UCSM Esc. Prof. de Ingeniería de Sistemas

Agosto - 2024

INFORME DE PRÁCTICAS

LENGUAJES DE PROGRAMACIÓN II

Práctica N° 2

Elaborado por:

Choque Taco, Salvador Lincoln

Cusirramos Chiri, Santiago Jesus

RECONOCIMIENTOS

El autor de este trabajo reconoce con gratitud a los creadores de los lenguajes JAVA y otras personalidades y autores de libros de programación Bjarne Stroustrup, Dennis Ritchie, Herb Sutter, Herb Sutter, James Gosling, James Gosling, Brian Kernighan, Brian Kernighan, Ken Thompson.

PALABRAS CLAVES

.

ÍNDICE

1. RESÚMEN

En la vida real existen clases sin elementos, por ejemplo, la clase animal no posee elementos, en cambio las clases descendientes como perro o vaca si los poseen, ese es el objetivo de tener calase abstractas, poseer definiciones de un tipo que sirve para especificar características para clases derivadas. Una clase abstracta no instancia objetos, las clases abstractas se convierten en tales debido a que incluyen un método virtual puro, una abstractas tiene por lo menos un método virtual puro; los métodos son virtuales puros al igualarlos a 0 en la declaración, la definición del método no se hace en la clase base, para cada clase derivada de una clase abstracta que no implemente dicho método se convierte siempre en abstracta también.

2. INTRODUCCIÓN

En esta sesión introductoria, exploraremos los fundamentos de Java, incluyendo su sintaxis básica, estructuras de control y el manejo de memoria. Compararemos Java con otros lenguajes populares como C++ y Python, destacando sus similitudes y diferencias. Además, examinaremos las diferencias entre lenguajes compilados e interpretados y discutiremos las ventajas que Java ofrece sobre otros lenguajes de programación. También abordaremos las versiones más utilizadas del JDK, destacando su importancia en el entorno de desarrollo.

A través de ejemplos prácticos y ejercicios, los estudiantes tendrán la oportunidad de aplicar sus conocimientos previos en C++ y Python para escribir código eficiente y funcional en Java, estableciendo una base sólida para un aprendizaje continuo y avanzado en programación.

3. MARCO TEÓRICO

En el campo de la programación, el entendimiento de los conceptos básicos y las características de los lenguajes de programación es crucial para el desarrollo de aplicaciones eficientes y efectivas. El marco teórico siguiente proporciona una base para comprender cómo Java se posiciona en el ecosistema de lenguajes de programación y su relación con otros lenguajes populares.

1. Lenguajes de Programación

Lenguajes Compilados e Interpretados: Los lenguajes de programación se dividen en dos categorías principales: compilados e interpretados. Un lenguaje compilado, como C++ o Go, requiere un proceso de compilación previo a la ejecución. Este proceso convierte el código fuente en lenguaje de máquina, que es directamente ejecutado por la computadora. En contraste, los lenguajes interpretados, como Python o JavaScript, son convertidos a lenguaje de máquina en tiempo real, durante la ejecución del programa.

Java: Un Caso Especial: Java combina características de ambos enfoques. El código Java se compila a un formato intermedio llamado bytecode, que es independiente de la plataforma. Este bytecode es luego interpretado por la Java Virtual Machine (JVM), permitiendo que el código Java sea ejecutado en cualquier plataforma que tenga una JVM instalada. Este enfoque permite a Java lograr portabilidad y flexibilidad, apoyando el principio de "escribir una vez, ejecutar en cualquier lugar" (WORA).

2. Características del Lenguaje Java

Portabilidad: Una de las principales ventajas de Java es su portabilidad. El bytecode generado por el compilador Java puede ejecutarse en cualquier plataforma que tenga una JVM compatible, lo que elimina la necesidad de recompilar el código para diferentes sistemas operativos y hardware.

Seguridad: Java proporciona un entorno de ejecución seguro mediante la JVM, que realiza una serie de verificaciones de seguridad durante la ejecución del bytecode. Estas verificaciones ayudan a prevenir la ejecución de código malicioso y a proteger el sistema operativo subyacente.

Robustez: Java incluye mecanismos avanzados de manejo de errores y excepciones, lo que contribuye a la estabilidad del programa. La recolección de basura automática también ayuda a gestionar la memoria de manera eficiente, reduciendo la posibilidad de errores relacionados con la memoria.

Multihilo: Java soporta la programación concurrente mediante su API de hilos, permitiendo que múltiples tareas se ejecuten simultáneamente dentro de un programa. Esto es especialmente útil para aplicaciones que requieren operaciones simultáneas, como servidores y aplicaciones de usuario.

3. Comparación con Otros Lenguajes

C++ vs. Java: C++ es un lenguaje compilado que ofrece control detallado sobre el hardware y la memoria, permitiendo una optimización más fina. Sin embargo, esto también puede resultar en una mayor complejidad y una mayor posibilidad de errores relacionados con la memoria. Java, por otro lado, proporciona una capa adicional de seguridad y manejo automático de memoria, lo que facilita el desarrollo a expensas de un menor control sobre el hardware.

Python vs. Java: Python es un lenguaje interpretado conocido por su simplicidad y facilidad de uso, ideal para desarrollo rápido y scripting. Aunque Python permite un desarrollo ágil, su rendimiento puede ser inferior al de Java en aplicaciones de alto rendimiento debido a la naturaleza interpretada del lenguaje. Java, con su bytecode y ejecución en la JVM, ofrece un rendimiento más consistente y escalable para aplicaciones más grandes y complejas.

4. MODELAMIENTO – ACTIVIDADES

4.1 Actividad 1

Diagrama de clases

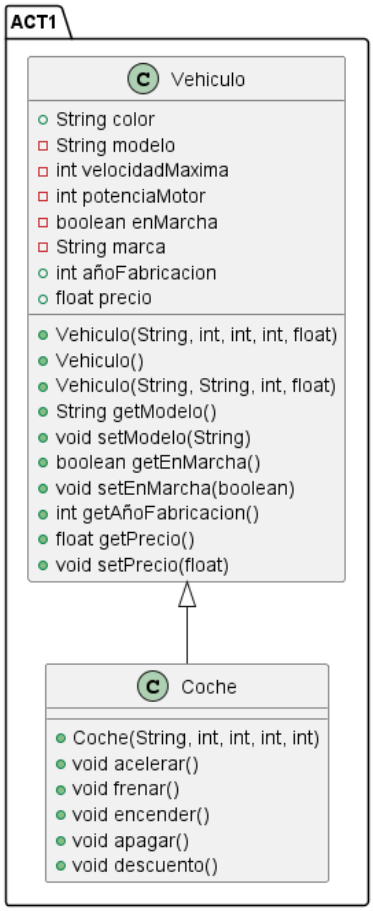
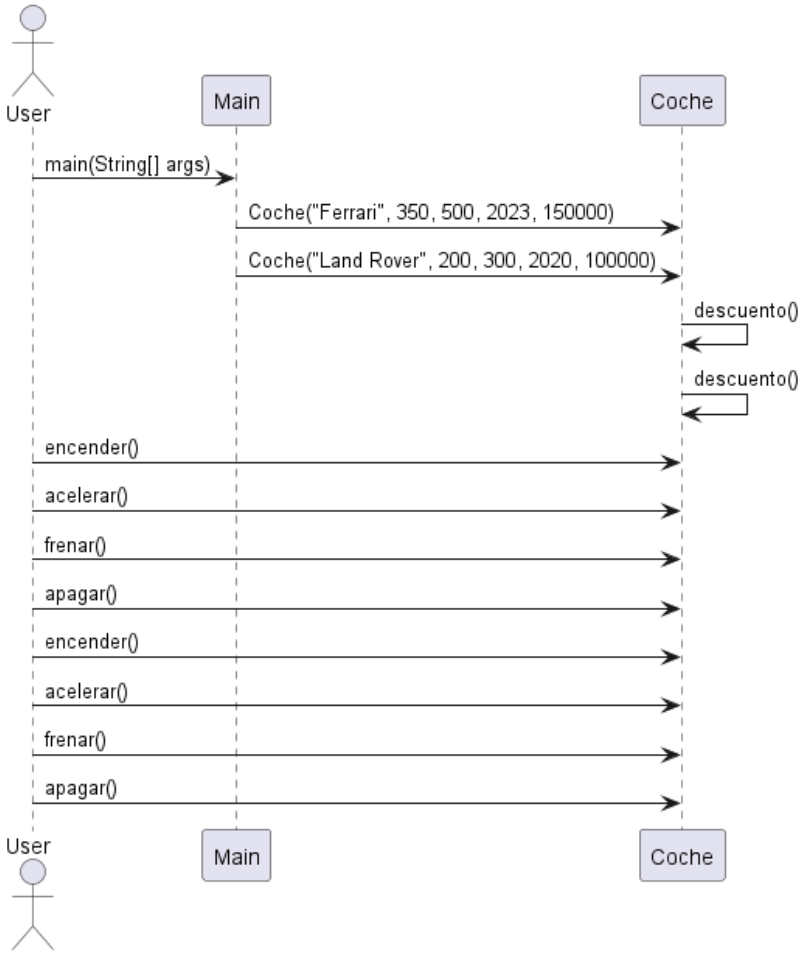


Diagrama de secuencia



4.2 Actividad 2

Diagrama de clases

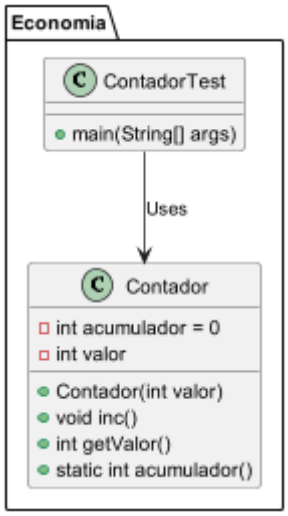
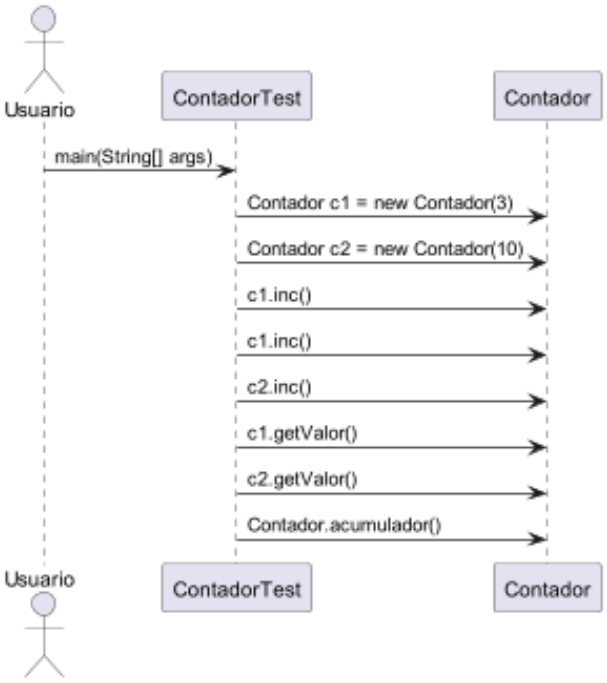


