Resolución Práctico 4: Interface y Excepciones en java

Parte 1: En este ejercicio debemos crear una interfaz llamada Pagable con un método calcularTotal(). Luego, debemos crear una clase Producto con atributos nombre y precio, que implemente la interfaz Pagable. También debemos crear una clase Pedido, que contenga una lista de productos y que también implemente Pagable, calculando el total del pedido sumando el precio de todos los productos.

A continuación, se debe ampliar el diseño agregando dos interfaces: Pago, con el método procesarPago(double), y PagoConDescuento, que extiende Pago y agrega el método aplicarDescuento(double). Luego, se deben crear clases como TarjetaCredito y PayPal que implementen estas interfaces según corresponda.

Por último, se debe crear una interfaz Notificable con un método para notificar cambios de estado. La clase Cliente debe implementar esta interfaz, y la clase Pedido debe notificar al cliente cuando cambie su estado.

Interfaz Pagable:

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

Clase Producto:

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

    public double getPrecio() {
        return precio;
    }

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

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

}
```

Clase Pedido:

```java
package Interfaces;

import java.util.ArrayList;

public class Pedido implements Pagable { // Extendemos de pagable
    // Declaramos atributos
    ArrayList<Producto> productos;
    private String estado;
    private Cliente cliente;

    // Creamos el constructor inicializando el array
    public Pedido(String estado, Cliente cliente) {
        this.estado = estado;
        this.cliente = cliente;
        this.productos = new ArrayList();
    }

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

    // Sobreescribimos el metodo calcular total
    @Override
    public double calcularTotal() {
        double total = 0;
        for (Producto p : productos) {
            total += p.getPrecio();
        }

        return total;
    }

    // Metodo para notificar el estado del pedido al cliente
    public void notificarEstado(String nuevoestado) {
        this.estado = nuevoestado;
        cliente.notificarCamboDeEstado(nuevoestado);
    }

}
```

Interfaz Pago:

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

Interfaz pagoConDescuento:

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

Clase TarjetaDeCredito:

```
public class PagoConTarjeta implements Pago{

    @Override
    public void procesarPago(double monto) {
        System.out.println("El total es: " + monto);
        System.out.println("El pago se realizo con éxito!");
    }

}
```

Clase PayPal:

```
public class PagoConPayPal implements PagoConDescuento{

    @Override
    public double aplicarDescuento(double monto) {
        return monto - (monto * 0.15);
    }

    @Override
    public void procesarPago(double monto) {
        double total = aplicarDescuento(monto);
        System.out.println("El total con descuento es: " + total);
        System.out.println("El pago se realizo correctamente");
    }

}
```

Interfaz Notificable:

```
public interface Notificable {
    public void notificarCamboDeEstado(String nuevoEstado);
}
```

Clase Cliente:

```
public class Cliente implements Notificable{

    private String nombre;

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

    @Override
    public void notificarCamboDeEstado(String nuevoEstado) {
        System.out.println(nombre + " tu pedido cambio de estado a " + nuevoEstado);
    }

}
```

Main:

```
public static void main(String[] args) {
    // Creamos dos clientes
    Cliente c1 = new Cliente("Cliente Uno");
    Cliente c2 = new Cliente("Cliente Dos");

    // Creamos un pedido para cada cliente
    Pedido pedido = new Pedido("Pendiente", c1);
    Pedido pedido2 = new Pedido("Pendiente", c2);

    // Creamos 3 productos
    Producto p1 = new Producto("Heladera", 1000000);
    Producto p2 = new Producto("Televisor", 500000);
    Producto p3 = new Producto("Lavarropa", 750000);

    // Agregamos 2 productos al pedido del cliente 1
    pedido.agregarProducto(p1);
    pedido.agregarProducto(p3);

    // Notificamos el estado en proceso del pedido del cliente 1
    pedido.notificarEstado("EN_PROCESO");

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

    // Creamos un nuevo pago con pay pal
    PagoConPayPal pago = new PagoConPayPal();

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

    // Notificamos el estado del pedido pagado
    pedido.notificarEstado("PAGADO");

    // Agregamos un producto al pedido del cliente 2
    pedido2.agregarProducto(p2);
    // Notificamos su estado en proceso
    pedido2.notificarEstado("EN_PROCESO");

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

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

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

    // Notificamos al cliente 2 su estado de pedido pagado
    pedido2.notificarEstado("PAGADO");
}
```

Salida por pantalla:

```
Cliente Uno tu pedido cambio de estado a EN_PROCESO
El total con descuento es: 1487500.0
El pago se realizo correctamente
Cliente Uno tu pedido cambio de estado a PAGADO
Cliente Dos tu pedido cambio de estado a EN_PROCESO
El total es: 500000.0
El pago se realizo con éxito!
Cliente Dos tu pedido cambio de estado a PAGADO
```

Parte 2.1: En este ejercicio debemos crear una clase llamada DivisionSegura, que solicite al usuario dos números mediante la consola y realice la división entre ellos. Antes de efectuar la operación, se debe verificar si el segundo número es cero. En ese caso, se debe lanzar y capturar una excepción del tipo ArithmeticException para evitar que el programa finalice de forma abrupta. Si la división es válida, se debe mostrar el resultado por pantalla.

Codigo:

