**PROGRAMACION II.**
**Modulo 8: Interfaces y excepciones.**

## Trabajo práctico 8: Interfaces y excepciones.

**Alumno:** LEPKA AGUSTIN
**Comisión:** 13

## OBJETIVO GENERAL

Desarrollar habilidades en el uso de interfaces y manejo de excepciones en Java para fomentar la modularidad, flexibilidad y robustez del código. Comprender la definición e implementación de interfaces como contratos de comportamiento y su aplicación en el diseño orientado a objetos. Aplicar jerarquías de excepciones para controlar y comunicar errores de forma segura. Diferenciar entre excepciones comprobadas y no comprobadas, y utilizar bloques try, catch, finally y throw para garantizar la integridad del programa. Integrar interfaces y manejo de excepciones en el desarrollo de aplicaciones escalables y mantenibles.

## MARCO TEÓRICO

| Concepto | Aplicación en el proyecto |
|---|---|
| Interfaces | Definición de contratos de comportamiento común entre distintas clases |
| Herencia múltiple con interfaces | Permite que una clase implementa múltiples comportamientos sin herencia de estado |
| Implementación de interfaces | Uso de **implements** para que una clase cumpla con los métodos definidos en una interfaz |
| Excepciones | Manejo de errores en tiempo de ejecución mediante estructuras **try-catch** |
| Excepciones checked y unchecked | Diferencias y usos según la naturaleza del error |
| Excepciones personalizadas | Creación de nuevas clases que extienden **Exception** |
| finally y try-with-resources | Buenas prácticas para liberar recursos correctamente |
| Uso de throw y throws | Declaración y lanzamiento de excepciones |


| | |
|---|---|
| Interfaces | Definición de contratos de comportamiento común entre distintas clases |
| Herencia múltiple con interfaces | Permite que una clase implementa múltiples comportamientos sin herencia de estado |

**Caso Practico**

**Parte 1: Interfaces en un sistema de E-commerce**

1. Crear una interfaz **Pagable** con el método **calcularTotal().**

2. Clase **Producto**: tiene nombre y precio, implementa **Pagable.**

3. Clase **Pedido**: tiene una lista de productos, implementa **Pagable** y calcula el total del pedido.

4. Ampliar con interfaces **Pago** y **PagoConDescuento** para distintos medios de pago (**TarjetaCredito, PayPal**), con métodos **procesarPago(double)** y **aplicarDescuento(double)**.

5. Crear una interfaz **Notificable** para notificar cambios de estado. La clase **Cliente** implementa dicha interfaz y **Pedido** debe notificarlo al cambiar de estado.

**Clase Main:**

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

public class TP8 {


    public static void main(String[] args) {

        Cliente cliente = new Cliente("Agustin");

        Pedido pedido = new Pedido(cliente);

        pedido.agregarProducto(new Producto("Heladera", 600000));
        pedido.agregarProducto(new Producto("Microondas", 50000));

        double total = pedido.calcularTotal();
        System.out.println("Total sin descuento: $" + total);

        TarjetaCredito tc = new TarjetaCredito();
        PayPal paypal = new PayPal();

        //Pago con tarjeta con descuento
        double totalDescuento = tc.aplicarDescuento(total);
        tc.procesarPago(totalDescuento);

        //Cambio de estado del pedido
        pedido.cambiarEstado("Preparando envio");

        //Pago con PayPal
        paypal.procesarPago(total);
    }
```

```
}
```

### Interfaz Pagable:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Interfaz Pagable: establece que cualquier clase que la implemente
debe definir el método calcularTotal()
public interface Pagable {

    //Metodo que retorna el total calculado
    double calcularTotal();

}
```

### Clase Producto:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */

package Parte1;

public class Producto implements Pagable {

    //Atributos del producto
    private String nombre; // Nombre del producto
    private double precio; // Precio del producto

    //Constructor para inicializar nombre y precio
    public Producto(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
    }

    //Uso del método calcularTotal() de Pagable
    @Override
    public double calcularTotal() {
        return precio; // El total de un producto es su precio
    }

    //Getter para usarlo fuera de la clase
    public String getNombre() {
        return nombre;
    }
}
```

## Clase Pedido:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;
import java.util.ArrayList;
import java.util.List;

//Clase Pedido representa una compra con varios productos
//Implementa Pagable para calcular el total del pedido
public class Pedido implements Pagable {

    //Lista de productos del pedido
    private List<Producto> productos = new ArrayList<>();

    //Estado del pedido
    private String estado = "Pendiente";

    // liente al cual se notificará cambios de estado
    private Notificable cliente;

    //Constructor del pedido
    public Pedido(Notificable cliente) {
        this.cliente = cliente;
    }

    //Metodo para agregar productos al pedido
    public void agregarProducto(Producto p) {
        productos.add(p); // Se añade el producto a la lista
    }

    //Implementación de calcularTotal()
    @Override
    public double calcularTotal() {
        double total = 0; // Acumulador del total
        for (Producto p : productos) {
            total += p.calcularTotal(); // Suma del precio de cada
producto
        }
        return total;
    }

    //Método para cambiar estado y notificar al cliente
    public void cambiarEstado(String nuevoEstado) {
        this.estado = nuevoEstado;
        cliente.notificar("El pedido cambió a: " + nuevoEstado);
    }
}
```