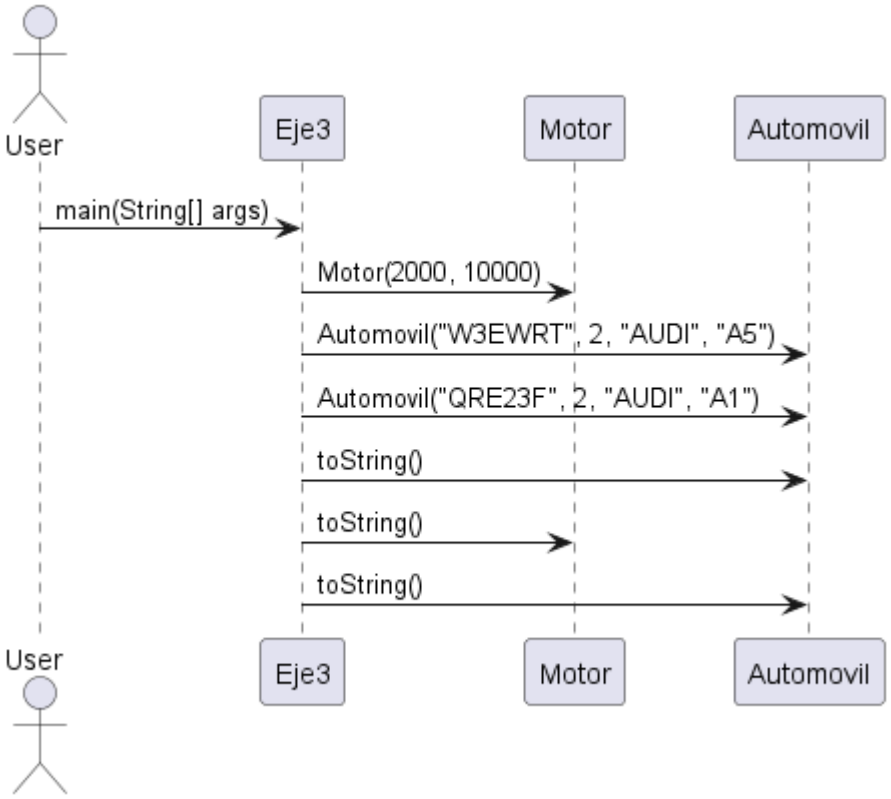
Diagrama de secuencia



4.3 Actividad 3

Diagrama de clases

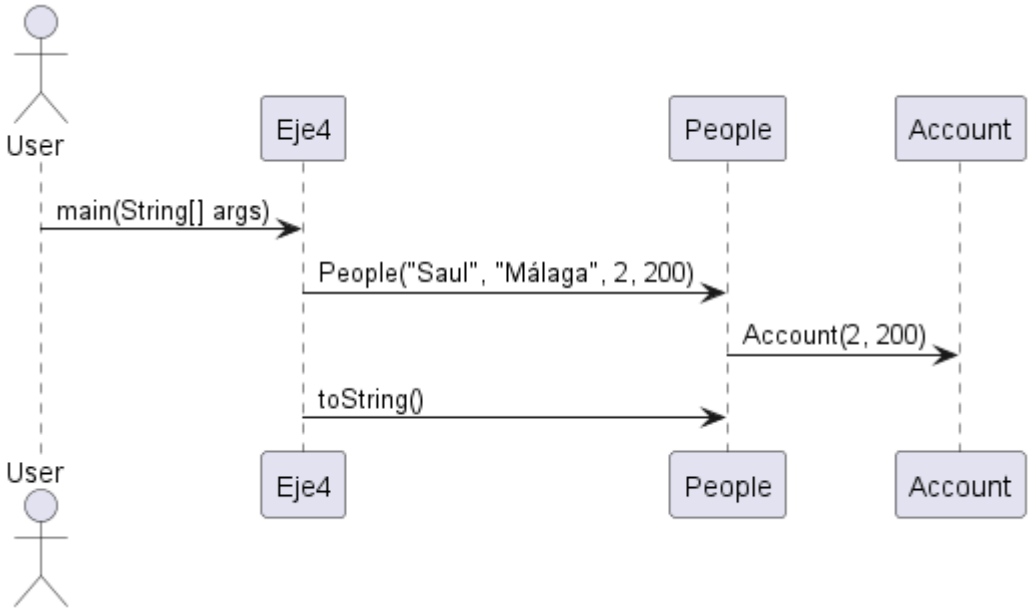
Diagrama de secuencia



4.4 Actividad 4

Diagrama de clases

Diagrama de secuencia



4.5 Actividad 5

Diagrama de clases

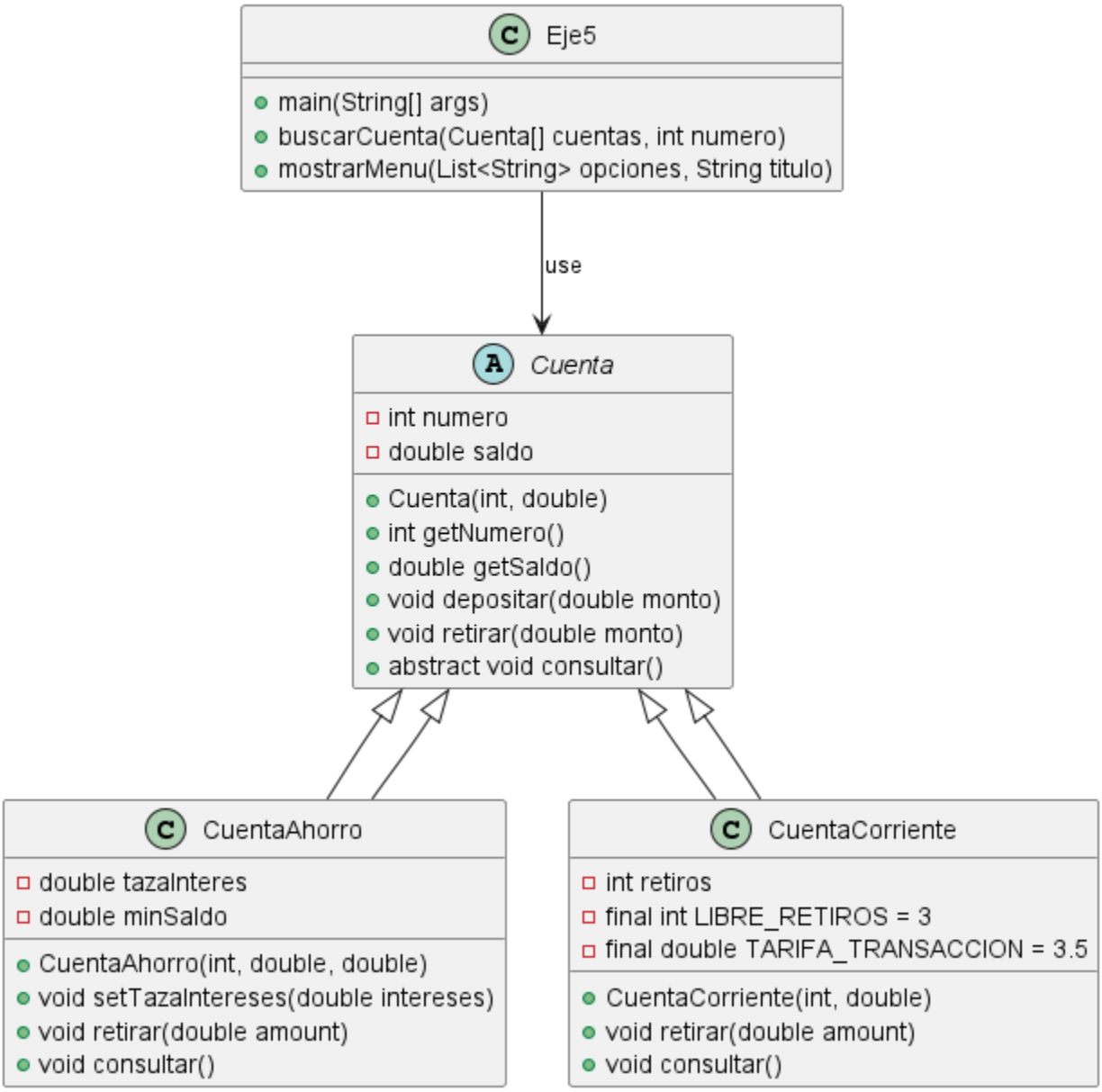
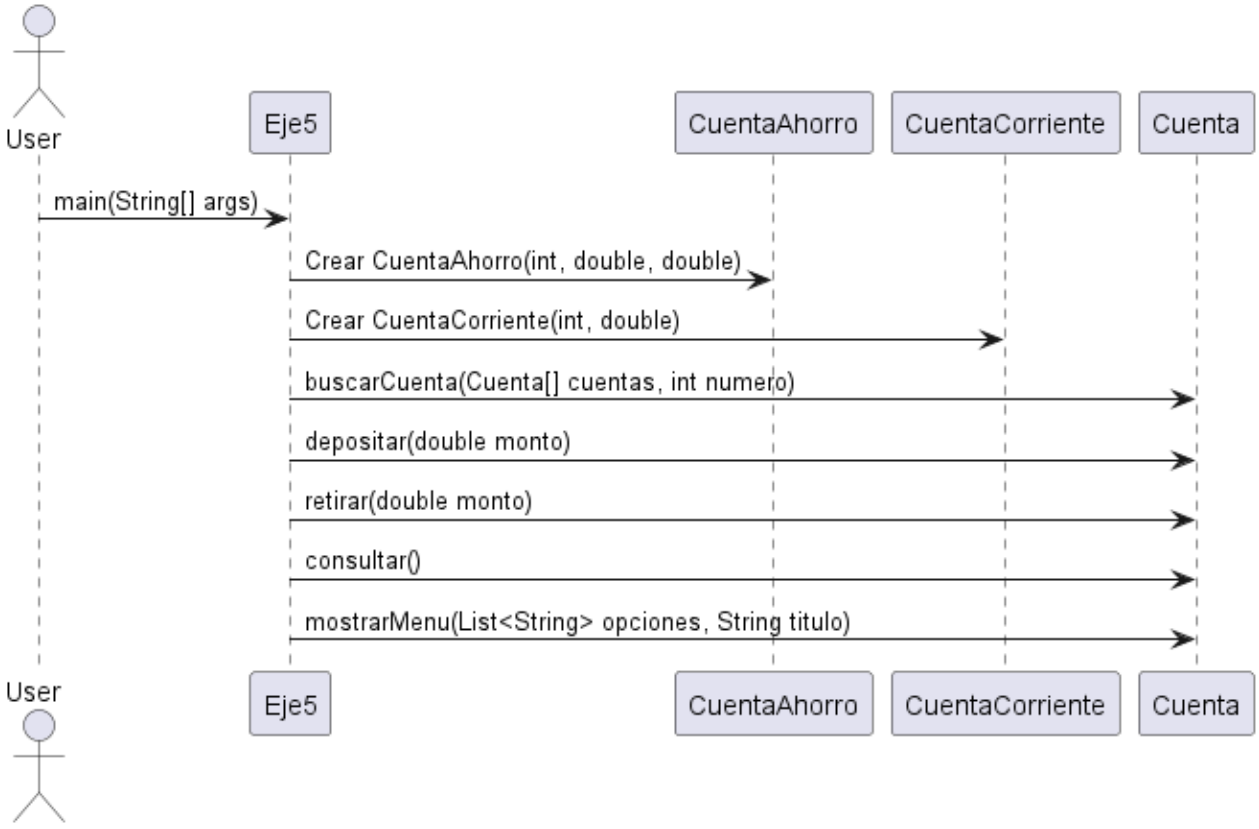


