

Trabajo Practico 8

Interface y Excepciones

Santiago Raúl Salinas

1.

Interfaz Pagable:

```
11 public interface Pagable {  
12  
13     // Este método es abstracto por defecto y obliga a las clases a implementarlo.  
14     double calcularTotal();  
15 }
```

Clase Producto:

```
11 public class Producto implements Pagable {  
12     private String id; // Identificador único del producto.  
13     private String nombre; // Nombre del producto.  
14     private double precio; // Precio unitario.  
15  
16     public Producto(String id, String nombre, double precio) {  
17         this.id = id;  
18         this.nombre = nombre;  
19         this.precio = precio;  
20     }  
21  
22     // Implementación Obligatoria del Contrato Pagable  
23     @Override  
24     public double calcularTotal() {  
25         // Para un producto individual, el total es simplemente su precio.  
26         return this.precio;  
27     }  
28  
29     // Getters necesarios  
30     public String getId() {  
31         return id;  
32     }  
33  
34     public double getPrecio() {  
35         return precio;  
36     }  
37  
38     public String getNombre() {  
39         return nombre;  
40     }  
41 }
```

Clase Pedido:

```
21 public class Pedido implements Pagable {
22
23     // ATRIBUTOS
24     private List<Producto> productos; // Colección dinámica 1:N
25     private Cliente cliente; // Referencia al Cliente
26     private String estado;
27
28     // CONSTRUCTOR
29     public Pedido(Cliente cliente) {
30         // Inicialización obligatoria del ArrayList para la colección
31         this.productos = new ArrayList<>();
32         this.cliente = cliente;
33         this.estado = "PENDIENTE";
34     }
35
36     public void agregarProducto(Producto p) {
37         this.productos.add(p);
38     }
39
40     // Implementación del contrato Pagable
41     @Override
42     public double calcularTotal() {
43         double total = 0;
44         // Polimorfismo: Llama a calcularTotal() del Producto
45         for (Producto p : productos) {
46             total += p.calcularTotal();
47         }
48         return total;
49     }
50
51     // Método de Negocio: Cambia el estado y notifica (usa la interfaz Notificable)
52     public void cambiarEstado(String nuevoEstado) {
53         this.estado = nuevoEstado;
54
55         if (cliente != null) {
56             cliente.notificarCambioEstado(nuevoEstado);
57         }
58     }
59 }
```

Interfaz Pago:

```
10 public interface Pago {
11     boolean procesarPago(double monto);
12 }
13
```

Interfaz PagoConDescuento:

```
10 public interface PagoConDescuento extends Pago {
11     double aplicarDescuento(double monto);
12 }
13
```

Clase TarjetaCredito:

```
11 public class TarjetaCredito implements PagoConDescuento {
12     private String numero;
13
14     public TarjetaCredito(String numero) {
15         this.numero = numero;
16     }
17
18     @Override
19     public boolean procesarPago(double monto) {
20         System.out.printf("$ Pago Tarjeta %s: Procesando $%.2f... \n", numero, monto);
21         // Lógica real (validaciones, etc.) iría aquí
22         return monto > 0;
23     }
24
25     @Override
26     public double aplicarDescuento(double monto) {
27         // Descuento del 10%
28         return monto * 0.90;
29     }
30 }
```

Clase PayPal:

```
11 // Implementa PagoConDescuento, por lo que debe implementar procesarPago() y aplicarDescuento()
12 public class PayPal implements PagoConDescuento {
13     private String email;
14
15     public PayPal(String email) {
16         this.email = email;
17     }
18
19     @Override
20     public boolean procesarPago(double monto) {
21         System.out.printf("$ Pago PayPal: Procesando $%.2f via %s... \n", monto, email);
22         return monto > 0;
23     }
24
25     @Override
26     public double aplicarDescuento(double monto) {
27         // Ejemplo: Aplica un descuento fijo del 5%
28         return monto * 0.95;
29     }
30 }
```

Interfaz Notificable:

```
11 // Archivo: Notificable.java
12 public interface Notificable {
13     // Contrato: Obliga a cualquier clase que implemente esto a saber cómo enviar un mensaje.
14     void notificarCambioEstado(String nuevoEstado);
15 }
```

Clase Cliente:

```
11 // Archivo: Cliente.java
12 public class Cliente implements Notificable {
13     private String nombre;
14     private String email;
15
16     public Cliente(String nombre, String email) {
17         this.nombre = nombre;
18         this.email = email;
19     }
20
21     // Implementación del contrato Notificable (Obligatorio)
22     @Override
23     public void notificarCambioEstado(String nuevoEstado) {
24         System.out.printf("CLIENTE (%s): Notificación a %s. Su pedido ahora está en estado: %s.\n",
25             this.nombre, this.email, nuevoEstado);
26     }
27
28     public String getNombre() {
29         return nombre;
30     }
31 }
```

Main:

