



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS



PROGRAMACIÓN ORIENTADA A OBJETOS

PROFESOR:

Ing. Yadira Franco R

PERÍODO ACADÉMICO:

2025-A

TAREA SEMANA 3

TÍTULO:

Métodos, Validaciones y Encapsulamiento



Estudiante

Balseca Odaliz

Ayol Nayely

Panchi Andrés

Zapata Felipe

2025-A

TAREA 1

Objetivo de la Tarea:

Aprender y Aplicar los Conceptos de Métodos, Validaciones y Encapsulamiento

- **Comprender los Métodos:**
Es importante entender cómo declarar y usar métodos con retorno y sin retorno, y cuándo es apropiado usar cada tipo de método según las necesidades del programa. Familiarizarse con los tipos de retorno como int, double, String, boolean y void, y comprender cómo cada uno cumple un propósito específico en la aplicación.
- **Aplicar Validaciones:**
Las validaciones aseguran que las operaciones realizadas sobre los datos sean correctas, evitando errores y comportamientos inesperados. Implementar validaciones dentro de los métodos para garantizar que las entradas sean válidas y que las operaciones (como depósitos, configuraciones o puntuaciones) solo se realicen cuando sean lógicas y apropiadas.
- **Encapsulamiento:**
El encapsulamiento permite proteger los datos internos de una clase y controlarlos mediante métodos públicos que validan y gestionan las operaciones de manera segura. Encapsular los atributos de una clase (como saldo, velocidad o puntuación) para evitar manipulaciones directas y permitir su modificación solo a través de métodos controlados y validados.

1. Comprender los Métodos Controlados y Tipos de Salida

Objetivo: Comprender los tipos de métodos (con retorno y sin retorno) y cómo declarar y utilizar métodos adecuadamente, tanto con valores de retorno como sin ellos.

Tipos de Métodos y Tipos de Retorno:

1. **Métodos con retorno:**
Son métodos que devuelven un valor de un tipo específico. Estos métodos pueden devolver valores de tipos primitivos como int, double, String, boolean, entre otros.
Ejemplo: Un método que reciba el precio de un producto y devuelva el monto del descuento.

2. Métodos sin retorno (void):

Son métodos que no devuelven ningún valor. Se utilizan cuando se quiere realizar una acción, como imprimir un mensaje o modificar un atributo, sin la necesidad de devolver algo.

Ejemplo: Un método que imprima un mensaje de confirmación en la consola.

Tarea Métodos y su Retorno:

1. Método con Retorno de int

Enunciado:

Crea un método llamado `calcularDescuento` que reciba el precio original de un producto y el porcentaje de descuento. El método debe devolver el monto del descuento en int.

Acciones a realizar:

- El método debe recibir el precio y el porcentaje de descuento.
- El programa debe mostrar el monto del descuento en la consola.

CÓDIGO:

```
public class DescuentoProducto {

    public static int calcularDescuento(double precio, double porcentaje) {
        double descuento = (precio * porcentaje) / 100;
        return (int) descuento;
    }
}

public class Main { // clase obligatoria

    public static void main(String[] args) {

        double precioOriginal = 150.75;
        double porcentajeDescuento = 20.0;

        int descuento = DescuentoProducto.calcularDescuento(precioOriginal, porcentajeDescuento);
        System.out.println("El precio es: $" + precioOriginal);
        System.out.println("El porcentaje de descuento es: $" + porcentajeDescuento);
        System.out.println("El descuento es: " + descuento);

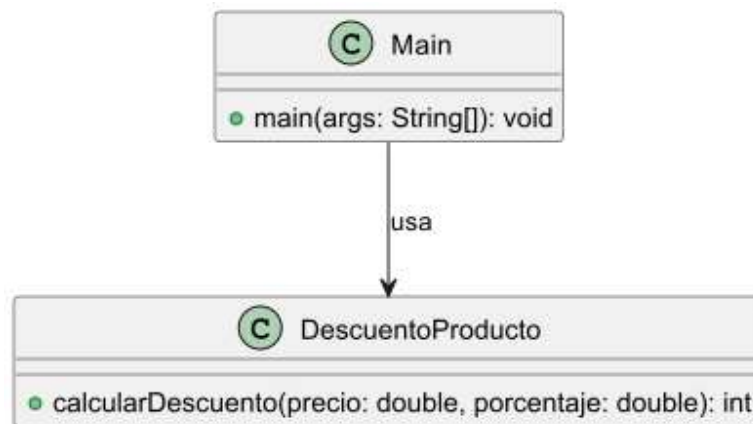
    }
}
```

EJECUCIÓN:

```
"C:\Users\Odaliz Aracely\.jdk\openjd
El precio es: $150.75
El porcentaje de descuento es: $20.0
El descuento es: 30

Process finished with exit code 0
```

DIAGRAMA UML:



2. Método con Retorno de double

Enunciado:

Crea un método llamado `calcularIVA` que reciba el precio de un producto y calcule el IVA (impuesto del 21%). El método debe devolver el valor del IVA en `double`.

Acciones a realizar:

- El método debe devolver el IVA calculado como un valor decimal (`double`).
- El programa debe mostrar el valor del IVA en la consola.

CÓDIGO:

```
public class CalcularIva {
    //Método con retorno double
    public static double calcularIva(double precio) {
        return precio * 0.21;
    }
}

public class Main {
    public static void main(String[] args) {
        double precioProducto = 200.0; // precio del producto

        //Llamada método desde la clase CalcularIva
        double iva = CalcularIva.calcularIva(precioProducto);

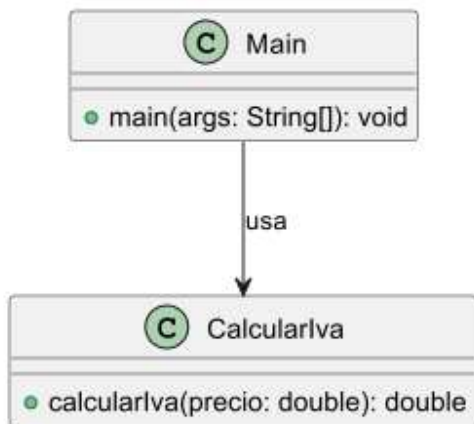
        System.out.println("El IVA del producto es: $" + iva);
    }
}
```

EJECUCIÓN:

```
"C:\Users\Odaliz Aracely\.jdk\open
El IVA del producto es: $42.0

Process finished with exit code 0
```

DIAGRAMA UML:



3. Método con Retorno de String

Enunciado:

Crea un método llamado saludarUsuario que reciba el nombre de un usuario y devuelva un mensaje de saludo personalizado.