Diagrama de secuencia



5. MODELAMIENTO - EJERCICIO

5.1 EJERICIO 1

Diagrama de clases

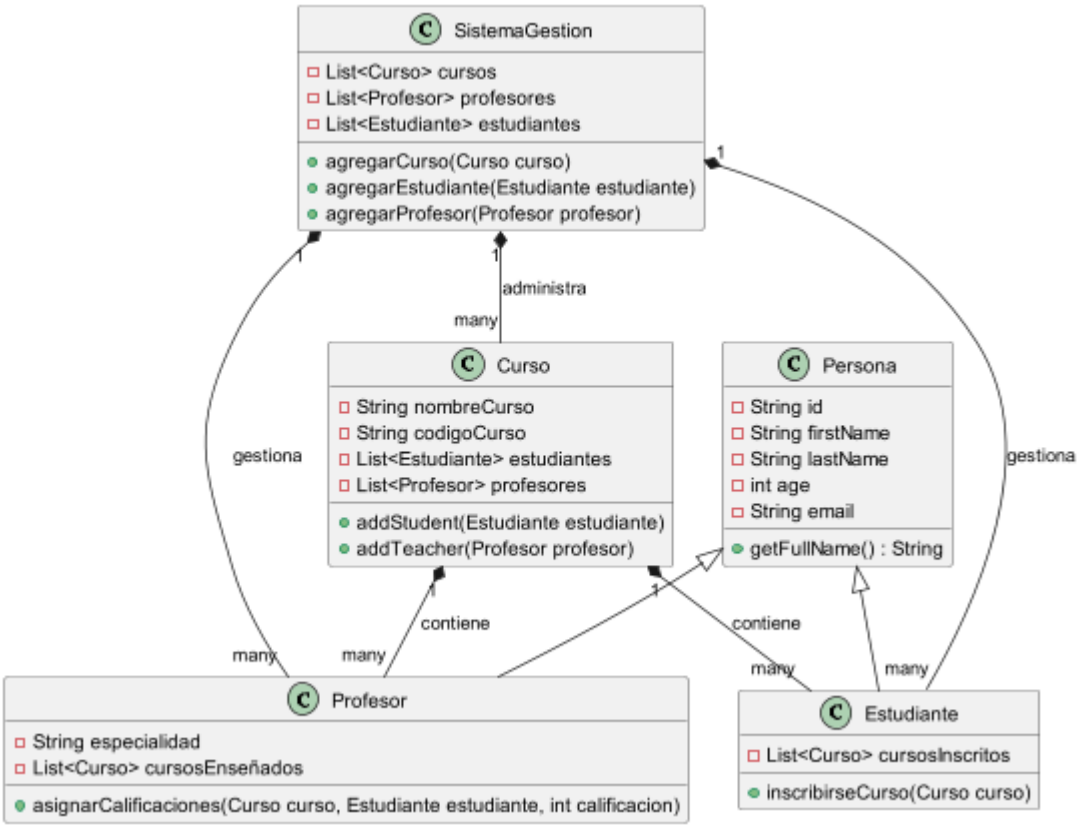
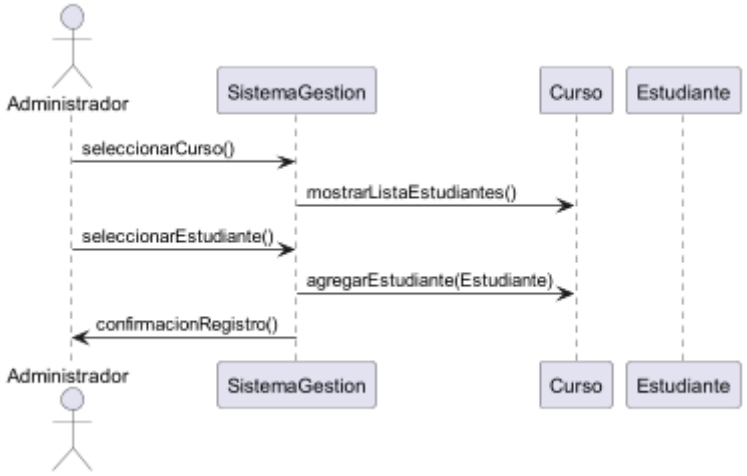


Diagram de secuencia



6. ACTIVIDADES

1. ACTIVIDAD 1

Vehículo.java

```
1. package ACT1;
2.
```

```
 3. class Vehiculo {
 4.     public String color;
 5.     private String modelo;
 6.     private int velocidadMaxima;
 7.     private int potenciaMotor;
 8.     private boolean enMarcha;
 9.     private String marca;
10.     public int añoFabricacion;
11.     public float precio;
12.
13.     public Vehiculo(String modelo, int velocidadMaxima, int potenciaMotor, int añoFabricacion,
float precio) {
14.         this.modelo = modelo;
15.         this.velocidadMaxima = velocidadMaxima;
16.         this.potenciaMotor = potenciaMotor;
17.         this.enMarcha = false;
18.         this.añoFabricacion = añoFabricacion;
19.         this.precio = precio;
20.     }
21.
22.     public Vehiculo() {
23.         this.modelo = "BMW";
24.         this.velocidadMaxima = 300;
25.         this.potenciaMotor = 120;
26.         this.enMarcha = false;
27.         this.añoFabricacion = 2012;
28.         this.precio = 50000;
29.     }
30.
31.     public Vehiculo(String marca, String modelo, int añoFabricacion, float precio) {
32.         this.marca = marca;
33.         this.modelo = modelo;
34.         this.añoFabricacion = 300;
35.         this.precio = 12000;
36.     }
37.
38.     public String getModelo() {
39.         return modelo;
40.     }
41.
42.     public void setModelo(String modelo) {
43.         this.modelo = modelo;
44.     }
45.
46.     public boolean getEnMarcha() {
47.         return enMarcha;
48.     }
49.
50.     public void setEnMarcha(boolean enMarcha) {
51.         this.enMarcha = enMarcha;
52.     }
53.
54.     public int getAñoFabricacion() {
55.         return añoFabricacion;
56.     }
57.
58.     public float getPrecio() {
59.         return precio;
60.     }
61.
62.     public void setPrecio(float precio) {
63.         this.precio = precio;
64.     }
65. }
66.
```

Coche.java

```java
package ACT1;

class Coche extends Vehiculo {

    Coche(String modelo, int velocidadMaxima, int potenciaMotor, int añoFabricacion, int precio)
{
        super(modelo, velocidadMaxima, potenciaMotor, añoFabricacion, precio);
    }

    public void acelerar() {
        if (getEnMarcha()) {
            System.out.println("El coche " + getEnMarcha() + " esta acelerando.");
        } else {
            System.out.println("Primero enciende el coche.");
        }
    }

    public void frenar() {
        if (getEnMarcha()) {
            System.out.println("El coche " + getEnMarcha() + " esta frenando.");
        } else {
            System.out.println("El coche esta apagado, no se puede frenar.");
        }
    }

    public void encender() {
        setEnMarcha(true);
        System.out.println("El coche " + getEnMarcha() + " se ha encendido.");
    }

    public void apagar() {
        setEnMarcha(false);
        System.out.println("El coche " + getModelo() + " se ha apagado.");
    }

    public void descuento() {
        if (getAñoFabricacion() > 2010) {
            System.out.println("El coche " + getModelo() + " esta con descuento.");
            float precioDescuento = (float) (getPrecio() * 0.90);
            setPrecio(precioDescuento);
        }
    }
}

```

Main.java

```java
package ACT1;

public class Main {
    public static void main(String[] args) {
        Coche cocheDeportivo = new Coche("Ferrari", 350, 500, 2023, 150000);
        Coche cocheTodoTerreno = new Coche("Land Rover", 200, 300, 2020, 100000);

        cocheDeportivo.descuento();
        cocheTodoTerreno.descuento();

        System.out.println("1ro el " + cocheDeportivo.getModelo() + " - Coche Deportivo");
        cocheDeportivo.encender();
        cocheDeportivo.acelerar();
        cocheDeportivo.frenar();
        cocheDeportivo.apagar();

```

```
17.         System.out.println("2do el " + cocheTodoTerreno.getModelo() + " - Coche Todo Terreno");
18.         cocheTodoTerreno.encender();
19.         cocheTodoTerreno.acelerar();
20.         cocheTodoTerreno.frenar();
21.         cocheTodoTerreno.apagar();
22.     }
23. }
24.
```

```
semestre\1. Lenguaje de Programacion III\1ra
El coche Ferrari esta con descuento.
El coche Land Rover esta con descuento.

1ro el Ferrari - Coche Deportivo
El coche true se ha encendido.
El coche true esta acelerando.
El coche true esta frenando.
El coche Ferrari se ha apagado.

2do el Land Rover - Coche Todo Terreno
El coche true se ha encendido.
El coche true esta acelerando.
El coche true esta frenando.
El coche Land Rover se ha apagado.

Process finished with exit code 0
```

