

PROGRAMACIÓN II

Trabajo Práctico 8: Interfaces y Excepciones en Java

Alumno: Mauro Gaspar
Comisión: 4

Caso Práctico

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.

Interfaces:

```
* @author Mauro
*/
public interface Pagable {
    double calcularTotal();
}
```

```
* @author Mauro
*/
public interface Notificable {
    void notificar(String mensaje);
}
```

```
* @author Mauro
*/
public interface Pago {
    boolean procesarPago(double monto) throws Exception;
}
```

```
* @author Mauro
*/
public interface PagoConDescuento {
    double aplicarDescuento(double monto, double porcentaje);
}
```

Clase Producto:

```
public class Producto implements Pagable {
    private String nombre;
    private double precio;

    public Producto(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
    }

    public String getNombre() { return nombre; }
    public double getPrecio() { return precio; }

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

    @Override
    public String toString() {
        return nombre + " ($" + precio + ")";
    }
}
```

Clase Pedido:

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

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

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

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

    public void quitarProducto(Producto p) {
        productos.remove(p);
    }

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

```
public void setEstado(EstadoPedido nuevoEstado) {
    EstadoPedido viejo = this.estado;
    this.estado = nuevoEstado;
    // Notificar solo si cambió
    if (viejo != nuevoEstado) {
        cliente.notificar("Estado del pedido cambiado de " + viejo +
            ". Total: $" + calcularTotal());
    }
}

public EstadoPedido getEstado() {
    return estado;
}

@Override
public String toString() {
    return "Pedido{" +
        "cliente=" + cliente +
        ", estado=" + estado +
        ", total=$" + calcularTotal() +
        ", productos=" + productos +
        '}';
}
}
```

Clase Cliente:

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

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

    @Override
    public void notificar(String mensaje) {
        // En un sistema real enviaríamos un correo. Aquí lo imprimimos.
        System.out.println("Notificación a " + nombre + " (" + email + "): " + mensaje);
    }

    @Override
    public String toString() {
        return nombre + " <" + email + ">";
    }
}
```

Clase enum EstadoPedido:

```
* @author Mauro
*/
public enum EstadoPedido {
    CREADO,
    PAGADO,
    EN_PREPARACION,
    ENVIADO,
    ENTREGADO,
    CANCELADO
}
```

Clase TarjetaCredito:

```
public class TarjetaCredito implements Pago, PagoConDescuento {
    private String titular;
    private String numero;

    public TarjetaCredito(String titular, String numero) {
        this.titular = titular;
        this.numero = numero;
    }

    @Override
    public boolean procesarPago(double monto) throws Exception {
        // Simulación: si número termina en 0, falla
        if (numero.endsWith("0")) {
            throw new Exception("Pago rechazado por el banco.");
        }
        System.out.println("Pago con tarjeta aprobado: $" + monto);
        return true;
    }

    @Override
    public double aplicarDescuento(double monto, double porcentaje) {
        return monto * (1 - porcentaje / 100.0);
    }
}
```

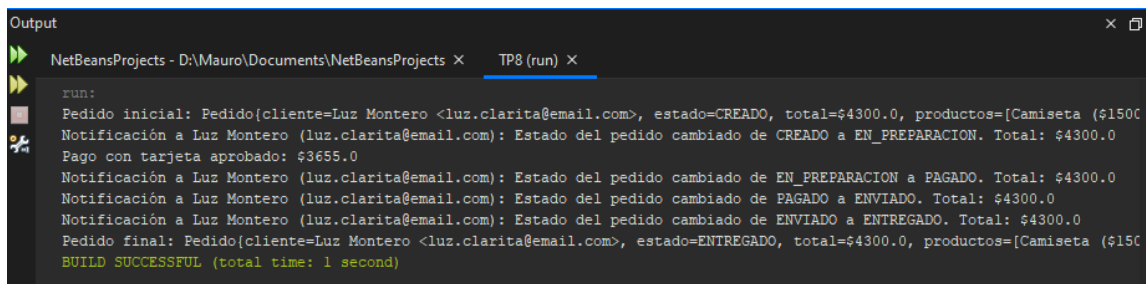
Clase PayPal:

```
public class PayPal implements Pago {  
    private String cuenta;  
  
    public PayPal(String cuenta) {  
        this.cuenta = cuenta;  
    }  
  
    @Override  
    public boolean procesarPago(double monto) throws Exception {  
        // Simulación simple  
        if (cuenta == null || cuenta.isEmpty()) {  
            throw new Exception("Cuenta PayPal inválida.");  
        }  
        System.out.println("Pago con PayPal aprobado: $" + monto);  
        return true;  
    }  
}
```

Clase Main:

```
public class Main {  
    public static void main(String[] args) {  
        Cliente cliente = new Cliente("Luz Montero", "luz.clarita@email.com");  
  
        Producto p1 = new Producto("Camiseta", 1500);  
        Producto p2 = new Producto("Sombrero", 800);  
        Producto p3 = new Producto("Pantalones", 2000);  
  
        Pedido pedido = new Pedido(cliente);  
        pedido.agregarProducto(p1);  
        pedido.agregarProducto(p2);  
        pedido.agregarProducto(p3);  
  
        System.out.println("Pedido inicial: " + pedido);  
  
        // Cambiar estado y observar notificación  
        pedido.setEstado(EstadoPedido.EN_PREPARACION);  
  
        // Procesar pago con descuento usando TarjetaCredito  
        TarjetaCredito tarjeta = new TarjetaCredito("Luz Montero", "2111982702");  
        try {  
            double total = pedido.calcularTotal();  
            double totalConDesc = tarjeta.aplicarDescuento(total, 15); // 10% off  
            if (tarjeta.procesarPago(totalConDesc)) {  
                pedido.setEstado(EstadoPedido.PAGADO);  
            }  
        } catch (Exception ex) {  
            System.out.println("Error al procesar pago: " + ex.getMessage());  
        }  
  
        // Simular envío  
        pedido.setEstado(EstadoPedido.ENVIADO);  
        pedido.setEstado(EstadoPedido.ENTREGADO);  
  
        System.out.println("Pedido final: " + pedido);  
    }  
}
```

Salida:



Output

NetBeansProjects - D:\Mauro\Documents\NetBeansProjects X TP8 (run) X

run:

Pedido inicial: Pedido(cliente=Luz Montero <luz.clarita@email.com>, estado=CREADO, total=\$4300.0, productos=[Camiseta (\$1500, Sombrero (\$800), Pantalones (\$2000)])

Notificación a Luz Montero (luz.clarita@email.com): Estado del pedido cambiado de CREADO a EN_PREPARACION. Total: \$4300.0

Pago con tarjeta aprobado: \$3655.0

Notificación a Luz Montero (luz.clarita@email.com): Estado del pedido cambiado de EN_PREPARACION a PAGADO. Total: \$4300.0

Notificación a Luz Montero (luz.clarita@email.com): Estado del pedido cambiado de PAGADO a ENVIADO. Total: \$4300.0

Notificación a Luz Montero (luz.clarita@email.com): Estado del pedido cambiado de ENVIADO a ENTREGADO. Total: \$4300.0

Pedido final: Pedido(cliente=Luz Montero <luz.clarita@email.com>, estado=ENTREGADO, total=\$4300.0, productos=[Camiseta (\$1500, Sombrero (\$800), Pantalones (\$2000)])

BUILD SUCCESSFUL (total time: 1 second)

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 `EdadInvalidaException`:

```
* @author Mauro
*/
public class EdadInvalidaException extends Exception { // 1
    public EdadInvalidaException(String message) {
        super(message);
    }
}
```


Clase Excepciones:

```
package tp8;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/**
 *
 * @author Mauro
 */

public class Excepciones {

    // 1. División
    // Demuestra la captura de la excepción ArithmeticException (una excepción Unchecked)
    public static void divisionSegura(int a, int b) {
        try {
            // Si b es cero, Java lanza la excepción.
            int r = a / b;
            System.out.println("Resultado: " + r);
        } catch (ArithmeticException e) {
            System.out.println("No se puede dividir por cero: " + e.getMessage());
        }
    }

    // 2. Conversión segura de String a int
    // Demuestra la captura de la excepción NumberFormatException
    public static void conversionSegura(String texto) {
        try {
            // El bloque try utiliza Integer.parseInt(texto) para convertir la cadena a entero.
            int valor = Integer.parseInt(texto);
            System.out.println("Valor convertido: " + valor);
        } catch (NumberFormatException e) {
            System.out.println("Formato inválido para entero: " + texto);
        }
    }
}
```