Acciones a realizar:

- El método debe recibir el nombre y devolver un saludo como un String.
- El programa debe mostrar el saludo personalizado en la consola.

Main_Usuario	Clase Usuario
<pre>import java.util.Scanner; public class Main_Usuario { public static void main(String[] args) { //Se crea un objeto scanner para leer los datos ingresados por teclado Scanner sc=new Scanner(System.in); //Se crea un objeto de la clase Usuario Usuario saludo1=new Usuario(); System.out.println("----- INICIO-----"); System.out.println("Ingrese su nombre: "); String nombre=sc.nextLine();//Se lee el nombre del usuario por teclado System.out.println("\n-----</pre>	<pre>public class Usuario{ //Se crea un metodo con retorno public String saludarUsuario(String nombre){ //Devuelve un saludo return ("Hola "+nombre+", ¡Bienvenid@ al sistema!"); } }</pre> <p>SALIDA</p>

```

SALUDO-----\n");

        //Se crea una nueva variable
y se guarda la llamada al metodo
saludarUsuario()
        String
saludo=saludo1.saludarUsuario(nombre)
;

        //Se imprime la variable para
que se muestre el contenido en
consola
        System.out.println(saludo);
    }
}

```

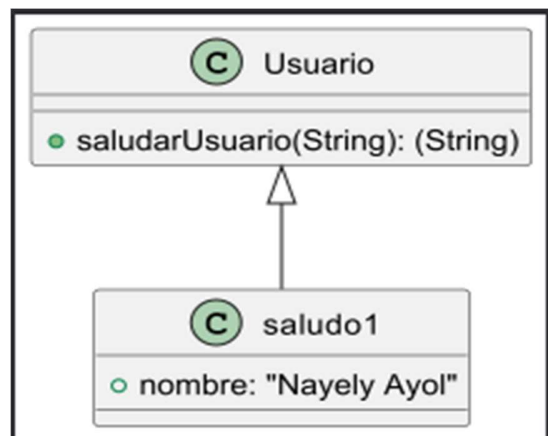
```

-----INICIO-----
Ingrese su nombre:
Nayely Ayol

-----SALUDO-----

Hola Nayely Ayol, ¡Bienvenid@ al sistema!

```



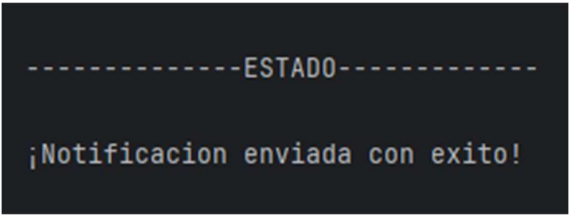
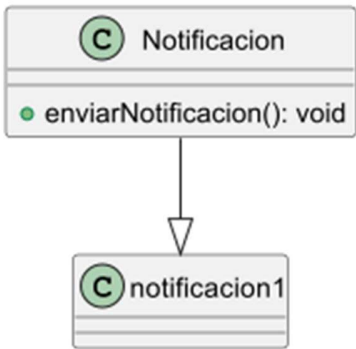
4. Método sin Retorno

(void) Enunciado:

Creará un método llamado `enviarNotificacion` que no devuelva nada. Este método debe imprimir en la consola un mensaje de notificación: "¡Notificación enviada con éxito!" cada vez que sea llamado.

Acciones a realizar:

- El método debe ser `void` y no debe retornar ningún valor.
- El programa debe imprimir el mensaje "¡Notificación enviada con éxito!" al ejecutar el método.

Main_Notificacion	Clase Notificacion
<pre>public class Main_Notificacion { public static void main(String[] args) { //Se crea un objeto de la clase Notificacion Notificacion notificacion1= new Notificacion(); System.out.println("\n----- -----ESTADO-----\n"); //Se llama al metodo notificacion1.enviarNotificacion(); } }</pre>	<pre>public class Notificacion { //Se crea un metodo sin retorno public void enviarNotificacion(){ System.out.println("¡Notificacion enviada con exito!"); } }</pre>
SALIDA	UML
	

5. Método con Retorno de boolean

Enunciado:

Crea un método llamado verificarEdad que reciba la edad de una persona y determine si es mayor de edad (18 años o más). El método debe devolver true si la persona es mayor de edad y false si no lo es.

Acciones a realizar:

- El método debe devolver un valor boolean (true o false).
- El programa debe mostrar si la persona es mayor de edad o no, según el resultado del método.

CODIGO:

```
public class Main {
    public static void main(String[] args) {
        //Crear el objeto persona
        Persona persona1 = new Persona(19);

        //Llamo al metodo

        if (persona1.verificarEdad()) {
            System.out.println("La persona es mayor de edad.");
        } else {
            System.out.println("La persona es menor de edad.");
        }
    }
}
```

```
public class Persona {
    //Atributos
    public int edad;

    //Constructor

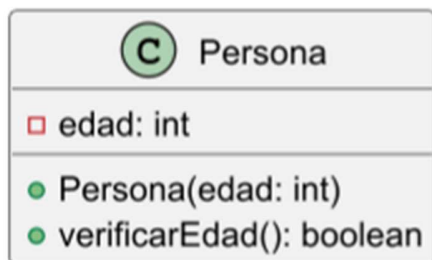
    public Persona(int edad) {
        this.edad = edad;
    }

    //Método
    public boolean verificarEdad() {
        return edad >= 18; //Retorna True o False dependiendo de la edad
    }
}
```


EJECUCION:

```
La persona es mayor de edad.  
  
Process finished with exit code 0
```

DIAGRAMA UML:



6. Método con Retorno de char

Enunciado:

Crea un método llamado `categoriaDeCalificacion` que reciba una calificación numérica y devuelva la categoría correspondiente como una letra (A, B, C, D, F).

