PROGRAMACIÓN II

TP 8: Interfaces y Excepciones en Java

OBJETIVO GENERAL

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.

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

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

MARCO TEÓRICO

Caso Practico

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.

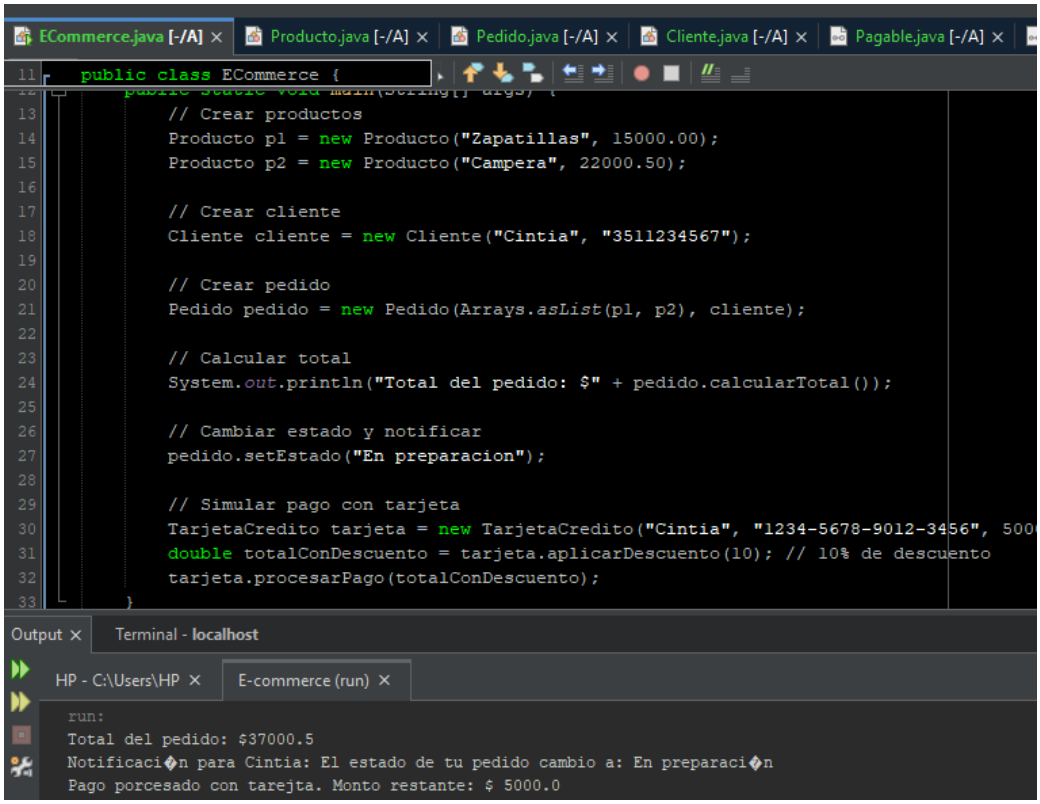
4. Ampliar con interfaces Pago y PagoConDescuento para distintos medios de

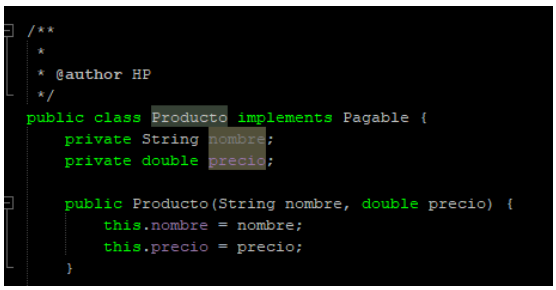
pago (TarjetaCredito, PayPal), con métodos procesarPago(double) y

