

# PROGRAMACIÓN II

## TP 8: Interfaces y Excepciones en Java

### 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.

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 <b>implements</b> para que una clase cumpla con los métodos definidos en una interfaz
Excepciones	Manejo de errores en tiempo de ejecución mediante estructuras <b>try-catch</b>
Excepciones checked y unchecked	Diferencias y usos según la naturaleza del error
Excepciones personalizadas	Creación de nuevas clases que extienden <b>Exception</b>
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

## MARCO TEÓRICO

### 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.

#### 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.

## CONCLUSIONES ESPERADAS

- Comprender la utilidad de las interfaces para lograr diseños desacoplados y reutilizables.
- Aplicar herencia múltiple a través de interfaces para combinar comportamientos.

- Utilizar correctamente estructuras de control de excepciones para evitar caídas del programa.
- Crear excepciones personalizadas para validar reglas de negocio.
- Aplicar buenas prácticas como **try-with-resources** y uso del bloque **finally** para manejar recursos y errores.
- Reforzar el diseño robusto y mantenible mediante la integración de interfaces y manejo de errores en Java.

## RESOLUCIÓN

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

#### 1. Estructura de Archivos

Interfaces: Pagable.java, Notificable.java, Pago.java, PagoConDescuento.java

Clases: Producto.java, Cliente.java, Pedido.java, TarjetaCredito.java, PayPal.java

Main: MainApp.java

#### 2. Código

Pagable.java

```
package ecommerce;
```

```
public interface Pagable {  
    double calcularTotal();  
}
```

Notificable.java

```
package ecommerce;
```

```
public interface Notificable {  
    void notificarCambio(String mensaje);  
}
```

Pago.java

```
package ecommerce;
```

```
public interface Pago {  
    boolean procesarPago(double monto);  
}
```

PagoConDescuento.java

```
package ecommerce;
```

```
public interface PagoConDescuento extends Pago {  
    double aplicarDescuento(double montoOriginal);  
}
```

Producto.java

```
package ecommerce;
```

```
public class Producto implements Pagable {  
    private String nombre;  
    private double precio;  
  
    public Producto(String nombre, double precio) {  
        this.nombre = nombre;  
        this.precio = precio;  
    }
```

```
    @Override  
    public double calcularTotal() {  
        return this.precio;  
    }
```

```
    public String getNombre() { return nombre; }  
}
```

Cliente.java

```
package ecommerce;
```

```
public class Cliente implements Notificable {  
    private String nombre;
```

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

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

Pedido.java

```
package ecommerce;
```

```
import java.util.ArrayList;
import java.util.List;

public class Pedido implements Pagable {
    private List<Producto> productos;
    private Cliente cliente;
    private String estado;

    public Pedido(Cliente cliente) {
        this.productos = new ArrayList<>();
        this.cliente = cliente;
        this.estado = "Creado";
    }

    public void agregarProducto(Producto producto) {
        this.productos.add(producto);
    }

    @Override
    public double calcularTotal() {
        double total = 0;
        for (Producto p : productos) {
            total += p.calcularTotal();
        }
        return total;
    }

    public void cambiarEstado(String nuevoEstado) {
        this.estado = nuevoEstado;
        this.cliente.notificarCambio("El estado del pedido ha cambiado a: " + nuevoEstado);
    }
}
```

TarjetaCredito.java

```
package ecommerce;

public class TarjetaCredito implements PagoConDescuento {
    private double tasaDescuento = 0.10; // 10%

    @Override
    public boolean procesarPago(double monto) {
        System.out.println("Pago con Tarjeta de Crédito por $" + monto + " procesado.");
        return true;
    }

    @Override
    public double aplicarDescuento(double montoOriginal) {
```

```
double montoConDescuento = montoOriginal * (1 - tasaDescuento);  
System.out.println("Descuento de " + (tasaDescuento * 100) + "% aplicado.");  
return montoConDescuento;  
}  
}
```

PayPal.java

```
package ecommerce;  
  
public class PayPal implements Pago {  
    @Override  
    public boolean procesarPago(double monto) {  
        System.out.println("Pago con PayPal por $" + monto + " procesado.");  
        return true;  
    }  
}
```

## Parte 2: Ejercicios sobre Excepciones

### 1. Estructura de Archivos

Excepción Personalizada: EdadInvalidaException.java

Main: MainApp.java (contiene todas las pruebas de excepciones)