Acciones a realizar:

- El método debe recibir una calificación numérica y devolver una letra (char).
- El programa debe mostrar la categoría de calificación en la consola.

CODIGO:

```
public class Main {  
    public static void main(String[] args) {  
        // Crear objeto  
        Calificacion calificación1 = new Calificacion(70.55);  
  
        // Obtener categoría  
        char categoria = calificación1.categoriaDeCalificacion();  
  
        // Mostrar resultado  
        System.out.println("Su calificación es " + calificación1.calificacion  
+" y corresponde a la categoría: " + categoria);  
    }  
}
```

```

public class Calificacion {
    //Atributos
    public double calificacion;

    //Constructor
    public Calificacion(double calificacion){
        this.calificacion=calificacion;
    }

    //Método
    public char categoriaDeCalificacion() {
        if (calificacion >= 90) {
            return 'A';
        } else if (calificacion >= 80) {
            return 'B';
        } else if (calificacion >= 70) {
            return 'C';
        } else if (calificacion >= 60) {
            return 'D';
        } else {
            return 'F';
        }
    } //Devuelve un char
}

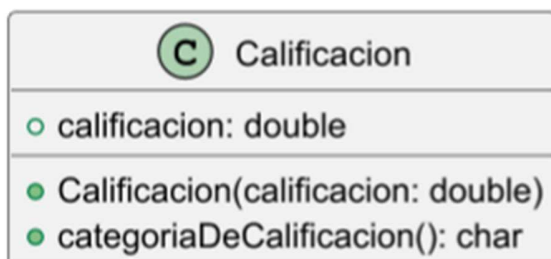
```

EJECUCION:

Su calificación es 70.55 y corresponde a la categoría: C

Process finished with exit code 0

DIAGRAMA UML:



7. Método con Retorno de float

Enunciado:

Crea un método llamado `calcularPrecioConDescuento` que reciba el precio original de un producto y un descuento en porcentaje. El método debe devolver el precio final después de aplicar el descuento, utilizando el tipo de dato `float`.

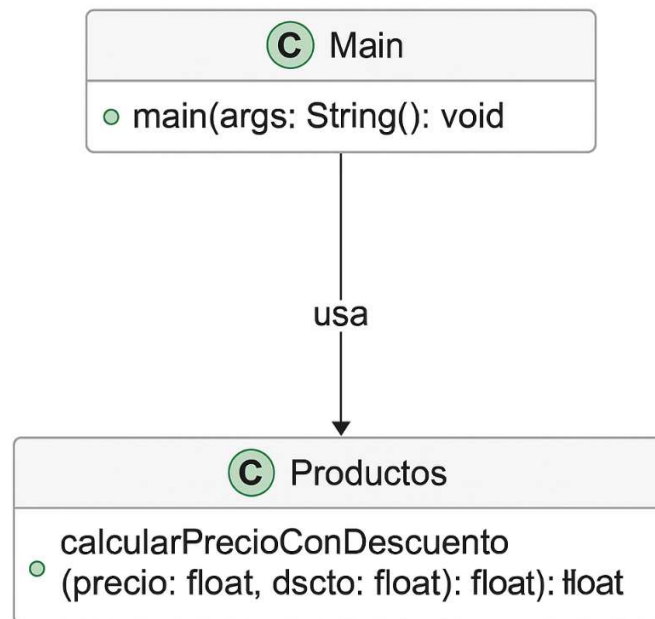
Acciones a realizar:

- El método debe recibir el precio original y el descuento, y devolver el precio final calculado como un valor de tipo `float`.
- El programa debe mostrar el precio final después de aplicar el descuento.

Código
<pre>import java.text.DecimalFormat; import java.util.Scanner; class Productos { float precio; float dscto; float total; public float calcularPrecioConDescuento(float precio, float dscto) { this.precio = precio; this.dscto = dscto; System.out.println("Tiene un descuento del " + dscto + "%" + " equivalente a " + precio + ""); dscto /= 100; this.total = precio * (1 - dscto); return this.total; } } public class Main { public static void main(String[] args) { Scanner teclado = new Scanner(System.in); DecimalFormat df = new DecimalFormat("#.##"); System.out.println("Ingresa el precio del producto (USD): "); float precio = teclado.nextFloat(); teclado.nextLine(); System.out.println("Ingresa el descuento del producto (n%): "); float dscto = teclado.nextFloat(); teclado.nextLine(); Productos Producto1 = new Productos(); Producto1.calcularPrecioConDescuento(precio, dscto); System.out.println("El valor total del producto es: " + df.format(Producto1.calcularPrecioConDescuento(precio, dscto))); teclado.close(); } }</pre>
Salida

```
Ingrese el precio del producto (USD):  
77  
Ingrese el descuento del producto ( n% ):  
7  
Tiene un descuento del 7.0%  
El valor total del producto es: 71,61
```

UML



2. Validaciones en los Métodos

Objetivo: Comprender la importancia de las validaciones en los métodos, para garantizar que solo se realicen operaciones correctas.

Validaciones a Implementar - EJERCICIOS:

1. Bancos - Control de Cuentas

Validación de Depósito: Verificar que el monto a depositar sea positivo. Si es negativo o cero, no debe permitirse el depósito.

Validación de Retiro: Verificar que el monto a retirar sea positivo y no exceda el saldo disponible en la cuenta.

