UCSM Esc. Prof. de Ingeniería de Sistemas

Agosto - 2024

INFORME DE PRÁCTICAS

LENGUAJES DE PROGRAMACIÓN II

Práctica N° 4

Elaborado por:

Cusirramos Chiri, Santiago Jesus

RECONOCIMIENTOS

El autor de este trabajo reconoce con gratitud a los creadores de los lenguajes JAVA y otras personalidades y autores de libros de programación Bjarne Stroustrup, Dennis Ritchie, Herb Sutter, Herb Sutter, James Gosling, James Gosling, Brian Kernighan, Brian Kernighan, Ken Thompson.

PALABRAS CLAVES

- ✓ Excepción
- ✓ Jerarquía de clases
- ✓ Manejo de excepciones
- ✓ Excepciones en Java
- ✓ Cierre de recursos
- ✓ Excepciones comprobadas (checked)
- ✓ Excepciones no comprobadas (unchecked)

ÍNDICE

1. RESÚMEN 1
2. INTRODUCCIÓN 1
3. MARCO TEÓRICO 1
4. MODELAMIENTO – ACTIVIDAD 1
5. ACTIVIDADES 1
6. EJERCICIOS DE PRÁCTICA 6
7. CONCLUSIONES DE LA PRÁCTICA: 21
8. CUESTIONARIO 22
9. Anexos 24
10. BIBLIOGRAFÍA 24

1. RESÚMEN

Java utiliza excepciones para manejar errores de forma eficiente durante la ejecución de un programa. Estas excepciones permiten que el programa no se detenga abruptamente cuando ocurre un problema inesperado. Existen dos tipos principales: las excepciones comprobadas, que deben ser gestionadas obligatoriamente, y las no comprobadas, que no requieren declaración previa.

El manejo de excepciones en Java se realiza mediante bloques `try`, `catch` y `finally`, que permiten capturar errores y ejecutar código de limpieza final. Además, con `try-with-resources`, Java facilita el cierre automático de recursos como archivos o conexiones.

También es posible crear excepciones personalizadas para adaptarse a necesidades específicas de un programa, proporcionando mayor control y precisión en la gestión de errores.

2. INTRODUCCIÓN

En Java, el manejo de excepciones es crucial para crear programas robustos y fiables. Las excepciones son eventos que interrumpen el flujo normal de ejecución de un programa cuando ocurre un error. Java ofrece un sistema de manejo de excepciones estructurado que se basa en una jerarquía de clases, permitiendo a los desarrolladores capturar y gestionar errores de manera efectiva.

Existen dos tipos principales de excepciones en Java: las comprobadas (checked) y las no comprobadas (unchecked). Las excepciones comprobadas deben ser declaradas en la firma del método o capturadas dentro del método, mientras que las excepciones no comprobadas, como las derivadas de `RuntimeException`, no requieren declaración explícita. Este sistema ayuda a garantizar que los errores sean manejados adecuadamente.

Además, Java permite a los programadores definir sus propias excepciones para casos específicos no cubiertos por las excepciones estándar. La declaración `try-with-resources` es otra característica importante, ya que facilita el manejo automático del cierre de recursos como archivos o conexiones, asegurando que se liberen correctamente incluso si se produce una excepción.

Comprender cómo lanzar, capturar y manejar excepciones, así como el diseño de excepciones personalizadas y el uso de `try-with-resources`, es fundamental para desarrollar aplicaciones en Java que sean tanto eficientes como fáciles de mantener. Este enfoque integral al manejo de errores contribuye a un código más limpio y a una mayor estabilidad en la ejecución de programas.

3. MARCO TEÓRICO

En el campo de la programación, el entendimiento de los conceptos básicos y las características de los lenguajes de programación es crucial para el desarrollo de aplicaciones eficientes y efectivas. El marco teórico siguiente proporciona una base para comprender cómo Java se posiciona en el ecosistema de lenguajes de programación y su relación con otros lenguajes populares.

1. Lenguajes de Programación

Lenguajes Compilados e Interpretados: Los lenguajes de programación se dividen en dos categorías principales: compilados e interpretados. Un lenguaje compilado, como C++ o Go, requiere un proceso de compilación previo a la ejecución. Este proceso convierte el código fuente en lenguaje de máquina, que es directamente ejecutado por la computadora. En contraste, los lenguajes interpretados, como Python o JavaScript, son convertidos a lenguaje de máquina en tiempo real, durante la ejecución del programa.

Java: Un Caso Especial: Java combina características de ambos enfoques. El código Java se compila a un formato intermedio llamado bytecode, que es independiente de la plataforma. Este bytecode es luego interpretado por la Java Virtual Machine (JVM), permitiendo que el código Java sea ejecutado en cualquier plataforma que tenga una JVM instalada. Este enfoque permite a Java lograr portabilidad y flexibilidad, apoyando el principio de "escribir una vez, ejecutar en cualquier lugar" (WORA).

2. Características del Lenguaje Java

Portabilidad: Una de las principales ventajas de Java es su portabilidad. El bytecode generado por el compilador Java puede ejecutarse en cualquier plataforma que tenga una JVM compatible, lo que elimina la necesidad de recompilar el código para diferentes sistemas operativos y hardware.

Seguridad: Java proporciona un entorno de ejecución seguro mediante la JVM, que realiza una serie de verificaciones de seguridad durante la ejecución del bytecode. Estas verificaciones ayudan a prevenir la ejecución de código malicioso y a proteger el sistema operativo subyacente.

Robustez: Java incluye mecanismos avanzados de manejo de errores y excepciones, lo que contribuye a la estabilidad del programa. La recolección de basura automática también ayuda a gestionar la memoria de manera eficiente, reduciendo la posibilidad de errores relacionados con la memoria.

