

---

# TRABAJO PRÁCTICO 8

**Interfaces y Excepciones en Java**

**Tecnicatura en Programación a Distancia**

**Programación II**

---

**Alumno:** Fabricio Puccio

**Fecha de Entrega:** 21-11-2025

**Repositorio GitHub:**

<https://github.com/FabricioPuccio/TP8-Interfaces-Excepciones>

---

## ÍNDICE

1. Introducción Teórica
  2. Parte 1: Sistema E-commerce
  3. Parte 2: Ejercicios de Excepciones
  4. Conclusión
  5. Repositorio GitHub
- 

## 1. INTRODUCCIÓN TEÓRICA

### 1.1 Interfaces en Java

Las interfaces en Java son contratos que definen comportamientos que las clases deben implementar. Permiten lograr:

- Desacoplamiento entre componentes
- Polimorfismo sin herencia múltiple
- Flexibilidad en el diseño del software

### 1.2 Manejo de Excepciones

El manejo de excepciones garantiza la robustez del software mediante:

- Control de errores en tiempo de ejecución
- Recuperación graceful de fallos
- Mantenimiento de la integridad del programa

## 2. PARTE 1: SISTEMA E-COMMERCE

## 2.1 Diseño de Interfaces

## Interfaz Pagable

Define el contrato para elementos que pueden calcular un total.

The screenshot shows a Java code editor with the following details:

- Title Bar:** TP TP8\_Interfaces\_... < main > Main > ⚡
- File:** Pagable.java
- Content:**

```
1 package ecommerce.interfaces;
2
3 public interface Pagable {
4     double calcularTotal();
5 }
6
```

- Annotations:** The code editor highlights the `public interface` and `double calcularTotal()` lines with blue boxes.
- Metrics:** The interface has 7 usages and 2 implementations, attributed to FabricioPuccio.
- Toolbars and Icons:** On the left, there are icons for file operations like Open, Save, Find, and Delete. On the right, there are icons for search, refresh, and settings.
- Status Bar:** The status bar at the bottom shows the path: TP8\_Interfaces\_Excepciones > src > ecommerce > interfaces > Pagable, and the status: 5:2 CRLF UTF-8 4 spaces.

## Interfaz Pago y PagoConDescuento

Especializan los métodos de procesamiento de pagos.

The screenshot shows a Java code editor with the following code:

```
package ecommerce.interfaces;
public interface Pago {
    boolean procesarPago(double monto);
}
```

The code editor interface includes:

- File navigation bar: TP8\_Interfaces\_... (selected), main, Main, etc.
- Toolbars: File, Edit, View, Insert, Tools, Window, Help.
- Left sidebar: Project tree (Pago.java selected), Search, Find, Replace, Diff, and various icons for file operations.
- Right sidebar: Status bar showing file path (TP8\_Interfaces\_Excepciones > src > ecommerce > interfaces > Pago), timestamp (3:24), encoding (CRLF), and file settings (4 spaces).

The screenshot shows a Java code editor with the following interface details:

- Project: TP8\_Interfaces\_...
- File: main / Main
- Code:

```
1 package ecommerce.interfaces;
2
3 public interface PagoConDescuento extends Pago {
4     double aplicarDescuento(double monto, double porcentajeDescuento);
5 }
```

The code editor includes a sidebar with icons for file operations like new, open, save, and delete. The status bar at the bottom shows the file path, timestamp (3:49), encoding (CRLF), and character set (UTF-8).

## Interfaz Notifiable

Permite notificar cambios de estado a los clientes.

The screenshot shows a Java code editor with the following interface details:

- Project: TP8\_Interfaces\_...
- File: main / Main
- Code:

```
1 package ecommerce.interfaces;
2
3 public interface Notifiable {
4     void notificar(String mensaje);
5 }
```

The code editor includes a sidebar with icons for file operations like new, open, save, and delete. The status bar at the bottom shows the file path, timestamp (3:31), encoding (CRLF), and character set (UTF-8).

## 2.2 Implementación de Clases

### Clase Producto

Implementa Pagable representando productos individuales.

The screenshot shows the code editor with the file `Producto.java` open. The class implements the `Pagable` interface. It has private fields `nombre` and `precio`, a constructor, and methods for calculating total price and getting/setting name and price. A `toString` method is also provided. The code editor's right panel displays the class hierarchy for `Producto`.

```
import ecommerce.interfaces.Pagable;
public class Producto implements Pagable {
    private String nombre; 4 usages & FabricioPuccio
    private double precio; 5 usages
    public Producto(String nombre, double precio) { 4 usages & FabricioPuccio
        this.nombre = nombre;
        this.precio = precio;
    }
    @Override 5 usages & FabricioPuccio
    public double calcularTotal() {
        return precio;
    }
    // Getters y Setters
    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 & FabricioPuccio
    public String toString() {
        return String.format("Producto: %s - %.2f", nombre, precio);
    }
}
```

Producto.java

- Inherited members (Ctrl+F12)
- Anonymous classes (Ctrl+I)
- Lambdas (Ctrl+L)

Producto

- Producto(String, double)
- calcularTotal(): double
- getNombre(): String
- getPrecio(): double
- setNombre(String): void
- setPrecio(double): void
- toString(): String ↑Object
- nombre: String
- precio: double

### Clase Pedido

Gestiona múltiples productos y notifica cambios.

The screenshot shows the code editor with the file `Pedido.java` open. The class implements the `Pagable` interface. It has a constructor for a client, a method to add products, and implementations for calculating total price and changing state. It also includes a notification mechanism for the client. The code editor's right panel displays the class hierarchy for `Pedido`.

```
public class Pedido implements Pagable {
    public Pedido(Cliente cliente) { 2 usages & FabricioPuccio
        this.productos = new ArrayList<>();
        this.estado = "PENDIENTE";
        this.cliente = cliente;
    }
    public void agregarProducto(Producto producto) { 3 usages & FabricioPuccio
        productos.add(producto);
    }
    @Override 5 usages & FabricioPuccio
    public double calcularTotal() {
        double total = 0;
        for (Producto producto : productos) {
            total += producto.calcularTotal();
        }
        return total;
    }
    public void cambiarEstado(String nuevoEstado) { 4 usages & FabricioPuccio
        String estadoAnterior = this.estado;
        this.estado = nuevoEstado;
        // Notificar al cliente del cambio de estado
        if (cliente != null) {
            String mensaje = "Tu pedido cambió de estado: " + estadoAnterior + " > " + nuevoEstado;
            cliente.notificar(mensaje);
        }
    }
}
```

Pedido.java

- Inherited members (Ctrl+F12)
- Anonymous classes (Ctrl+I)
- Lambdas (Ctrl+L)

Pedido

- Pedido(Cliente)
- agregarProducto(Producto): void
- calcularTotal(): double
- cambiarEstado(String): void
- getCliente(): Cliente
- getEstado(): String
- getProductos(): List<Producto>
- toString(): String ↑Object
- cliente: Cliente
- estado: String
- productos: List<Producto>

## Clase Cliente

Recibe notificaciones sobre sus pedidos.

The screenshot shows an IDE interface with the following details:

- Project:** TP8\_interfaces\_Excepciones
- File:** Cliente.java
- Code Content:**

```
package ecommerce.modelos;
import ecommerce.interfaces.Notifiable;
public class Cliente implements Notifiable {
    private String nombre; 3 usages & FabricioPuccio
    private String email; 3 usages
    public Cliente(String nombre, String email) {
        this.nombre = nombre;
        this.email = email;
    }
    @Override 1 usage & FabricioPuccio
    public void notificar(String mensaje) {
        System.out.println("Notificación para " + nombre + "(" + email + "): " + mensaje);
    }
    // Getters
    public String getNombre() { 2 usages & FabricioPuccio
        return nombre;
    }
    public String getEmail() { no usages & FabricioPuccio
        return email;
    }
}
```
- Right Panel:** Shows the interface `Notifiable` with its methods: `notificar(String)`, `getNombre()`, `getEmail()`, and `String nombre`.
- Status Bar:** Shows the file path `TP8_interfaces_Excepciones > src > ecommerce > modelos > Cliente.java`, and the time `5:46`.

## Medios de Pago

Implementaciones concretas de procesamiento de pagos.

The screenshot shows an IDE interface with the following details:

- Project:** TP8\_interfaces\_Excepciones
- File:** TarjetaCredito.java
- Code Content:**

```
package ecommerce.modelos;
import ecommerce.interfaces.PagoConDescuento;
public class TarjetaCredito implements PagoConDescuento {
    private String numeroTarjeta; 2 usages & FabricioPuccio
    private String titular; 2 usages
    private double limite; 5 usages
    public TarjetaCredito(String numeroTarjeta, String titular, double limite) {...}
    @Override 2 usages & FabricioPuccio
    public boolean procesarPago(double monto) {
        if (monto <= limite) {
            System.out.println("Pago con Tarjeta de Crédito procesado: $" + monto);
            limite -= monto;
            return true;
        } else {
            System.out.println("Límite insuficiente en tarjeta. Límite: $" + limite);
            return false;
        }
    }
    @Override 1 usage & FabricioPuccio
    public double aplicarDescuento(double monto, double porcentajeDescuento) {
        double descuento = monto * (porcentajeDescuento / 100);
        double montoConDescuento = monto - descuento;
        System.out.println("Descuento aplicado: " + porcentajeDescuento + "% - Ahorro: $" + descuento);
        return montoConDescuento;
    }
}
```
- Right Panel:** Shows the interface `PagoConDescuento` with its methods: `procesarPago(double)`, `aplicarDescuento(double, double)`, and `double limite`.
- Status Bar:** Shows the file path `TP8_interfaces_Excepciones > src > ecommerce > modelos > TarjetaCredito.java`, and the time `5:58`.

```
TP8_Interfaces_Excepciones > src > ecommerce > modelos > PayPal.java
```

```
1 package ecommerce.modelos;
2
3 import ecommerce.interfaces.Pago;
4
5 public class PayPal implements Pago {
6     private String email; 3 usages  ↳ FabricioPuccio
7     private double saldo; 5 usages
8
9     public PayPal(String email, double saldo) { 2 usages  ↳ FabricioPuccio
10        this.email = email;
11        this.saldo = saldo;
12    }
13
14    @Override 2 usages  ↳ FabricioPuccio
15    public boolean procesarPago(double monto) {
16        if (monto <= saldo) {
17            System.out.println("Pago con PayPal procesado: $" + monto + " - Email: " + email);
18            saldo -= monto;
19            return true;
20        } else {
21            System.out.println("Saldo insuficiente en PayPal. Saldo: $" + saldo);
22            return false;
23        }
24    }
25
26    // Getters
27    public String getEmail() { return email; }
28
29    public double getSaldo() { return saldo; }
30
31
32
33
34
```

PayPal.java

- Inherited members (Ctrl+F12)
- Anonymous classes (Ctrl+I)
- Lambdas (Ctrl+Shift+L)

PayPal

- ⚡ PayPal(String, double)
- ⚡ getEmail(): String
- ⚡ getSaldo(): double
- ⚡ procesarPago(double): boolean
- ⚡ email: String
- ⚡ saldo: double

5:38 CRLF UTF-8 4 spaces

## 2.3 Clase Principal y Ejecución

```
TP8_Interfaces_E... > src > main > Main.java
```

```
1 package ecommerce;
2
3 public class Main {
4     public static void ejecutarDemoEcommerce() { 1 usage  ↳ FabricioPuccio
5         System.out.println("DEMOSTRACIÓN SISTEMA E-COMMERCE");
6         System.out.println("=====\\n");
7
8         // 1. Crear cliente
9         Cliente cliente = new Cliente("Fabricio Puccio", "fabricio@email.com");
10        System.out.println("Cliente creado: " + cliente.getNombre());
11
12        // 2. Crear productos
13        Producto laptop = new Producto("Laptop Gamer", 1500000.00);
14        Producto mouse = new Producto("Mouse Inalámbrico", 50000.00);
15        Producto teclado = new Producto("Teclado Mecánico", 120000.00);
16
17        System.out.println("\nProductos creados:");
18        System.out.println(" - " + laptop);
19        System.out.println(" - " + mouse);
20        System.out.println(" - " + teclado);
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Main.java

⚠ 1 ✅ 3

21:12 CRLF UTF-8 4 spaces

The screenshot shows a Java code editor with the file `Main.java` open. The code implements a `Main` class with a static method `ejecutarDemoEcommerce`. The code demonstrates creating a `Pedido` object and adding `laptop`, `mouse`, and `teclado` products to it. It then prints the created pedido and its products.

```
public class Main {    private static void ejecutarDemoEcommerce() {        // 3. Crear pedido y agregar productos        Pedido pedido = new Pedido(cliente);        pedido.agregarProducto(laptop);        pedido.agregarProducto(mouse);        pedido.agregarProducto(teclado);        System.out.println("\nPedido creado:");        System.out.println(" " + pedido);        // 4. Mostrar productos en el pedido        System.out.println("\nProductos en el pedido:");        for (Producto producto : pedido.getProductos()) {            System.out.println(" - " + producto.getNombre() + ": $" + producto.getPrecio());        }    }}
```

The screenshot shows a Java code editor with the file `Main.java` open. The code implements a `Main` class with a static method `ejecutarDemoEcommerce`. The code demonstrates changing pedido states (NOTIFICACIONES DE ESTADO), demonstrating polymorphism with interfaces (DEMOSTRACION DE POLIMORFISMO), and printing payable elements and payment methods.

```
private static void ejecutarDemoEcommerce() {    // 7. Cambiar estados del pedido (notificaciones)    System.out.println("\nNOTIFICACIONES DE ESTADO:");    System.out.println("-----");    pedido.cambiarEstado("EN PROCESO");    pedido.cambiarEstado("ENVIADO");    pedido.cambiarEstado("ENTREGADO");    // 8. Demostrar polimorfismo con interfaces    System.out.println("\nDEMOSTRACION DE POLIMORFISMO:");    System.out.println("-----");    Pagable[] elementosPagables = {laptop, pedido};    Pago[] metodosPago = {tarjeta, paypal};    System.out.println("Elementos pagables:");    for (Pagable pagable : elementosPagables) {        System.out.println(" - " + pagable.getClass().getSimpleName() + ": $" + pagable.calcularTotal());    }    System.out.println("Métodos de pago disponibles:");    for (Pago pago : metodosPago) {        System.out.println(" - " + pago.getClass().getSimpleName());    }}
```

## Resultados de Ejecución

== TRABAJO PRÁCTICO 8 ==

Interfaces y Excepciones en Java

Alumno: Fabricio Puccio

=====

## **DEMOSTRACIÓN SISTEMA E-COMMERCE**

---

**Cliente creado: Fabricio Puccio**

**Productos creados:**

- Producto: Laptop Gamer - \$1500000,00
- Producto: Mouse Inalámbrico - \$50000,00
- Producto: Teclado Mecánico - \$120000,00

**Pedido creado:**

**Pedido [Cliente: Fabricio Puccio, Estado: PENDIENTE, Productos: 3, Total: \$1670000,00]**

**Productos en el pedido:**

- Laptop Gamer: \$1500000.0
- Mouse Inalámbrico: \$50000.0
- Teclado Mecánico: \$120000.0

**Total del pedido: \$1670000,00**

**PROCESANDO PAGOS:**

---

**--- Pago con Tarjeta de Crédito ---**

**Descuento aplicado: 10.0% - Ahorro: \$167000.0**

**Pago con Tarjeta de Crédito procesado: \$1503000.0**

**Notificación para Fabricio Puccio (fabricio@email.com): Tu pedido cambió de estado: PENDIENTE → PAGADO**

**--- Pago con PayPal ---**

**Pago con PayPal procesado: \$50000.0 - Email: fabricio.puccio@email.com**

**Pago PayPal exitoso**

#### **NOTIFICACIONES DE ESTADO:**

-----

**Notificación para Fabricio Puccio (fabricio@email.com): Tu pedido cambió de estado: PAGADO → EN PROCESO**

**Notificación para Fabricio Puccio (fabricio@email.com): Tu pedido cambió de estado: EN PROCESO → ENVIADO**

**Notificación para Fabricio Puccio (fabricio@email.com): Tu pedido cambió de estado: ENVIADO → ENTREGADO**

#### **DEMOSTRACIÓN DE POLIMORFISMO:**

-----

##### **Elementos pagables:**

- **Producto: \$1500000.0**
- **Pedido: \$1670000.0**

##### **Métodos de pago disponibles:**

- **TarjetaCredito**
- **PayPal**

**Análisis:** El sistema demuestra polimorfismo mediante interfaces, notificaciones en tiempo real y procesamiento flexible de pagos.

### 3. PARTE 2: EJERCICIOS DE EXCEPCIONES

#### 3.1 División Segura

Manejo de `ArithmeticException` para división por cero.

Código:

The screenshot shows the IntelliJ IDEA interface with the file `DivisionSegura.java` open. The code implements a method `ejecutarEjercicio()` that reads two numbers from the user and divides them. It catches `InputMismatchException` and `ArithmeticException`. If an `ArithmeticException` occurs, it prints an error message. A `finally` block ensures resources are released. The code is annotated with `usage` and `FabricioPuccio`.

```
1 package ejercicios_excepciones;
2
3 import java.util.Scanner;
4
5 public class DivisionSegura {
6
7     public static void ejecutarEjercicio() {
8         System.out.println("\nEJERCICIO 1: DIVISIÓN SEGURA");
9         System.out.println("=====");
10
11         Scanner scanner = new Scanner(System.in);
12
13         try {
14             System.out.print("Ingrese el dividendo: ");
15             double dividendo = scanner.nextDouble();
16
17             System.out.print("Ingrese el divisor: ");
18             double divisor = scanner.nextDouble();
19
20             double resultado = dividir(dividendo, divisor);
21             System.out.printf("Resultado: %.2f / %.2f = %.2f\n", dividendo, divisor, resultado);
22
23         } catch (java.util.InputMismatchException e) {
24             System.out.println("Error: Debe ingresar números válidos");
25         } catch (ArithmeticException e) {
26             System.out.println("Error: No se puede dividir por cero");
27         } finally {
28             System.out.println("Bloque finally ejecutado - Recursos liberados");
29         }
30     }
31 }
```

Ejecución:

The screenshot shows the IntelliJ IDEA interface with the file `Main.java` open. The program runs and outputs the following text to the terminal:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
== TRABAJO PRÁCTICO 8 ==
Interfaces y Excepciones en Java
Alumno: Fabricio Puccio
=====

EJERCICIO 1: DIVISIÓN SEGURA
=====
Ingrese el dividendo: 12
Ingrese el divisor: 0
Error: No se puede dividir por cero
Bloque finally ejecutado - Recursos liberados

Process finished with exit code 0
```

## 3.2 Conversión de Cadena a Número

Manejo de NumberFormatException para conversiones inválidas.

Código:

The screenshot shows the IntelliJ IDEA interface with the code editor open. The file is named `ConversionCadenaNumero.java`. The code implements a static method `ejecutarEjercicio()` that reads a string from the user and tries to convert it to an integer. If the conversion fails, it catches `NumberFormatException` and prints an error message. A tooltip for the `convertirAEntero` method is visible, showing its signature: `convertirAEntero(String): int`.

```
package ejercicios_excepciones;
import java.util.Scanner;

public class ConversionCadenaNumero {
    public static void ejecutarEjercicio() {
        System.out.println("\nEJERCICIO 2: CONVERSIÓN DE CADENA A NÚMERO");
        System.out.println("=====");

        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese un número entero: ");
        String entrada = scanner.nextLine();

        try {
            int numero = convertirAEntero(entrada);
            System.out.println("Número convertido: " + numero);
            System.out.println("El doble del número es: " + (numero * 2));
        } catch (NumberFormatException e) {
            System.out.println("Error: '" + entrada + "' no es un número entero válido");
            System.out.println("Mensaje técnico: " + e.getMessage());
        }
    }

    private static int convertirAEntero(String cadena) {
        return Integer.parseInt(cadena);
    }
}
```

Ejecución:

The screenshot shows the IntelliJ IDEA interface with the run tool window open. The run configuration is set to `Main`. The output pane displays the execution of the program. It starts with some initial text, then enters the exercise loop where it asks for a number. When the user types `Hola`, it catches the `NumberFormatException` and prints the error message and technical message.

```
== TRABAJO PRÁCTICO 8 ==
Interfaces y Excepciones en Java
Alumno: Fabricio Puccio
=====

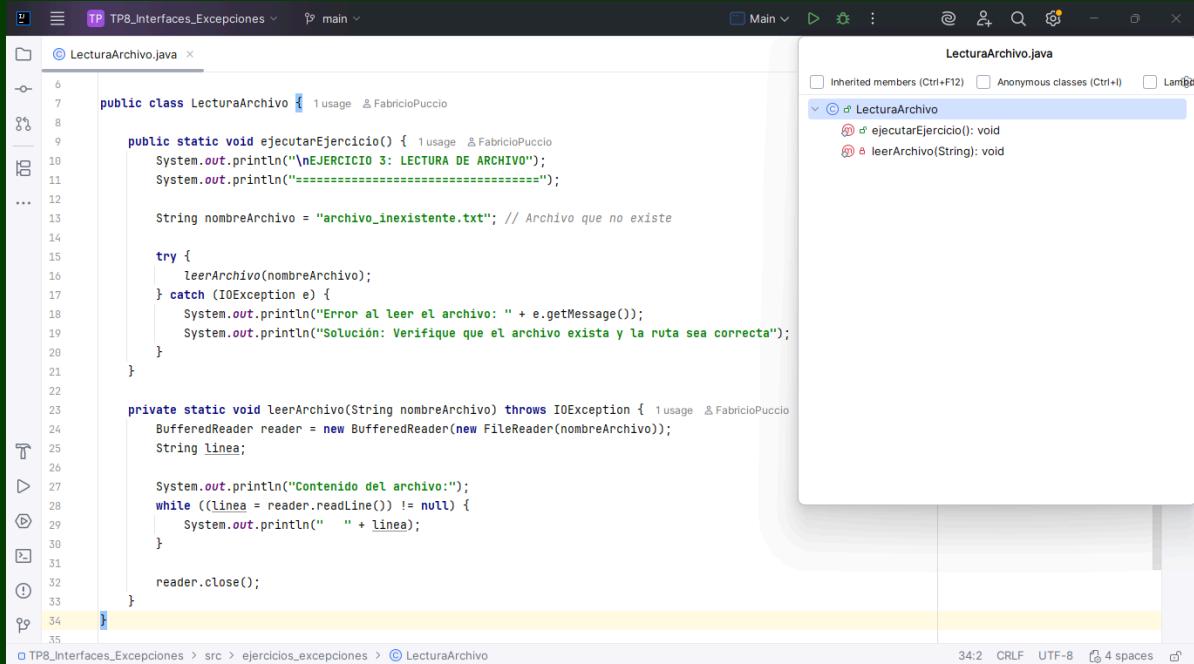
EJERCICIO 2: CONVERSIÓN DE CADENA A NÚMERO
=====
Ingrese un número entero: Hola
Error: 'Hola' no es un número entero válido
Mensaje técnico: For input string: "Hola"

Process finished with exit code 0
```

### 3.3 Lectura de Archivo

Manejo de FileNotFoundException para archivos inexistentes.

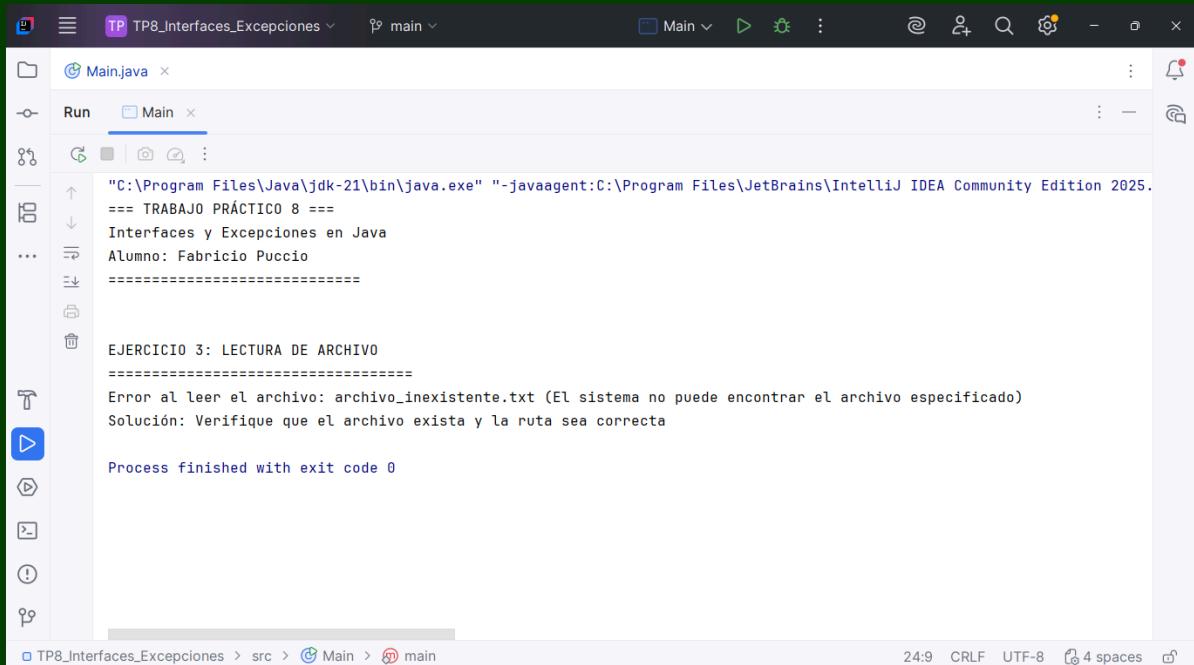
Código:



```
public class LecturaArchivo {
    public static void ejecutarEjercicio() {
        System.out.println("\nEJERCICIO 3: LECTURA DE ARCHIVO");
        System.out.println("=====");
        String nombreArchivo = "archivo_inexistente.txt"; // Archivo que no existe
        try {
            leerArchivo(nombreArchivo);
        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
            System.out.println("Solución: Verifique que el archivo exista y la ruta sea correcta");
        }
    }

    private static void leerArchivo(String nombreArchivo) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(nombreArchivo));
        String linea;
        System.out.println("Contenido del archivo:");
        while ((linea = reader.readLine()) != null) {
            System.out.println(" " + linea);
        }
        reader.close();
    }
}
```

Ejecución:



```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.
== TRABAJO PRÁCTICO 8 ==
Interfaces y Excepciones en Java
Alumno: Fabricio Puccio
=====

EJERCICIO 3: LECTURA DE ARCHIVO
=====
Error al leer el archivo: archivo_inexistente.txt (El sistema no puede encontrar el archivo especificado)
Solución: Verifique que el archivo exista y la ruta sea correcta

Process finished with exit code 0
```

### 3.4 Excepción Personalizada

Creación y uso de EdadInvalidaException para validaciones.

Excepción Personalizada:

The screenshot shows a Java code editor with the following code:

```
package ejercicios_excepciones.excepciones;

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

    public EdadInvalidaException(String mensaje, Throwable causa) {
        super(mensaje, causa);
    }
}
```

The code editor interface includes a sidebar with icons for file operations, a status bar at the bottom showing the path "Excepciones > src > ejercicios\_excepciones > excepciones > EdadInvalidaException", and a status bar at the bottom right showing "3:55 CRLF UTF-8 4 spaces".

Código de Validación:

The screenshot shows a Java code editor with the following code:

```
public class ValificacionEdad {
    public static void ejecutarEjercicio() {
        System.out.println("\nEJERCICIO 4: EXCEPCIÓN PERSONALIZADA - VALIDACIÓN DE EDAD");
        System.out.println("=====");

        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Ingrese su edad: ");
            int edad = scanner.nextInt();

            validarEdad(edad);
            System.out.println("Edad válida: " + edad + " años");

        } catch (EdadInvalidaException e) {
            System.out.println(e.getMessage());
        } catch (java.util.InputMismatchException e) {
            System.out.println("Error: Debe ingresar un número entero válido");
        }
    }

    private static void validarEdad(int edad) throws EdadInvalidaException {
        if (edad < 0) {
            throw new EdadInvalidaException("La edad no puede ser negativa: " + edad);
        } else if (edad > 120) {
            throw new EdadInvalidaException("La edad no puede ser mayor a 120 años: " + edad);
        } else if (edad < 18) {
            throw new EdadInvalidaException("Edad insuficiente para realizar la operación: " + edad);
        }
    }
}
```

A code completion tooltip is visible on the right side of the editor, listing the methods of the `ValificacionEdad` class:

- ejecutarEjercicio(): void
- validarEdad(int): void

The code editor interface includes a sidebar with icons for file operations, a status bar at the bottom showing the path "TP8\_Interfaces\_Excepciones > src > ejercicios\_excepciones > ValificacionEdad", and a status bar at the bottom right showing "7:30 CRLF UTF-8 4 spaces".

## Ejecución:

The screenshot shows the IntelliJ IDEA interface during the execution of a Java program. The top bar displays the project name "TP TP8\_Interfaces\_E..." and the file "main". The left sidebar has a "Run" tab selected, showing a "Main" configuration. The main window displays the following text output:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
== TRABAJO PRÁCTICO 8 ==
Interfaces y Excepciones en Java
Alumno: Fabricio Puccio
=====
EJERCICIO 4: EXCEPCIÓN PERSONALIZADA - VALIDACIÓN DE EDAD
=====
Ingrese su edad: 200
La edad no puede ser mayor a 120 años: 200

Process finished with exit code 0
```

At the bottom, the status bar shows the path "TP8\_Interfaces\_Excepciones > src > Main > main", the time "25:44", and encoding "UTF-8".

## 3.5 Try-with-Resources

Uso de try-with-resources para gestión automática de recursos.

Código:

The screenshot shows the IntelliJ IDEA interface displaying the code for "TryWithResourcesDemo.java". The code demonstrates the use of the try-with-resources statement to automatically close resources. The code reads from a file named "datos\_ejemplo.txt".

```
public class TryWithResourcesDemo {
    public static void ejecutarEjercicio() {
        System.out.println("\nEJERCICIO 5: TRY-WITH-RESOURCES");
        System.out.println("=====");

        // Ruta del archivo
        String nombreArchivo = "resources/datos_ejemplo.txt";

        System.out.println("Leyendo archivo: " + nombreArchivo);
        System.out.println("-----");

        // Try-with-resources - LOS RECURSOS SE CIERRAN AUTOMÁATICAMENTE
        try (FileReader fileReader = new FileReader(nombreArchivo);
             BufferedReader reader = new BufferedReader(fileReader)) {

            String linea;
            int numeroLinea = 1;

            while ((linea = reader.readLine()) != null) {
                System.out.println("Línea " + numeroLinea + ": " + linea);
                numeroLinea++;
            }

            System.out.println("Archivo leido exitosamente");
            System.out.println("Recursos cerrados automáticamente por try-with-resources");

        } catch (IOException e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
            System.out.println("Verifique que el archivo exista en: " + nombreArchivo);
        }
    }
}
```

At the bottom, the status bar shows the path "TP8\_Interfaces\_Excepciones > src > ejercicios\_excepciones > TryWithResourcesDemo", the time "7:36", and encoding "UTF-8".

## Ejecución:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.1\lib\idea_rt.jar" -Dfile.encoding=UTF-8 main
== TRABAJO PRÁCTICO 8 ==
Interfaces y Excepciones en Java
Alumno: Fabricio Puccio
=====
EJERCICIO 5: TRY-WITH-RESOURCES
=====
Leyendo archivo: resources/datos_ejemplo.txt
-----
Línea 1: Línea 1: Este es un archivo de ejemplo
Línea 2: Línea 2: Para demostrar try-with-resources
Línea 3: Línea 3: En el Trabajo Práctico 8
Línea 4: Línea 4: Programación II - Interfaces y Excepciones
Línea 5: Línea 5: Alumno: Fabricio Puccio
Archivo leído exitosamente
Recursos cerrados automáticamente por try-with-resources
Process finished with exit code 0
```

---

## 4. CONCLUSIÓN

### 4.1 Aprendizajes Adquiridos

- ❖ Interfaces como contratos para diseño modular
- ❖ Polimorfismo sin herencia múltiple mediante interfaces
- ❖ Manejo robusto de excepciones para software confiable
- ❖ Try-with-resources para gestión automática de recursos
- ❖ Excepciones personalizadas para reglas de negocio específicas

### 4.2 Dificultades Superadas

- Comprensión del problema del diamante y su solución con interfaces
- Diferenciación entre excepciones checked y unchecked
- Implementación de sistemas de notificación desacoplados
- Organización de paquetes y estructura modular

### 4.3 Aplicación Práctica

Los conocimientos adquiridos permiten desarrollar software más:

- Modular y mantenable
- Robusto ante errores
- Escalable para futuras extensiones
- Profesional en estándares de calidad

## 5. REPOSITORIO GITHUB

**Enlace al código fuente completo:**

<https://github.com/FabricioPuccio/TP8-Interfaces-Excepciones>

- ❖ El repositorio contiene:
  - ❖ Código fuente completo y organizado
  - ❖ README con documentación del proyecto
  - ❖ Estructura de paquetes profesional
  - ❖ Historial de commits del desarrollo
- 

Fabricio Puccio - Programación II - 21-11-2025