CÓDIGO:

```
public class CuentaBancaria {
    private double saldo; // solo esta clase puede ver o cambiar el dinero

    // Constructor: se llama automáticamente al crear la cuenta
    public CuentaBancaria() {
        this.saldo = 0.0; // el saldo empieza en 0
    }

    // Método para depositar dinero
    public void depositar(double monto) {
        if (monto > 0) {
            saldo += monto; // suma al saldo
            System.out.println("Depósito exitoso. Nuevo saldo: $" + saldo);
        } else {
            System.out.println("No se puede depositar un monto menor o igual a cero.");
        }
    }

    // Método para retirar dinero
    public void retirar(double monto) {
        if (monto <= 0) {
            System.out.println("El monto a retirar debe ser mayor que cero.");
        } else if (monto > saldo) {
            System.out.println("No puedes retirar más de lo que tienes. Saldo: $" + saldo);
        } else {
            saldo -= monto; // resta al saldo
            System.out.println("Retiro exitoso. Nuevo saldo: $" + saldo);
        }
    }

    // Método para ver el saldo actual
    public double obtenerSaldo() {
        return saldo;
    }
}
```

```

public class Main {
    public static void main(String[] args) {
        // Creamos una cuenta
        CuentaBancaria miCuenta = new CuentaBancaria();

        // Probamos depositar dinero
        miCuenta.depositar(100);
        miCuenta.depositar(-50);

        // Probamos retirar dinero
        miCuenta.retirar(30);
        miCuenta.retirar(100);
        miCuenta.retirar(-10);

        // Vemos cuánto queda
        System.out.println("Saldo final: $" + miCuenta.obtenerSaldo());
    }
}

```

EJECUCIÓN:

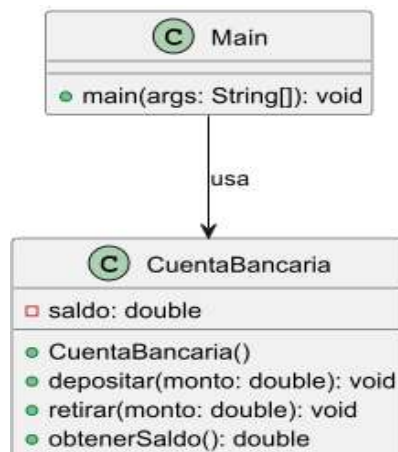
```

"C:\Users\Odaliz Aracely\.jdk\openjdk-24.0.1\bin\java.exe"
Depósito exitoso. Nuevo saldo: $100.0
No se puede depositar un monto menor o igual a cero.
Retiro exitoso. Nuevo saldo: $70.0
No puedes retirar más de lo que tienes. Saldo: $70.0
El monto a retirar debe ser mayor que cero.
Saldo final: $70.0

Process finished with exit code 0

```

DIAGRAMA UML:



2. Automóviles - Control de Velocidad

Validación de Aceleración: Verificar que la velocidad no supere el límite máximo establecido (por ejemplo, 200 km/h).

Validación de Frenado: Verificar que la velocidad no sea negativa después de frenar.

Main_Auto	Clase Auto
<pre> import java.util.Scanner; public class Main_Auto { public static void main(String[] args) { //Se crea un objeto scanner para leer la informacion ingresada por el usuario Scanner sc=new Scanner(System.in); //Se solicitan los datos System.out.println("----- ----ESTADO DEL AUTO-----"); System.out.println("Ingrese los datos\n"); System.out.println("Velocidad Maxima: "); double velocidadMaxima=sc.nextDouble();//Se lee la velocidad maxima System.out.println("Velocidad (Actual): "); double velocidad=sc.nextDouble();//Se lee la velocidad System.out.println("Velocidad (después del frenado): "); double velocidadFrenado=sc.nextDouble();//Se lee la velocidad del frenado //Se crea un objeto Auto Auto auto1=new Auto(velocidadMaxima, velocidad, velocidadFrenado); //Se llama a los metodos para mostrar los resultados System.out.println("----- -----RESULTADOS-----"); auto1.acelerar(); auto1.frenar(); } } </pre>	<pre> public class Auto{ //Se inicializan los atributos public double velocidadMaxima, velocidad, velocidadFrenado; //Se crea un constructor para inicializar los atributos public Auto(double velocidadMaxima, double velocidad, double velocidadFrenado){ this.velocidadMaxima=velocidadMaxima; this.velocidad=velocidad; this.velocidadFrenado=velocidadFrenad o; } //Se crea un metodo acelerar para controlar que la velocidad supere el limite permitido public void acelerar(){ System.out.println("\nVelocidad actual: "+velocidad+" km/h"); if(velocidad>velocidadMaxima){ System.out.println("¡ALERTA!\n Reduzca la velocidad. Velocidad permitida: "+velocidadMaxima+" km/h"); } else{ System.out.println("La velocidad es segura, ¡SIGA ASI!"); } } //Se crea el metodo frenar para verificar que la velocidad no sea menor que cero public void frenar(){ System.out.println("\nVelocidad después del frenado: "+velocidadFrenado+" km/h"); if(velocidadFrenado<0){ System.out.println("¡ALERTA!\n La velocidad después del frenado no puede ser negativa"); </pre>

	<pre> }else if (velocidadFrenado==0) { System.out.println("El auto se ha detenido"); } else{ System.out.println("Ha disminuido la velocidad"); } } } </pre>
SALIDA	UML
<pre> C:\Users\ric\Documents\openjdk-24\bin\java.exe -----ESTADO DEL AUTO----- Ingrese los datos Velocidad Maxima: 200 Velocidad (Actual): 50 Velocidad (después del frenado): 30 -----RESULTADOS----- Velocidad actual: 50.0 km/h La velocidad es segura, ¡SIGA ASI! Velocidad despues del frenado: 30.0 km/h Ha disminuido la velocidad </pre>	<pre> classDiagram class Auto { +velocidadMaxima: double +velocidad: double +velocidadFrenado: double +acelerar():void +frenar():void } class auto1 { +velocidadMaxima = 200 +velocidad = 150 +velocidadFrenado = 50 } Auto --> auto1 </pre>

3. Juegos - Gestión de Puntuación

Validación de Puntos: Verificar que los puntos sumados sean positivos. No debe permitirse sumar una cantidad negativa de puntos.

Validación de Rango de Puntuación: Verificar que la puntuación no se modifique a un valor inválido.