```
// 3 y 5. Lectura de archivo con try-with-resources
// Demuestra la gestión de recursos (FileReader, BufferedReader) y la captura de IOException
public static void leerArchivo(String ruta) {
    System.out.println("Intentando leer el archivo en: " + ruta);
    try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
        String linea;
        System.out.println("Contenido:");
        while ((linea = br.readLine()) != null) {
            System.out.println("-> " + linea);
        }
    } catch (IOException e) { // captura FileNotFoundException y otros IOExceptions
        System.out.println("Error leyendo archivo: " + e.getMessage());
    }
}

// 4. Excepción personalizada para edad
// Demuestra cómo lanzar manualmente una excepción y el concepto de Excepciones Verificadas (
public static void validarEdad(int edad) throws EdadInvalidaException {
    if (edad < 0 || edad > 120) {
        throw new EdadInvalidaException("Edad inválida fuera del rango (0-120): " + edad);
    }
    System.out.println("Edad válida: " + edad);
}
```

Main:

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    // 1. Prueba de División Segura (ArithmeticException)  
    System.out.println("\n--- 1. Validación de División por Cero ---");  
    System.out.print("Ingrese el numerador: ");  
    int num = scanner.nextInt();  
    System.out.print("Ingrese el divisor (pruebe con 0 para error, o 2 para éxito): ");  
    int den = scanner.nextInt();  
    divisionSegura(num, den);  
  
    // 2. Prueba de Conversión Segura (NumberFormatException)  
    scanner.nextLine(); // Consumir el salto de línea pendiente  
    System.out.println("\n--- 2. Validación de Formato Numérico ---");  
    System.out.print("Ingrese texto para convertir (pruebe '123' o 'texto'): ");  
    String texto = scanner.nextLine();  
    conversionSegura(texto);  
  
    // 3. Prueba de Lectura de Archivo (IOException - Checked)  
    System.out.println("\n--- 3. Validación de Acceso a Archivo (try-with-resources) ---");  
    System.out.print("Ingrese la ruta de un archivo (pruebe 'archivo_inexistente.txt' para  
    String rutaArchivo = scanner.nextLine();  
    leerArchivo(rutaArchivo);  
  
    // 4. Prueba de Excepción Personalizada (EdadInvalidaException - Checked)  
    System.out.println("\n--- 4. Validación de Edad (Excepción Personalizada) ---");  
    System.out.print("Ingrese una edad válida (ej. 30): ");  
    int edadValida = scanner.nextInt();  
    System.out.print("Ingrese una edad inválida (ej. 150 o negativo): ");  
    int edadInvalida = scanner.nextInt();  
  
    try {  
        // Intenta el valor válido  
        validarEdad(edadValida);  
  
        // Intenta el valor inválido (aquí se lanzará la excepción)  
        validarEdad(edadInvalida);  
    } catch (EdadInvalidaException e) {  
        // Captura la excepción personalizada lanzada  
        System.out.println("*** Capturada excepción personalizada: " + e.getMessage());  
    }  
  
    scanner.close();  
}
```

Salida:

```
run:

--- 1. Validación de División por Cero ---
Ingrese el numerador: 21
Ingrese el divisor (pruebe con 0 para error, o 2 para éxito): 0
No se puede dividir por cero: / by zero

--- 2. Validación de Formato Numérico ---
Ingrese texto para convertir (pruebe '123' o 'texto'): "369"
Formato inválido para entero: "369"

--- 3. Validación de Acceso a Archivo (try-with-resources) ---
Ingrese la ruta de un archivo (pruebe 'archivo_inexistente.txt' para error): leer.txt
Intentando leer el archivo en: leer.txt
Error leyendo archivo: leer.txt (El sistema no puede encontrar el archivo especificado)

--- 4. Validación de Edad (Excepción Personalizada) ---
Ingrese una edad válida (ej. 30): 33
Ingrese una edad inválida (ej. 150 o negativo): -27
Edad válida: 33
*** Capturada excepción personalizada: Edad inválida fuera del rango (0-120): -27
BUILD SUCCESSFUL (total time: 48 seconds)
|
```

LINK:

<https://github.com/27mau/UTN-Programacion-2/tree/main>