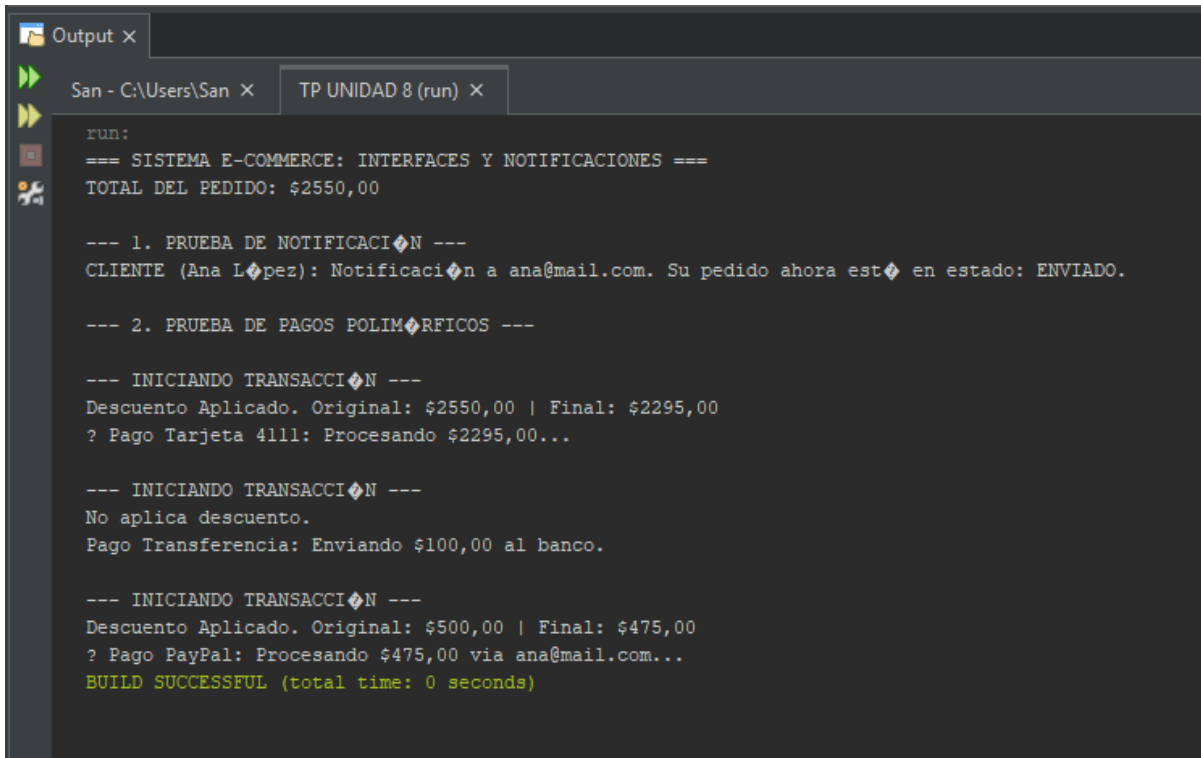
```
14 public class Main {
15
16     // Método Estático que utiliza el Polimorfismo por Interfaz
17     // Acepta cualquier objeto que cumpla el contrato Pago
18     public static void realizarTransaccion(Pago medioDePago, double monto) {
19
20         System.out.println("\n--- INICIANDO TRANSACCIÓN ---");
21
22         // 1. Uso de instanceof para chequear si el medio es elegible para descuento
23         if (medioDePago instanceof PagoConDescuento) {
24
25             // 2. Downcasting Seguro: Convertimos la referencia genérica a PagoConDescuento
26             PagoConDescuento medioConDesc = (PagoConDescuento) medioDePago;
27
28             // 3. Aplicar descuento (Lógica específica)
29             double montoConDescuento = medioConDesc.aplicarDescuento(monto);
30
31             System.out.printf("Descuento Aplicado. Original: $%.2f | Final: $%.2f\n", monto, montoConDescuento);
32             // Llamada polimórfica a procesarPago (con el monto con descuento)
33             medioConDesc.procesarPago(montoConDescuento);
34
35         } else {
36             // 4. Procesar pago normal (Efectivo, Transferencia, etc.)
37             System.out.println("No aplica descuento.");
38             medioDePago.procesarPago(monto); // Llamada polimórfica
39         }
40     }
41 }
```

```

42 public static void main(String[] args) {
43
44     // 1. Crear el Cliente y Medios de Pago
45     Cliente clienteAna = new Cliente("Ana López", "ana@mail.com");
46
47     // Crear las implementaciones concretas de la interfaz Pago
48     TarjetaCredito tarjetaVisa = new TarjetaCredito("4111");
49     PayPal cuentaPayPal = new PayPal("ana@mail.com");
50
51     // Creamos una forma simple de pago sin descuento para el ejemplo
52     class Transferencia implements Pago {
53         @Override
54         public boolean procesarPago(double monto) {
55             System.out.printf("Pago Transferencia: Enviando $%.2f al banco.\n", monto);
56             return true;
57         }
58     }
59
60     Transferencia banco = new Transferencia();
61
62     System.out.println("=== SISTEMA E-COMMERCE: INTERFACES Y NOTIFICACIONES ===");
63
64     // 2. Crear un Pedido (Objeto Complejo)
65     Pedido pedidol = new Pedido(clienteAna); // Pedido asociado a Ana
66
67     // Añadir productos
68     pedidol.agregarProducto(new Producto("T101", "Laptop", 2500.00));
69     pedidol.agregarProducto(new Producto("T102", "Mouse", 50.00));
70
71     double totalPedido = pedidol.calcularTotal();
72
73     System.out.printf("TOTAL DEL PEDIDO: $%.2f\n", totalPedido);
74
75     // 3. Demostración de Notificación (Interfaz Notificable)
76     System.out.println("\n--- 1. PRUEBA DE NOTIFICACIÓN ---");
77     pedidol.cambiarEstado("ENVIADO"); // Llama a clienteAna.notificarCambioEstado()
78
79     // 4. Demostración de Pago Polimórfico
80     System.out.println("\n--- 2. PRUEBA DE PAGOS POLIMÓRFICOS ---");
81
82     // Pago 1: Con Descuento (Tarjeta, implementa PagoConDescuento)
83     realizarTransaccion(tarjetaVisa, totalPedido);
84
85     // Pago 2: Sin Descuento (Transferencia, implementa solo Pago)
86     realizarTransaccion(banco, 100.00);
87
88     // Pago 3: Con Descuento (PayPal, implementa PagoConDescuento)
89     realizarTransaccion(cuentaPayPal, 500.00);
90 }