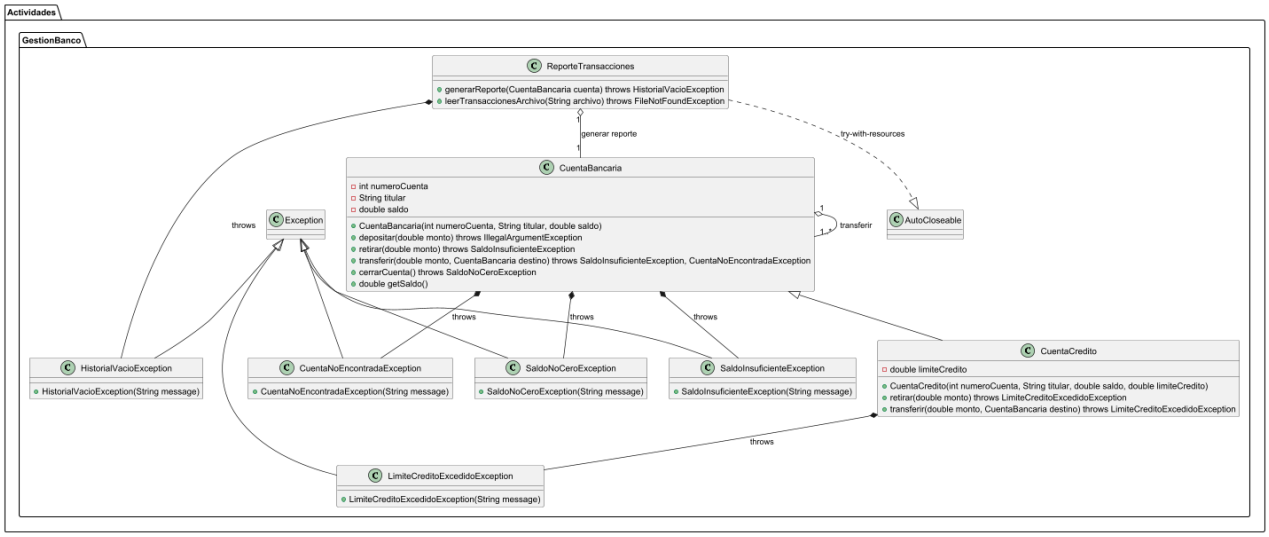
Multihilo: Java soporta la programación concurrente mediante su API de hilos, permitiendo que múltiples tareas se ejecuten simultáneamente dentro de un programa. Esto es especialmente útil para aplicaciones que requieren operaciones simultáneas, como servidores y aplicaciones de usuario.

3. Comparación con Otros Lenguajes

C++ vs. Java: C++ es un lenguaje compilado que ofrece control detallado sobre el hardware y la memoria, permitiendo una optimización más fina. Sin embargo, esto también puede resultar en una mayor complejidad y una mayor posibilidad de errores relacionados con la memoria. Java, por otro lado, proporciona una capa adicional de seguridad y manejo automático de memoria, lo que facilita el desarrollo a expensas de un menor control sobre el hardware.

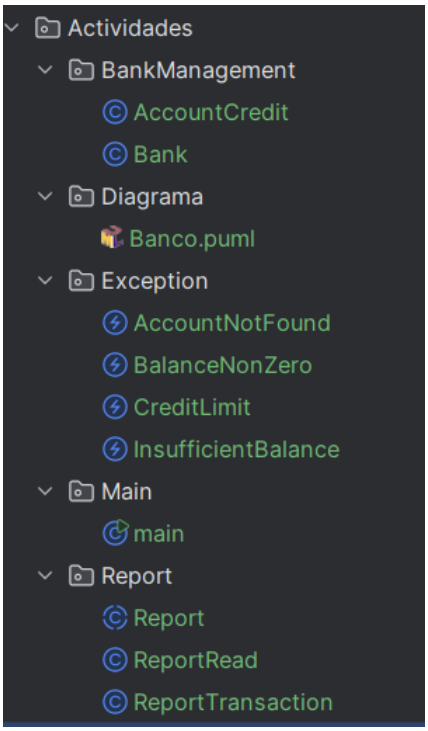
Python vs. Java: Python es un lenguaje interpretado conocido por su simplicidad y facilidad de uso, ideal para desarrollo rápido y scripting. Aunque Python permite un desarrollo ágil, su rendimiento puede ser inferior al de Java en aplicaciones de alto rendimiento debido a la naturaleza interpretada del lenguaje. Java, con su bytecode y ejecución en la JVM, ofrece un rendimiento más consistente y escalable para aplicaciones más grandes y complejas.

4. MODELAMIENTO – ACTIVIDAD



Este modelo fue la base para el programa, no lo implemente al 100%. Sin embargo, actuo como un diagrama del cual poder guiarme.

5. ACTIVIDADES



Este proyecto consiste en desarrollar un Sistema de Gestión de Cuentas Bancarias en Java, aplicando los principios de Programación Orientada a Objetos (POO) y el manejo de excepciones. El sistema permitirá crear cuentas bancarias, realizar operaciones como depósitos, retiros, transferencias entre cuentas, consultas de saldo, cierre de cuentas, y ofrecerá funcionalidades adicionales como la generación de reportes de transacciones y la gestión de límites de crédito

Requisitos Funcionales:

- Creación de cuentas bancarias.
- Depósito de dinero.
- Retiro de dinero.
- Transferencias entre cuentas.
- Cierre de cuentas si el saldo es cero.
- Consulta del saldo de una cuenta.
- Gestión de límites de crédito para cuentas especiales.
- Generación de reportes de transacciones.

Requisitos Técnicos:

- Manejo de excepciones proporcionadas por Java como IllegalArgumentException.
- Creación de excepciones personalizadas como SaldoInsuficienteException,
- CuentaNoEncontradaException, SaldoNoCeroException, y HistorialVacioException.
- Manejo de recursos utilizando bloques try-with-resources o finally para garantizar la correcta
- liberación de recursos, como archivos y Scanner.
- Aplicación de POO para modelar las operaciones y relaciones entre las clases.