2. ACTIVIDAD 2

```
1. package Economia;
2.
3. public class ContadorTest {
4.     public static void main(String [] args ) {
5.         Contador c1, c2;
6.         System.out.println(Contador.acumulador());
7.
8.         c1 = new Contador(3);
9.         c2 = new Contador(10);
10.
11.        c1.inc();
12.        c1.inc();
13.        c2.inc();
14.
15.        System.out.println(c1.getValor());
16.        System.out.println(c2.getValor());
17.
```

```
18.         System.out.println(Contador.acumulador());
19.     }
20. }
21.
```

```
 1. package Economia;
 2.
 3. public class Contador {
 4.     static int acumulador  = 0;
 5.     private int valor;
 6.
 7.     public static int acumulador() {
 8.         return acumulador;
 9.     }
10.
11.     public Contador(int valor) {
12.         this.valor = valor;
13.         acumulador += valor;
14.     }
15.
16.     public void inc() {
17.         valor++;
18.         acumulador++;
19.     }
20.
21.     public int getValor(){
22.         return this.valor;
23.     }
24. }
25.
```

a) ¿Se pueden realizar las siguientes modificaciones en el código de la clase Contador, sin que cambie el funcionamiento de la clase? ¿Por qué?

a.1. Cambiar en el constructor Contador, la instrucción:

acumulador += valor por this.acumulador += valor

No cambia el resultado, ya que el THIS hace referencia a la variable estatica que esta presente en la clase

```
1. public Contador(int valor) {
2.     this.valor = valor;
3.     this.acumulador+= valor;
4. }
5.
```

a.2. Cambiar en el constructor Contador, la instrucción:

acumulador += valor por Contador.acumulador += valor

No cambia el resultado, ya que se utiliza el nombre de la clase “Contador, esto hace referencia a la variable estática que está presente en la clase.

```
1. public Contador(int valor) {
2.     this.valor = valor;
3.     Contador.acumulador+= valor;
4. }
```

```
5.
```

a.3. Cambiar en el método inc(), la instrucción:

valor++ por this.valor++

No cambia el resultado, ya que el THIS hace referencia a la variable priavda que esta presente en la clase

b) Luego de los cambios realizados, ¿Qué valores imprime el programa ContadorTest? ¿Por qué?

Los valores imprimidos de ContadorTest son los mismos, esto se debe a que los valores modificados o añadidos, lo que realizan es hacer referencia a esa variables.

c) Si cambiamos en la clase Contador la línea:

static int acumulador = 0 por private static int acumulador = 0

¿aparece algún error? ¿por qué?

```
1. java: non-static variable valor cannot be referenced from a static context
2.
```

Nos da este error, este error se debe a que la variable fue modificada a privada, eso quiere decir que únicamente la variable estática privada la podríamos usar dentro de la clase y no fuera.

Por otro lado, si intentamos imprimir valor siendo privado y modificándolo a público, este también nos daría un error ya que no sería estático.

d) Qué sucede si no inicializamos el valor de la variable acumulador?

```
1. public class Contador {
2.     static int acumulador;
3.     private int valor;
4.
```

En este caso, nosotros no estamos inicializando nuestra variable estatica, sin embargo, cuando llamamos al constructor, volvemos a inicializar el acumulador

```
1. public Contador(int valor) {
2.     this.valor = valor;
3.     Contador.acumulador+= valor;
4. }
5.
```

e) Vamos a agregar una constante (VALOR_INICIAL) a la clase Contador y otro constructor, tal como se muestra a continuación:

```
 1. package Economia;
 2.
 3. public class Contador {
 4.     static int acumulador = 0;
 5.     final static int VALOR_INICIAL = 10;
 6.     private int valor;
 7.
 8.     public static int acumulador() {
 9.         return acumulador;
10.     }
11.
12.     public Contador(int valor) {
13.         this.valor = valor;
14.         Contador.acumulador+= valor;
15.     }
16.
17.     public Contador() {
18.         this(Contador.VALOR_INICIAL);;
19.     }
20.
21.     public void inc() {
22.         this.valor++;
23.         acumulador++;
24.     }
25.
26.     public int getValor(){
27.         return this.valor;
28.     }
29.
30. }
31.
```

f) Fijase en la instrucción "this(Contador.VALOR_INICIAL)". ¿Qué hace está instrucción?

Es una manera de inicializar el VALOR_INICIAL, esto funciona ya que actúa como un encadenamiento de constructores. Es cuando se llama a un constructor desde otro constructor, el encadenamiento de constructores.

g) Agregue las instrucciones que usted crea por conveniente a la clase ContadorTest, las mismas que evidencien el funcionamiento de la clase modificada.

```
 1. public Contador() {
 2.     this(Contador.VALOR_INICIAL);
 3. }
 4.
 5. public static int getValorInicial() {
 6.     return VALOR_INICIAL;
 7. }
 8.
 9. c1 = new Contador(3);
10. c2 = new Contador(10);
11. Contador c3 = new Contador();
12.
```

```
13. c1.inc();
14. c1.inc();
15. c2.inc();
16.
17. System.out.println(c1.getValor());
18. System.out.println(c2.getValor());
19.
20. System.out.println(Contador.acumulador);
21. System.out.println(Contador.VALOR_INICIAL);
22.
```

h) ¿Qué sucede si cambiamos la instrucción "this(Contador.VALOR_INICIAL)" por "new Contador(Contador.VALOR_INICIAL)" ?

En este caso no sucede algo diferente a lo que ya teniamos, ya que es una manera de encadenar los constructores.

i) ¿Qué sucede si el primer constructor lo modificamos de la siguiente forma?

```
1. public Contador(int valor) {
2.     this.valor = valor;
3.     Contador.acumulador+= valor;
4.     Contador.VALOR_INICIAL += valor;
5. }
6.
```

En nuestro constructor dará un error debido a que una variable "FINAL" no puede ser modificada, y lo que hacemos en el constructor es tratar de modificar VALOR_INICIAL, aumentando la cantidad del valor que tome VALOR

j) Realice las siguientes modificaciones en la clase Contador:

j.1. Añada una variable de clase nContadores que contenga el número de contadores creados.

```
1. static int contadorTotal = 0;
2.
3. public Contador(int valor) {
4.     this.valor = valor;
5.     Contador.acumulador+= valor;
6.     contadorTotal += 1;
7. }
8.
```

j.2. Añada una variable de clase ultimoContador que almacene el valor inicial del último contador creado.

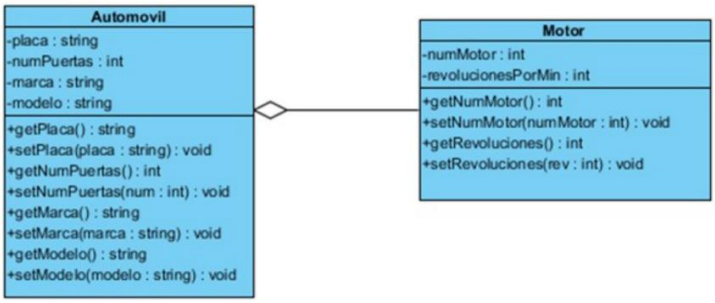
```
1. static int ultimoContador = 0;
2. public Contador(int valor) {
3.     this.valor = valor;
4.     Contador.acumulador+= valor;
5.     contadorTotal += 1;
6.     ultimoContador = valor;
7. }
8.
```

j.3. Agregue las instrucciones que usted crea por conveniente a la clase ContadorTest, las mismas que evidencien el funcionamiento de la clase modificada.

```
1. System.out.println(Contador.acumulador);
2. System.out.println(Contador.VALOR_INICIAL);
3. System.out.println(Contador.contadorTotal);
4. System.out.println(Contador.ultimoContador);
5.
```

3. ACTIVIDAD 3

a. Relación de "agregación": Sea el siguiente diagrama de clases, en donde observamos la relación de agregación entre la clase Automóvil y Motor. En un proyecto nuevo, escriba el código que implemente este diagrama.



Agregue los siguientes método:

i. Métodos getters y setters de ambas clases
ii. El método toString() de ambas clases deben de permitir retornar la información de un automóvil.

```
1. public class Eje3 {
2.     public static void main(String[] args) {
3.         Motor motor1 = new Motor(2000, 10000);
4.         Automovil auto1 = new Automovil("W3EWRT", 2, "AUDI", "A5", motor1);
5.         Automovil auto2 = new Automovil("QRE23F", 2, "AUDI", "A1", new Motor(3000, 12000));
6.         System.out.println(auto1.toString());
7.         System.out.println(auto2.toString());
8.     }
9. }
10.
```

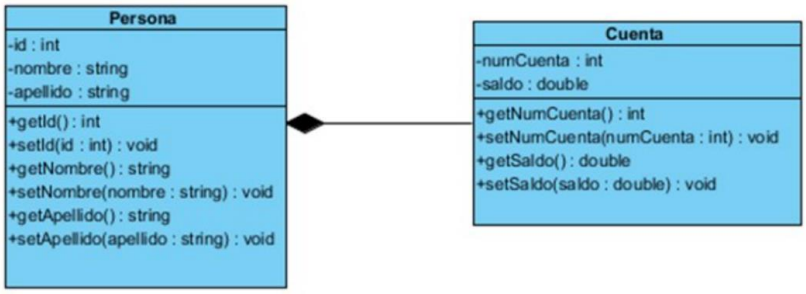
```
1. class Automovil {
2.     private String placa;
3.     private int numPuertas;
4.     private String marca;
5.     private String modelo;
6.     private Motor motor; // Agregamos la relación de agregación
7.
8.     public Automovil(String placa, int numPuertas, String marca, String modelo, Motor motor) {
9.         this.placa = placa;
10.        this.numPuertas = numPuertas;
11.        this.marca = marca;
12.        this.modelo = modelo;
13.        this.motor = motor; // Asignamos el motor en el constructor
14.    }
15.
16.    // GETTERS Y SETTERS
```

```
17.     public String getPlaca() {return placa;}
18.     public void setPlaca(String placa) {
19.         this.placa = placa;
20.     }
21.     public int getNumeroPuertas() {return numPuertas;}
22.     public void setNumeroPuertas(int numeroPuertas) {
23.         this.numPuertas = numPuertas;
24.     }
25.     public String getMarca() {return marca;}
26.     public void setMarca(String marca) {
27.         this.marca = marca;
28.     }
29.     public String getModelo() {return modelo;}
30.     public void setModelo(String modelo) {
31.         this.modelo = modelo;
32.     }
33.     public Motor getMotor() {return motor;}
34.     public void setMotor(Motor motor) {
35.         this.motor = motor;
36.     }
37.
38.     @Override
39.     public String toString() {
40.         return "placa='" + placa +
41.                 "\nnumPuertas=" + numPuertas +
42.                 "\nmarca='" + marca +
43.                 "\nmodelo='" + modelo +
44.                 "\nmotor=" + motor.toString();
45.     }
46. }
47.
```

```
 1. class Motor {
 2.     private int numMotor;
 3.     private int revPorMin;
 4.
 5.     public Motor(int numMotor, int revPorMin) {
 6.         this.numMotor = numMotor;
 7.         this.revPorMin = revPorMin;
 8.     }
 9.     // GETTERS AND SETTERS
10.     public int getNumMotor() {return numMotor;}
11.     public void setNumMotor(int numMotor) {
12.         this.numMotor = numMotor;
13.     }
14.
15.     public int getRevPorMin() {return revPorMin;}
16.     public void setRevPorMin(int revPorMin) {
17.         this.revPorMin = revPorMin;
18.     }
19.
20.     @Override
21.     public String toString() {
22.         return "numMotor=" + numMotor+
23.                 "\nevPorMin=" + revPorMin;
24.     }
25. }
26.
```

```
semestre\1. Lenguaje de Programacion
placa='W3EWRT
numPuertas=2
marca='AUDI
modelo='A5
motor=numMotor=2000
evPorMin=10000
placa='QRE23F
numPuertas=2
marca='AUDI
modelo='A1
motor=numMotor=3000
evPorMin=12000

Process finished with exit code 0
```