Código

```

import java.util.Scanner;

class Juego {
    int puntaje;
    int total;

    public Juego(int puntajeInicial) {
        this.puntaje = puntajeInicial;
        this.total = 0;
    }

    public void agregarPuntaje(int puntosAAgregar) {

```



```

        this.total += puntosAAgregar;
        System.out.println("Se agregaron " + puntosAAgregar + " puntos. Puntaje
total: " + this.total);
    }

    public int getTotal() {
        return this.total;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int puntos;
        Juego juegoActual = new Juego(0);

        do {
            System.out.println("Ingrese el puntaje a agregar ( 0 para salir ): ");
            puntos = teclado.nextInt();
            teclado.nextLine();

            if (puntos > 0) {
                juegoActual.agregarPuntaje(puntos);
            } else if (puntos == 0) {
                System.out.println("Saliendo del sistema de ingreso de
puntajes...");
                break;
            } else {
                System.out.println("Error, no puede ingresar cantidades negativas.
Intente de nuevo");
            }
        } while (puntos != 0);

        System.out.print("Finalizado con un puntaje total de: " +
juegoActual.getTotal());
        teclado.close();
    }
}

```

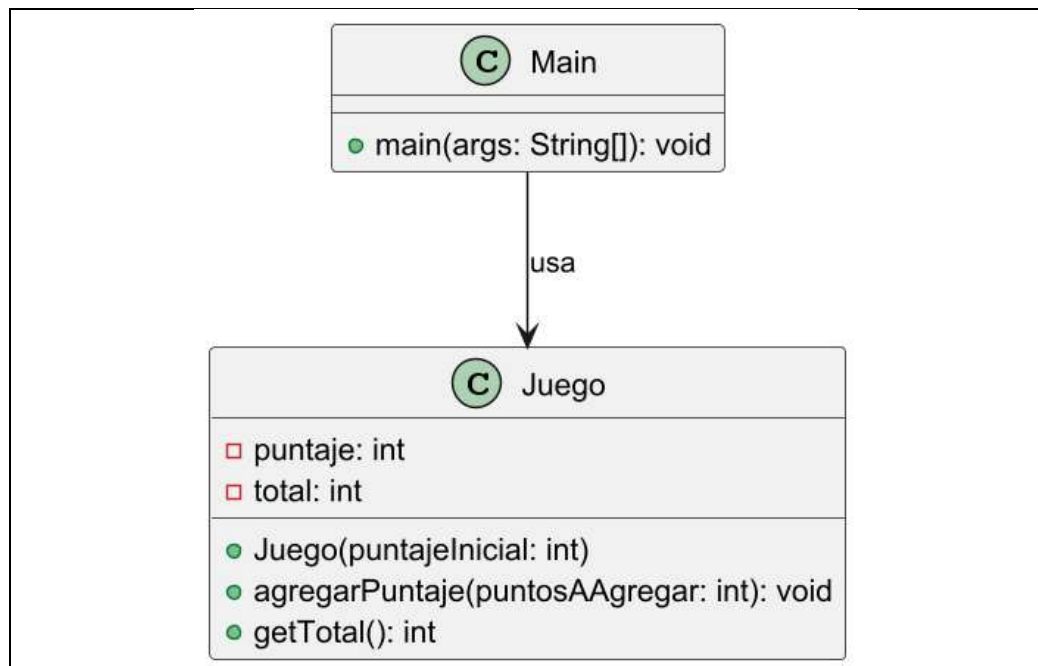
Salida

```

Ingrese el puntaje a agregar ( 0 para salir ):
-7
Error, no puede ingresar cantidades negativas. Intente de nuevo
Ingrese el puntaje a agregar ( 0 para salir ):
77
Se agregaron 77 puntos. Puntaje total: 77
Ingrese el puntaje a agregar ( 0 para salir ):
0
Saliendo del sistema de ingreso de puntajes...
Finalizado con un puntaje total de: 77
Process finished with exit code 0

```

UML



4. Aplicaciones Móviles - Configuración del Usuario

Validación de Idioma: Verificar que el idioma elegido sea uno de los idiomas válidos soportados por la aplicación (por ejemplo, "Español", "Inglés", "Francés").

Validación de Notificaciones: Verificar que la opción de notificaciones esté correctamente activada o desactivada.

CODIGO:

```
public class Main {
    public static void main(String[] args) {
        // Crear objeto Usuario
        Usuario usuario = new Usuario("Español", true);
        // Validar idioma
        if (usuario.validarIdioma()) {
            System.out.println("Idioma válido.");
        } else {
            System.out.println("Idioma no válido.");
        }

        // Validar notificaciones
        if (usuario.validarNotificaciones()) {
            System.out.println("Estado de notificaciones correcto.");
        } else {
            System.out.println("Estado de notificaciones incorrecto.");
        }
    }
}
```

```

public class Usuario {
    //Atributos
    public String idioma;
    public boolean notificacionesActivadas;

    // Constructor
    public Usuario(String idioma, boolean notificacionesActivadas) {
        this.idioma = idioma;
        this.notificacionesActivadas = notificacionesActivadas;
    }

    // Método para validar el idioma
    public boolean validarIdioma() {
        if (idioma.equals("Español") || idioma.equals("Inglés") ||
idioma.equals("Francés")) {
            return true;
        } else {
            return false;
        }
    }

    // Método para validar notificaciones
    public boolean validarNotificaciones() {
        return notificacionesActivadas || !notificacionesActivadas;
    }

    public boolean isNotificacionesActivadas() {
        return notificacionesActivadas;
    }
}

```

EJECUCION:

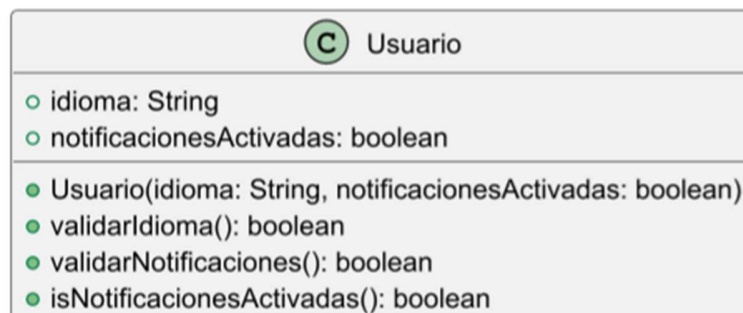
```

Idioma válido.
Estado de notificaciones correcto.

Process finished with exit code 0

```

DIAGRAMA UML:



3. Encapsulamiento

Objetivo: Aplicar el principio de encapsulamiento para proteger los datos y controlarlos mediante métodos controlados.