```

Output:



```
run:
=== SISTEMA E-COMMERCE: INTERFACES Y NOTIFICACIONES ===
TOTAL DEL PEDIDO: $2550,00

--- 1. PRUEBA DE NOTIFICACION ---
CLIENTE (Ana Lopez): Notificación a ana@mail.com. Su pedido ahora está en estado: ENVIADO.

--- 2. PRUEBA DE PAGOS POLIMORFICOS ---

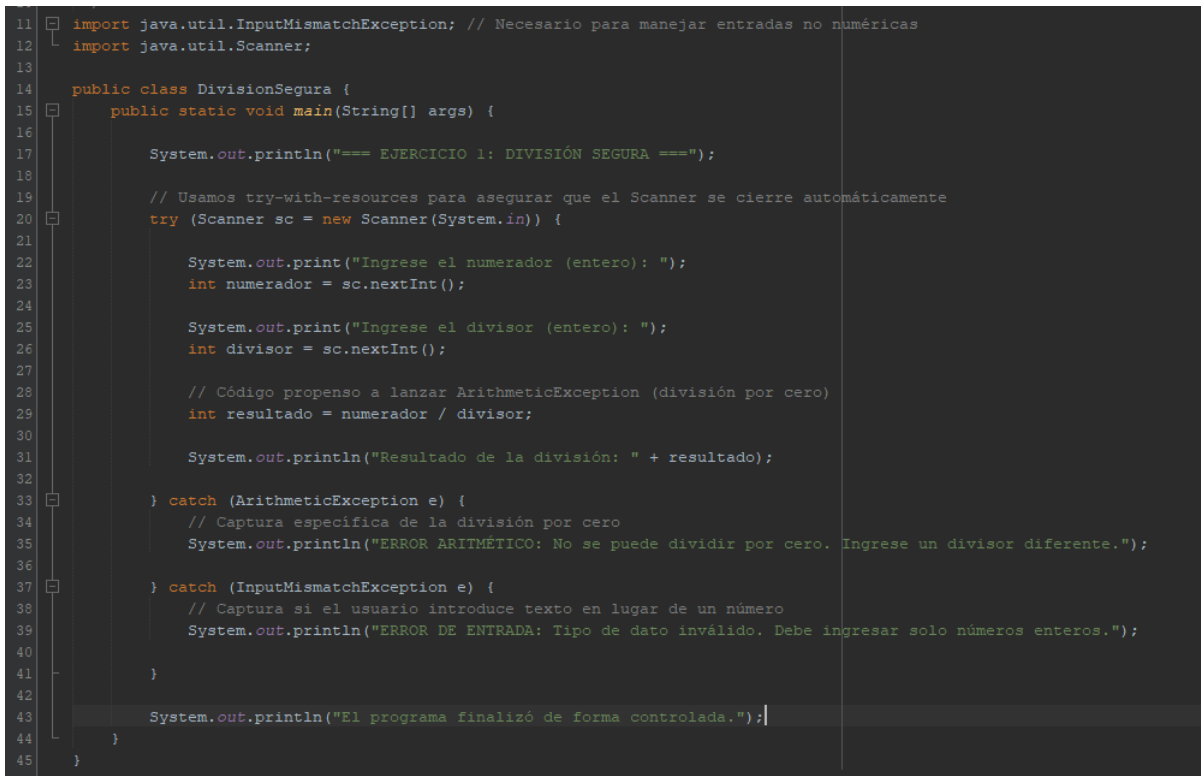
--- INICIANDO TRANSACCION ---
Descuento Aplicado. Original: $2550,00 | Final: $2295,00
? Pago Tarjeta 4111: Procesando $2295,00...

--- INICIANDO TRANSACCION ---
No aplica descuento.
Pago Transferencia: Enviando $100,00 al banco.

--- INICIANDO TRANSACCION ---
Descuento Aplicado. Original: $500,00 | Final: $475,00
? Pago PayPal: Procesando $475,00 via ana@mail.com...
BUILD SUCCESSFUL (total time: 0 seconds)
```