### 2. Código de la Excepción Personalizada

EdadInvalidaException.java

```
package excepciones; // Crear un nuevo paquete para excepciones  
  
public class EdadInvalidaException extends Exception {  
    public EdadInvalidaException(String mensaje) {  
        super(mensaje);  
    }  
}
```

### 3. Código del Main (Integración y Pruebas)

MainApp.java

```
package ecommerce;  
  
import excepciones.EdadInvalidaException;  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;
```

```
public class MainApp {

    public static void main(String[] args) {
        System.out.println("=== PRUEBAS DE E-COMMERCE (INTERFACES) ===");
        probarEcommerce();
        System.out.println("\n=== PRUEBAS DE EXCEPCIONES ===");

        // Ejercicio 1: División Segura
        ejercicioDivisionSegura(10, 2);
        ejercicioDivisionSegura(10, 0);

        // Ejercicio 2: Conversión de Cadena
        ejercicioConversionCadena("123");
        ejercicioConversionCadena("Hola");

        // Ejercicio 3 y 5: Lectura de Archivo
        // Nota: Asegurar que "archivo_prueba.txt" exista para la prueba de T-W-R.
        ejercicioLecturaArchivoConFinally("archivo_inexistente.txt");
        ejercicioTryWithResources("archivo_prueba.txt");

        // Ejercicio 4: Excepción Personalizada
        ejercicioExcepcionPersonalizada(25);
        ejercicioExcepcionPersonalizada(150);
    }

    private static void probarEcommerce() {
        Cliente cliente = new Cliente("Marta Gómez");
        Pedido pedido = new Pedido(cliente);

        pedido.agregarProducto(new Producto("Libro POO", 500.0));
        pedido.agregarProducto(new Producto("Licencia Java", 750.0));

        double totalOriginal = pedido.calcularTotal();
        System.out.println("Total a Pagar (Pagable): $" + totalOriginal);

        // Pago con Descuento
        TarjetaCredito tarjeta = new TarjetaCredito();
        double totalDescontado = tarjeta.aplicarDescuento(totalOriginal);
        tarjeta.procesarPago(totalDescontado);

        // Notificable
        pedido.cambiarEstado("Preparando Envío");
    }

    // 1. División segura (Manejo de ArithmeticException)
    private static void ejercicioDivisionSegura(double a, double b) {
        try {
            if (b == 0) {
```

```
        throw new ArithmeticException("El divisor no puede ser cero.");
    }
    double resultado = a / b;
    System.out.println(a + " / " + b + " = " + resultado);
} catch (ArithmeticException e) {
    System.err.println("Error (División Segura): " + e.getMessage());
}
}

// 2. Conversión de cadena a número (Manejo de NumberFormatException)
private static void ejercicioConversionCadena(String texto) {
    try {
        int numero = Integer.parseInt(texto);
        System.out.println("Cadena " + texto + " convertida a int: " + numero);
    } catch (NumberFormatException e) {
        System.err.println("Error (Conversión de Cadena): " + texto + " no es un número entero válido.");
    }
}

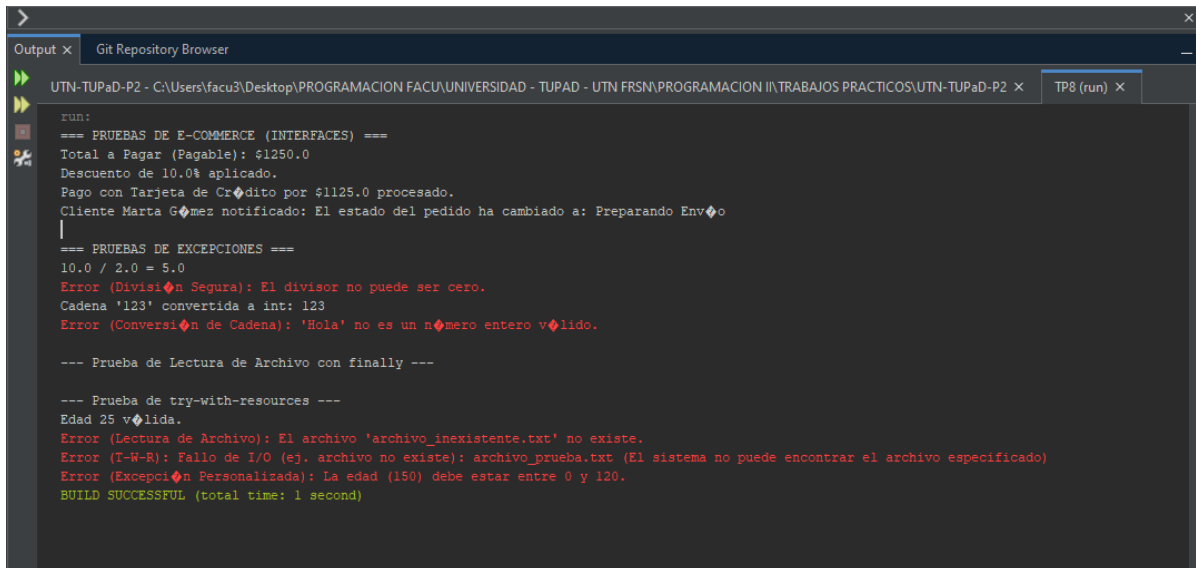
// 3. Lectura de archivo (Manejo de FileNotFoundException y uso de finally)
private static void ejercicioLecturaArchivoConFinally(String nombreArchivo) {
    System.out.println("\n--- Prueba de Lectura de Archivo con finally ---");
    FileReader fr = null;
    try {
        fr = new FileReader(nombreArchivo);
        // Simulación de lectura...
        System.out.println("Archivo " + nombreArchivo + " encontrado y abierto.");
    } catch (java.io.FileNotFoundException e) {
        System.err.println("Error (Lectura de Archivo): El archivo " + nombreArchivo + " no existe.");
    } finally {
        if (fr != null) {
            try {
                fr.close();
                System.out.println("Recurso FileReader cerrado en finally.");
            } catch (IOException e) {
                System.err.println("Error al cerrar recurso: " + e.getMessage());
            }
        }
    }
}

// 4. Excepción personalizada (Creación y lanzamiento de EdadInvalidaException)
private static void ejercicioExcepcionPersonalizada(int edad) {
    try {
        validarEdad(edad);
        System.out.println("Edad " + edad + " válida.");
    }
```



```
} catch (EdadInvalidaException e) {  
    System.err.println("Error (Excepción Personalizada): " + e.getMessage());  
}  
}  
  
private static void validarEdad(int edad) throws EdadInvalidaException {  
    if (edad < 0 || edad > 120) {  
        throw new EdadInvalidaException("La edad (" + edad + ") debe estar entre 0 y 120.");  
    }  
}  
  
// 5. Uso de try-with-resources (Manejo de IOException)  
private static void ejercicioTryWithResources(String nombreArchivo) {  
    System.out.println("\n--- Prueba de try-with-resources ---");  
    // Asegura que BufferedReader se cierre automáticamente  
    try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {  
        String linea = br.readLine();  
        System.out.println("Contenido del archivo (T-W-R): " + linea);  
    } catch (IOException e) {  
        System.err.println("Error (T-W-R): Fallo de I/O (ej. archivo no existe): " + e.getMessage());  
    }  
}
```