Packet: BankManagement

Bank

```
package Actividades.BankManagement;
import Actividades.Exception.*;
import Actividades.Report.*;

import javax.security.auth.login.AccountNotFoundException;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;


class Bank {
    int accountNumber;
    String accountHolder;
    double balance = 0.0;
    protected static List<AccountCredit> account = new ArrayList<>();
    protected InsufficientBalance insufficientBalance;
    protected AccountNotFound accountNotFound;
    protected BalanceNonZero balanceNonZero;
    protected CreditLimit creditLimit;
    protected ReportTransaction reportTransaction;
    protected ReportRead reportRead;


    public Bank(int _accountNumber, String _accountHolder, double _balance) {
        this.accountNumber = _accountNumber;
        this.accountHolder = _accountHolder;
        checkDepositedBalance(_balance);
        this.insufficientBalance = new InsufficientBalance();
        this.accountNotFound = new AccountNotFound();
        this.balanceNonZero = new BalanceNonZero();
        this.creditLimit = new CreditLimit();
        this.reportTransaction = new ReportTransaction();
        this.reportRead = new ReportRead();
    }

    @Override     public String toString() {
        return "Account: " + accountNumber + " " + accountHolder  +" " + balance + "\n";
    }

    public void ReportReadTransaction() throws FileNotFoundException {
        reportRead.readFile();
    }

    public void checkDepositedBalance(double _balance) {
        if (_balance <= 0) {
            throw new IllegalArgumentException("Saldo insuficiente");
        }
        this.balance += _balance;
    }

    public void closeBankAccount(AccountCredit verifyAccount) throws BalanceNonZero,
AccountNotFoundException {
        accountNotFound.AccountNotFoundException(account, verifyAccount);
        balanceNonZero.BalanceNonZeroException(verifyAccount);
        verifyAccount = null;
        System.gc();
        accountNotFound.AccountNotFoundException(account, verifyAccount);
    }

    public int getAccountNumber() {
        return accountNumber;
    }
```

```
    public String getAccountHolder() {
        return accountHolder;
    }

    public double getBalance() {
        return balance;
    }

    public void setAccountNumber(int accountNumber) {
        this.accountNumber = accountNumber;
    }

    public void setAccountHolder(String accountHolder) {
        this.accountHolder = accountHolder;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }
}
```

AccountCredit

```
import Actividades.Exception.CreditLimit;
import javax.security.auth.login.AccountNotFoundException;

public class AccountCredit extends Bank {
    double credit;
    double bankDebt;

    public AccountCredit(int accountNumber, String accountHolder, double balance, double credit) {
        super(accountNumber, accountHolder, balance);
        this.credit = credit;
        this.bankDebt = 0;
    }

    public void balanceWithdrawal (double amount) throws CreditLimit {
        if (amount <= 0) {
            throw new IllegalArgumentException("El monto a retirar debe ser mayor a 0");
        }
        creditLimit.checkCreditLimit(credit, amount);
        this.balance -= amount;
    }

    public void bankTransfer(AccountCredit originAccount, AccountCredit destinationAccount, double
_balance) throws CreditLimit, AccountNotFoundException {
        accountNotFound.AccountNotFoundException(account, originAccount);
        accountNotFound.AccountNotFoundException(account, destinationAccount);
        originAccount.balanceWithdrawal(_balance);
        destinationAccount.checkDepositedBalance(_balance);
        reportTransaction.reportTransaction(originAccount, destinationAccount);
        System.out.println("Transaferencia completada");
    }

}
```

Packet: Exceptions.

AccountNotFound

```
package Actividades.Exception;

import Actividades.BankManagement.AccountCredit;
import javax.security.auth.login.AccountNotFoundException;
import java.util.List;

public class AccountNotFound extends Exception {
    public AccountNotFound() {
        super();
    }

    public void AccountNotFoundException(List<AccountCredit> account, AccountCredit
destinationAccount) throws AccountNotFoundException {
        if (account.contains(destinationAccount)) {
            throw new AccountNotFoundException("La cuenta no existe");
        }
    }
}
```

BalanceNonZero

```
import Actividades.BankManagement.AccountCredit;

public class BalanceNonZero extends Exception {
    public BalanceNonZero() {
        super();
    }

    public void BalanceNonZeroException(AccountCredit verifyAccount) throws BalanceNonZero {
        if (verifyAccount.getBalance() > 0.0) {
            throw new IllegalArgumentException("El saldo de la cuenta no es 0 ");
        }
    }
}
```

CreditLimit

```
public class CreditLimit extends Exception {
    public CreditLimit() {
        super();
    }

    public void checkCreditLimit(double credit, double amount) throws CreditLimit {
        if (credit < amount) {
            throw new IllegalArgumentException("Limite de credito excedido");
        }
    }
}
```

InsufficientBalance

```
public class InsufficientBalance extends Exception {
    public InsufficientBalance() {
        super();
    }
```

```
    public void checkBalance(double balance, double amount) throws InsufficientBalance {
        if (balance < amount) {
            throw new IllegalArgumentException("Saldo insuficiente");
        }
    }
}
```

Packet: Report

Report

```
import java.io.FileNotFoundException;

abstract class Report {
    static protected String pathName = "transactions.txt";

    void reportTransaction() {}
    void readFile() throws FileNotFoundException {}

}
```

ReportRead

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReportRead extends Report {

    public void readFile() throws FileNotFoundException {
        int count = 0;
        try (Scanner scan = new Scanner(new File(pathName))){
            while (scan.hasNextLine()) {
                System.out.println(scan.nextLine());
                count += 1;
            }
        } catch (FileNotFoundException e) {
            System.err.println("Archivo no encontrado: " + pathName);
        }
        System.out.println("Transacciones totales: " + count);
    }
}
```

ReportTransaction

```
import Actividades.BankManagement.AccountCredit;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ReportTransaction extends Report{
    public void reportTransaction(AccountCredit accountHolder, AccountCredit account) {
        try (BufferedWriter writerinformation = new BufferedWriter(new FileWriter(pathName, true))) {
            writerinformation.write(" Origen: " + accountHolder + " Destino: " + account.toString());
        } catch (IOException e) {
            System.out.println("Ocurrió un error al guardar en el archivo: " + e.getMessage());
        }
    }
}
```

6. EJERCICIOS DE PRÁCTICA

1. Sea una aplicación en la que debe crear cuatro clases de excepciones que representen respectivamente:

• Excepción de vocal

```
public class ExcepcionVocal extends Exception {
    public ExcepcionVocal(char c) {
        super("Excepción: El carácter '" + c + "' es una vocal.");
    }
}
```

• Excepción de número

```
public class ExcepcionNumero extends Exception {
    public ExcepcionNumero(char c) {
        super("Excepción: El carácter '" + c + "' es un número.");
    }
}
```

• Excepción de blanco

```
public class ExcepcionBlanco extends Exception {
    public ExcepcionBlanco() {
        super("Excepción: El carácter es un espacio en blanco.");
    }
}
```

• Excepción de salida