## Interfaz Pago:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Interfaz Pago define el método para procesar pagos
public interface Pago {
    void procesarPago(double monto);
}
```

## Interfaz PagoConDescuento:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Interfaz que agrega método para aplicar descuento
public interface PagoConDescuento {
    double aplicarDescuento(double monto);
}
```

## Clase TarjetaCredito:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Implementa Pago y PagoConDescuento porque puede pagar y tener
descuento
public class TarjetaCredito implements Pago, PagoConDescuento {

    @Override
    public void procesarPago(double monto) {
        System.out.println("Pago con tarjeta realizado. Monto final:
$" + monto);
    }

    @Override
    public double aplicarDescuento(double monto) {
        return monto * 0.90; //10% de descuento
    }
}
```

### Clase PayPal:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//PayPal solo realiza pagos, no tiene descuento
public class PayPal implements Pago {

    @Override
    public void procesarPago(double monto) {
        System.out.println("Pago vía PayPal realizado. Monto: $" +
monto);
    }
}
```

### Interfaz Notificable:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Interfaz que obliga a implementar método de notificación
public interface Notificable {
    void notificar(String mensaje);
}
```

### Clase Cliente:

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte1;

//Clase Cliente implementa Notificable
public class Cliente implements Notificable {

    private String nombre; // Nombre del cliente

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

    @Override
    public void notificar(String mensaje) {
        System.out.println("Cliente " + nombre + " fue notificado: " +
mensaje);
    }
}
```

**Parte 2: Ejercicios sobre Excepciones**

1. División segura
○ Solicitar dos números y dividirlos. Manejar **ArithmeticException** si el divisor es cero.

2. Conversión de cadena a número
○ Leer texto del usuario e intentar convertirlo a int. Manejar **NumberFormatException** si no es válido.

3. Lectura de archivo
○ Leer un archivo de texto y mostrarlo. Manejar **FileNotFoundException** si el archivo no existe.

4. Excepción personalizada
○ Crear **EdadInvalidaException**. Lanzarla si la edad es menor a 0 o mayor a 120. Capturarla y mostrar mensaje.

5. **Uso de try-with-resources**
○ Leer un archivo con **BufferedReader** usando **try-with-resources**. Manejar **IOException** correctamente.

**Clase Main:**

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */

package Parte2;
import java.io.*;
import java.util.Scanner;

public class ExcepcionesTP {


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

        //1. Division segura
        try {
            System.out.print("Ingrese dividendo: ");
            int a = sc.nextInt();
            System.out.print("Ingrese divisor: ");
            int b = sc.nextInt();
            System.out.println("Resultado: " + (a / b));
        } catch (ArithmeticException e) {
            System.out.println("Error: division por cero.");
        }

        sc.nextLine(); //limpiar buffer

        //2. Conversion segura
        try {
            System.out.print("Ingrese un número: ");
```

```
            String texto = sc.nextLine();
            int numero = Integer.parseInt(texto);
            System.out.println("Número ingresado: " + numero);
        } catch (NumberFormatException e) {
            System.out.println("Error: no es un número válido.");
        }

        //3. Lectura de archivo
        try {
            BufferedReader br = new BufferedReader(new
FileReader("archivo.txt"));
            System.out.println("Contenido: " + br.readLine());
            br.close();
        } catch (FileNotFoundException e) {
            System.out.println("Error: archivo no encontrado.");
        } catch (IOException e) {
            System.out.println("Error de lectura.");
        }

        //4. Excepción personalizada
        try {
            validarEdad(200);
        } catch (EdadInvalidaException e) {
            System.out.println(e.getMessage());
        }

        //5. Try-with-resources
        try (BufferedReader br = new BufferedReader(new
FileReader("archivo.txt"))) {
            System.out.println("Leyendo archivo con try with
resources...");
            System.out.println(br.readLine());
        } catch (IOException e) {
            System.out.println("Error al leer archivo.");
        }
    }

    //Metodo para validar edad
    public static void validarEdad(int edad) throws
EdadInvalidaException {
        if (edad < 0 || edad > 120) {
            throw new EdadInvalidaException("Edad invalida: " + edad);
        }
    }

}
```

**Clase EdadInvalidaException:**

```
/*
PROGRAMACION II
ALUMNO: LEPKA AGUSTIN
COMISION: 13
 */
package Parte2;

//Excepcion personalizada que extiende Exception
public class EdadInvalidaException extends Exception {

    public EdadInvalidaException(String mensaje) {
        super(mensaje);
    }
}
```