Acciones a Realizar:

1. Bancos - Control de Cuentas

Encapsula el atributo saldo: Asegúrate de que el atributo saldo sea privado y solo accesible a través de métodos controlados.

Métodos depositar() y retirar(): Implementa estos métodos para que solo puedan modificar el saldo después de realizar las validaciones necesarias.

CÓDIGO:

```
public class cuentaBancaria {
    private double saldo;

    public cuentaBancaria() {
        this.saldo = 0.0;
    }

    public void depositar(double monto) {
        if (monto > 0) {
            saldo += monto;
            System.out.println("Depósito exitoso. Nuevo saldo: $" + saldo);
        } else {
            System.out.println("Error: El depósito debe ser mayor que cero.");
        }
    }

    public void retirar(double monto) {
        if (monto <= 0) {
            System.out.println("Error: El retiro debe ser mayor que cero.");
        } else if (monto > saldo) {
            System.out.println("Error: Fondos insuficientes. Saldo actual: $" + saldo);
        } else {
            saldo -= monto;
            System.out.println("Retiro exitoso. Nuevo saldo: $" + saldo);
        }
    }

    public double obtenerSaldo() {
        return saldo;
    }
}
```

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        cuentaBancaria miCuenta = new cuentaBancaria();
        int opcion;

        do {
            System.out.println("\n----- MENÚ -----");
            System.out.println("1. Depositar");
            System.out.println("2. Retirar");
            System.out.println("3. Consultar saldo");
            System.out.println("4. Salir");
            System.out.print("Elige una opción: ");
            opcion = scanner.nextInt();

            switch (opcion) {
                case 1:
                    System.out.print("Ingresa el monto a depositar: ");
                    double deposito = scanner.nextDouble();
                    miCuenta.depositar(deposito);
                    break;

                case 2:
                    System.out.print("Ingresa el monto a retirar: ");
                    double retiro = scanner.nextDouble();
                    miCuenta.retirar(retiro);
                    break;

                case 3:
                    System.out.println("Saldo actual: $" + miCuenta.obtenerSaldo());
                    break;

                case 4:
                    System.out.println("Gracias por usar el sistema bancario.");
                    break;

                default:
                    System.out.println("Opción no válida. Intenta de nuevo.");
            }
        } while (opcion != 4);

        scanner.close();
    }
}

```

EJECUCIÓN:

```

----- MENÚ -----
1. Depositar
2. Retirar
3. Consultar saldo
4. Salir
Elige una opción: 1
Ingresa el monto a depositar: 200
Depósito exitoso. Nuevo saldo: $200.0

```

```

----- MENÚ -----
1. Depositar
2. Retirar
3. Consultar saldo
4. Salir
Elige una opción: 2
Ingresa el monto a retirar: -5
Error: El retiro debe ser mayor que cero.