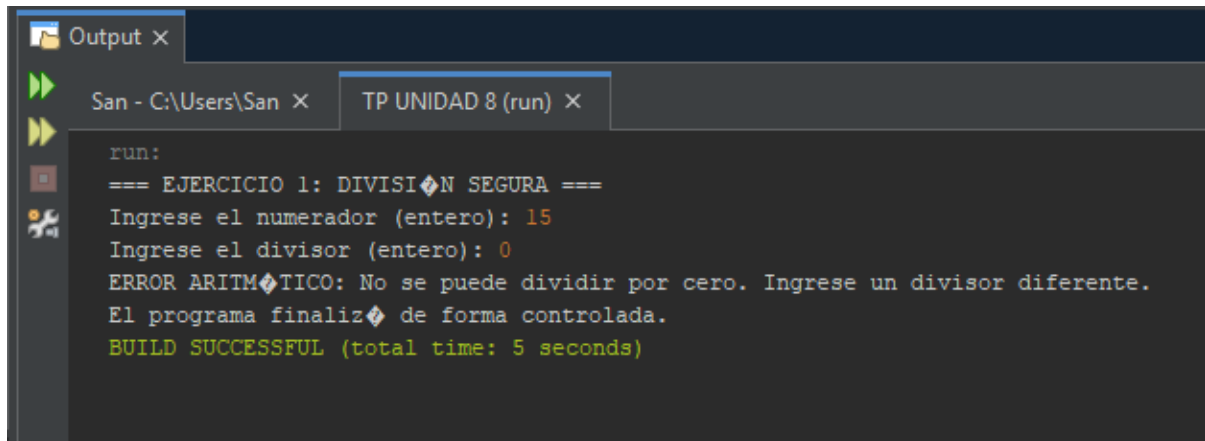
2.1

DivisionSegura:



```
11 import java.util.InputMismatchException; // Necesario para manejar entradas no numéricas
12 import java.util.Scanner;
13
14 public class DivisionSegura {
15     public static void main(String[] args) {
16
17         System.out.println("=== EJERCICIO 1: DIVISIÓN SEGURA ===");
18
19         // Usamos try-with-resources para asegurar que el Scanner se cierre automáticamente
20         try (Scanner sc = new Scanner(System.in)) {
21
22             System.out.print("Ingrese el numerador (entero): ");
23             int numerador = sc.nextInt();
24
25             System.out.print("Ingrese el divisor (entero): ");
26             int divisor = sc.nextInt();
27
28             // Código propenso a lanzar ArithmeticException (división por cero)
29             int resultado = numerador / divisor;
30
31             System.out.println("Resultado de la división: " + resultado);
32
33         } catch (ArithmeticException e) {
34             // Captura específica de la división por cero
35             System.out.println("ERROR ARITMÉTICO: No se puede dividir por cero. Ingrese un divisor diferente.");
36
37         } catch (InputMismatchException e) {
38             // Captura si el usuario introduce texto en lugar de un número
39             System.out.println("ERROR DE ENTRADA: Tipo de dato inválido. Debe ingresar solo números enteros.");
40
41         }
42
43         System.out.println("El programa finalizó de forma controlada.");
44     }
45 }
```