```
public class ExcepcionSalida extends Exception {
    public ExcepcionSalida() {
        super("Excepción: Carácter de salida detectado. Terminando programa.");
    }
}
```

Requisitos del programa:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;

public class LeerEntrada {
    private Reader stream;
         public LeerEntrada(InputStream fuente) {
        stream = new InputStreamReader(fuente);
    }
         public char getChar() throws IOException {
        return (char) this.stream.read();
    }
}
```

```
import java.io.IOException;

class ProcesarEntrada {
    private LeerEntrada entrada;

    public ProcesarEntrada(LeerEntrada entrada) {
        this.entrada = entrada;
    }
```

```
    public void procesar() throws IOException, ExcepcionVocal , ExcepcionNumero, ExcepcionBlanco,
ExcepcionSalida {
        char c = entrada.getChar();

        if (c == 'q') {
            throw new ExcepcionSalida();

        } else if (Character.isDigit(c)) {
            throw new ExcepcionNumero(c);

        } else if (c == ' ') {
            throw new ExcepcionBlanco();

        } else if (esVocal(c)) {
            throw new ExcepcionVocal(c);

        } else {
            System.out.println("Carácter ingresado: " + c);
        }
    }

    private boolean esVocal(char c) {
        return "AEIOUaeiou".indexOf(c) != -1;
    }
}
```

Main

```
package Ejercicio.Ejer1;

import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        LeerEntrada entrada = new LeerEntrada(System.in);
        ProcesarEntrada procesador = new ProcesarEntrada(entrada);
        char c = ' ';

        System.out.println("Programa interactivo de lectura de caracteres.");
        System.out.println("Introduce caracteres (ingresa 'q' para salir):");

        while (true) {
            try {
                procesador.procesar();

            } catch (ExcepcionVocal e) {
                System.out.println(e.getMessage());

            } catch (ExcepcionNumero e) {
                System.out.println(e.getMessage());

            } catch (ExcepcionBlanco e) {
                System.out.println(e.getMessage());

            } catch (ExcepcionSalida e) {
                System.out.println(e.getMessage());
                break;

            } catch (IOException e) {
                System.out.println("Error de entrada/salida: " + e.getMessage());
            }
        }

        System.out.println("Programa finalizado.");
```

```
    }
}
```

2. Calculadora Básica: Crea una clase Calculadora que permita realizar operaciones matemáticas básicas como suma, resta, multiplicación y división. Implementa métodos para cada operación y asegúrate de que el método dividir lance una excepción personalizada DivisionPorCeroException si se intenta dividir por cero. Maneja las excepciones en el código de invocación utilizando bloques catch múltiples para capturar tanto IllegalArgumentException como ArithmeticException.

Calculadora.java

```
package Ejercicio.Ejer2;

public class Calculadora {

    public double sumar(double a, double b) {
        return a + b;
    }

    public double restar(double a, double b) {
        return a - b;
    }

    public double multiplicar(double a, double b) {
        return a * b;
    }

    public double dividir(double a, double b) throws DivisionPorCeroException {
        if (b == 0) {
            throw new DivisionPorCeroException();
        }

        return a / b;
    }
}
```

DivisionPorCeroException

```
public class DivisionPorCeroException extends Exception {
    public DivisionPorCeroException() {
        super("Error: No se puede dividir por cero.");
    }
}
```

ExcepcionSalida

```
public class ExcepcionSalida extends Exception {
    public ExcepcionSalida() {
        super("Salir del programa.");
    }
}
```

Main.

```
import java.util.InputMismatchException;
import java.util.Scanner;
```

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Calculadora calculadora = new Calculadora();
        double num1, num2;
        char operacion;

        while (true) {
            System.out.println("\n--- Calculadora Básica ---");
            System.out.println("Introduce el primer número");
            System.out.println("\nIntroduce 'exit' para terminar el programa: ");


            try {
                if (scanner.hasNextDouble()) {
                    num1 = scanner.nextDouble();
                } else {
                    String input = scanner.next();
                    if (input.equalsIgnoreCase("exit")) {
                        throw new ExcepcionSalida();
                    } else {
                        throw new InputMismatchException("Entrada inválida.");
                    }
                }

                System.out.print("Introduce el segundo número: ");
                num2 = scanner.nextDouble();

                System.out.print("Introduce la operación (+, -, *, /): ");
                operacion = scanner.next().charAt(0);

                double resultado = 0;

                switch (operacion) {
                    case '+':
                        resultado = calculadora.sumar(num1, num2);
                        break;
                    case '-':
                        resultado = calculadora.restar(num1, num2);
                        break;
                    case '*':
                        resultado = calculadora.multiplicar(num1, num2);
                        break;
                    case '/':
                        resultado = calculadora.dividir(num1, num2);
                        break;
                    default:
                        throw new IllegalArgumentException("Operación no válida. Elige +, -, * o
/.");
                }

                System.out.println("El resultado de la operación es: " + resultado);

            } catch (DivisionPorCeroException e) {
                System.out.println(e.getMessage());

            } catch (ExcepcionSalida e) {
                System.out.println(e.getMessage());
                break;

            } catch (IllegalArgumentException e) {
                System.out.println("Error: " + e.getMessage());

            } catch (ArithmeticException e) {
                System.out.println("Error aritmético: " + e.getMessage());

            } catch (InputMismatchException e) {
                System.out.println("Error de entrada: " + e.getMessage());
```

```
                scanner.nextLine();
            }

        }
        scanner.close();
    }
}
```

3. Gestión de Números Negativos: Implementa una clase Numero que tenga un atributo valor (double) y métodos para establecer y obtener el valor. Asegúrate de que el método setValor() lance una IllegalArgumentException si el valor es negativo. Maneja esta excepción correctamente en el código que utiliza la clase.

Numero.

```
package Ejercicio.Ejer3;

public class Numero {
    private double valor;

    public Numero(double valor) {
        setValor(valor);      }

    public void setValor(double valor) {
        if (valor < 0) {
            throw new IllegalArgumentException("El valor no puede ser negativo.");
        }
        this.valor = valor;
    }

    public double getValor() {
        return valor;
    }
}
```

Main.