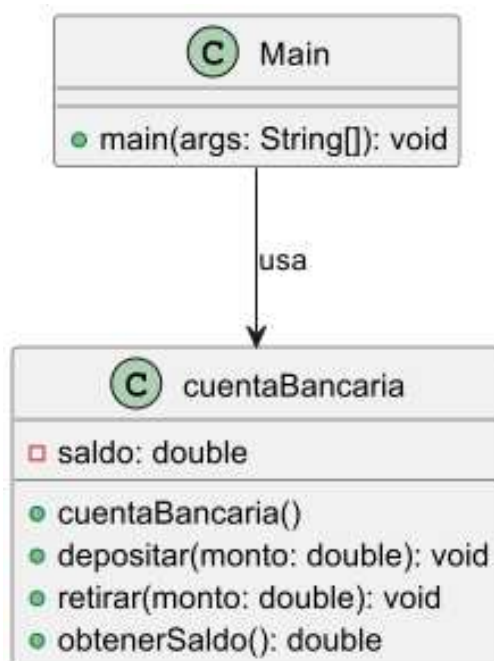
```

```

----- MENÚ -----
1. Depositar
2. Retirar
3. Consultar saldo
4. Salir
Elige una opción: 2
Ingresa el monto a retirar: 50
Error: Fondos insuficientes. Saldo actual: $0.0

```

DIAGRAMA UML:



2. Automóviles - Control de Velocidad

Encapsula el atributo velocidad: La velocidad debe ser privada y no accesible directamente desde fuera de la clase.

Métodos acelerar() y frenar(): Implementa estos métodos para controlar la velocidad, con validaciones de límite máximo y mínima.

Main_Auto2	Auto2
<pre> import java.util.Scanner; public class Main_Auto2 { public static void main(String[] args) { //Se crea un objeto Scanner para leer los datos ingresados por el usuario Scanner sc=new Scanner(System.in); System.out.println("----- --ESTADO DEL AUTO-----"); System.out.println("Ingrese los datos\n"); System.out.println("Velocidad Maxima: "); double vmaxima=sc.nextDouble();//Se lee la velocidad maxima System.out.println("Velocidad (Actual): "); double velocidad=sc.nextDouble();//Se lee la velocidad System.out.println("Velocidad (después del frenado): "); double vfrenado=sc.nextDouble();//Se lee la velocidad despues del frenado //Se crea un objeto con los datos ingresados Auto2 auto1=new Auto2(vmaxima,vfrenado); System.out.println("\n----- ---VELOCIDAD ACTUAL-----"); System.out.print("Estado: "); //Se establece la velocidad actual del auto auto1.setVelocidad(velocidad); //Se obtiene y se muestra la velocidad actual del auto System.out.println(auto1.getVelocidad ()); //Se llama a los metodos para mostrar los resultados System.out.println("\n----- -----RESULTADOS-----"); </pre>	<pre> public class Auto2{ //Se crean los atributos y se determinan como privados private double velocidad; private final double vmaxima; private final double vfrenado; private final double vminima=0; //Se crea un constructor que inicializa los atributos public Auto2(double vmaxima, double vfrenado){ this.vmaxima=vmaxima; this.vfrenado=vfrenado; } //Metodo que usa el set para establecer la velocidad actual public void setVelocidad(double velocidad){ if (velocidad< vminima){ System.out.println("Encienda el auto"); } else if (velocidad>vmaxima) { System.out.println("La velocidad no puede ser mayor que "+vmaxima+"km/h"); } else { this.velocidad=velocidad; System.out.println("Puede acelerar un poco mas"); } } //Metodo que usa el get para devolver la velocidad actual public double getVelocidad(){ return velocidad; } //Se crea un metodo acelerar() para verificar que la velocidad no supere la permitida public void acelerar(){ if (velocidad>vmaxima){ System.out.println("¡ALERTA!\n Reduzca la velocidad. Velocidad permitida: "+vmaxima+" km/h"); } } </pre>

<pre> System.out.println("Al acelerar: "); auto1.acelerar(); System.out.println("Al frenar: "); auto1.frenar(); } } </pre>	<pre> else{ System.out.println("La velocidad es segura, ¡SIGA ASI!\n"); } } //Se crea el metodo frenar para verificar que la velocidad no sea negativa public void frenar(){ if(vfrenado<0){ System.out.println("¡ALERTA!\n La velocidad después del frenado no puede ser negativa"); } else if (vfrenado==vminima) { System.out.println("El auto se ha detenido"); } else{ System.out.println("Ha disminuido la velocidad"); } } } </pre>
<p>SALIDA</p>	<p>UML</p>
<pre> -----ESTADO DEL AUTO----- Ingrese los datos Velocidad Maxima: 200 Velocidad (Actual): 150 Velocidad (después del frenado): 40 -----VELOCIDAD ACTUAL----- Estado: Puede acelerar un poco mas 150.0 -----RESULTADOS----- Al acelerar: La velocidad es segura, ¡SIGA ASI! Al frenar: Ha disminuido la velocidad </pre>	 <pre> classDiagram class Auto2 { +double velocidad +double vmaxima +double vfrenado +double vminima +Auto2(double, double) +setVelocidad(double): void +getVelocidad(): double +acelerar(): void +fenar(): void } class auto1 { +double velocidad: 150 +double vmaxima: 200 +double vfrenado: 40 +double vminima: 0 } Auto2 < -- auto1 </pre>

3. Juegos - Gestión de Puntuación

Encapsula el atributo puntuación: La puntuación debe ser privada y solo modificada a través de un método como sumarPuntos().

Método sumarPuntos(): Implementa este método para validar que los puntos sumados sean positivos antes de aumentar la puntuación.

Código
<pre>import java.util.Scanner; class Juegos { private int puntaje; private int total; public Juegos(int puntaje) { this.puntaje = puntaje; this.total = 0; } public void sumarPuntos(int puntosASumar) { this.total += puntosASumar; System.out.println("Se sumaron " + puntosASumar + " puntos. Puntaje total: " + this.total); } public int getTotal() { return this.total; } } public class Encapsulado { public static void main(String[] args) { Scanner teclado = new Scanner(System.in); int puntos; Juegos juegoActual = new Juegos(0); do { System.out.println("Ingrese el puntaje a sumar (0 para salir); "); puntos = teclado.nextInt(); teclado.nextLine(); if (puntos > 0) { juegoActual.sumarPuntos(puntos); } else if (puntos == 0) { System.out.println("Saliendo del programa de ingreso de puntajes..."); break; } else { System.out.println("Error. No puede ingresar cantidades negativas. Intente de nuevo."); } } while (puntos != 0); } }</pre>

```

        System.out.print("Finalizado con un puntaje total de: " +
juegoActual.getTotal());