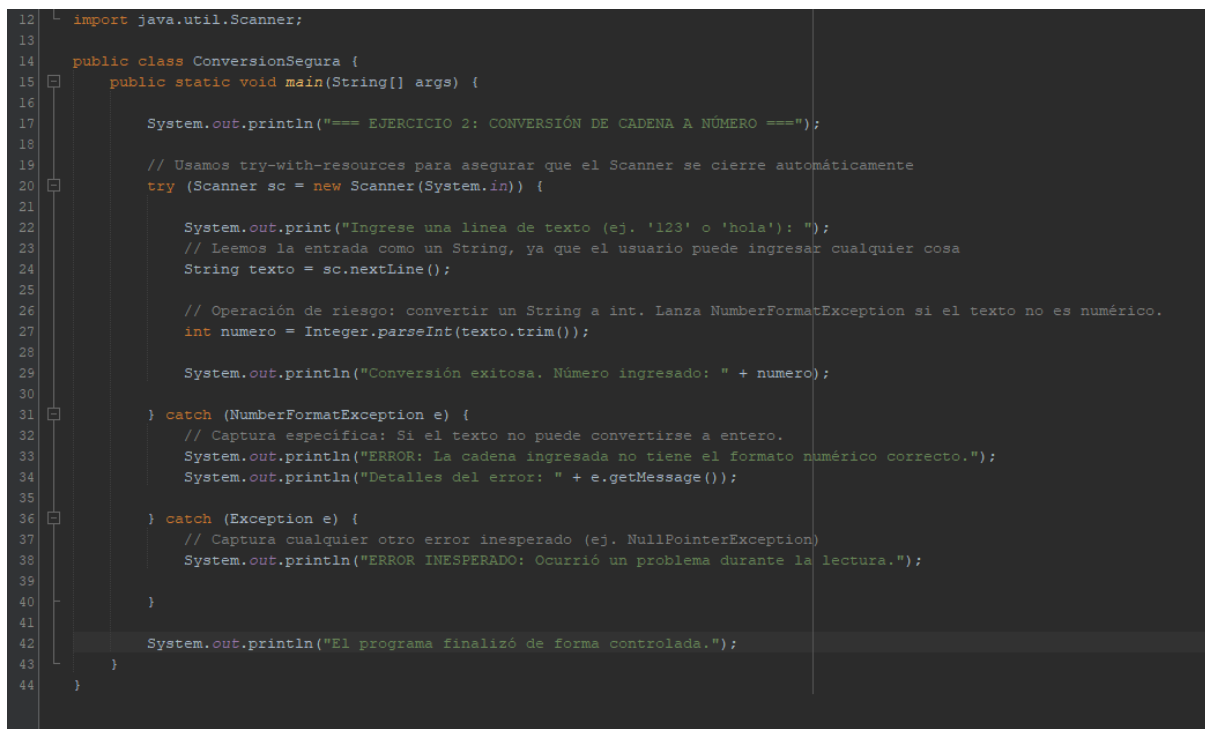
Output:



```
run:
=== EJERCICIO 1: DIVISION SEGURA ===
Ingrese el numerador (entero): 15
Ingrese el divisor (entero): 0
ERROR ARITMETICO: No se puede dividir por cero. Ingrese un divisor diferente.
El programa finalizó de forma controlada.
BUILD SUCCESSFUL (total time: 5 seconds)
```

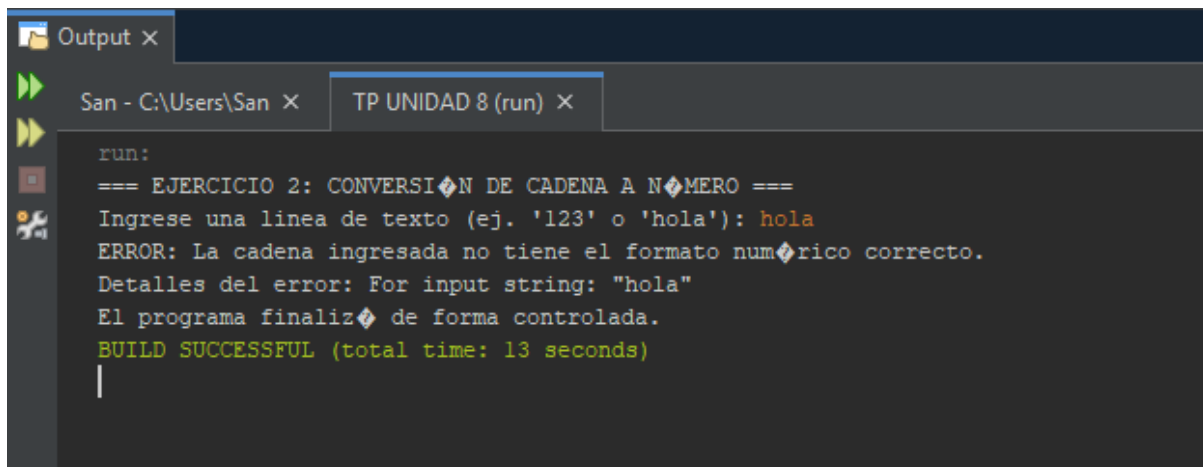
2.2

ConversionSegura:



```
12 import java.util.Scanner;
13
14 public class ConversionSegura {
15     public static void main(String[] args) {
16
17         System.out.println("=== EJERCICIO 2: CONVERSIÓN DE CADENA A NÚMERO ===");
18
19         // Usamos try-with-resources para asegurar que el Scanner se cierre automáticamente
20         try (Scanner sc = new Scanner(System.in)) {
21
22             System.out.print("Ingrese una línea de texto (ej. '123' o 'hola'): ");
23             // Leemos la entrada como un String, ya que el usuario puede ingresar cualquier cosa
24             String texto = sc.nextLine();
25
26             // Operación de riesgo: convertir un String a int. Lanza NumberFormatException si el texto no es numérico.
27             int numero = Integer.parseInt(texto.trim());
28
29             System.out.println("Conversión exitosa. Número ingresado: " + numero);
30
31         } catch (NumberFormatException e) {
32             // Captura específica: Si el texto no puede convertirse a entero.
33             System.out.println("ERROR: La cadena ingresada no tiene el formato numérico correcto.");
34             System.out.println("Detalles del error: " + e.getMessage());
35
36         } catch (Exception e) {
37             // Captura cualquier otro error inesperado (ej. NullPointerException)
38             System.out.println("ERROR INESPERADO: Ocurrió un problema durante la lectura.");
39
40         }
41
42         System.out.println("El programa finalizó de forma controlada.");
43     }
44 }
```

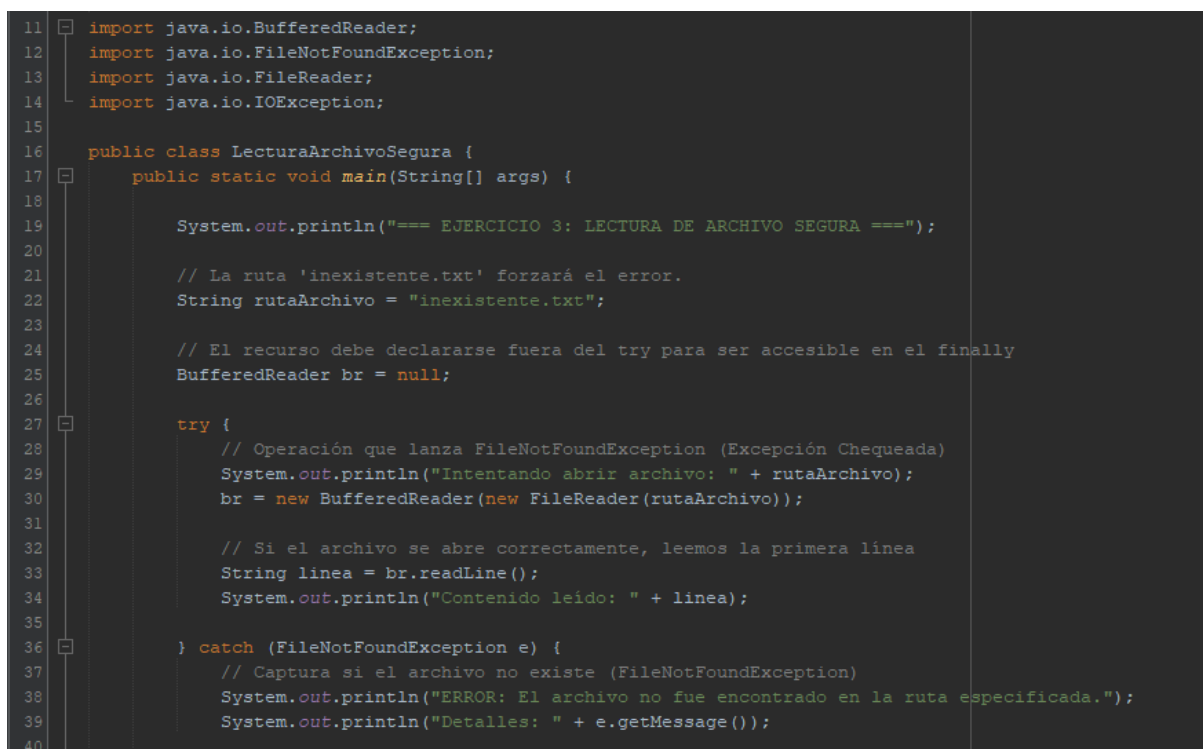
Output:



```
run:
=== EJERCICIO 2: CONVERSION DE CADENA A NUMERO ===
Ingrese una linea de texto (ej. '123' o 'hola'): hola
ERROR: La cadena ingresada no tiene el formato numerico correcto.
Detalles del error: For input string: "hola"
El programa finalizó de forma controlada.
BUILD SUCCESSFUL (total time: 13 seconds)
```

2.3

LecturaArchivoSegura:



```
11 import java.io.BufferedReader;
12 import java.io.FileNotFoundException;
13 import java.io.FileReader;
14 import java.io.IOException;
15
16 public class LecturaArchivoSegura {
17     public static void main(String[] args) {
18
19         System.out.println("=== EJERCICIO 3: LECTURA DE ARCHIVO SEGURA ===");
20
21         // La ruta 'inexistente.txt' forzará el error.
22         String rutaArchivo = "inexistente.txt";
23
24         // El recurso debe declararse fuera del try para ser accesible en el finally
25         BufferedReader br = null;
26
27         try {
28             // Operación que lanza FileNotFoundException (Excepción Chequeada)
29             System.out.println("Intentando abrir archivo: " + rutaArchivo);
30             br = new BufferedReader(new FileReader(rutaArchivo));
31
32             // Si el archivo se abre correctamente, leemos la primera línea
33             String linea = br.readLine();
34             System.out.println("Contenido leído: " + linea);
35
36         } catch (FileNotFoundException e) {
37             // Captura si el archivo no existe (FileNotFoundException)
38             System.out.println("ERROR: El archivo no fue encontrado en la ruta especificada.");
39             System.out.println("Detalles: " + e.getMessage());
40         }
```



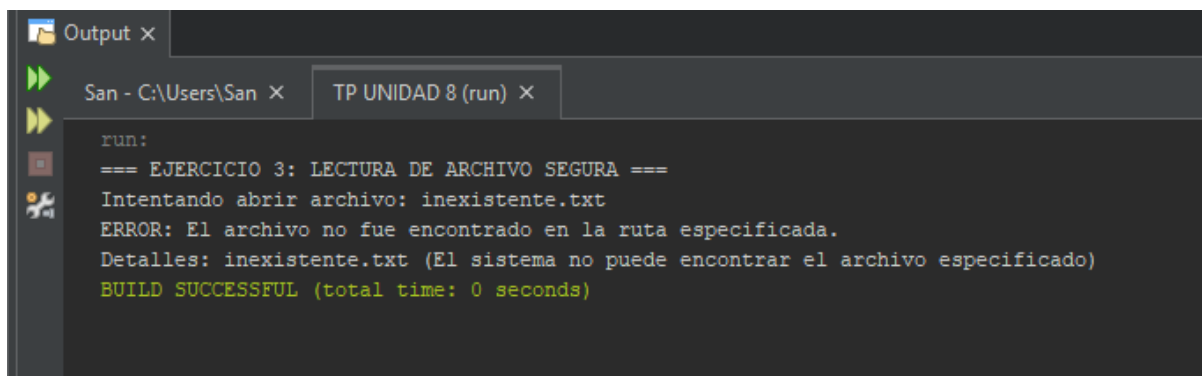
```

41         } catch (IOException e) {
42             // Captura errores de lectura/escritura que no sean la inexistencia del archivo
43             System.out.println("ERROR DE IO: Ocurrió un problema al leer el contenido.");
44         }
45     } finally {
46         // Bloque finally: Garantiza que el recurso se cierre siempre
47         if (br != null) {
48             try {
49                 br.close();
50                 System.out.println("Recurso cerrado en el bloque finally.");
51             } catch (IOException ex) {
52                 // Manejo del error si el cierre falla (se ignora o se registra)
53             }
54         }
55     }
56 }
57 }

```

Output:

Archivo no encontrado:



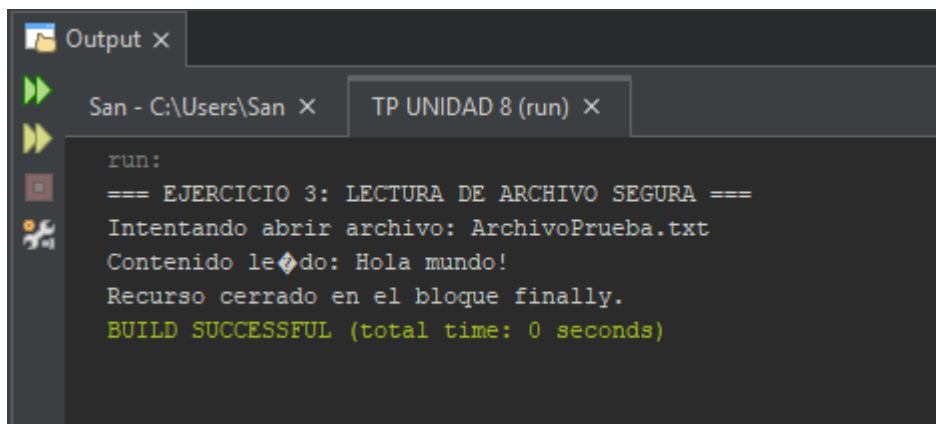
The screenshot shows an IDE output window with the following text:

```

run:
=== EJERCICIO 3: LECTURA DE ARCHIVO SEGURA ===
Intentando abrir archivo: inexistente.txt
ERROR: El archivo no fue encontrado en la ruta especificada.
Detalles: inexistente.txt (El sistema no puede encontrar el archivo especificado)
BUILD SUCCESSFUL (total time: 0 seconds)

```

Archivo Leído Exitosamente:



The screenshot shows an IDE output window with the following text:

```

run:
=== EJERCICIO 3: LECTURA DE ARCHIVO SEGURA ===
Intentando abrir archivo: ArchivoPrueba.txt
Contenido leído: Hola mundo!
Recurso cerrado en el bloque finally.
BUILD SUCCESSFUL (total time: 0 seconds)

```

2.4

Clase EdadInvalidaException:

```
5  package tp.unidad.pkg8;
6
7  /**
8   *
9   * @author San
10  */
11  // Clase que define la excepción personalizada
12  public class EdadInvalidaException extends RuntimeException {
13
14      // Solo necesitamos el constructor que recibe el mensaje
15      public EdadInvalidaException(String message) {
16          super(message);
17      }
18  }
```

