UNIVERSIDAD TECNOLÓGICA NACIONAL

# PROGRAMACIÓN II

## *Trabajo Práctico N.º 8: Interfaces y excepciones*

Estudiante: Emilia Gómez Juárez

Objetivo:

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.

### Parte 1: Interfaces en un sistema E-Commerce

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

```
1   package partel;
2
    public interface Pagable {
        double calcularTotal();
5   }
```

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

```java
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 precio;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

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

```java
public class Pedido implements Pagable {

    private List<Producto> productos = new ArrayList<>();
    private Cliente cliente;
    private String estado;

    public Pedido(Cliente cliente) {
        this.cliente = cliente;
        this.estado = "CREADO";
    }
    public void agregarProducto(Producto producto) {
        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;
        cliente.notificarCambioEstado("El estado del pedido cambió a: " + nuevoEstado);
    }
    @Override
    public String toString() {
        return "Pedido de " + cliente.getNombre() + " - Estado: " + estado + " - Total: $" + calcularTotal();
    }
}
```

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

```java
package partel;

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

```java
package partel;

public interface PagoConDescuento extends Pago {

    double aplicarDescuento(double monto);
}
```

```java
package partel;

public class PagoPayPal implements Pago {
    @Override
    public void procesarPago(double monto) {
        System.out.println("Pago con PayPal procesado por $" + monto);
    }
}
```

```java
package partel;

public class PagoTarjetaCredito implements PagoConDescuento {
    public static double DESCUENTO = 0.95;

    @Override
    public double aplicarDescuento(double monto) {
        return monto * DESCUENTO;
    }

    @Override
    public void procesarPago(double monto) {
        System.out.println("Pago con tarjeta procesado por $" + monto);
    }
}
```

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

```java
package partel;

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

***MAIN***

```java
public class Main {
    public static void main(String[] args) {
        Cliente cliente = new Cliente("Juan");
        Pedido pedido = new Pedido(cliente);

        pedido.agregarProducto(new Producto("Mouse", 1500));
        pedido.agregarProducto(new Producto("Teclado", 2500));

        System.out.println(pedido);

        pedido.cambiarEstado("PAGADO");

        PagoConDescuento pagoTarjeta = new PagoTarjetaCredito();
        double totalConDescuento = pagoTarjeta.aplicarDescuento(pedido.calcularTotal());
        pagoTarjeta.procesarPago(totalConDescuento);
    }
}
```

***RESULTADO***

```
Pedido de Juan - Estado: CREADO - Total: $4000.0
Notificaci�n para Juan: El estado del pedido cambi� a: PAGADO
Pago con tarjeta procesado por $3800.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Parte 2: Ejercicios sobre excepciones

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

```java
package parte2;

import java.util.Scanner;

public class DivisionSegura {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Ingrese el dividendo: ");
            int a = sc.nextInt();
            System.out.print("Ingrese el divisor: ");
            int b = sc.nextInt();

            int resultado = a / b;
            System.out.println("Resultado: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: No se puede dividir por cero.");
        } finally {
            sc.close();
        }
    }
}
```

***RESULTADO***

```
run:
Ingrese el dividendo: 10
Ingrese el divisor: 0
Error: No se puede dividir por cero.
BUILD SUCCESSFUL (total time: 11 seconds)
```

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

```java
package parte2;

import java.util.Scanner;

public class ConversionCadena {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        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: Entrada no válida, debe ser un número.");
        } finally {
            sc.close();
        }
    }
}
```

***RESULTADO***

```
Ingrese un n□mero: HOLA
Error: Entrada no v□lida, debe ser un n□mero.
BUILD SUCCESSFUL (total time: 3 seconds)
```

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

```java
package parte2;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class LecturaArchivo {
    public static void main(String[] args) {
        try {
            File archivo = new File("archivo-de-prueba.txt");
            Scanner sc = new Scanner(archivo);
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("Error: El archivo no existe.");
        }
    }
}
```

***RESULTADO***

```
run:
Error: El archivo no existe.
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

```java
package parte2;

public class EdadInvalidaException extends Exception {
    public EdadInvalidaException(String mensaje) {
        super(mensaje);
    }
}
```

```java
package parte2;

import java.util.Scanner;

public class ValidarEdad {
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in);){
            System.out.print("Ingrese su edad: ");
            int edad = sc.nextInt();
            validarEdad(edad);
            System.out.println("Edad válida: " + edad);
        } catch (EdadInvalidaException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void validarEdad(int edad) throws EdadInvalidaException {
        if (edad < 0 || edad > 120) {
            throw new EdadInvalidaException("La edad debe estar entre 0 y 120 años.");
        }
    }
}

```

***RESULTADO***

```
run:
Ingrese su edad: 130
Error: La edad debe estar entre 0 y 120 a◆os.
BUILD SUCCESSFUL (total time: 4 seconds)
```

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

UNIVERSIDAD TECNOLÓGICA NACIONAL

# PROGRAMACIÓN II

```java
package parte2;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class LecturaArchivoTryWithResources {
    public static void main(String[] args) {
        String ruta = "archivo-de-prueba.txt";
        try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
        }
    }
}
```

***RESULTADO***

```
run:
Error al leer el archivo: archivo.txt (El sistema no puede encontrar el archivo especificado)
BUILD SUCCESSFUL (total time: 0 seconds)
```

**REPOSITORIO REMOTO:** https://github.com/GomezJEmilia/UTN-Programacion2-TPs-EmiliaGJ/tree/8319eb7c9b24985e9c4971a068ef22a87ae59651/08%20Interfaces%20y%20excepciones/08%20Interfaces%20y%20excepciones

Emilia Gómez Juárez
2025