores
        persona.setNombre("Paco");
        persona.setEdad(25);

        // Usando el getter para obtener valores
        System.out.println("Nombre: " + persona.getNombre());
        System.out.println("Edad: " + persona.getEdad());

      }

}
class Persona {
    private String nombre;
    private int edad;

    // Getter para el atributo 'nombre'
    public String getNombre() {
        return nombre;
    }

    // Setter para el atributo 'nombre'
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Getter para el atributo 'edad'
    public int getEdad() {
        return edad;
    }

    // Setter para el atributo 'edad'
    public void setEdad(int edad) {
        if (edad >= 0) {
            this.edad = edad;
        } else {
            System.out.println("La edad no puede ser negativa.");
        }
    }
}
```

5. Supongamos que tienes una clase llamada “Persona” con los siguientes atributos: “nombre”, “edad” y “sexo”. Implementa una sobrecarga de constructores para la clase “Persona”

```java
package poo;

public class Ejercicio5 {

	public static void main(String[] args) {

		Persona persona1 = new Persona(); // Sin nombre, edad 0, sexo desconocido
		Persona persona2 = new Persona("Juan", 25); // Nombre: Juan, edad: 25, sexo desconocido
		Persona persona3 = new Persona("María", 30, "Femenino"); // Nombre: María, edad: 30, sexo: Femenino

		// Imprimir información de las personas
		System.out.println("Persona 1: " + persona1.getNombre() + ", " + persona1.getEdad() + ", " + persona1.getSexo());
		System.out.println("Persona 2: " + persona2.getNombre() + ", " + persona2.getEdad() + ", " + persona2.getSexo());
		System.out.println("Persona 3: " + persona3.getNombre() + ", " + persona3.getEdad() + ", " + persona3.getSexo());


	}

}
class Persona {
    private String nombre;
    private int edad;
    private String sexo;

    // Constructor sin parámetros
    public Persona3() {
        this.nombre = "Sin nombre";
        this.edad = 0;
        this.sexo = "Desconocido";
    }

    // Constructor con nombre y edad
    public Persona3(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
        this.sexo = "Desconocido";
    }

    // Constructor con nombre, edad y sexo
    public Persona3(String nombre, int edad, String sexo) {
        this.nombre = nombre;
        this.edad = edad;
        this.sexo = sexo;
    }

    // Getter para el atributo 'nombre'
    public String getNombre() {
        return nombre;
    }
    // Getter para el atributo 'edad'
    public int getEdad() {
        return edad;
    }
    // Getter para el atributo 'edad'
    public String getSexo() {
```

```
        return sexo;
    }

}
```

6. Supongamos que tenemos una jerarquía de clases que representa diferentes tipos de animales. En el método **main** crear un array de objetos de tipo **Animal** que pueden contener instancias de **Perro**, **Gato** y **Vaca**. Luego llamar al método **hacerSonido()** en cada uno de ellos, y utilizar el polimorfismo que se encargará de ejecutar la versión correspondiente de ese método para cada tipo de animal

```
package poo;

public class Ejercicio6 {

      public static void main(String[] args) {
            Animal[] animales = new Animal[3];
        animales[0] = new Perro2();
        animales[1] = new Gato2();
        animales[2] = new Vaca2();

        for (Animal animal : animales) {
            animal.hacerSonido();
        }

      }

}
class Animal {
    public void hacerSonido() {
        System.out.println("Sonido desconocido");
    }
}

class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("Woof woof");
    }
}

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

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

7. Supongamos que tenemos una clase padre “Animal” y una subclase “Perro” que comparten un método. Realizar un casting de una subclase a una superclase y un casting de una superclase a una subclase (con instanceof para evitar excepciones.

```
package poo;

public class Ejercicio7 {

	public static void main(String[] args) {
		// Casting de una subclase a una superclase
		Perro perro = new Perro();
		Animal animal = (Animal) perro;
		animal.hacerSonido(); // Salida: El perro hace: ¡Guau!

		//Casting de una superclase a una subclase
		Animal animal2 = new Animal();
		if (animal instanceof Perro) {
		    Perro perro2 = (Perro) animal;
		    perro2.hacerSonido();
		} else {
		    System.out.println("El animal no es un perro");
		}

	}

}
class Animal3 {
    public void hacerSonido() {
        System.out.println("Haciendo un sonido indefinido");
    }
}

class Perro3 extends Animal3 {
    public void hacerSonido() {
        System.out.println("El perro hace: ¡Guau!");
    }

}
```

8. Creamos una clase abstracta llamada **Figura** que servirá como base para otras clases de figuras geométricas concretas como **Circulo** y **Rectangulo**. Cada figura tiene un método abstracto para calcular su área.

```
package poo;

public class Ejercicio8 {

	public static void main(String[] args) {
	  Circulo circulo = new Circulo(5);
	  System.out.println("Área del círculo: " +
circulo.calcularArea());

	  Rectangulo rectangulo = new Rectangulo(4, 6);
```

```
        System.out.println("Área del rectángulo: " +
rectangulo.calcularArea());

      }

}
abstract class Figura {
    abstract double calcularArea();
}

class Circulo extends Figura {
    double radio;

    Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    double calcularArea() {
        return Math.PI * radio * radio;
    }
}

class Rectangulo extends Figura {
    double base;
    double altura;

    Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    double calcularArea() {
        return base * altura;
    }
}
```

9. Crear y utilizar un tipo enumerado en Java para los días de la semana, que nos indique si es laboral o fin de semana.

```
package poo;

public enum DiaSemana {

      LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO

}
```

```
package poo;

import java.util.Scanner;

public class Ejercicio9 {

      public static void main(String[] args) {
            Scanner sn = new Scanner(System.in);
```

```
        System.out.println("Escribe un dia de la semana");
        String dia = sn.next();

        DiaSemana diaS = DiaSemana.valueOf(dia.toUpperCase());

        switch(diaS){
            case LUNES:
            case MARTES:
            case MIERCOLES:
            case JUEVES:
            case VIERNES:
                System.out.println("El dia
"+diaS.name().toLowerCase()+" es laborable");
                break;
            case SABADO:
            case DOMINGO:
                System.out.println("El dia "+diaS+" no es laborable");
                break;

        }

    }

}
```