## RESULTADOS OUTPUT



```
run:  
=== PRUEBAS DE E-COMMERCE (INTERFACES) ===  
Total a Pagar (Pagable): $1250.0  
Descuento de 10.0% aplicado.  
Pago con Tarjeta de Crédito por $1125.0 procesado.  
Cliente Marta Gómez notificado: El estado del pedido ha cambiado a: Preparando Envío  
|  
=== PRUEBAS DE EXCEPCIONES ===  
10.0 / 2.0 = 5.0  
Error (División Segura): El divisor no puede ser cero.  
Cadena '123' convertida a int: 123  
Error (Conversión de Cadena): 'Hola' no es un número entero válido.  
  
--- Prueba de Lectura de Archivo con finally ---  
  
--- Prueba de try-with-resources ---  
Edad 25 válida.  
Error (Lectura de Archivo): El archivo 'archivo_inexistente.txt' no existe.  
Error (T-W-R): Fallo de I/O (ej. archivo no existe): archivo_prueba.txt (El sistema no puede encontrar el archivo especificado)  
Error (Excepción Personalizada): La edad (150) debe estar entre 0 y 120.  
BUILD SUCCESSFUL (total time: 1 second)
```

Link de repo: <https://github.com/Dario-Cabrera/UTN-TUPaD-P2.git>

Alumno: Cabrera Dario Ezequiel

DNI: 41375492