```
package Ejercicio.Ejer3;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Numero numero;

        while (true) {
            try {
                System.out.print("Introduce un número: ");
                double valor = scanner.nextDouble();

                numero = new Numero(valor);
                System.out.println("Número seleccionado: " + numero.getValor());

            } catch (IllegalArgumentException e) {
                System.out.println("Excepción capturada: " + e.getMessage());
                System.out.print("¿Deseas salir del programa? (Y/N): ");
                String salida = scanner.next();

                if (salida.equalsIgnoreCase("Y")) {
                    System.out.println("Saliendo del programa...");
                    break;
```

```
                }
            }
        }
        scanner.close();
    }
}
```

4. Registro de Estudiantes: Diseña una clase RegistroEstudiantes que permita agregar estudiantes a un arreglo y buscar estudiantes por su nombre. Implementa el método agregarEstudiante() para que lance una IllegalArgumentException si el nombre es nulo o vacío, y el método buscarEstudiante() para que lance una NoSuchElementException si el estudiante no está en el arreglo. Asegúrate de manejar estas excepciones en el código de invocación utilizando bloques catch adecuados.

Estudiante.

```
package Ejercicio.Ejer4;

public class Estudiante {
    private String nombre;

    public Estudiante(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }
}
```

RegistroEstudiantes

```
package Ejercicio.Ejer4;

import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;


public class RegistroEstudiantes {
    private List<Estudiante> estudiantes;

    public RegistroEstudiantes() {
        estudiantes = new ArrayList<>();
    }

public void agregarEstudiante(String nombre) {
        if (nombre == null || nombre.isEmpty()) {
            throw new IllegalArgumentException("El nombre del estudiante no puede ser nulo o
vacío.");
        }
        estudiantes.add(new Estudiante(nombre));
        System.out.println("Estudiante agregado: " + nombre);
    }

    public Estudiante buscarEstudiante(String nombre) {
        for (Estudiante estudiante : estudiantes) {
            if (estudiante.getNombre().equalsIgnoreCase(nombre)) {
                return estudiante;
            }
        }
        throw new NoSuchElementException("Estudiante no encontrado: " + nombre);
    }

    public List<Estudiante> obtenerEstudiantes() {
```

```
        return estudiantes;
    }
}
```

Main

```
package Ejercicio.Ejer4;

import java.util.NoSuchElementException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        RegistroEstudiantes registro = new RegistroEstudiantes();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n--- Registro de Estudiantes ---");
            System.out.println("1. Agregar estudiante");
            System.out.println("2. Buscar estudiante");
            System.out.println("3. Salir");
            System.out.print("Elige una opción: ");
            int opcion = scanner.nextInt();
            scanner.nextLine();  // Limpiar el buffer

            switch (opcion) {
                case 1:
                    System.out.print("Introduce el nombre del estudiante: ");
                    String nombre = scanner.nextLine();
                    try {
                        registro.agregarEstudiante(nombre);
                    } catch (IllegalArgumentException e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 2:
                    System.out.print("Introduce el nombre del estudiante a buscar: ");
                    String nombreBuscado = scanner.nextLine();
                    try {
                        Estudiante estudiante = registro.buscarEstudiante(nombreBuscado);
                        System.out.println("Estudiante encontrado: " + estudiante.getNombre());
                    } catch (NoSuchElementException e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 3:
                    System.out.println("Saliendo del programa...");
                    scanner.close();
                    return;

                default:
                    System.out.println("Opción no válida. Intenta de nuevo.");
            }
        }
    }
}
```

5. Registro de Contactos: Creen el código para una clase Contacto, Gestor de contacto y otras 2 clases adicionales que guarden relación, definan como mínimo tres atributos que asociados a la clase contacto, la clase gestor de contacto debe tener una lista de contactos, creen funciones en la clase de gestor de contacto que permita ingresar nuevos contactos, modificar contactos existentes y eliminar contactos, entre otras. Consideren el manejo de errores y excepciones en su programa tanto al recibir los datos como al procesarlos

Teléfono

```
package Ejercicio.Ejer5;

public class Telefono {
    private String numero;

    public Telefono(String numero) {
        this.numero = numero;
    }

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }

    @Override     public String toString() {
        return numero;
    }
}
```

Direccion

```
public class Direccion {
    private String calle;
    private String ciudad;
    private String codigoPostal;

    public Direccion(String calle, String ciudad, String codigoPostal) {
        this.calle = calle;
        this.ciudad = ciudad;
        this.codigoPostal = codigoPostal;
    }

    public String getCalle() {
        return calle;
    }

    public void setCalle(String calle) {
        this.calle = calle;
    }

    public String getCiudad() {
        return ciudad;
    }

    public void setCiudad(String ciudad) {
        this.ciudad = ciudad;
    }

    public String getCodigoPostal() {
```

```
        return codigoPostal;
    }

    public void setCodigoPostal(String codigoPostal) {
        this.codigoPostal = codigoPostal;
    }

    @Override     public String toString() {
        return calle + ", " + ciudad + ", " + codigoPostal;
    }
}
```

Contacto:

```
package Ejercicio.Ejer5;

public class Contacto {
    private String nombre;
    private String email;
    private Telefono telefono;
    private Direccion direccion;

    public Contacto(String nombre, String email, Telefono telefono, Direccion direccion) {
        this.nombre = nombre;
        this.email = email;
        this.telefono = telefono;
        this.direccion = direccion;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Telefono getTelefono() {
        return telefono;
    }

    public void setTelefono(Telefono telefono) {
        this.telefono = telefono;
    }

    public Direccion getDireccion() {
        return direccion;
    }

    public void setDireccion(Direccion direccion) {
        this.direccion = direccion;
    }

    @Override     public String toString() {
        return "Nombre: " + nombre + ", Email: " + email + ", Telefono: " + telefono + ", Direccion:
" + direccion;
    }
}
```

GestorDeContactos

