

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

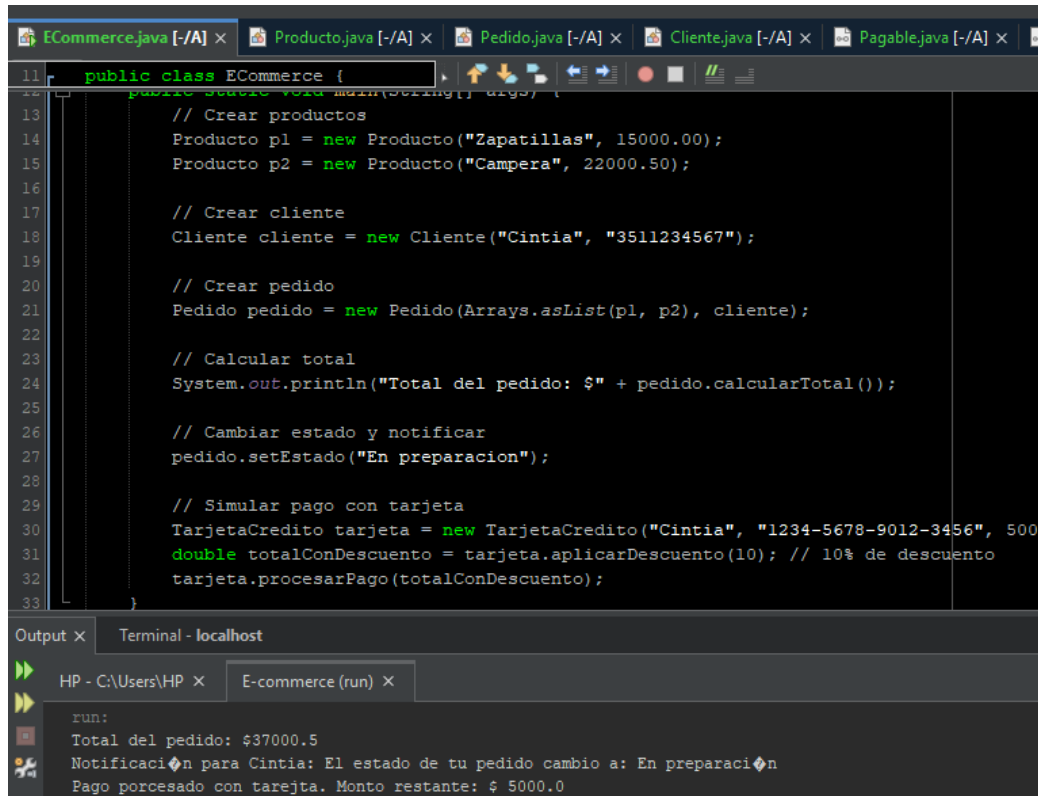
MARCO TEÓRICO

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.

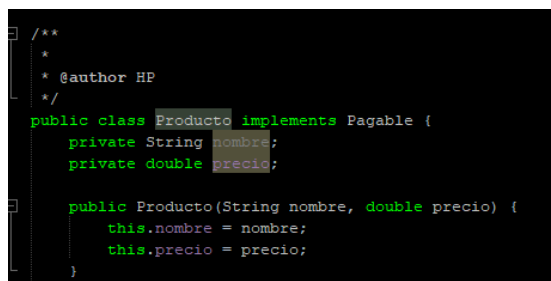


The screenshot shows an IDE with several tabs: ECommerce.java, Producto.java, Pedido.java, Cliente.java, and Pagable.java. The ECommerce.java file is active, showing the following code:

```
11 public class ECommerce {
12     public static void main(String[] args) {
13         // Crear productos
14         Producto p1 = new Producto("Zapatillas", 15000.00);
15         Producto p2 = new Producto("Campera", 22000.50);
16
17         // Crear cliente
18         Cliente cliente = new Cliente("Cintia", "3511234567");
19
20         // Crear pedido
21         Pedido pedido = new Pedido(Arrays.asList(p1, p2), cliente);
22
23         // Calcular total
24         System.out.println("Total del pedido: $" + pedido.calcularTotal());
25
26         // Cambiar estado y notificar
27         pedido.setEstado("En preparacion");
28
29         // Simular pago con tarjeta
30         TarjetaCredito tarjeta = new TarjetaCredito("Cintia", "1234-5678-9012-3456", 5000);
31         double totalConDescuento = tarjeta.aplicarDescuento(10); // 10% de descuento
32         tarjeta.procesarPago(totalConDescuento);
33     }
34 }
```

The Output window shows the following text:

```
run:
Total del pedido: $37000.5
Notificación para Cintia: El estado de tu pedido cambio a: En preparacion
Pago procesado con tarjeta. Monto restante: $ 5000.0
```



The screenshot shows the Producto.java file with the following code:

```
/**
 *
 * @author HP
 */
public class Producto implements Pagable {
    private String nombre;
    private double precio;

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

```
ECommerce.java [-/A] x Producto.java [-/A] x Pedido.java [-/A] x Cliente.java [-/A] x Pagable.java [-/A]
source History
3  */
4  public class Pedido implements Pagable{
5      private List<Producto> productos;
6      private Cliente cliente;
7      private String estado;
8
9      public Pedido(List<Producto> productos, Cliente cliente) {
10         this.productos = productos;
11         this.cliente = cliente;
12     }
13
14
15
16     public void setEstado(String nuevoEstado) {
17         this.estado = nuevoEstado;
18         if (cliente != null) {
19             cliente.notificar("El estado de tu pedido cambio a: " + nuevoEstado);
20         }
21     }
22
23     @Override
24     public double calcularTotal() {
25         double total = 0;
26         for (Producto p : productos) {
27             total += p.calcularTotal();
28         }
29         return total;
30     }
31 }
```

```
/**
 *
 * @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);
    }
}
```

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

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

```

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

```

```

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 procesado con tarjeta. Monto restante: $ " + montoDisponible);
        }else{
            System.out.println("Fondos insuficientes");
        }
    }

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

```

```

6
7 /**
8  *
9  * @author HP
10 */
11 public class Paypal implements Pago{
12     private String telefono;
13     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
24     public void procesarPago(double monto) {
25         // lógica para descontar del saldo
26     }
27

```

```

* @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.

```
11 public class Excepciones {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         System.out.println("Division segura:");
18         DivisionSegura.ejecutar();
19
20
21
22     }
23
24
25 }
```

Output × Terminal - localhost

HP - C:\Users\HP × Excepciones (run) ×

run:
Division segura:
Ingrese un numero para el dividendo:
25
Ingrese un nuemro oara el divisor:
5
El resultado es : 5
BUILD SUCCESSFUL (total time: 12 seconds)

```
23
24
25 }
```

Output × Terminal - localhost

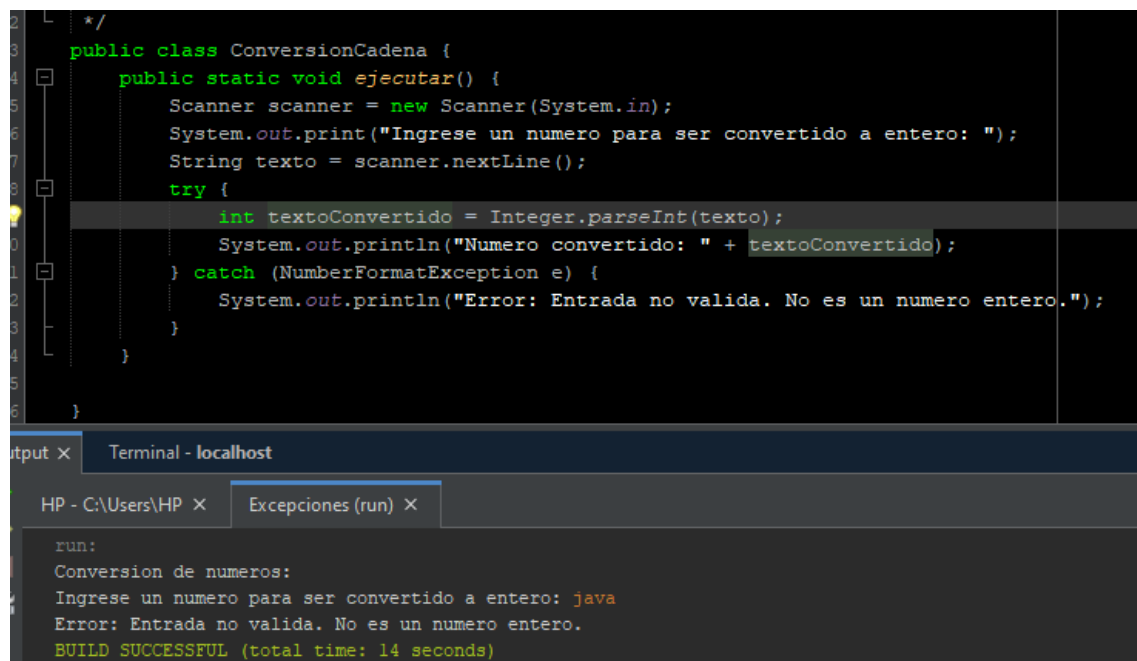
HP - C:\Users\HP × Excepciones (run) ×

run:
Division segura:
Ingrese un numero para el dividendo:
25
Ingrese un nuemro oara el divisor:
0
No se puede dividir
BUILD SUCCESSFUL (total time: 7 seconds)

2. Conversión de cadena a número

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

NumberFormatException si no es válido.



The screenshot shows an IDE with a Java class named `ConversionCadena`. The `ejecutar()` method uses a `Scanner` to read input and a `try-catch` block to handle `NumberFormatException` when parsing the input to an integer. The output window shows the program running, prompting for input, receiving the word "java", and displaying an error message.

```
2  */
3  public class ConversionCadena {
4      public static void ejecutar() {
5          Scanner scanner = new Scanner(System.in);
6          System.out.print("Ingrese un numero para ser convertido a entero: ");
7          String texto = scanner.nextLine();
8          try {
9              int textoConvertido = Integer.parseInt(texto);
10             System.out.println("Numero convertido: " + textoConvertido);
11         } catch (NumberFormatException e) {
12             System.out.println("Error: Entrada no valida. No es un numero entero.");
13         }
14     }
15 }
```

Output x Terminal - localhost

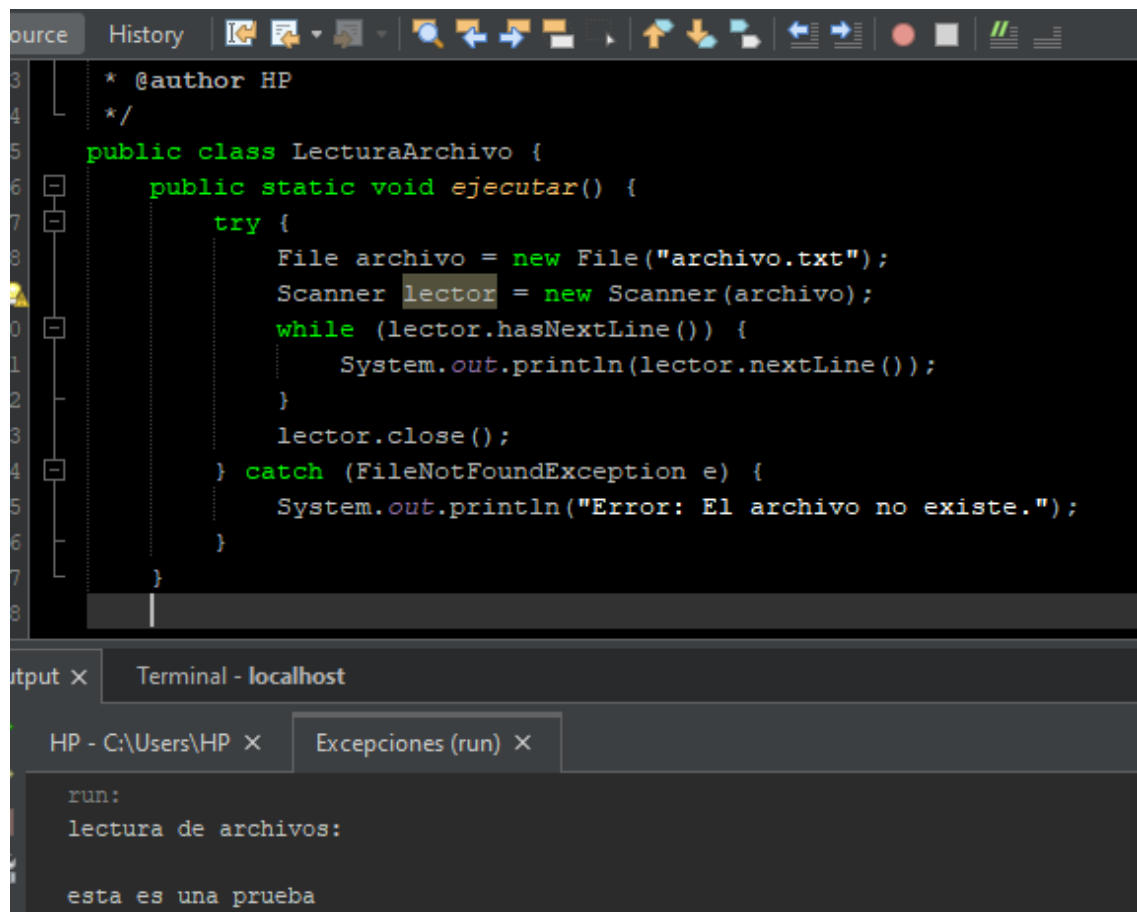
HP - C:\Users\HP x Excepciones (run) x

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.



The screenshot shows an IDE with a Java source file. The code defines a class `LecturaArchivo` with a static method `ejecutar()`. This method attempts to read a file named `archivo.txt` using a `Scanner`. It uses a `try-catch` block to handle `FileNotFoundException`. The output window shows the execution results.

```
3  * @author HP
4  */
5  public class LecturaArchivo {
6      public static void ejecutar() {
7          try {
8              File archivo = new File("archivo.txt");
9              Scanner lector = new Scanner(archivo);
10             while (lector.hasNextLine()) {
11                 System.out.println(lector.nextLine());
12             }
13             lector.close();
14         } catch (FileNotFoundException e) {
15             System.out.println("Error: El archivo no existe.");
16         }
17     }
18 }
```

Output × Terminal - localhost

HP - C:\Users\HP × Excepciones (run) ×

```
run:
lectura de archivos:
esta es una prueba
```

4. Excepción personalizada

- Crear `EdadInvalidaException`. Lanzarla si la edad es menor a 0 o mayor

a 120. Capturarla y mostrar mensaje.


```
6
7 /**
8  *
9  * @author HP
10  */
11 public class EdadInvalidaException extends Exception{
12     public EdadInvalidaException (String mensaje){
13         super(mensaje);
14     }
15
16

```

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`.

Manejar `IOException` correctamente.

```
4 /**
5  *
6  */
7 public class LecturaSegura {
8     public static void ejecutar() {
9         try (BufferedReader lector = new BufferedReader(new FileReader("archivo.txt"))) {
10             String linea;
11             while ((linea = lector.readLine()) != null) {
12                 System.out.println(linea);
13             }
14         } catch (IOException e) {
15

```

Output × Terminal - localhost

HP - C:\Users\HP × Excepciones (run) ×

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