4. ACTIVIDAD 4
   Relación de "composición": Sea el siguiente diagrama de clases, en donde observamos la relación de composición entre la clase Persona y Cuenta. En un proyecto nuevo escriba el código que implemente el diagrama

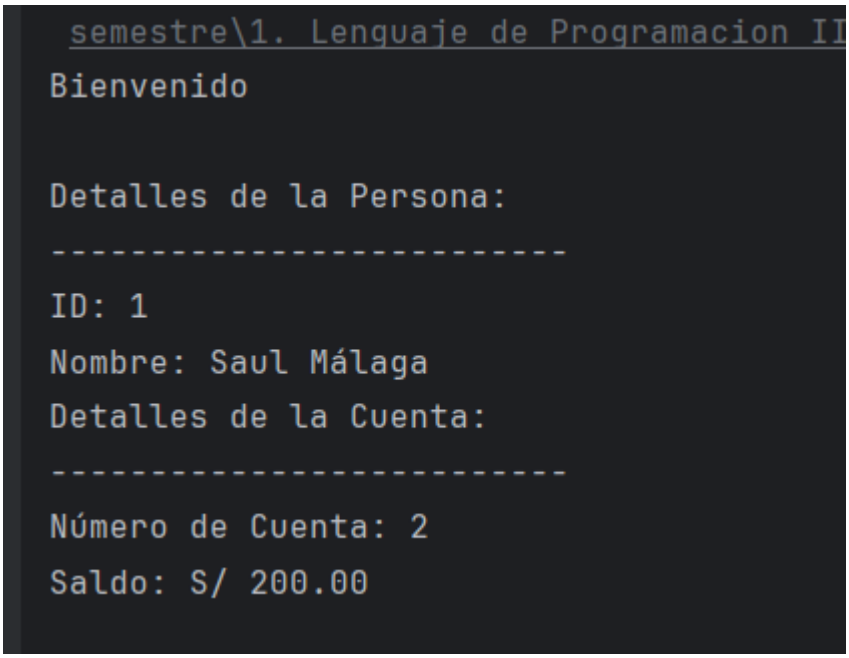



```java
public class Eje4 {
    public static void main(String[] args) {
        System.out.println("Bienvenido\n");
        People persona1 = new People("Saul", "Málaga", 2, 200);
        System.out.println(persona1);
    }
}

```

```java
class People {
    private static int idCounter = 0;
    private int id;
    private String nombre;
    private String apellido;
    private Account cuenta;

    public People(String nombre, String apellido, int numeroCuenta, double saldoInicial) {
        this.id = ++idCounter;
        this.nombre = nombre;
        this.apellido = apellido;
```

```
12.         this.cuenta = new Account(numeroCuenta, saldoInicial); // La cuenta es creada
internamente
13.     }
14.
15.     public int getId() { return id; }
16.
17.     public String getNombre() { return nombre; }
18.     public void setNombre(String nombre) {
19.         this.nombre = nombre;
20.     }
21.
22.     public String getApellido() { return apellido; }
23.     public void setApellido(String apellido) {
24.         this.apellido = apellido;
25.     }
26.
27.     @Override
28.     public String toString() {
29.         return "Detalles de la Persona:\n" +
30.                 "--------------------------\n" +
31.                 "ID: " + id + "\n" +
32.                 "Nombre: " + nombre + " " + apellido + "\n" +
33.                 cuenta.toString() + "\n";
34.     }
35. }
36.
```

```
 1. class Account {
 2.     private int numero;
 3.     private double saldo;
 4.
 5.     public Account(int numero, double saldo) {
 6.         this.numero = numero;
 7.         this.saldo = saldo;
 8.     }
 9.
10.     public int getNumero() { return numero; }
11.     public void setNumero(int numero) {
12.         this.numero = numero;
13.     }
14.
15.     public double getSaldo() { return saldo; }
16.     public void setSaldo(double saldo) {this.saldo = saldo;}
17.
18.     @Override
19.     public String toString() {
20.         return "Detalles de la Cuenta:\n" +
21.                 "--------------------------\n" +
22.                 "Número de Cuenta: " + numero + "\n" +
23.                 "Saldo: S/ " + String.format("%.2f", saldo) + "\n";
24.     }
25. }
26.
```



5. ACTIVIDAD 5
Herencia: Diseñe e implemente un programa que administre un conjunto de cuentas de un banco. Entre las cuales están:

- Cuenta de ahorros que genera intereses. Los intereses se capitalizan mensualmente y se calculan sobre el saldo mensual mínimo.
- Cuenta corriente que no tiene intereses, le brinda tres retiros gratuitos por mes y cobra una tarifa de transacción de S/.3.0 por cada retiro adiciona

El programa inicialmente debe gestionar estos dos tipos de cuentas, sin embargo, más adelante es posible que sean muchos otros tipos mas. Por lo tanto, la solución planteada debe permitir expandir los tipos de cuentas sin afectar el ciclo de procesamiento principal.
Se debe proporcionar menú.

```
 1. public class Eje5 {
 2.     public static void main(String[] args) {
 3.         Cuenta[] cuentas = new Cuenta[10];
 4.         int cuentaIndex = 0;
 5.
 6.         Scanner in = new Scanner(System.in);
 7.         boolean done = false;
 8.
 9.         while (!done) {
10.             List<String> opciones = List.of("Crear cuenta de ahorros", "Crear cuenta corriente",
"Depositar", "Retirar", "Consultar", "Salir");
11.             String titulo = "Menú Principal";
12.             mostrarMenu(opciones, titulo);
13.             System.out.println("Seleccione una opción: ");
14.             int op = in.nextInt();
15.
16.             switch (op) {
17.                 case 1:
18.                     if (cuentaIndex < cuentas.length) {
19.                         System.out.print("Ingrese número de cuenta: ");
20.                         int numero = in.nextInt();
21.                         System.out.print("Ingrese saldo inicial: ");
22.                         double saldo = in.nextDouble();
```

```
                        System.out.print("Ingrese tasa de interés: ");
                        double interes = in.nextDouble();
                        cuentas[cuentaIndex] = new CuentaAhorro(numero, saldo, interes);
                        cuentaIndex++;
                        System.out.println("Cuenta de ahorros creada exitosamente.");
                    } else {
                        System.out.println("No se pueden crear más cuentas.");
                    }
                    break;
                case 2:
                    if (cuentaIndex < cuentas.length) {
                        System.out.print("Ingrese número de cuenta: ");
                        int numero = in.nextInt();
                        System.out.print("Ingrese saldo inicial: ");
                        double saldo = in.nextDouble();
                        cuentas[cuentaIndex] = new CuentaCorriente(numero, saldo);
                        cuentaIndex++;
                        System.out.println("Cuenta corriente creada exitosamente.");
                    } else {
                        System.out.println("No se pueden crear más cuentas.");
                    }
                    break;
                case 3:
                    System.out.print("Ingrese número de cuenta: ");
                    int numeroDepositar = in.nextInt();
                    System.out.print("Ingrese monto a depositar: ");
                    double montoDepositar = in.nextDouble();
                    Cuenta cuentaDeposito = buscarCuenta(cuentas, numeroDepositar);
                    if (cuentaDeposito != null) {
                        cuentaDeposito.depositar(montoDepositar);
                        System.out.println("Depósito realizado con éxito.");
                    } else {
                        System.out.println("Cuenta no encontrada.");
                    }
                    break;
                case 4:
                    System.out.print("Ingrese número de cuenta: ");
                    int numeroRetirar = in.nextInt();
                    System.out.print("Ingrese monto a retirar: ");
                    double montoRetirar = in.nextDouble();
                    Cuenta cuentaRetiro = buscarCuenta(cuentas, numeroRetirar);
                    if (cuentaRetiro != null) {
                        cuentaRetiro.retirar(montoRetirar);
                        System.out.println("Retiro realizado con éxito.");
                    } else {
                        System.out.println("Cuenta no encontrada.");
                    }
                    break;
                case 5:
                    System.out.print("Ingrese número de cuenta: ");
                    int numeroConsultar = in.nextInt();
                    Cuenta cuentaConsulta = buscarCuenta(cuentas, numeroConsultar);
                    if (cuentaConsulta != null) {
                        cuentaConsulta.consultar();
                    } else {
                        System.out.println("Cuenta no encontrada.");
                    }
                    break;
                case 6:
                    done = true;
                    System.out.println("Saliendo del programa...");
                    break;
                default:
                    System.out.println("Opción no válida.");
            }
        }
    }