```java
import java.util.ArrayList;
import java.util.List;

public class GestorDeContactos {
    private List<Contacto> contactos;

    public GestorDeContactos() {
        this.contactos = new ArrayList<>();
    }

    public void agregarContacto(Contacto contacto) throws Exception {
        if (contacto == null) {
            throw new IllegalArgumentException("El contacto no puede ser nulo.");
        }
        if (contacto.getEmail() == null || contacto.getEmail().isEmpty()) {
            throw new IllegalArgumentException("El email del contacto no puede estar vacío.");
        }
        contactos.add(contacto);
    }

    public void modificarContacto(String email, Contacto nuevoContacto) throws Exception {
        for (int i = 0; i < contactos.size(); i++) {
            Contacto contacto = contactos.get(i);
            if (contacto.getEmail().equals(email)) {
                contactos.set(i, nuevoContacto);
                return;
            }
        }
        throw new Exception("Contacto no encontrado.");
    }

    public void eliminarContacto(String email) throws Exception {
        for (int i = 0; i < contactos.size(); i++) {
            Contacto contacto = contactos.get(i);
            if (contacto.getEmail().equals(email)) {
                contactos.remove(i);
                return;
            }
        }
        throw new Exception("Contacto no encontrado.");
    }

    public Contacto buscarContacto(String email) throws Exception {
        for (Contacto contacto : contactos) {
            if (contacto.getEmail().equals(email)) {
                return contacto;
            }
        }
        throw new Exception("Contacto no encontrado.");
    }

    public void listarContactos() {
        for (Contacto contacto : contactos) {
            System.out.println(contacto);
        }
    }

    public void menuGestionContactos() {
        System.out.println("\n--- Menú de Gestion de Contactos ---");
        System.out.println("1. Agregar contacto");
        System.out.println("2. Modificar contacto");
        System.out.println("3. Eliminar contacto");
        System.out.println("4. Buscar contacto");
        System.out.println("5. Listar contactos");
        System.out.println("6. Salir");
        System.out.print("Seleccione una opcion: ");
```

```
    }
}
```

Main.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        GestorDeContactos gestor = new GestorDeContactos();

        while (true) {
            gestor.menuGestionContactos();
            int opcion = scanner.nextInt();
            scanner.nextLine();

            switch (opcion) {
                case 1:
                    try {
                        System.out.print("Ingrese el nombre: ");
                        String nombre = scanner.nextLine();
                        System.out.print("Ingrese el email: ");
                        String email = scanner.nextLine();
                        System.out.print("Ingrese el número de teléfono: ");
                        String numeroTelefono = scanner.nextLine();
                        System.out.print("Ingrese la calle: ");
                        String calle = scanner.nextLine();
                        System.out.print("Ingrese la ciudad: ");
                        String ciudad = scanner.nextLine();
                        System.out.print("Ingrese el código postal: ");
                        String codigoPostal = scanner.nextLine();

                        Telefono telefono = new Telefono(numeroTelefono);
                        Direccion direccion = new Direccion(calle, ciudad, codigoPostal);
                        Contacto contacto = new Contacto(nombre, email, telefono, direccion);

                        gestor.agregarContacto(contacto);
                        System.out.println("Contacto agregado exitosamente.");
                    } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 2:
                    try {
                        System.out.print("Ingrese el email del contacto a modificar: ");
                        String email = scanner.nextLine();

                        System.out.print("Ingrese el nuevo nombre: ");
                        String nombre = scanner.nextLine();
                        System.out.print("Ingrese el nuevo número de teléfono: ");
                        String numeroTelefono = scanner.nextLine();
                        System.out.print("Ingrese la nueva calle: ");
                        String calle = scanner.nextLine();
                        System.out.print("Ingrese la nueva ciudad: ");
                        String ciudad = scanner.nextLine();
                        System.out.print("Ingrese el nuevo código postal: ");
                        String codigoPostal = scanner.nextLine();

                        Telefono telefono = new Telefono(numeroTelefono);
                        Direccion direccion = new Direccion(calle, ciudad, codigoPostal);
                        Contacto nuevoContacto = new Contacto(nombre, email, telefono, direccion);

                        gestor.modificarContacto(email, nuevoContacto);
                        System.out.println("Contacto modificado exitosamente.");
```

```
                    } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 3:
                    try {
                        System.out.print("Ingrese el email del contacto a eliminar: ");
                        String email = scanner.nextLine();
                        gestor.eliminarContacto(email);
                        System.out.println("Contacto eliminado exitosamente.");
                    } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 4:
                    try {
                        System.out.print("Ingrese el email del contacto a buscar: ");
                        String email = scanner.nextLine();
                        Contacto contacto = gestor.buscarContacto(email);
                        System.out.println("Contacto encontrado: " + contacto);
                    } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 5:
                    System.out.println("Lista de contactos:");
                    gestor.listarContactos();
                    break;

                case 6:
                    System.out.println("Saliendo del programa.");
                    scanner.close();
                    System.exit(0);
                    break;

                default:
                    System.out.println("Opción no válida. Inténtelo de nuevo.");
            }
        }
    }
}
```

6. Registro de Transacciones Inmobiliarias:

- Crea una clase Propiedad con atributos como dirección, precio y tamaño. Implementa un constructor que valide que el precio y el tamaño sean positivos. Lanza una excepción personalizada DatosInvalidosException si cualquiera de estos atributos no cumple con los requisitos.
- Implementa una clase TransaccionInmobiliaria que registre las transacciones de compra y venta de propiedades. Cada transacción debe tener un id, una Propiedad y un precio. Si el precio de la transacción es menor que el precio de la propiedad, lanza una excepción personalizada PrecioInferiorException.
- Implementa una clase ContratoAlquiler que maneje contratos de alquiler. La clase debe tener métodos para establecer la duración del contrato y el monto del alquiler. Si la duración del contrato es negativa o el monto del alquiler es cero o negativo, lanza una excepción personalizada ContratoInvalidoException.
- Implementa una clase HistorialDePagos que registre los pagos realizados por inquilinos. Cada pago debe tener una fecha y un monto. Si se intenta registrar un pago con una fecha futura o un monto negativo, lanza una excepción personalizada PagoInvalidoException

Packete: Exception

ContratoInvalidoException

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

DatosInvalidosException

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

PagoInvalidoException

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

PrecioInferiorException

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

Packete General.

Contrato Alquiler

```
import Ejercicio.Ejer6.Exception.ContratoInvalidoException;

public class ContratoAlquiler {
    private int duracion;
    private double montoAlquiler;

    public ContratoAlquiler(int duracion, double montoAlquiler) throws ContratoInvalidoException {
        if (duracion < 0) {
            throw new ContratoInvalidoException("La duración del contrato no puede ser negativa.");
        }
        if (montoAlquiler <= 0) {
            throw new ContratoInvalidoException("El monto del alquiler debe ser positivo.");
        }
        this.duracion = duracion;
        this.montoAlquiler = montoAlquiler;
    }
         public int getDuracion() {
        return duracion;
    }
```

```
    public double getMontoAlquiler() {
        return montoAlquiler;
    }

    @Override     public String toString() {
        return "Duración: " + duracion + " meses, Monto Alquiler: " + montoAlquiler;
    }
}
```

Historial de Pagos

```
import Ejercicio.Ejer6.Exception.PagoInvalidoException;

import java.util.Date;

public class HistorialDePagos {
    private Date fecha;
    private double monto;

    public HistorialDePagos(Date fecha, double monto) throws PagoInvalidoException {
        if (fecha.after(new Date())) {
            throw new PagoInvalidoException("La fecha del pago no puede ser futura.");
        }
        if (monto < 0) {
            throw new PagoInvalidoException("El monto del pago debe ser positivo.");
        }
        this.fecha = fecha;
        this.monto = monto;
    }

    public Date getFecha() {
        return fecha;
    }

    public double getMonto() {
        return monto;
    }

    @Override     public String toString() {
        return "Fecha: " + fecha + ", Monto: " + monto;
    }
}
```

Propiedad

```
import Ejercicio.Ejer6.Exception.DatosInvalidosException;

public class Propiedad {
    private String direccion;
    private double precio;
    private double tamaño;

    public Propiedad(String direccion, double precio, double tamaño) throws DatosInvalidosException {
        if (precio <= 0) {
            throw new DatosInvalidosException("El precio debe ser positivo.");
        }
        if (tamaño <= 0) {
            throw new DatosInvalidosException("El tamaño debe ser positivo.");
        }
        this.direccion = direccion;
        this.precio = precio;
        this.tamaño = tamaño;
    }
```

```
    public String getDireccion() {
        return direccion;
    }

    public double getPrecio() {
        return precio;
    }

    public double getTamaño() {
        return tamaño;
    }

    @Override     public String toString() {
        return "Dirección: " + direccion + ", Precio: " + precio + ", Tamaño: " + tamaño;
    }
}
```

Transaccion Inmobiliaria

```
import Ejercicio.Ejer6.Exception.PrecioInferiorException;

public class TransaccionInmobiliaria {
    private String id;
    private Propiedad propiedad;
    private double precio;

    public TransaccionInmobiliaria(String id, Propiedad propiedad, double precio) throws
PrecioInferiorException {
        if (precio < propiedad.getPrecio()) {
            throw new PrecioInferiorException("El precio de la transacción no puede ser menor que el
precio de la propiedad.");
        }
        this.id = id;
        this.propiedad = propiedad;
        this.precio = precio;
    }
         public String getId() {
        return id;
    }

    public Propiedad getPropiedad() {
        return propiedad;
    }

    public double getPrecio() {
        return precio;
    }

    @Override     public String toString() {
        return "ID: " + id + ", Propiedad: [" + propiedad + "], Precio: " + precio;
    }
}
```

Main.

```
package Ejercicio.Ejer6;

import Ejercicio.Ejer6.Exception.ContratoInvalidoException;
import Ejercicio.Ejer6.Exception.DatosInvalidosException;
import Ejercicio.Ejer6.Exception.PagoInvalidoException;
import Ejercicio.Ejer6.Exception.PrecioInferiorException;

import java.util.Date;
```

```
public class Main {
    public static void main(String[] args) {
        try {
            // Crear una propiedad
            Propiedad propiedad = new Propiedad("Calle Falsa 123", 150000, 120);
            System.out.println("Propiedad creada: " + propiedad);

            // Crear una transacción inmobiliaria
            TransaccionInmobiliaria transaccion = new TransaccionInmobiliaria("TX001", propiedad,
160000);
            System.out.println("Transacción creada: " + transaccion);

            // Crear un contrato de alquiler
            ContratoAlquiler contrato = new ContratoAlquiler(12, 1000);
            System.out.println("Contrato de alquiler creado: " + contrato);

            // Crear un historial de pagos
            HistorialDePagos pago = new HistorialDePagos(new Date(), 500);
            System.out.println("Pago registrado: " + pago);

        } catch (DatosInvalidosException | PrecioInferiorException | ContratoInvalidoException |
PagoInvalidoException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

7. CONCLUSIONES DE LA PRÁCTICA:

a) Una excepción en Java es un evento que interrumpe el flujo normal de ejecución de un programa. Se produce cuando ocurre un error o una situación inesperada, como una división por cero o un acceso a un archivo inexistente.

b) Java organiza sus excepciones en una jerarquía de clases, donde todas heredan de la clase base Throwable. Dentro de esta, Exception maneja errores recuperables, mientras que Error cubre fallos más graves del sistema.

c) Las excepciones se manejan mediante bloques try, catch y finally, permitiendo al programador anticipar errores y proporcionar soluciones sin interrumpir el programa. El uso de finally asegura la ejecución de código importante, como la liberación de recursos.

d) Existen dos tipos principales: las excepciones comprobadas (checked), que deben ser manejadas explícitamente, y las no comprobadas (unchecked), que son errores del tiempo de ejecución, como NullPointerException.

e) Java proporciona la declaración try-with-resources para gestionar automáticamente el cierre de recursos como archivos o conexiones, lo que evita fugas de memoria y otros errores relacionados con la falta de liberación de recursos.

f) Es posible crear excepciones personalizadas que hereden de Exception o RuntimeException para manejar errores específicos en nuestros programas, facilitando la identificación y el manejo de situaciones excepcionales particulares.

8. CUESTIONARIO

- ✓ ¿Qué sucede si intentas abrir un archivo para lectura que no existe? ¿Qué sucede si intentas abrir un archivo para escritura que no existe?

  Si intentamos abrir un archivo para READ que no existe, Java lanzará una excepción FileNotFoundException, ya que no puede encontrar el archivo especificado. Por otro lado, si un archivo para WRITE no existe, Java creará un nuevo archivo vacío. Si el archivo ya existía, su contenido será sobrescrito. (*FileReader, FileWrite (Java SE 17 &Amp; JDK 17)*, 2024)

- ✓ ¿Cuál es la diferencia entre lanzar una excepción y capturar una excepción?

  Lanzar una excepción significa crear un objeto de excepción y señalar que ha ocurrido un error. Esto se hace con la declaración throw. Capturar una excepción, en cambio, implica interceptar esa excepción usando un bloque catch para manejarla adecuadamente y evitar que el programa falle abruptamente. (*The Try Block (the Java™ Tutorials > Essential Java Classes > Exceptions)*, n.d.)

  Ejemplos.