        teclado.close();
    }
}

```

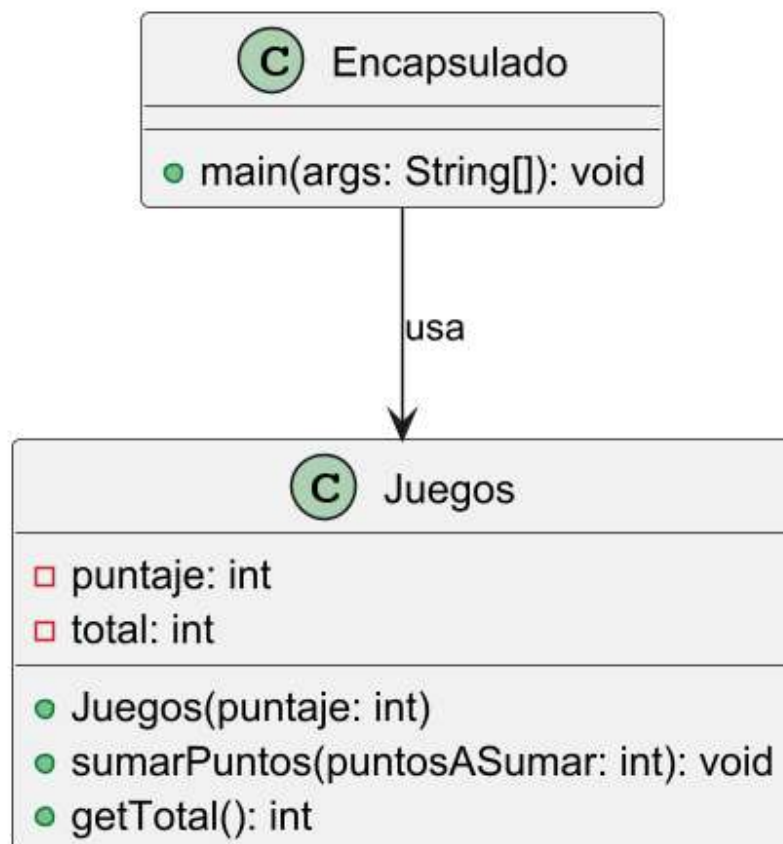
Salida

```

Ingrese el puntaje a sumar ( 0 para salir ):
-7
Error. No puede ingresar cantidades negativas. Intente de nuevo.
Ingrese el puntaje a sumar ( 0 para salir ):
77
Se sumaron 77 puntos. Puntaje total: 77
Ingrese el puntaje a sumar ( 0 para salir ):
0
Saliendo del programa de ingreso de puntajes...
Finalizado con un puntaje total de: 77
Process finished with exit code 0

```

UML



4. Aplicaciones Móviles - Configuración del Usuario

Encapsula los atributos idioma y notificaciones: Asegúrate de que estos atributos sean privados y solo modificables a través de los métodos `cambiarIdioma()` y `activarNotificaciones()`.

Métodos `cambiarIdioma()` y `activarNotificaciones()`: Implementa estos métodos para validar que los valores sean correctos antes de modificarlos.

CODIGO:

```
public class Main {
    public static void main(String[] args) {
        // Crear objeto
        Usuario usuario = new Usuario("Inglés", true);

        // Validar idioma inicial
        if (usuario.validarIdioma()) {
            System.out.println("Idioma válido.");
        } else {
            System.out.println("Idioma no válido.");
        }

        // Validar notificaciones iniciales
        if (usuario.validarNotificaciones()) {
            System.out.println("Estado de notificaciones correcto.");
        } else {
            System.out.println("Estado de notificaciones incorrecto.");
        }

        // Cambiar idioma a uno válido
        usuario.cambiarIdioma("Francés");

        // Cambiar idioma a uno no válido
        usuario.cambiarIdioma("Alemán");

        // Cambiar estado de notificaciones
        usuario.activarNotificaciones(false);
    }
}
```

```
public class Usuario {
    private String idioma;
    private boolean notificacionesActivadas;

    // Constructor
    public Usuario(String idioma, boolean notificacionesActivadas) {
        this.idioma = idioma;
        this.notificacionesActivadas = notificacionesActivadas;
    }

    // Método para validar el idioma actual
    public boolean validarIdioma() {
        return idioma.equals("Español") || idioma.equals("Inglés") ||

```

```

idioma.equals("Francés");
    }

    // Método para validar notificaciones
    public boolean validarNotificaciones() {
        return notificacionesActivadas || !notificacionesActivadas;
    }

    // Método para cambiar el idioma, solo si es válido
    public void cambiarIdioma(String nuevoIdioma) {
        if (nuevoIdioma.equals("Español") || nuevoIdioma.equals("Inglés") ||
nuevoIdioma.equals("Francés")) {
            this.idioma = nuevoIdioma;
            System.out.println("Idioma cambiado correctamente.");
        } else {
            System.out.println("Idioma no válido. No se realizó el cambio.");
        }
    }

    // Método para activar o desactivar notificaciones
    public void activarNotificaciones(boolean activar) {
        this.notificacionesActivadas = activar;
        System.out.println("Estado de notificaciones actualizado.");
    }

    public boolean isNotificacionesActivadas() {
        return notificacionesActivadas;
    }
}

```

EJECUCION:

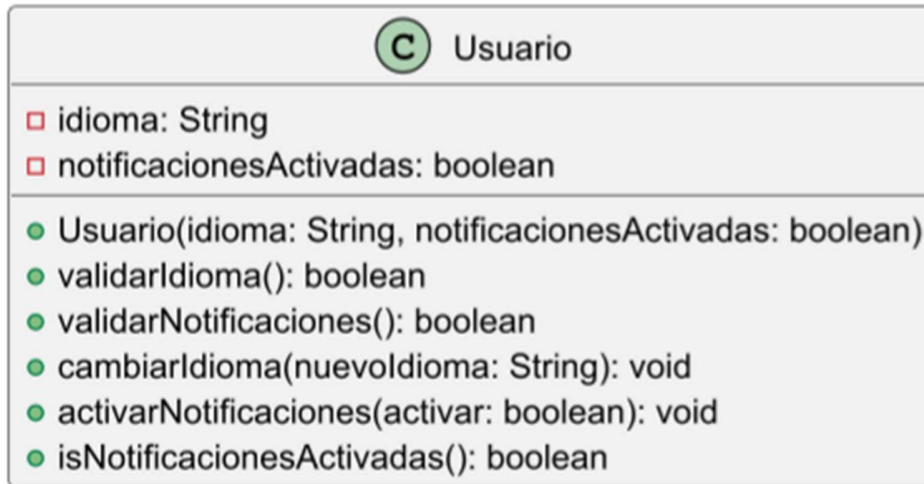
```

Idioma válido.
Estado de notificaciones correcto.
Idioma cambiado correctamente.
Idioma no válido. No se realizó el cambio.
Estado de notificaciones actualizado.

Process finished with exit code 0

```

DIAGRAMA UML:



Conclusión General:

- **Comprender los Métodos:** Es importante entender cómo declarar y usar métodos con retorno y sin retorno, y cuándo es apropiado usar cada tipo.
- **Aplicar Validaciones:** Las validaciones garantizan que las operaciones realizadas sobre los datos sean correctas, evitando errores y comportamientos inesperados.
- **Encapsulamiento:** Ayuda a proteger los datos internos de una clase y a controlarlos mediante métodos públicos que validan y gestionan las operaciones de manera segura.

PRESENTACIÓN:

Subir el archivo de la tarea

Crear el diagrama de UML

Subir el enlace del git hub , 1 por grupo

Defensa 8 de mayo 2025

Link de Github:

- https://github.com/Odaliz2105/TareaS3_Grupal.git

Link de la Presentación de Canva

- https://www.canva.com/design/DAGmT7fceoQ/8P1_bo5tG4f_uzTG74UWgQ/edit?utm_content=DAGmT7fceoQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton