

# 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.

**Alumno:**

Franco Sarrú

**Link público de GitHub:**

<https://github.com/fsarru/Programacion2.git>

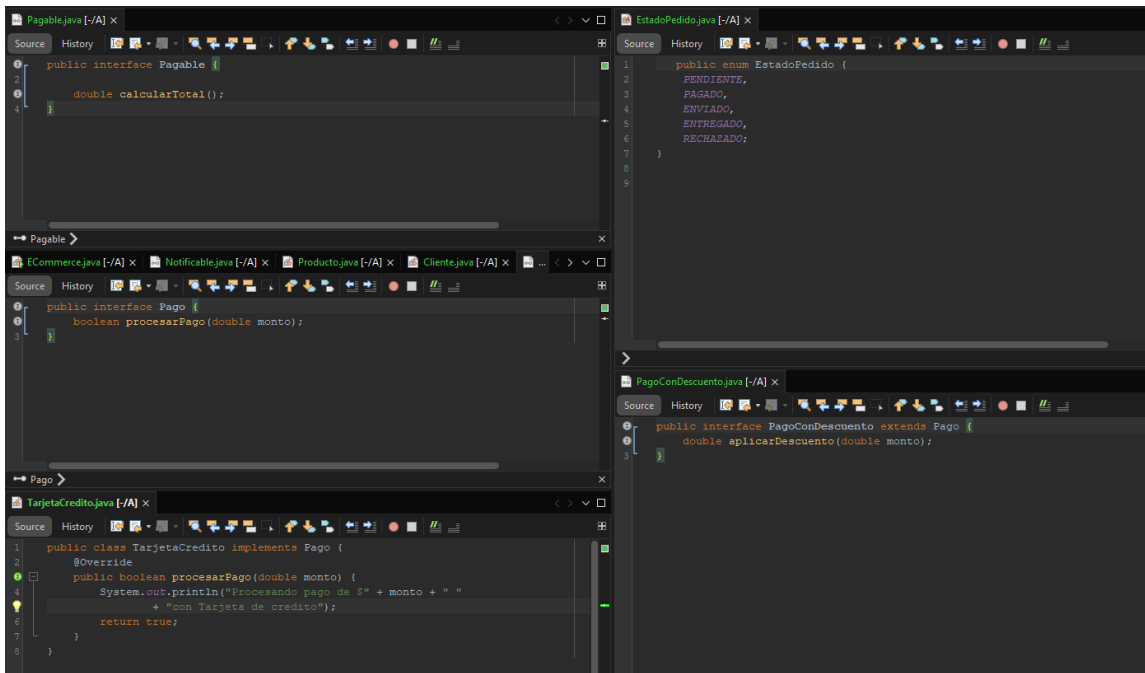
Concepto	Aplicación en el proyecto
Interfaces	

	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 <b>implements</b> para que una clase cumpla con los métodos definidos en una interfaz
Excepciones	Manejo de errores en tiempo de ejecución mediante estructuras <b>try-catch</b>
Excepciones checked y unchecked	Diferencias y usos según la naturaleza del error
Excepciones personalizadas	Creación de nuevas clases que extienden <b>Exception</b>
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

## 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.



```
public interface Pagable {  
    double calcularTotal();  
}  
  
public interface Pago {  
    boolean procesarPago(double monto);  
}  
  
public interface PagoConDescuento extends Pago {  
    double aplicarDescuento(double monto);  
}  
  
public enum EstadoPedido {  
    PENDIENTE,  
    PAGADO,  
    ENVIADO,  
    ENTREGADO,  
    RECHAZADO;  
}  
  
public class TarjetaCredito implements Pago {  
    @Override  
    public boolean procesarPago(double monto) {  
        System.out.println("Procesando pago de $" + monto + " "  
            + "con Tarjeta de credito");  
        return true;  
    }  
}
```

```

ECommerce.java [-/A] x  Notificable.java [-/A] x  Producto.java [-/A] x  Pedido.java [-/A] x  PayPal.java [-/A] x
Source History
1 public class Producto implements Pagable {
2     private String nombre;
3     private double precio;
4
5     public Producto(String nombre, double precio) {
6         this.nombre = nombre;
7         this.precio = precio;
8     }
9
10    public String getNombre() { return nombre; }
11    public double getPrecio() { return precio; }
12
13    @Override
14    public double calcularTotal() {
15        return this.precio;
16    }
17 }

Producto ▶ calcularTotal ▶
Cliente.java [-/A] x
Source History
1 public class Cliente implements Notificable {
2     private String nombre;
3
4     public Cliente(String nombre) {
5         this.nombre = nombre;
6     }
7
8     public String getNombre() { return nombre; }
9
10    @Override
11    public void recibirNotificacion(String mensaje) {
12        System.out.println("Cliente " + nombre + " Notificado: " + mensaje);
13    }
14 }

Pago.java [-/A] x
Source History
1 public interface Pago {
2     boolean procesarPago(double monto);
3 }
    
```

```

ECommerce.java [-/A] x  Pedido.java [-/A] x  Notificable.java [-/A] x
Source History
1 public class ECommerce {
2
3     public static void main(String[] args) {
4
5         Cliente ana = new Cliente("Ana Lopez");
6         Pedido pedidoAna = new Pedido(ana);
7
8         Producto libro = new Producto("Libro de Java", 50.0);
9         Producto teclado = new Producto("Teclado Mecanico", 120.0);
10
11         pedidoAna.agregarProducto(libro);
12         pedidoAna.agregarProducto(teclado);
13
14         System.out.println("Total a Pagar antes de descuentos: $" +
15             pedidoAna.calcularTotal());
16
17         pedidoAna.setMetodoPago(new PayPal());
18         pedidoAna.realizarPago();
19
20         pedidoAna.cambiarEstado(EstadoPedido.ENVIADO);
21         pedidoAna.cambiarEstado(EstadoPedido.ENTREGADO);
22
23         Cliente luis = new Cliente("Luis Gomez");
24         Pedido pedidoLuis = new Pedido(luis);
25         pedidoLuis.agregarProducto(new Producto("Pelota", 300.0));
26
27         pedidoLuis.setMetodoPago(new TarjetaCredito());
28         pedidoLuis.realizarPago();
29         pedidoLuis.cambiarEstado(EstadoPedido.RECHAZADO);
30
31     }
32 }

Notificable.java [-/A] x
Source History
1 public interface Notificable {
2
3     void recibirNotificacion(String mensaje);
4 }
    
```

```
Pedido.java [-/A] x
Source History
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Pedido implements Pagable {
5     private List<Producto> productos;
6     private Cliente cliente;
7     private EstadoPedido estado;
8     private Pago metodoPago;
9
10    public Pedido(Cliente cliente) {
11        this.cliente = cliente;
12        this.productos = new ArrayList<>();
13        this.estado = EstadoPedido.PENDIENTE;
14    }
15
16    public void agregarProducto(Producto p) {
17        this.productos.add(p);
18    }
19
20    public void setMetodoPago(Pago metodoPago) {
21        this.metodoPago = metodoPago;
22    }
23
24    public void cambiarEstado(EstadoPedido nuevoEstado) {
25        this.estado = nuevoEstado;
26        String mensaje = String.format("El estado de su pedido ha cambiado a: " + nuevoEstado.name());
27        cliente.recibirNotificacion(mensaje);
28    }
29 }
```

```
29
30 @Override
31 public double calcularTotal() {
32     double total = 0;
33     for (Producto p : productos) {
34         total += p.calcularTotal();
35     }
36     return total;
37 }
38
39 public boolean realizarPago() {
40     if (metodoPago == null) {
41         System.out.println("ERROR: No se ha seleccionado un método de pago.");
42         return false;
43     }
44
45     double totalAPagar = calcularTotal();
46     System.out.println("Total del Pedido: $" + totalAPagar);
47     boolean pagoExitoso = metodoPago.procesarPago(totalAPagar);
48     if (pagoExitoso) {
49         cambiarEstado(EstadoPedido.PAGADO);
50         return true;
51     } else {
52         cambiarEstado(EstadoPedido.RECHAZADO);
53         return false;
54     }
55 }
56 }
```

```
Output x
Programacion2 - C:\Users\franc\Documents\Git\Programacion2 x E-Commerce (run) x
run:
Total a Pagar antes de descuentos: $170.0
Total del Pedido: $170.0
Descuento por pago con PayPal aplicado. Nuevo total: 161.5Procesando pago final de $161.5 con PayPal
Cliente Ana Lopez Notificado: El estado de su pedido ha cambiado a: PAGADO
Cliente Ana Lopez Notificado: El estado de su pedido ha cambiado a: ENVIADO
Cliente Ana Lopez Notificado: El estado de su pedido ha cambiado a: ENTREGADO
Total del Pedido: $300.0
Procesando pago de $300.0 con Tarjeta de credito
Cliente Luis Gomez Notificado: El estado de su pedido ha cambiado a: PAGADO
Cliente Luis Gomez Notificado: El estado de su pedido ha cambiado a: RECHAZADO
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.

### 2. Conversión de cadena a número

- Leer texto del usuario e intentar convertirlo a **int**. Manejar **NumberFormatException** si no es válido.

### 3. Lectura de archivo

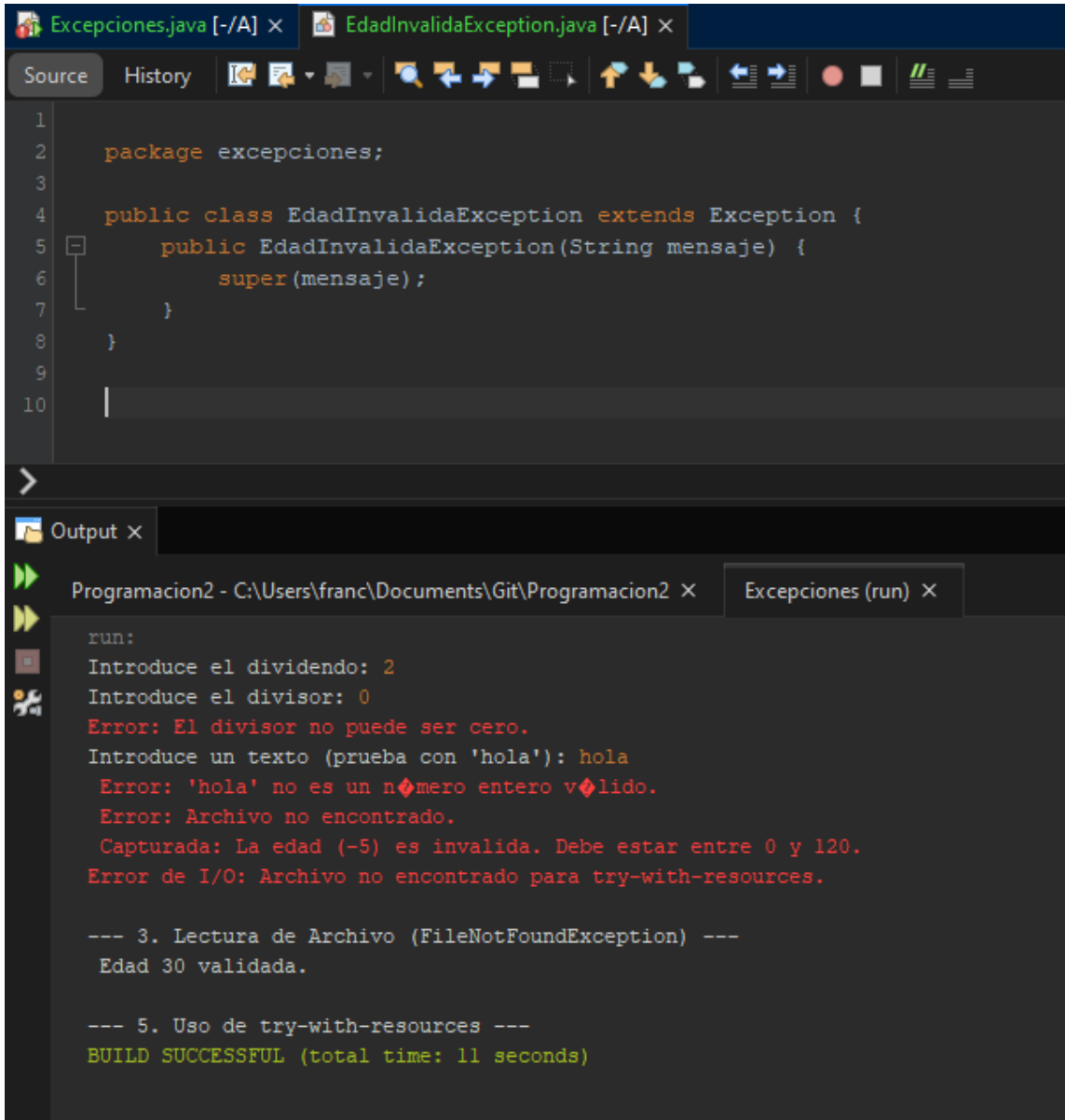
- Leer un archivo de texto y mostrarlo. Manejar **FileNotFoundException** si el archivo no existe.

### 4. Excepción personalizada

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

### 5. Uso de try-with-resources

- Leer un archivo con **BufferedReader** usando **try-with-resources**. Manejar **IOException** correctamente.



The screenshot shows an IDE with two tabs: 'Excepciones.java [-/A] x' and 'EdadInvalidaException.java [-/A] x'. The 'Source' tab is active, displaying the following Java code:

```
1
2 package excepciones;
3
4 public class EdadInvalidaException extends Exception {
5     public EdadInvalidaException(String mensaje) {
6         super(mensaje);
7     }
8 }
9
10
```

Below the code editor is the 'Output' tab, which shows the execution results of a program named 'Programacion2'. The output includes user input, error messages, and a successful build status.

```
run:
Introduce el dividendo: 2
Introduce el divisor: 0
Error: El divisor no puede ser cero.
Introduce un texto (prueba con 'hola'): hola
Error: 'hola' no es un número entero válido.
Error: Archivo no encontrado.
Capturada: La edad (-5) es invalida. Debe estar entre 0 y 120.
Error de I/O: Archivo no encontrado para try-with-resources.

--- 3. Lectura de Archivo (FileNotFoundException) ---
Edad 30 validada.

--- 5. Uso de try-with-resources ---
BUILD SUCCESSFUL (total time: 11 seconds)
```

```
Excepciones.java [-/A] x EdadInvalidaException.java [-/A] x
Source History
1 package excepciones;
2
3
4
5 import java.io.BufferedReader;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.util.InputMismatchException;
9 import java.util.Scanner;
10
11
12 public class Excepciones {
13
14
15     private static final Scanner scanner = new Scanner(System.in);
16     private static final String NOMBRE_ARCHIVO = "datos.txt";
17
18
19
20     public static void verificarEdad(int edad) throws EdadInvalidaException {
21         if (edad < 0 || edad > 120) {
22             throw new EdadInvalidaException("La edad (" + edad + ") es invalida. Debe estar entre 0 y 120.");
23         }
24         System.out.println("Edad " + edad + " validada.");
25     }
26
27
28     public static void main(String[] args) {
29
30         // 1. División segura (ArithmeticException)
31         try {
32             System.out.print("Introduce el dividendo: ");
33             double num1 = scanner.nextDouble();
34
35             System.out.print("Introduce el divisor: ");
36             double num2 = scanner.nextDouble();
37
38             if (num2 == 0) {
39                 throw new ArithmeticException("El divisor no puede ser cero.");
40             }
41         }
42     }
43 }
```



```
41
42     double resultado = num1 / num2;
43     System.out.printf("Resultado: " + resultado);
44
45 } catch (ArithmeticException e) {
46     System.err.println("Error: " + e.getMessage());
47 } catch (InputMismatchException e) {
48     System.err.println(" Error: Se esperaba un número.");
49 }
50
51
52 // 2. Conversión de cadena a número (NumberFormatException)
53 System.out.print("Introduce un texto (prueba con 'hola'): ");
54 if (scanner.hasNextLine()) scanner.nextLine();
55 String texto = scanner.nextLine();
56
57 try {
58     int numero = Integer.parseInt(texto);
59     System.out.println("Conversión exitosa. Número: " + numero);
60 } catch (NumberFormatException e) {
61     System.err.println(" Error: '" + texto + "' no es un número entero válido.");
62 }
63
64
65 // 3. Lectura de archivo (FileNotFoundException)
66 System.out.println("\n--- 3. Lectura de Archivo (FileNotFoundException) ---");
67 try {
68     FileReader fileReader = new FileReader(NOMBRE_ARCHIVO);
69
70     System.out.println("Archivo encontrado.");
71     fileReader.close();
72 } catch (java.io.FileNotFoundException e) {
73     System.err.println(" Error: Archivo no encontrado.");
74 } catch (IOException e) {
75     System.err.println(" Error de I/O: " + e.getMessage());
76 }
```

```
77
78
79 // 4. Excepción personalizada (EdadInvalidaException)
80 try {
81     verificarEdad(30);
82     verificarEdad(-5);
83 } catch (EdadInvalidaException e) {
84     System.err.println(" Capturada: " + e.getMessage());
85 }
86
87
88 // 5. Uso de try-with-resources (IOException)
89 System.out.println("\n--- 5. Uso de try-with-resources ---");
90 try (BufferedReader reader = new BufferedReader(new FileReader(NOMBRE_ARCHIVO))) {
91
92     String linea;
93     System.out.println("✓ Lectura con try-with-resources:");
94     while ((linea = reader.readLine()) != null) {
95         System.out.println("Línea: " + linea);
96     }
97
98 } catch (java.io.FileNotFoundException e) {
99     // Captura si el archivo no existe
100     System.err.println("Error de I/O: Archivo no encontrado para try-with-resources.");
101 } catch (IOException e) {
102     // Captura cualquier otro error de I/O (lectura, cierre, etc.)
103     System.err.println("Error de I/O: Ocurrió un error al leer el archivo: " + e.getMessage());
104 }
105
106
107
108
109
```

## CONCLUSIONES ESPERADAS

- Comprender la utilidad de las interfaces para lograr diseños desacoplados y reutilizables.
- Aplicar herencia múltiple a través de interfaces para combinar comportamientos.
- Utilizar correctamente estructuras de control de excepciones para evitar caídas del programa.
- Crear excepciones personalizadas para validar reglas de negocio.
- Aplicar buenas prácticas como **try-with-resources** y uso del bloque **finally** para manejar recursos y errores.
- Reforzar el diseño robusto y mantenible mediante la integración de interfaces y manejo de errores en Java.