ValidadorEdad:

```
11  public class ValidadorEdad {
12
13      // Este método lanza la excepción si la edad es inválida
14      public static void validar(int edad) {
15          if (edad <= 0 || edad >= 120) {
16              // Se utiliza 'throw new' para lanzar nuestra excepción personalizada
17              throw new EdadInvalidaException("Ingreso una edad invalida");
18          }
19      }
20  }
```

Main:

```
13 public class MainExcepcion {
14     public static void main(String[] args) {
15
16         // Usamos un bloque try para intentar la operación que puede fallar
17         try (Scanner scan = new Scanner(System.in)) {
18
19             System.out.println("Ingrese una edad");
20
21             // Leemos la entrada del usuario
22             int edad = Integer.parseInt(scan.nextLine());
23
24             // 1. Lógica para forzar la excepción si la edad es 124 (o cualquier valor fuera de 0-120)
25             if (edad <= 0 || edad >= 120) {
26                 // Aquí llamamos directamente a la clase de excepción para obtener el formato deseado
27                 throw new EdadInvalidaException("Ingreso una edad invalida");
28             }
29
30         } catch (NumberFormatException e) {
31             System.out.println("Error: El valor ingresado no es un número válido.");
32
33         } catch (EdadInvalidaException e) {
34             // 2. Bloque catch que captura nuestra excepción personalizada.
35             // La salida debe ser exactamente como la pide el profesor: Nombre de la Excepción y el mensaje.
36             System.out.println("Exception in thread \"main\" Excepciones.EdadInvalidaException: " + e.getMessage());
37
38         } finally {
39             System.out.println("\nCommand execution failed.");
40         }
41     }
42 }
```

Output:

```
Output x
San - C:\Users\San x TP UNIDAD 8 (run) x
run:
Ingrese una edad
124
Exception in thread "main" Excepciones.EdadInvalidaException: Ingreso una edad invalida

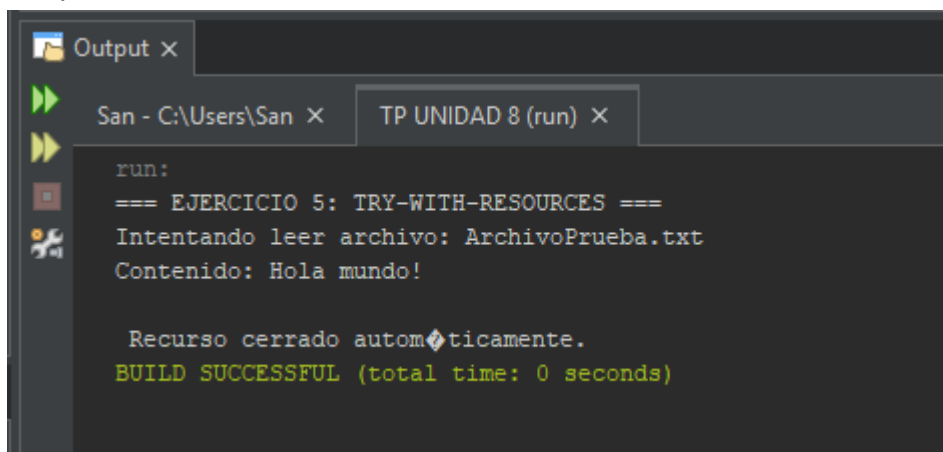
Command execution failed.
BUILD SUCCESSFUL (total time: 7 seconds)
```

2.5

TryWithResourceDemo:

```
21 public class TryWithResourcesDemo {
22     public static void main(String[] args) {
23
24         System.out.println("=== EJERCICIO 5: TRY-WITH-RESOURCES ===");
25
26         // Simplemente para dar un nombre al archivo, como en el ejemplo del profesor
27         String nombreArchivo = "ArchivoPrueba.txt";
28
29         // Se utiliza la clase File
30         File archivo = new File(nombreArchivo);
31
32         // Implementación de la solución oficial: encadenando File e InputStream
33         // Java se encarga de cerrar ambas instancias al salir del try.
34         try (FileReader fr = new FileReader(archivo);
35             BufferedReader reader = new BufferedReader(fr)) {
36
37             System.out.println("Intentando leer archivo: " + nombreArchivo);
38
39             String line;
40             while ((line = reader.readLine()) != null) {
41                 System.out.println("Contenido: " + line);
42             }
43
44         } catch (IOException e) {
45             System.out.println("ERROR: Ocurrió un problema de I/O (Input/Output).");
46             System.out.println("Detalles: " + e.getMessage());
47         }
48
49         System.out.println("\nRecurso cerrado automáticamente.");
50     }
51 }
```

Output:



The screenshot shows the 'Output' window of a Java IDE. It contains the following text:

```
run:
=== EJERCICIO 5: TRY-WITH-RESOURCES ===
Intentando leer archivo: ArchivoPrueba.txt
Contenido: Hola mundo!

Recurso cerrado automáticamente.
BUILD SUCCESSFUL (total time: 0 seconds)
```