```

```
    public static Cuenta buscarCuenta(Cuenta[] cuentas, int numero) {
        for (Cuenta cuenta : cuentas) {
            if (cuenta != null && cuenta.getNumero() == numero) {
                return cuenta;
            }
        }
        return null;
    }
    public static void mostrarMenu(List<String> opciones, String titulo) {
        String separador = "+";
        for (int i = 0; i < titulo.length(); i++) {
            separador += "-";
        }
        separador += "+";

        System.out.println(separador);
        System.out.println("| " + titulo + " |");
        System.out.println(separador);

        for (int i = 0; i < opciones.size(); i++) {
            System.out.println("| " + (i + 1) + ". " + opciones.get(i));
        }

        System.out.println(separador);
    }
}
```

```
abstract class Cuenta {
    private int numero;
    private double saldo;

    public Cuenta(int numero, double saldo) {
        this.numero = numero;
        this.saldo = saldo;
    }

    public int getNumero() {
        return numero;
    }

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double monto) {
        if (monto > 0) {
            saldo += monto;
            System.out.println("Depositado: S/ " + monto);
        } else {
            System.out.println("Monto inválido.");
        }
    }

    public void retirar(double monto) {
        if (monto > 0 && saldo >= monto) {
            saldo -= monto;
            System.out.println("Retirado: S/ " + monto);
        } else {
            System.out.println("Fondos insuficientes o monto inválido.");
        }
    }

    public abstract void consultar();
}
```

```
 1. class CuentaAhorro extends Cuenta {
 2.     private double tazaInteres;
 3.     private double minSaldo;
 4.
 5.     public CuentaAhorro(int numero, double saldo, double tazaInteres) {
 6.         super(numero, saldo);
 7.         this.tazaInteres = tazaInteres;
 8.         this.minSaldo = saldo;
 9.     }
10.
11.     public void setTazaIntereses(double intereses) {
12.         this.tazaInteres = intereses;
13.     }
14.
15.     @Override
16.     public void retirar(double amount) {
17.         super.retirar(amount);
18.         double saldo = getSaldo();
19.         if (saldo < minSaldo) {
20.             minSaldo = saldo;
21.         }
22.     }
23.
24.     @Override
25.     public void consultar() {
26.         double interes = minSaldo * tazaInteres / 100;
27.         depositar(interes);
28.         minSaldo = getSaldo();
29.         System.out.println("Intereses aplicados. Nuevo saldo: S/ " + getSaldo());
30.     }
31. }
32.
```

```
 1. class CuentaCorriente extends Cuenta {
 2.     private int retiros = 0;
 3.     private final int LIBRE_RETIROS = 3;
 4.     private final double TARIFA_TRANSACCION = 3.5;
 5.
 6.     public CuentaCorriente(int numero, double saldo) {
 7.         super(numero, saldo);
 8.     }
 9.
10.     @Override
11.     public void retirar(double amount) {
12.         super.retirar(amount);
13.         retiros++;
14.         if (retiros > LIBRE_RETIROS) {
15.             super.retirar(TARIFA_TRANSACCION);
16.             System.out.println("Se ha aplicado una tarifa de S/ " + TARIFA_TRANSACCION + " por
retiro adicional.");
17.         }
18.     }
19.
20.     @Override
21.     public void consultar() {
22.         retiros = 0;
23.         System.out.println("Consultado.");
24.     }
25. }
26.
```

7. EJERCICIOS DE PRÁCTICA

Sistema de Gestión de Cursos Universitarios.

El proyecto consiste en desarrollar un sistema para gestionar cursos universitarios. En este sistema, los estudiantes podrán inscribirse en cursos que están a cargo de profesores. Cada curso pertenece a una categoría específica (como Matemáticas, Programación, etc.). El sistema también debe gestionar la información personal de los estudiantes y profesores, así como la inscripción y la asignación de cursos. Se necesita saber la cantidad de estudiantes matriculados en los diferentes cursos, así como, los cursos que estan disponibles.

Requisitos del Proyecto

1. Clases y Objetos:

- Crear clases que representen entidades como Estudiante, Profesor, Curso, SistemaGestion, entre otras.
- Cada clase debe tener atributos y métodos específicos.
- Identifique cuales de los atributos son de instancia para cada clase y cuales son de clase.
- Identificar la posibilidad de utilizar constantes para valores que no cambian durante la ejecución del programa.
- Identifique las relaciones de agregación y composición entre las clases.
- Instanciar objetos de las clases para representar los elementos del sistema (por ejemplo, crear objetos que representen estudiantes y cursos específicos).

2. Herencia:

- Definir por lo menos una jeraquía de clases, en la cual se pueda definir clases abstractas con métdoso abstractos.

3. Polimorfismo:

- Identificar cuáles de los métodos son polimórfico.

Todos los archivos/ paquetes parten de la carpeta de SRC.

Paquete Person

Interface.java

```
1. package Person;
2.
3. interface InterfacePerson {
4.     void showInfo();
5. }
6.
```

Person.java

```
 1. package Person;
 2.
 3. public class Person {
 4.     protected String Id;
 5.     protected String name;
 6.     protected String lastname;
 7.     protected int age;
 8.     protected String occupation;
```

```
 9.     protected String email;
10.
11.     public Person(String Id, String occupation, String name, String lastname, int age, String
email) {
12.         this.Id = Id;
13.         this.occupation = occupation;
14.         this.name = name;
15.         this.lastname = lastname;
16.         this.age = age;
17.         this.email = email;
18.     }
19.
20.     @Override
21.     public String toString() {
22.         return Id + " " + occupation + " " + name + " " + lastname + " " + age + " " + email;
23.     }
24.
25.     String getOccupation() {
26.         return occupation;
27.     }
28.     String getID() {
29.         return Id;
30.     }
31. }
32.
```

Student.java

```
 1. package Person;
 2. import java.io.Serializable;
 3.
 4. public class Student extends Person implements Serializable, InterfacePerson {
 5.
 6.     private static final long serialVersionUID = 1L;
 7.     public Student(String studentID, String occupation, String name, String lastName, int age,
String email) {
 8.         super(studentID, occupation, name, lastName, age, email);
 9.     }
10.
11.     @Override
12.     public void showInfo() {
13.         System.out.println(super.toString());
14.     }
15.
16.     @Override
17.     public String toString() {
18.         return super.toString();
19.     }
20.
21. }
22.
```

Teacher.java

```
 1. package Person;
 2. import java.io.Serializable;
 3.
 4. public class Teacher extends Person implements Serializable, InterfacePerson {
 5.
 6.     private static final long serialVersionUID = 1L;
 7.     private String courseSubject;
 8.
 9.     public Teacher(String teacherID, String occupation, String name, String lastName, int age,
String email, String courseSubject) {
```

```
10.         super(teacherID, occupation, name, lastName, age, email);
11.         this.courseSubject = courseSubject;
12.     }
13.
14.     @Override
15.     public void showInfo() {
16.         System.out.println(super.toString());
17.     }
18.
19.     @Override
20.     public String toString() {
21.         return super.toString() + " " + courseSubject;
22.     }
23.
24.     String getCourseSubject() {
25.         return courseSubject;
26.     }
27.
28. }
29.
```

Paquete Course

InterfaceCourse.java

```
 1. package Course;
 2.
 3. import Person.Student;
 4. import Person.Teacher;
 5. import java.util.List;
 6.
 7. public interface IntercafeCouse {
 8.
 9.     String getCourseName();
10.     List<Teacher> getTeacher();
11.     boolean addTeacher(Teacher person);
12.     List<Student> getStudent();
13.     boolean addStudent(Student person);
14.
15. }
16.
```

Course.java

```
 1. package Course;
 2.
 3. import Person.Student;
 4. import Person.Teacher;
 5.
 6. import java.util.ArrayList;
 7. import java.util.List;
 8.
 9. public class Course {
10.     private String courseName;
11.     private String courseID;
12.     private List<Student> student;
13.     private List<Teacher> teacher;
14.     static protected int TotalityStudents = 5;
15.     static protected int TotalityTeachers = 1;
16.
17.     public Course(String courseName, String courseID) {
18.         this.courseName = courseName;
19.         this.courseID = courseID;
```

```
20.         this.student = new ArrayList<>();
21.         this.teacher = new ArrayList<>();
22.     }
23.
24.     String getCourseID() {
25.         return courseID;
26.     }
27. }
28.
```

Mathematics.java

```
 1. package Course;
 2.
 3. import Person.Student;
 4. import Person.Teacher;
 5.
 6. import java.util.ArrayList;
 7. import java.util.List;
 8.
 9. public class Mathematics extends Course implements IntercafeCouse {
10.
11.     private String courseName;
12.     static int QuantityStudents = 0;
13.     static int QuantityTeachers = 0;
14.     private List<Student> student;
15.     private List<Teacher> teacher;
16.
17.     public Mathematics(String courseName, String courseID){
18.         super(courseName, courseID);
19.         this.courseName = courseName;
20.         this.student = new ArrayList<>();
21.         this.teacher = new ArrayList<>();
22.     }
23.
24.     public String getCourseName() {
25.         return courseName;
26.     }
27.
28.     public List<Teacher> getTeacher() {
29.         return teacher;
30.     }
31.     public boolean addTeacher(Teacher person) {
32.         if (QuantityStudents == TotalityTeachers) {
33.             System.out.println("Ya no se admiten mas profesores");
34.             QuantityTeachers++;
35.             return false;
36.         }
37.         if (!teacher.contains(person)) {
38.             teacher.add(person);
39.             System.out.println("Registro exitoso");
40.             return true;
41.         }
42.         return false;
43.     }
44.
45.
46.     public List<Student> getStudent() {
47.         return student;
48.     }
49.     public boolean addStudent(Student person) {
50.         if (QuantityStudents == TotalityStudents) {
51.             System.out.println("Los cupos de este curso se han completado");
52.             QuantityStudents++;
53.             return false;
54.         }
```

```
55.         if (!student.contains(person)) {
56.             student.add(person);
57.             System.out.println("Registro exitoso");
58.             return true;
59.         }
60.         return false;
61.     }
62. }
63.
```