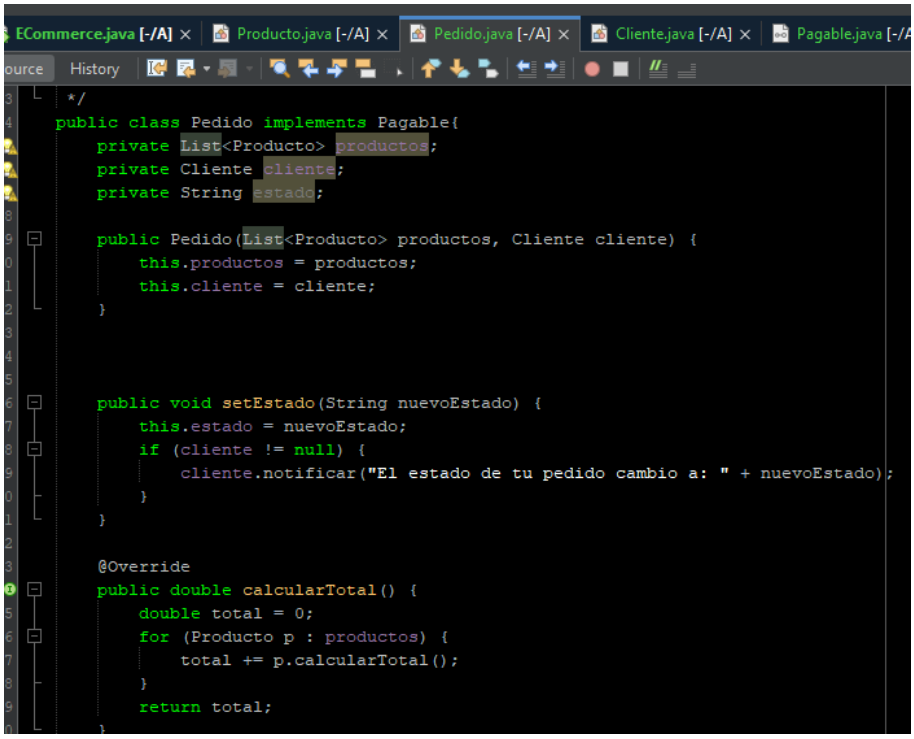
aplicarDescuento(double).

5. Crear una interfaz Notificable para notificar cambios de estado. La clase

Cliente implementa dicha interfaz y Pedido debe notificarlo al cambiar de estado.








```java
*/
public class Pedido implements Pagable{
    private List<Producto> productos;
    private Cliente cliente;
    private String estado;

    public Pedido(List<Producto> productos, Cliente cliente) {
        this.productos = productos;
        this.cliente = cliente;
    }



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

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

```java
/**
 *
 * @author HP
 */
public class Cliente implements Notificable{
    private String nombre;
    private String telefono;

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



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

```java
/**
 *
 * @author HP
 */
public interface Pagable {
    double calcularTotal();

}
```

```java
/**
 *
 * @author HP
 */
public interface PagoConDesuento extends Pago{
    double aplicarDescuento(double porcentaje);
```

```java
/**
 *
 * @author HP
 */
public interface Pago {
    void procesarPago (double monto);

}
```

```java
public class TarjetaCredito implements PagoConDescuento{
    private String titular;
    private String numero;
    private double montoDisponible;

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

    @Override
    public void procesarPago(double monto) {
        if (monto <=montoDisponible){
            montoDisponible -= monto;
            System.out.println("Pago porcesado con tarejta. Monto restante: $ " + montoDisponible);

        }else{
            System.out.println("Fondos insuficientes");
        }
    }

    @Override
    public double aplicarDescuento(double porcentaje) {
        // lógica para calcular descuento
        return montoDisponible * (1 - porcentaje / 100);
    }
```

```java
6
7   /**
8    *
9    * @author HP
10   */
11  public class Paypal implements Pago{
        private String telefono;
        private double saldo;
14
15      public Paypal(String telefono, double saldo) {
16          this.telefono = telefono;
17          this.saldo = saldo;
18      }
19
20
21
22
23      @Override
25      public void procesarPago(double monto) {
26          // lógica para descontar del saldo
        }
```

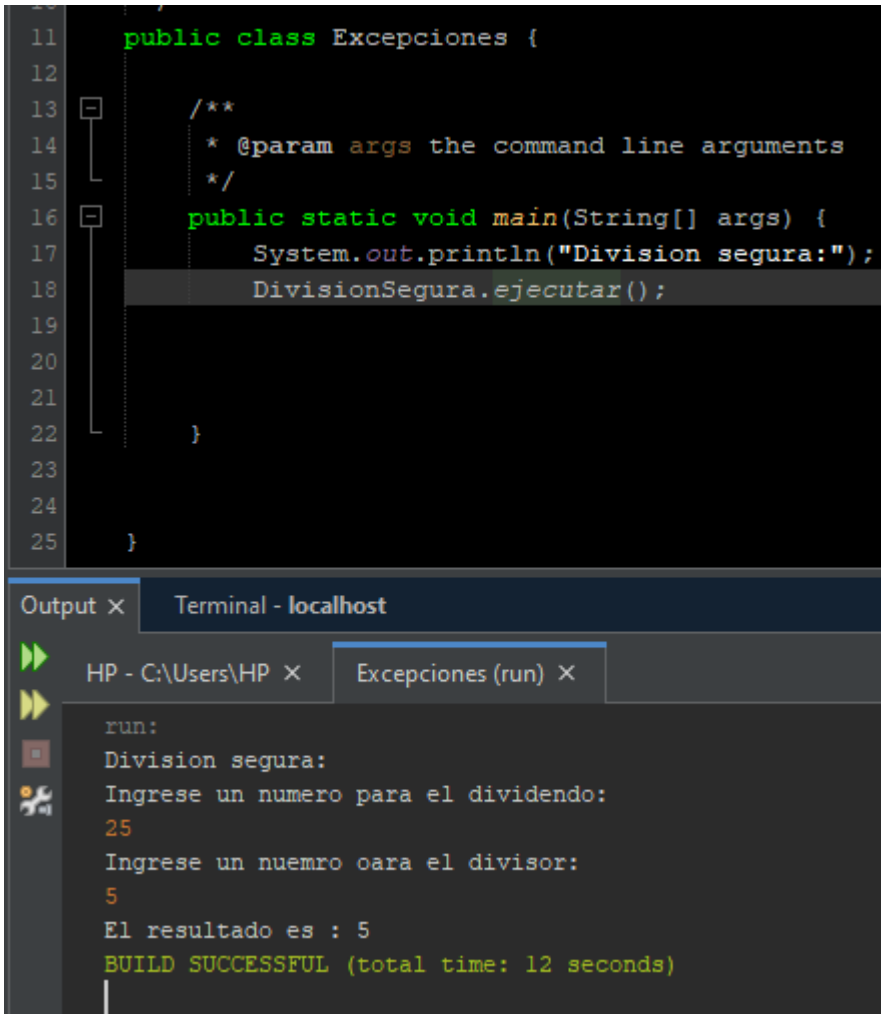
```java
 * @author HP
 */
public interface Notificable {
    void notificar(String mensaje);

}
```

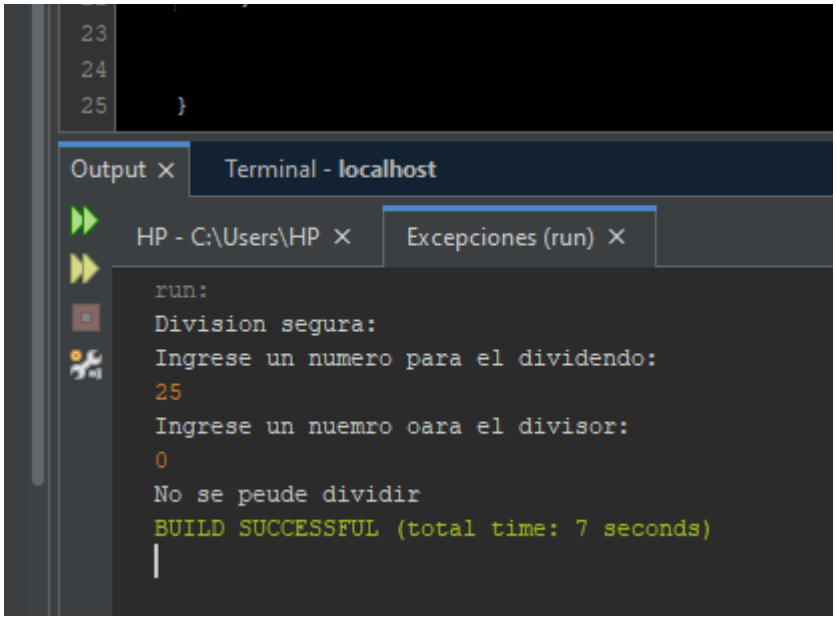
Parte 2: Ejercicios sobre Excepciones

1. División segura

- Solicitar dos números y dividirlos. Manejar ArithmeticException si el

divisor es cero.

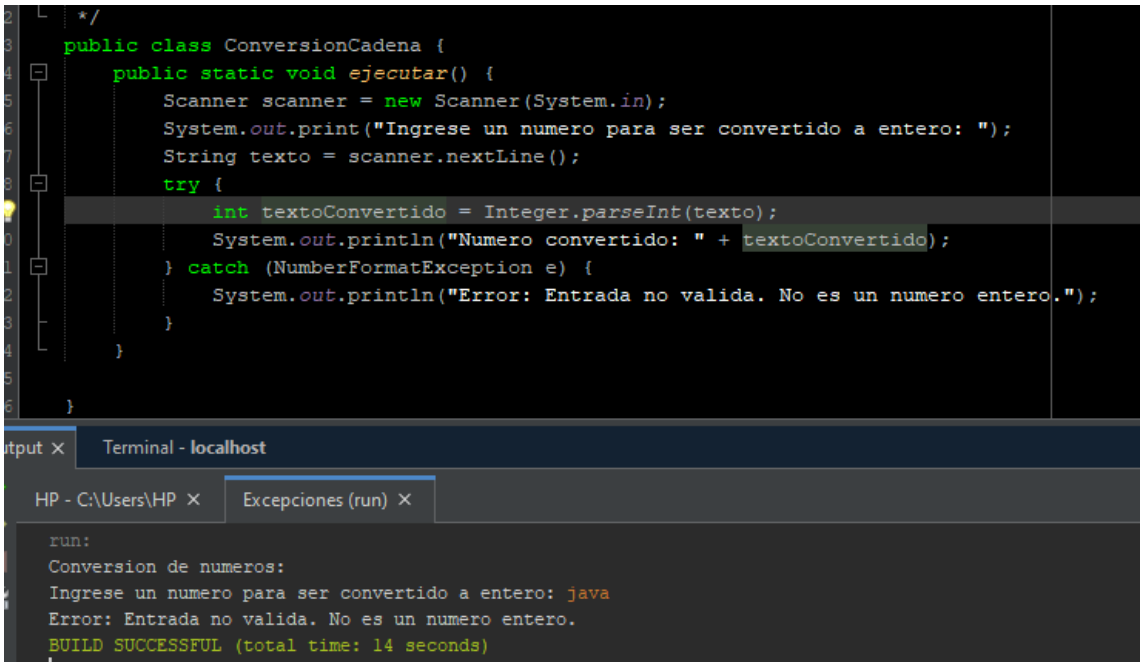




2. Conversión de cadena a número

- Leer texto del usuario e intentar convertirlo a int. Manejar

NumberFormatException si no es válido.



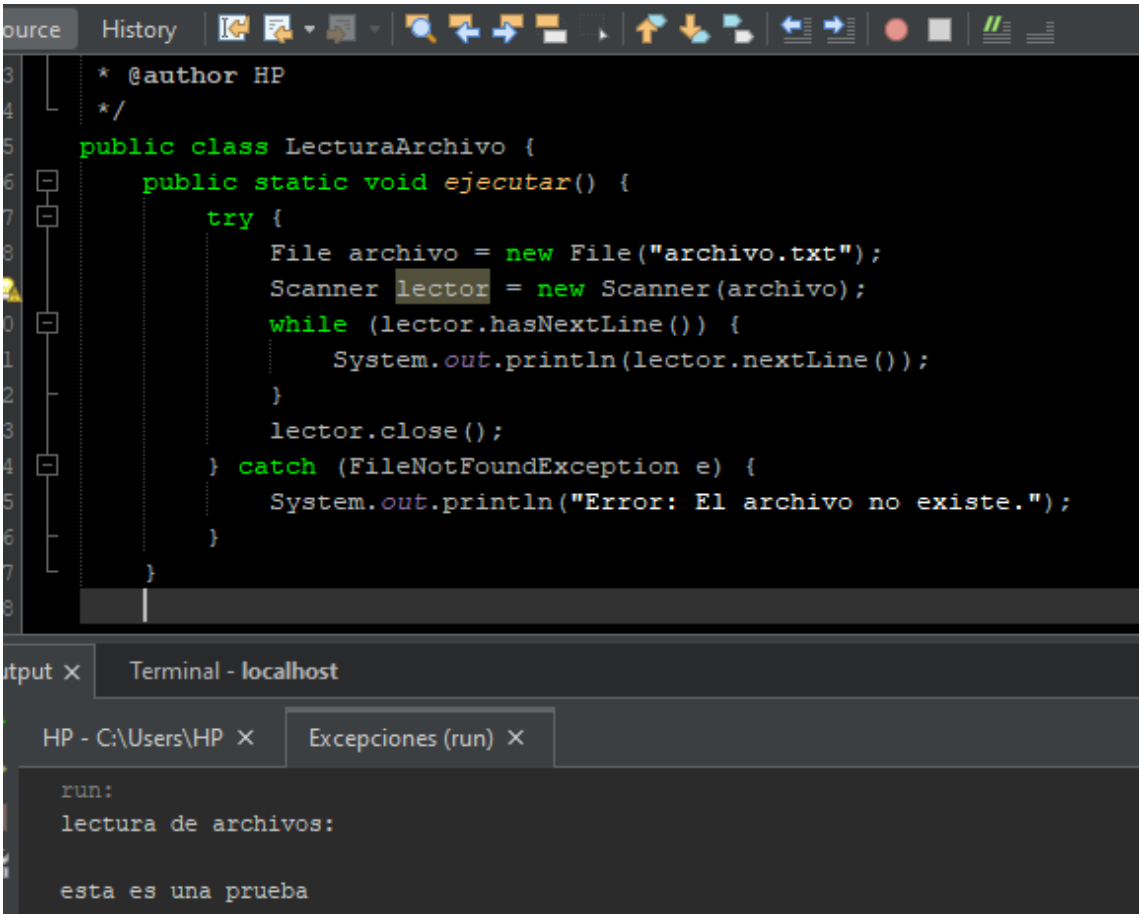
```
*/
public class ConversionCadena {
    public static void ejecutar() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese un numero para ser convertido a entero: ");
        String texto = scanner.nextLine();
        try {
            int textoConvertido = Integer.parseInt(texto);
            System.out.println("Numero convertido: " + textoConvertido);
        } catch (NumberFormatException e) {
            System.out.println("Error: Entrada no valida. No es un numero entero.");
        }
    }

}
```

```
utput ×  Terminal - localhost
HP - C:\Users\HP ×  Excepciones (run) ×
run:
Conversion de numeros:
Ingrese un numero para ser convertido a entero: java
Error: Entrada no valida. No es un numero entero.
BUILD SUCCESSFUL (total time: 14 seconds)
```

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.

```java
/**
 *
 * @author HP
 */
public class EdadInvalidaException extends Exception{
    public EdadInvalidaException (String mensaje){
        super(mensaje);
    }
```

```
Output ×   Terminal - localhost
HP - C:\Users\HP ×   Excepciones (run) ×
run:
Edad Invalida:
Ingrese su edad: 130
Error: Edad invalida: debe estar entre 0 y 120.
BUILD SUCCESSFUL (total time: 7 seconds)
```

5. Uso de try-with-resources

- Leer un archivo con BufferedReader usando try-with-resources.p

Manejar IOException correctamente.

```java
 */
public class LecturaSegura {
    public static void ejecutar() {
        try (BufferedReader lector = new BufferedReader(new FileReader("archivo.txt"))) {
            String linea;
            while ((linea = lector.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
```

```
utput ×   Terminal - localhost
HP - C:\Users\HP ×   Excepciones (run) ×
run:

esta es una prueba
BUILD SUCCESSFUL (total time: 0 seconds)
```