```java
public class DivisonSegura {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        double num1, num2;

        System.out.println("Ingrese dos numeros");
        System.out.println("Numero uno: ");
        num1 = scan.nextDouble();

        System.out.println("Numero dos:");
        num2 = scan.nextDouble();

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

Salida por pantalla:

```
Ingrese dos numeros
Numero uno:
12
Numero dos:
0
Error: No se puede dividir por 0
------------------------------------------------------------------------
BUILD SUCCESS
```

Parte 2.2: Este ejercicio solicita al usuario que ingrese un texto por consola e intenta convertirlo a un número entero utilizando Integer.parseInt(). Si el texto ingresado no representa un valor numérico válido, se lanza una excepción NumberFormatException, la cual es capturada para informar al usuario que la conversión no fue posible.

Codigo:

```
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");
        String textoAConvertir = scan.nextLine();

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

}
```

Salida por pantalla:

```
Ingrese un texto que pueda ser convertido a entero
hola
Error: El texto ingresado no es un número entero válido.
------------------------------------------------------------------------
BUILD SUCCESS
```

Parte 2.3: en este ejercicio debemos leer un archivo de texto cuyo nombre es ingresado por el usuario. Mostrar por pantalla la primera línea del archivo. Si el archivo no existe, capturar la excepción FileNotFoundException y mostrar un mensaje de error informativo.
Codigo:

```
public class LecturaArchivo {

    public static void main(String[] args) throws IOException {
        Scanner scan = new Scanner(System.in);

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

        try {
            File archivo = new File(nombre);
            BufferedReader br = new BufferedReader(new FileReader(archivo));
            System.out.println(br.readLine());
        } catch (FileNotFoundException e) {
            System.out.println("Error: El archivo no fue encontrado.");
        } catch(IOException ex){
            System.out.println("Error de E/S");
            System.out.println(ex.getMessage());
        }
    }
```

Salida por pantalla:

```
Ingrese el nombre del archivo txt:
archivo.txt
Error: El archivo no fue encontrado.
------------------------------------------------------------------------
BUILD SUCCESS
```

Salida por pantalla con lectura exitosa (debe estar la ruta al archivo completa):

```
Ingrese el nombre del archivo txt:
C:\Users\Lenovo\OneDrive\Documentos\NetBeansProjects\EjercitacionTpOcho\src\main\java\Excepciones\ArchivoALeer.txt
Â¡Hola, Java!
------------------------------------------------------------------------
BUILD SUCCESS
```

Parte 2.4: En este ejercicio se debe crear una excepción personalizada llamada EdadInvalidaException, la cual se lanzará si el usuario ingresa una edad menor a 0 o mayor a 120. La excepción será capturada para mostrar un mensaje de error adecuado.

Excepcion personalizada:

```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);
    }

}
```

Main:

```java
public class ExcepcionPersonalizada {

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

        System.out.println("Ingrese una edad");
        int edad = Integer.parseInt(scan.nextLine());
        if (edad <= 0 || edad >= 120) {
            throw new EdadInvalidaException("Ingreso una edad invalida");
        }

    }
```

Salida por pantalla:

```
 Ingrese una edad
 124
] Exception in thread "main" Excepciones.EdadInvalidaException: Ingreso una edad invalida
        at Excepciones.ExcepcionPersonalizada.main(ExcepcionPersonalizada.java:13)
 Command execution failed.
] org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit value: 1)
     at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor.java:404)
     at org.apache.commons.exec.DefaultExecutor.execute (DefaultExecutor.java:166)
     at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:982)
     at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:929)
     at org.codehaus.mojo.exec.ExecMojo.execute (ExecMojo.java:457)
```

Parte 2.5:

En este ejercicio se debe leer un archivo de texto ubicado en una ruta específica. Se utiliza un bloque try-with-resources para manejar la lectura y asegurar el cierre automático del recurso. Se captura la excepción IOException para mostrar un mensaje de error en caso de que ocurra un problema de entrada/salida, como que el archivo no exista o no se pueda abrir.

Codigo:

```
public class TryWithResources {

    public static void main(String[] args) {
        File archivo = new File("C:\\Users\\Lenovo\\OneDrive\\Documentos\\NetBeansProjects\\EjercitacionTpOcho\\src\\main\\java\\Excepciones\\ArchivoALeer.txt");
        try(BufferedReader br = new BufferedReader(new FileReader(archivo))){
            System.out.println(br.readLine());
        } catch(IOException ex) {
            System.out.println("Error de E/S: "+ ex.getMessage());
        }
    }

}
```

Salida por pantalla:

```
------------------< com.mycompany:EjercitacionTpOcho >------------------
Building EjercitacionTpOcho 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ EjercitacionTpOcho ---
Â¡Hola, Java!
------------------------------------------------------------------------
BUILD SUCCESS
```