TP 8: Interfaces y Excepciones en Java

Parte 1: Interfaces y E-commerce

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

// INTERFACES
interface Pagable {
   double calcularTotal();
}

interface Pago {
   void procesarPago(double monto);
}

interface PagoConDescuento extends Pago {
   double aplicarDescuento(double monto);
}

interface Notificable {
   void notificar(String mensaje);
}

// CLASES
class Producto implements Pagable {
   private String nombre;
```

```
    private double precio;

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

    public double getPrecio() {
        return precio;
    }

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

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

    public Pedido(Notificable cliente) {
        this.productos = new ArrayList<>();
        this.cliente = cliente;
        this.estado = "CREADO";
    }
```

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

    public void cambiarEstado(String nuevoEstado) {
        this.estado = nuevoEstado;
        if (cliente != null) {
            cliente.notificar("El pedido cambió de estado a: " + estado);
        }
    }

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

class Cliente implements Notificable {
    private String nombre;

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

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

// MEDIOS DE PAGO
class TarjetaCredito implements PagoConDescuento {
    @Override
    public void procesarPago(double monto) {
        System.out.println("Procesando pago con tarjeta de: $" + monto);
    }

    @Override
    public double aplicarDescuento(double monto) {
        double total = monto * 0.9; // 10% descuento
        System.out.println("Monto con descuento: $" + total);
        return total;
    }
}

class PayPal implements Pago {
    @Override
    public void procesarPago(double monto) {
        System.out.println("Procesando pago con PayPal de: $" + monto);
    }
}
```

Parte 2: Ejercicios de Excepciones

```
import java.io.*;
import java.util.Scanner;

// EXCEPCION PERSONALIZADA
class EdadInvalidaException extends Exception {
  public EdadInvalidaException(String mensaje) {
    super(mensaje);
  }
}

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

    // 1. División 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: División por cero no permitida.");
    }

    // 2. Conversión de cadena a número
```

```
sc.nextLine(); // limpiar buffer
try {
   System.out.print("Ingrese un número: ");
   String input = sc.nextLine();
   int numero = Integer.parseInt(input);
   System.out.println("Número ingresado: " + numero);
} catch (NumberFormatException e) {
   System.out.println("Error: Entrada inválida, no es un número entero.");
}

// 3. Lectura de archivo
try {
   BufferedReader br = new BufferedReader(new FileReader("archivo.txt"));
   String linea;
   while ((linea = br.readLine()) != null) {
      System.out.println(linea);
   }
   br.close();
} catch (FileNotFoundException e) {
   System.out.println("Error: Archivo no encontrado.");
} catch (IOException e) {
   System.out.println("Error al leer el archivo.");
}

// 4. Excepción personalizada
try {
   System.out.print("Ingrese edad: ");
   int edad = sc.nextInt();
```

```
            if (edad < 0 || edad > 120) {
                throw new EdadInvalidaException("Edad inválida: " + edad);
            } else {
                System.out.println("Edad válida: " + edad);
            }
        } catch (EdadInvalidaException e) {
            System.out.println("Excepción personalizada: " + e.getMessage());
        }

        // 5. Try-with-resources
        try (BufferedReader br = new BufferedReader(new FileReader("archivo.txt"))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            System.out.println("Error al leer el archivo con try-with-resources: " +
e.getMessage());
        }

        sc.close();
    }
}
```

Conclusiones

Las interfaces permiten definir contratos claros y reutilizables entre distintas clases (Pagable, Pago, Notificable).

Se puede lograr herencia múltiple de comportamiento implementando varias interfaces sin compartir estado.

La gestión de excepciones evita que el programa caiga ante errores inesperados (ArithmeticException, NumberFormatException, FileNotFoundException).

Las excepciones personalizadas (EdadInvalidaException) ayudan a validar reglas de negocio específicas.

El uso de try-with-resources y finally garantiza que los recursos se cierren correctamente, reforzando el diseño robusto y mantenible.