ProgrammingLanguage.java

```
 1. package Course;
 2.
 3. import Person.Student;
 4. import Person.Teacher;
 5.
 6. import java.util.ArrayList;
 7. import java.util.List;
 8.
 9. public class ProgrammingLanguage extends Course implements IntercafeCouse {
10.
11.     private String courseName;
12.     static int QuantityStudents = 0;
13.     static int QuantityTeachers = 0;
14.     private List<Student> student;
15.     private List<Teacher> teacher;
16.
17.     public ProgrammingLanguage(String courseName, String courseID){
18.         super(courseName, courseID);
19.         this.courseName = courseName;
20.         this.student = new ArrayList<>();
21.         this.teacher = new ArrayList<>();
22.     }
23.
24.     public String getCourseName() {
25.         return courseName;
26.     }
27.
28.     public List<Teacher> getTeacher() {
29.         return teacher;
30.     }
31.     public boolean addTeacher(Teacher person) {
32.         if (QuantityStudents == TotalityTeachers) {
33.             System.out.println("Ya no se admiten mas profesores");
34.             QuantityTeachers++;
35.             return false;
36.         }
37.         if (!teacher.contains(person)) {
38.             teacher.add(person);
39.             System.out.println("Registro exitoso");
40.             return true;
41.         }
42.         return false;
43.     }
44.
45.
46.     public List<Student> getStudent() {
47.         return student;
48.     }
49.     public boolean addStudent(Student person) {
50.         if (QuantityStudents == TotalityStudents) {
51.             System.out.println("Los cupos de este curso se han completado");
52.             QuantityStudents++;
53.             return false;
54.         }
```

```
55.         if (!student.contains(person)) {
56.             student.add(person);
57.             System.out.println("Registro exitoso");
58.             return true;
59.         }
60.         return false;
61.     }
62.
63. }
64.
```

Cisco.java

```
 1. package Course;
 2.
 3. import Person.Student;
 4. import Person.Teacher;
 5.
 6. import java.util.ArrayList;
 7. import java.util.List;
 8.
 9. public class Cisco extends Course implements IntercafeCouse {
10.
11.     private String courseName;
12.     static int QuantityStudents = 0;
13.     static int QuantityTeachers = 0;
14.     private List<Student> studentCisco;
15.     private List<Teacher> teacherCisco;
16.
17.     public Cisco(String courseName, String courseID){
18.         super(courseName, courseID);
19.         this.courseName = courseName;
20.         this.studentCisco = new ArrayList<>();
21.         this.teacherCisco = new ArrayList<>();
22.     }
23.
24.     public String getCourseName() {
25.         return courseName;
26.     }
27.
28.     public List<Teacher> getTeacher() {
29.         return teacherCisco;
30.     }
31.     public boolean addTeacher(Teacher person) {
32.         if (QuantityStudents == TotalityTeachers) {
33.             System.out.println("Ya no se admiten mas profesores");
34.             QuantityTeachers++;
35.             return false;
36.         }
37.         if (!teacherCisco.contains(person)) {
38.             teacherCisco.add(person);
39.             System.out.println("Registro exitoso");
40.             return true;
41.         }
42.         return false;
43.     }
44.
45.
46.     public List<Student> getStudent() {
47.         return studentCisco;
48.     }
49.     public boolean addStudent(Student person) {
50.         if (QuantityStudents == TotalityStudents) {
51.             System.out.println("Los cupos de este curso se han completado");
52.             QuantityStudents++;
53.             return false;
```

```
54.         }
55.         if (!studentCisco.contains(person)) {
56.             studentCisco.add(person);
57.             System.out.println("Registro exitoso");
58.             return true;
59.         }
60.         return false;
61.     }
62. }
63.
```

Paquete Resource

```
 1. Main.java
 2. package Resource;
 3.
 4. import Course.Cisco;
 5. import Course.Mathematics;
 6. import Course.ProgrammingLanguage;
 7. import Person.Student;
 8. import Person.Teacher;
 9.
10. public class Main {
11.     public static void main(String[] args) {
12.
13.         Cisco ciscoCourse = new Cisco("Cisco Networking", "FA98");
14.         Mathematics mathCourse = new Mathematics("Advanced Mathematics", "MATH101");
15.         ProgrammingLanguage progCourse = new ProgrammingLanguage("Java Programming", "CS102");
16.
17.
18.         Student student1 = new Student("DF34", "Student", "Alice", "Smith", 14,
"alice@gmail.com");
19.         Student student2 = new Student("GH56", "Student", "Bob", "Johnson", 19, "bob@gmail.com");
20.         Student student3 = new Student("JK78", "Student", "Charlie", "Brown", 20,
"charlie@gmail.com");
21.
22.         Teacher teacher1 = new Teacher("AD78", "Teacher", "Jimmy", "Smith", 35,
"smith@gmail.com", "Networking");
23.         Teacher teacher2 = new Teacher("KL89", "Teacher", "Sarah", "Connor", 40,
"connor@gmail.com", "Mathematics");
24.         Teacher teacher3 = new Teacher("MN90", "Teacher", "John", "Doe", 45, "doe@gmail.com",
"Programming");
25.
26.         ciscoCourse.addStudent(student1);
27.         ciscoCourse.addStudent(student2);
28.         mathCourse.addStudent(student2);
29.         mathCourse.addStudent(student3);
30.         progCourse.addStudent(student1);
31.         progCourse.addStudent(student3);
32.
33.         ciscoCourse.addTeacher(teacher1);
34.         mathCourse.addTeacher(teacher2);
35.         progCourse.addTeacher(teacher3);
36.
37.         System.out.println("Información de los Cursos:");
38. //------------------------------------------------------------------------
39.         System.out.println("\nCurso: " + ciscoCourse.getCourseName());
40.         for (Teacher teacher : ciscoCourse.getTeacher()) {
41.             System.out.println("Profesor: " + teacher.toString());
42.         }
43.         for (Student student : ciscoCourse.getStudent()) {
44.             System.out.println("Estudiante: " + student.toString());
45.         }
46. //------------------------------------------------------------------------
47.         System.out.println("\nCurso: " + mathCourse.getCourseName());
48.         for (Teacher teacher : mathCourse.getTeacher()) {
```

```
49.             System.out.println("Profesor: " + teacher.toString());
50.         }
51.         for (Student student : mathCourse.getStudent()) {
52.             System.out.println("Estudiante: " + student.toString());
53.         }
54. //-------------------------------------------------------------------------
55.         System.out.println("\nCurso: " + progCourse.getCourseName());
56.         for (Teacher teacher : progCourse.getTeacher()) {
57.             System.out.println("Profesor: " + teacher.toString());
58.         }
59.         for (Student student : progCourse.getStudent()) {
60.             System.out.println("Estudiante: " + student.toString());
61.         }
62.     }
63. }
64.
```

```
 semestre\1. Lenguaje de Programacion III\1ra fase\Practica\Guia2\Ejercic
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Registro exitoso
Información de los Cursos:

Curso: Cisco Networking
Profesor: AD78 Teacher Jimmy Smith 35 smith@gmail.com Networking
Estudiante: DF34 Student Alice Smith 14 alice@gmail.com
Estudiante: GH56 Student Bob Johnson 19 bob@gmail.com

Curso: Advanced Mathematics
Profesor: KL89 Teacher Sarah Connor 40 connor@gmail.com Mathematics
Estudiante: GH56 Student Bob Johnson 19 bob@gmail.com
Estudiante: JK78 Student Charlie Brown 20 charlie@gmail.com

Curso: Java Programming
Profesor: MN90 Teacher John Doe 45 doe@gmail.com Programming
Estudiante: DF34 Student Alice Smith 14 alice@gmail.com
Estudiante: JK78 Student Charlie Brown 20 charlie@gmail.com

Process finished with exit code 0
```

8. CONCLUSIONES DE LA PRÁCTICA:

a) Las clases son fundamentales en la POO, ya que definen las propiedades y comportamientos que los objetos pueden tener. Los objetos, que son instancias de clases, permiten la manipulación directa de datos y comportamientos específicos dentro de un programa.

b) Es crucial distinguir entre variables y métodos de instancia y de clase. Las variables de instancia pertenecen a objetos específicos, mientras que las variables de clase son compartidas por todas las instancias de una clase.

c) Las relaciones permiten la construcción de sistemas más complejos y modulares, donde la agregación y la composición definen cómo las clases trabajan juntas, y la herencia facilita la reutilización del código.

d) El polimorfismo es un concepto clave que permite a los métodos comportarse de diferentes maneras según el contexto, mejorando la flexibilidad y la escalabilidad del código.

e) Se enfatiza la importancia de una buena documentación y el seguimiento de las mejores prácticas de programación, como el uso de nombres descriptivos, una adecuada indentación, y la modularidad en el código, para asegurar la claridad y mantenibilidad del software.

9. CUESTIONARIO

1. ¿Qué es el polimorfismo en Java y cómo se relaciona con la herencia? Investiga cómo el polimorfismo mejora la flexibilidad del código y proporciona un ejemplo propio que no haya sido discutido en clase.

Según [5] una de las características del polimorfismo es la posibilidad de enviar mensajes sintácticamente iguales a objetos de distintas clases, la misma operación se realiza de distinta forma según sea el objeto al que se le envía el mensaje.
Se relaciona con la herencia basándose en sus clases derivadas, implementar métodos en la clase base, pero con sus propios comportamientos. De esta manera con el polimorfismo se mejora la flexibilidad del código al permitir comportamientos propios.

Ejemplo:

```
 1. public class test {
 2.     public static void main(String[] args) {
 3.         // herencia y polimorfismo figuras
 4.         Figura[] figuras = new Figura[3];
 5.         figuras[0] = new Circulo("rojo", true, 5);
 6.         figuras[1] = new Rectangulo("azul", false, 3, 4);
 7.         figuras[2] = new Circulo("verde", true, 10);
 8.
 9.         for (Figura figura : figuras) {
10.             System.out.println(figura);
11.             System.out.println("Area: " + figura.area());
12.             System.out.println("Perimetro: " + figura.perimetro());
13.         }
14.
15.     }
16. }
17.
18. class Figura {
19.     private String color;
20.     private boolean relleno;
21.
22.     public Figura(String color, boolean relleno) {
23.         this.color = color;
24.         this.relleno = relleno;
25.     }
26.
27.     public String getColor() {return color;}
28.     public void setColor(String color) {
29.         this.color = color;
30.     }
31.
32.     public boolean isRelleno() {return relleno;}
33.     public void setRelleno(boolean relleno) {
34.         this.relleno = relleno;
```

```
    }

    public double area() {
        return 0;
    }

    public double perimetro() {
        return 0;
    }

    @Override
    public String toString() {
        return "Figura{" +
                "color='" + color + '\'' +
                ", relleno=" + relleno +
                '}';
    }
}

class Circulo extends Figura {
    private double radio;

    public Circulo(String color, boolean relleno, double radio) {
        super(color, relleno);
        this.radio = radio;
    }

    public double getRadio() {return radio;}
    public void setRadio(double radio) {
        this.radio = radio;
    }

    @Override
    public double area() {
        return Math.PI * Math.pow(radio, 2);
    }

    @Override
    public double perimetro() {
        return 2 * Math.PI * radio;
    }

    @Override
    public String toString() {
        return "radio=" + radio +
                ", " + super.toString();
    }
}

class Rectangulo extends Figura {
    private double ancho;
    private double largo;

    public Rectangulo(String color, boolean relleno, double ancho, double largo) {
        super(color, relleno);
        this.ancho = ancho;
        this.largo = largo;
    }

    public double getAncho() {return ancho;}
    public void setAncho(double ancho) {
        this.ancho = ancho;
    }

    public double getLargo() {return largo;}
    public void setLargo(double largo) {
        this.largo = largo;
    }
```

```
103.
104.     @Override
105.     public double area() {
106.         return ancho * largo;
107.     }
108.
109.     @Override
110.     public double perimetro() {
111.         return 2 * (ancho + largo);
112.     }
113.
114.     @Override
115.     public String toString() {
116.         return "ancho=" + ancho +
117.                 ", largo=" + largo +
118.                 ", " + super.toString();
119.     }
120. }
121.
```

- Compilando

2. ¿Cuáles son las diferencias clave entre una clase abstracta y una interfaz en Java? Investiga y menciona una situación en la que sería más apropiado usar una clase abstracta en lugar de una interfaz, y viceversa.

