Programación II - TP8: Interfaces y Excepciones

Alumno: *Romero, Abel Tomás (Comisión 5)*

Link del repo de GitHub:

https://github.com/Tomu98/UTN-TUPaD-P2-TPs/tree/main/08%20Interfaces%20y%20Excepciones

Objetivo General:

Desarrollar habilidades en el uso de Genéricos en Java para mejorar la seguridad, reutilización y escalabilidad del código. Comprender la implementación de clases, métodos e interfaces genéricas en estructuras de datos dinámicas. Aplicar comodines (?, extends, super) para gestionar diferentes tipos de datos en colecciones. Utilizar Comparable y Comparator para ordenar y buscar elementos de manera flexible. Integrar Genéricos en el diseño modular del software.

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 extiendan `Exception`.
finally y try-with-resources	Buenas prácticas para liberar recursos correctamente.
Uso de throw y throws	Declaración y lanzamiento de excepciones.

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.

Interfaz `Pagable`:

```java
public interface Pagable {

    public double calcularTotal();

}
```

Clase `Producto`:

```java
public class Producto implements Pagable {

    // Atributos
    private String nombre;
    private double precio;

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

    // Setters y Getters
    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;
    }

    // Método sobrescrito implementado de Pagable
    @Override
    public double calcularTotal() {
        return this.precio;
    }
```

Clase `Pedido`:

```java
package Interfaces;

import java.util.ArrayList;

public class Pedido implements Pagable {

    // Atributos
    ArrayList<Producto> productos;
    private String estado;
    private Cliente cliente;

    // Constructor
    public Pedido(String estado, Cliente cliente) {
        this.estado = estado;
        this.cliente = cliente;
        this.productos = new ArrayList();
    }

    // Método para añadir un producto al array
    public void agregarProducto(Producto producto) {
        productos.add(producto);
    }

    // Método sobrescrito implementado de Pagable
    @Override
    public double calcularTotal() {
        double total = 0;
        for (Producto p : productos) {
            total += p.getPrecio();
        }

        return total;
    }

    // Método para notificar el estado del pedido
    public void notificarEstado(String nuevoEstado) {
        this.estado = nuevoEstado;
        cliente.notificarCambioDeEstado(nuevoEstado);
    }
```

Interfaz `Pago`:

```java
public interface Pago {

    public void procesarPago(double monto);

}
```

Interfaz `PagoConDescuento`:

```java
public interface PagoConDescuento extends Pago {

    // Método para aplicar descuento
    public double aplicarDescuento(double monto);

}
```

Clase `TarjetaCredito`:

```java
public class TarjetaCredito implements Pago {

    // Método sobrescrito implementado de Pago
    @Override
    public void procesarPago(double monto) {
        System.out.println("\nInformación: ");
        System.out.println("- Total: " + monto);
        System.out.println("Pago realizado con exito.");
    }
```

Clase `PayPal`:

```java
public class PayPal implements PagoConDescuento {

    // Método sobrescrito implementado de PagoConDescuento
    @Override
    public double aplicarDescuento(double monto) {
        return monto - (monto * 0.21);
    }

    // Método sobrescrito implementado de PagoConDescuento <- Pago
    @Override
    public void procesarPago(double monto) {
        double total = aplicarDescuento(monto);
        System.out.println("\nInformación:");
        System.out.println("- Total con descuento: " + total);
        System.out.println("Pago realizado con exito");
    }
```

Interfaz `Notificable`:

```java
public interface Notificable {

    // Método para notificar un cambio de estado
    public void notificarCambioDeEstado(String nuevoEstado);

}
```

Clase `Cliente`:

```java
public class Cliente implements Notificable {

    // Atributo
    private String nombre;

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

    // Método sobrescrito implementado de Notificable
    @Override
    public void notificarCambioDeEstado(String nuevoEstado) {
        System.out.println("\nAVISO: " + nombre + " tu pedido ha cambiado de estado.
        System.out.println("- Nuevo estado: " + nuevoEstado);
    }
```

Clase `Main`:

```java
public class Main {

    public static void main(String[] args) {

        System.out.println("PARTE 1: Interfaces en un sistema de E-commerce.");

        // Instanciamos dos clientes
        Cliente cliente1 = new Cliente("Cliente 1");
        Cliente cliente2 = new Cliente("Cliente 2");

        // Cremaos un pedido para cada cliente
        Pedido pedido1 = new Pedido("Pendiente", cliente1);
        Pedido pedido2 = new Pedido("Pendiente", cliente2);

        // Creamos 3 productos
        Producto producto1 = new Producto("PS5", 1400000);
        Producto producto2 = new Producto("GPU", 2300000);
        Producto producto3 = new Producto("Monitor", 1100000);

        // Agregamos dos productos al pedido del cliente 1
        pedido1.agregarProducto(producto2);
        pedido1.agregarProducto(producto3);

        // Notificamos el estado del pedido al cliente 1
        pedido1.notificarEstado("EN_PROCESO");

        // Calculamos el total dle pedido del cliente 1
        double total = pedido1.calcularTotal();

        // Creamos un nuevo pago con PayPal
        PayPal pago = new PayPal();

        // Procesamos el pago del cliente 1
        pago.procesarPago(total);

        // Notificamos el estado del pedido pagado al cliente 1
        pedido1.notificarEstado("PAGADO");
```

```java

        // Agregamos un producto al pedido del cliente 2
        pedido2.agregarProducto(producto1);

        // Notificamos su estado en proceso al cliente 2
        pedido2.notificarEstado("EN_PROCESO");

        // Calculamos el total del pedido del cliente 2
        double total2 = pedido2.calcularTotal();

        // Creamos un pago con tarjeta
        TarjetaCredito pago2 = new TarjetaCredito();

        // Procesamos el pago con el monto del pedido del cliente 2
        pago2.procesarPago(total2);

        // Notificamos el estado del pedido pagado al cliente 2
        pedido2.notificarEstado("PAGADO");

        System.out.println("\nEjercicios finalizados.");
```