```
public void checkDepositedBalance(double _balance) {
    if (_balance <= 0) {
        throw new IllegalArgumentException("Saldo insuficiente");
    }
    this.balance += _balance;
}
```

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

- ✓ ¿Qué es una excepción comprobada (checked)? ¿Qué es una excepción no comprobada (unchecked)? Da un ejemplo de cada una. ¿Qué excepciones necesitas declarar con la palabra reservada throws?

  Una **excepción comprobada (checked)** es aquella que el compilador exige manejar explícitamente, ya sea con un bloque try-catch o declarando que el método la puede lanzar usando throws. Ejemplo: IOException. Una **excepción no comprobada (unchecked)** es un error de tiempo de ejecución que no necesita ser declarado ni capturado. Ejemplo: NullPointerException.
  Debes declarar excepciones comprobadas con la palabra reservada throws.

- ✓ ¿Por qué no necesitas declarar que tu método podría lanzar una IndexOutOfBoundsException?

  IndexOutOfBoundsException porque es una excepción no comprobada (unchecked). (*Unchecked Exceptions — the Controversy (the JavaTM Tutorials > Essential Java Classes > Exceptions)*, n.d.)

- ✓ Cuando tu programa ejecuta una declaración throw, ¿qué declaración se ejecuta a continuación? ¿Qué sucede si una excepción no tiene una cláusula catch coincidente?

  Cuando se ejecuta una declaración throw, el flujo de ejecución salta al bloque catch correspondiente si existe uno adecuado. Si no hay una cláusula catch que coincida, la excepción propaga hacia arriba en la pila de llamadas hasta que encuentra un bloque adecuado o provoca el fallo del programa. (*How to Throw Exceptions (the JavaTM Tutorials > Essential Java Classes > Exceptions)*, n.d.)

- ✓ ¿Qué puede hacer tu programa con el objeto de excepción que recibe una cláusula catch?

  El programa puede obtener información detallada sobre la excepción, a partir de esto obtenemos un mensaje de error, registramos la información. (*Catching and Handling Exceptions (the JavaTM Tutorials > Essential Java Classes > Exceptions)*, n.d.)
  Como programadores podríamos tomar acciones para identificar y corregir todos los errores posibles, ahora un programa siempre tendrá errores, allí entra el uso del CATCH que permite decirnos un error, ya sea por parte del programador o del usuario.

- ✓ ¿Es el tipo del objeto de excepción siempre el mismo que el tipo declarado en la cláusula catch que lo captura? Si no, ¿por qué no?

  No necesariamente. Una excepción capturada puede ser de una subclase de la excepción declarada en el catch. Esto se debe a que las excepciones en Java siguen una jerarquía de clases, por lo que una cláusula catch puede capturar una excepción más específica. (*The Catch or Specify Requirement (the JavaTM Tutorials > Essential Java Classes > Exceptions)*, n.d.)

- ✓ ¿Cuál es el propósito de la declaración try-with-resources? Da un ejemplo de cómo se puede usar.

  Como yo entiendo el uso del try-with-resources, se uso para identificar los errores del programa de manera automática, usado para el cierro de los recursos como archivos y conexiones.

```
try {
    Italo.ReportReadTransaction();
} catch (FileNotFoundException e) {
    throw new RuntimeException(e);
}
```

- ✓ ¿Qué sucede cuando se lanza una excepción, una declaración try-with-resources llama a close, y esa llamada lanza una excepción de un tipo diferente al de la original? ¿Cuál de las excepciones es capturada por una cláusula catch que rodea el bloque? Escribe un programa de ejemplo para probarlo.
  La excepción generada por el método close() se considera una excepción suprimida. Esta excepción suprimida no se pierde, sino que puede ser accedida a través del método getSuppressed() en la excepción principal.

```
try (MyResource resource = new MyResource()) {
            throw new Exception("Error en bloque try");
        } catch (Exception e) {
            System.out.println("Excepción capturada: " + e.getMessage());
            for (Throwable suppressed : e.getSuppressed()) {
                System.out.println("Excepción suprimida: " + suppressed.getMessage());
            }
        }
```

el programa lanza una excepción en el bloque try, seguida de otra excepción cuando se llama a close(). La excepción del bloque try es la que se captura en el bloque catch, mientras que la excepción de close() es suprimida y puede ser accedida mediante getSuppressed(). (*Chapter 14. Blocks and Statements*, n.d.)

- ✓ ¿Qué excepciones pueden lanzar los métodos next y nextInt de la clase Scanner? ¿Son excepciones comprobadas (checked) o no comprobadas (unchecked)?
  Los métodos next y nextInt pueden lanzar InputMismatchException, NoSuchElementException y IllegalStateException, todas ellas son excepciones unchecked.

9. Anexos

Practica4.zip

10. BIBLIOGRAFÍA

Hortsmann, C. (2015). Big Java. San José: Pearson. Oracle. (1994). The Java™ Tutorials. (Oracle) Recuperado el 28 de 07 de 2024, de https://docs.oracle.com/javase/tutorial/index.html

Programiz. (s.f.). Obtenido de https://www.programiz.com/java-programming/exception-handling baeldung. (24 de 05 de 2024). Exception Handling in Java. Obtenido de Baeldung: https://www.baeldung.com/java-exceptions

FileReader (Java SE 17 & JDK 17). (2024, July 11). https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/FileReader.html

FileWriter (Java SE 17 & JDK 17). (2024, July 11). https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/FileWriter.html

The try Block (The Java™ Tutorials > Essential Java Classes > Exceptions). (n.d.). https://docs.oracle.com/javase/tutorial/essential/exceptions/try.html

Unchecked Exceptions — the controversy (The JavaTM Tutorials > Essential Java Classes > Exceptions). (n.d.). https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

How to throw Exceptions (The JavaTM Tutorials > Essential Java Classes > Exceptions). (n.d.). https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Catching and handling Exceptions (The JavaTM Tutorials > Essential Java Classes > Exceptions). (n.d.). https://docs.oracle.com/javase/tutorial/essential/exceptions/handling.html

The catch or specify requirement (The JavaTM Tutorials > Essential Java Classes > Exceptions). (n.d.). https://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html

Chapter 14. Blocks and Statements. (n.d.). https://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.20.3