- Una clase abstracta puede heredar o extender cualquier clase (independientemente de que esta sea abstracta o no), mientras que una interfaz solamente puede extender o implementar otras interfaces.
- Una clase abstracta puede heredar de una sola clase (abstracta o no) mientras que una interfaz puede extender varias interfaces de una misma vez.
- En una clase abstracta pueden existir variables static, final o static final con cualquier modificador de acceso (public, private, protected o default). En una interfaz sólo puedes tener constantes (public static final).
- Enlace adicional

- Las interfaces se les conoce como un contrato. Esto porque te obligan a la implementación de sus métodos lo que le asegura a toda lógica de negocio que todo objeto que implemente de ella, tendrá el acceso a los métodos definidos en la misma.
- Mientras que las clases abstractas son más utilizadas para objetos de tipo base. Es como la principal en la jerarquía en un conjunto de objetos que comparten el mismo padre.

Cuando usar:

- Cuando tenemos clases que deben implementar un comportamiento idéntico, el método que implementa en el comportamiento se puede llevar a una clase abstracta.
  Si una clase va a tener un mismo método, pero con distintas implementaciones, estos métodos se pueden definir en una interfaz.

3. Explica las diferencias entre composición y agregación en el diseño orientado a objetos. Investiga y proporciona un ejemplo real (fuera del contexto del sistema de gestión de cursos) en el que se utilizaría cada uno.

- Agregación: Un objeto solo puede estar contenido en otro, pero su existencia no depende del objeto contenedor. si la referencia del objeto que contiene al otro se destruye, no se destruirá el objeto contenido.
- Composición: Un objeto solo puede estar contenido en otro, su existencia esta ligada directamente al objeto contenedor, si se destruye la referencia del objeto contenedor ya no hay razón para que exista la referencia del objeto contenido, también se eliminara según esta relación.

- **Ejemplo:**

```
 1. public class Cuestionario03 {
 2.     public static void main(String[] args) {
 3.         procesador procesador1 = new procesador("Intel", 4, 8);
 4.         computador computador1 = new computador("HP", "Pavilion", 8, 500, procesador1);
 5.         System.out.println(computador1.toString());
 6.     }
 7. }
 8.
 9. class computador {
10.     private String marca;
11.     private String modelo;
12.     private int ram;
13.     private int disco;
14.     private procesador procesador;
15.
16.     public computador(String marca, String modelo, int ram, int disco, procesador
procesador) {
17.         this.marca = marca;
18.         this.modelo = modelo;
19.         this.ram = ram;
20.         this.disco = disco;
21.         this.procesador = procesador;
22.     }
23.
24.     public String getMarca() {return marca;}
25.     public void setMarca(String marca) {
26.         this.marca = marca;
27.     }
28.
29.     public String getModelo() {return modelo;}
30.     public void setModelo(String modelo) {
31.         this.modelo = modelo;
32.     }
33.
34.     public int getRam() {return ram;}
35.     public void setRam(int ram) {
36.         this.ram = ram;
37.     }
38.
39.     public int getDisco() {return disco;}
40.     public void setDisco(int disco) {
41.         this.disco = disco;
42.     }
43.
44.     public procesador getProcesador() {return procesador;}
45.     public void setProcesador(procesador procesador) {
46.         this.procesador = procesador;
47.     }
```

```
48.
49.     @Override
50.     public String toString() {
51.         return "computador{" +
52.                 "marca='" + marca + '\'' +
53.                 ", modelo='" + modelo + '\'' +
54.                 ", ram=" + ram +
55.                 ", disco=" + disco +
56.                 ", procesador=" + procesador +
57.                 '}';
58.     }
59. }
60.
61. class procesador {
62.     private String marca;
63.     private int nucleos;
64.     private int hilos;
65.
66.     public procesador(String marca, int nucleos, int hilos) {
67.         this.marca = marca;
68.         this.nucleos = nucleos;
69.         this.hilos = hilos;
70.     }
71.
72.     public String getMarca() {return marca;}
73.     public void setMarca(String marca) {
74.         this.marca = marca;
75.     }
76.
77.     public int getNucleos() {return nucleos;}
78.     public void setNucleos(int nucleos) {
79.         this.nucleos = nucleos;
80.     }
81.
82.     public int getHilos() {return hilos;}
83.     public void setHilos(int hilos) {
84.         this.hilos = hilos;
85.     }
86.
87.     @Override
88.     public String toString() {
89.         return "\nprocesador{" +
90.                 "marca='" + marca + '\'' +
91.                 ", nucleos=" + nucleos +
92.                 ", hilos=" + hilos +
93.                 '}';
94.     }
95. }
96.
```

- **Compilando**

```
Run    Cuestionario03 ×

"C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" "-javaagent:C:\Program Files\J
computador{marca='HP', modelo='Pavilion', ram=8, disco=500, procesador=
procesador{marca='Intel', nucleos=4, hilos=8}}

Process finished with exit code 0
```

4. **¿Por qué es recomendable utilizar constantes en las aplicaciones Java? Investiga las mejores prácticas para definir y utilizar constantes, y explica cómo el uso correcto de constantes puede mejorar la mantenibilidad del código**

Una constante es un valor que se define una vez y no puede ser modificado posteriormente. Las constantes son útiles cuando necesitas asegurar que ciertos valores permanezcan iguales en todo tu programa, lo que evita errores accidentales y mejora la legibilidad del código [6].

- Evita errores, nos aseguramos de que los valores críticos no se convienen accidentalmente en el código.
- Mejora la legibilidad, usualmente al definir valores constantes, solemos colocarles nombres descriptivos o que den a entender la utilidad de esa variable
- Facilita el mantenimiento, ya que nos permite modificar la constante en un lugar en lugar de buscar y reemplazar cada instancia.

Las constantes son una herramienta poderosa en Java que te permiten definir valores inmutables y utilizarlos de manera segura y eficiente en tu código. No solo ayudan a prevenir errores, sino que también mejoran la legibilidad y mantenibilidad de tus programas.

```
1. public class Configuracion {
2.     public static final int MAX_USUARIOS = 100;
3.     public static final String URL_BASE = "https://miaplicacion.com/api/";
4.     public static final int TIEMPO_ESPERA = 5000;
5.     public static final double PI = 3.14159;
6. }
7.
```

5. **¿Cuál es la diferencia entre un método de clase (static) y un método de instancia en Java? Investiga y proporciona un ejemplo de cuándo sería más útil utilizar un método de clase frente a un método de instancia en una aplicación del mundo real.**

En Java, la diferencia principal entre un método de clase (también conocido como método estático) y un método de instancia radica en cómo se asocian con la clase y sus objetos:

1. **Método de Clase (static):**

Asociación: Un método de clase pertenece a la clase en sí misma, no a una instancia particular de la clase.

Acceso: Puede ser llamado directamente usando el nombre de la clase sin crear una instancia de ella.

Contexto: Dentro de un método estático, no se puede acceder a variables de instancia (no estáticas) ni a métodos de instancia directamente, ya que no se asocian con ningún objeto.

Uso: Los métodos estáticos suelen utilizarse para operaciones que no dependen de los datos de una instancia específica. Por ejemplo, métodos utilitarios, como Math.pow(), o cuando se desea compartir datos comunes entre todas las instancias de una clase.

```
 1. public class BankAccount {
 2.     private double balance;
 3.     private static int numberOfAccounts = 0;
 4.
 5.     public BankAccount(double initialBalance) {
 6.         this.balance = initialBalance;
 7.         numberOfAccounts++;
 8.     }
 9.
10.     public static int getNumberOfAccounts() {
11.         return numberOfAccounts;
12.     }
13. }
14.
```

2. **Método de Instancia:**
   Asociación: Un método de instancia está asociado con una instancia específica de la clase.
   Acceso: Requiere la creación de un objeto (instancia) de la clase para ser llamado.
   Contexto: Puede acceder tanto a variables de instancia como a métodos estáticos de la clase.
   Uso: Se utiliza cuando el método necesita operar sobre los datos de una instancia particular de la clase.

```
 1. public class BankAccount {
 2.     private double balance;
 3.
 4.     public BankAccount(double initialBalance) {
 5.         this.balance = initialBalance;
 6.     }
 7.
 8.     public void deposit(double amount) {
 9.         balance += amount;
10.     }
11.
12.     public void withdraw(double amount) {
13.         if (balance >= amount) {
14.             balance -= amount;
15.         } else {
16.             System.out.println("Insufficient funds");
17.         }
18.     }
19.
20.     public double getBalance() {
21.         return balance;
22.     }
23. }
24.
```

10. Anexos

Practica

11. BIBLIOGRAFÍA

1. [1] R. M. F. L. E. R. M. M. D. F. Silva, «"A Study on the Performance of Java Virtual Machine GarbageCollectors,» 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), vol. 1, nº 35, pp. 23-31, 2021.
2. [2] C. Hortsmann, Big Java, San José: Pearson, 2009.
3. [3] Oracle, «The Java™ Tutorials,» Oracle, 1994. [En línea]. Available: https://docs.oracle.com/javase/tutorial/index.html. [Último acceso: 28 07 2024].
4. [4] Deitel, H. M., & Deitel, P. J. (2003). Cómo programar en Java. Pearson educación
5. [5] González, A. H., & Gallo, S. L. (2021). Herencia y polimorfismo.
6. [6] “Constantes en Java: Cómo Definir y Utilizar Valores Inmutables | Blog Coders Free,” Coders Free. https://codersfree.com/posts/constantes-en-java