Output:

```
run:
PARTE 1: Interfaces en un sistema de E-commerce.

AVISO: Cliente 1 tu pedido ha cambiado de estado.
- Nuevo estado: EN_PROCESO

Informaci□n:
- Total con descuento: 2686000.0
Pago realizado con exito

AVISO: Cliente 1 tu pedido ha cambiado de estado.
- Nuevo estado: PAGADO

AVISO: Cliente 2 tu pedido ha cambiado de estado.
- Nuevo estado: EN_PROCESO

Informaci□n:
- Total: 1400000.0
Pago realizado con exito.

AVISO: Cliente 2 tu pedido ha cambiado de estado.
- Nuevo estado: PAGADO

Ejercicios finalizados.
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
public class DivisionSegura {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        double num1, num2;

        System.out.println("Ingrese dos numeros...");
        System.out.print("Primer numero: ");
        num1 = scan.nextDouble();

        System.out.print("Segundo numero: ");
        num2 = scan.nextDouble();

        try {
            if (num2 == 0) {
                throw new ArithmeticException("ERROR: No se puede dividir por cero.");
            }
            double resultado = num1 / num2;
            System.out.println("Resultado de la división: " + resultado);
        } catch (ArithmeticException ex) {
            System.out.println(ex.getMessage());
        }
```

Output:

```
run:
Ingrese dos numeros...
Primer numero: 12
Segundo numero: 0
ERROR: No se puede dividir por cero.
BUILD SUCCESSFUL (total time: 6 seconds)
```

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

```java
public class ConversionNumeroACadena {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese un texto que pueda ser convertido a entero.");
        System.out.print("Texto: ");
        String textoAConvertir = scan.nextLine();

        try {
            int textoConvertido = Integer.parseInt(textoAConvertir);
            System.out.println("Numero convertido: " + textoConvertido);
        } catch (NumberFormatException ex) {
            System.out.println("ERROR: El texto ingresado no es un entero valido.");
        }
```

Output:

```
run:
Ingrese un texto que pueda ser convertido a entero.
Texto: Profe apruebeme :)
ERROR: El texto ingresado no es un entero valido.
BUILD SUCCESSFUL (total time: 27 seconds)
```

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

```java
public class LecturaArchivo {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.println("Ingrese el nombre del archivo txt: ");
        String nombre = scan.nextLine();

        File archivo = new File(nombre);
        System.out.println("¿Existe?: " + archivo.exists());

        try {
            BufferedReader br = new BufferedReader(new FileReader(archivo));

            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }

            br.close();

        } catch (FileNotFoundException e) {
            System.out.println("ERROR: El archivo no fue encontrado.");
        } catch(IOException ex){
            System.out.println("ERROR de E/S: " + ex.getMessage());
        } finally {
            scan.close();
        }
```

Outputs:

```
run:
Ingrese el nombre del archivo txt:
archivo.txt
Existe?: false
ERROR: El archivo no fue encontrado.
BUILD SUCCESSFUL (total time: 3 seconds)
```

```
run:
Ingrese el nombre del archivo txt:
C:\Users\romer\Documents\NetBeansProjects\TP8\src\Excepciones\ArchivoParaLeer.txt
Existe?: true
Este es un archivo de prueba para el ejercicio 2.3 del tp8 sobre Excepciones :)
BUILD SUCCESSFUL (total time: 13 seconds)
```

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

```java
public class EdadInvalidaException extends RuntimeException {

    public EdadInvalidaException() {
    }

    public EdadInvalidaException(String message) {
        super(message);
    }

    public EdadInvalidaException(String message, Throwable cause) {
        super(message, cause);
    }

    public EdadInvalidaException(Throwable cause) {
        super(cause);
    }

}
```

```java
public class ExcepcionPersonalizada {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.print("Ingrese una edad: ");
        int edad = Integer.parseInt(scan.nextLine());

        if (edad <= 0 || edad >= 120) {
            throw new EdadInvalidaException("ERROR: Edad invalida.");
        } else {
            System.out.println("Edad ingresada: " + edad);
        }

    }

}
```

Output:

```
run:
Ingrese una edad: -1
Exception in thread "main" Excepciones.EdadInvalidaException: ERROR: Edad invalida.
        at Excepciones.ExcepcionPersonalizada.main(ExcepcionPersonalizada.java:15)
C:\Users\romer\AppData\Local\NetBeans\Cache\26\executor-snippets\run.xml:111: The following error occurred while execu
C:\Users\romer\AppData\Local\NetBeans\Cache\26\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 5 seconds)
```

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

```java
public class TryWithResources {

    public static void main(String[] args) {

        File archivo = new File("C:\\Users\\romer\\Documents\\NetBeansProjects\\TP8\\src\\Excepciones\\ArchivoParaLeer.txt");

        try(BufferedReader br = new BufferedReader(new FileReader(archivo))){
            System.out.println("Contenido del archivo:\n");
            System.out.println(br.readLine());
        } catch(IOException ex) {
            System.out.println("Error de E/S: "+ ex.getMessage());
        }
```

Output:

```
run:
Contenido del archivo:

Este es un archivo de prueba para el ejercicio 2.3 del tp8 sobre Excepciones :)
BUILD SUCCESSFUL (total time: 0 seconds)
```