Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

до лабораторної роботи №4

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Виключення»

Варіант 5

Виконала:

ст. гр. КІ-305

Гринь С.М.

Прийняв:

Іванов Ю.С.

Львів 2023

Мета роботи: оволодіти навиками використання механізму виключень при написанні програм мовою Java.

Завдання:

1. Створити клас, що реалізує метод обчислення виразу заданого варіантом. Написати на мові Java та налагодити програму-драйвер для розробленого класу. Результат обчислень записати у файл. При написанні програми застосувати механізм виключень для виправлення помилкових ситуацій, що можуть виникнути в процесі виконання програми. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
   2. Автоматично згенерувати документацію до розробленого пакету.
   3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
   4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС. 5. Дати відповідь на контрольні запитання.
   Варіант 5: y=2x/sin(x)

Код програми:

Class Equations:

```
package KI305.Hryn.Lab4;


/**
 * The Equations class provides methods for calculating mathematical equations.
 */
public class Equations
{
    /**
     * Calculates the value of a mathematical equation based on the input parameter.
     *
     * @param x The input value in degrees.
     * @return The result of the equation.
     * @throws CalcException If an exception occurs during calculation.
     */
    public double calculate(int x) throws CalcException
    {
        double y, rad;
        rad = x * Math.PI / 180.0;
```

```
        try
        {
            y = 2 * x / Math.sin(rad);
            if (Double.isNaN(y) || Double.isInfinite(y) || x == 90 || x == -90)
                throw new ArithmeticException();
        }
        catch (ArithmeticException ex)
        {
            if (rad == Math.PI / 2.0 || rad == -Math.PI / 2.0)
                throw new CalcException("Exception reason: Illegal value of X for sin calculation");
            else if (x == 0)
                throw new CalcException("Exception reason: X = 0");
            else
                throw new CalcException("Unknown reason of the exception during exception
calculation");
        }
        return y;
    }
}
```

Class CalcException:

```
package KI305.Hryn.Lab4;

/**
 * The CalcException class is a custom exception class that extends ArithmeticException.
 * It is used to handle exceptions related to equation calculations in the Equations class.
 */
class CalcException extends ArithmeticException
{
    /**
     * Default constructor for CalcException.
     */
    public CalcException(){}

    /**
     * Constructor for CalcException with a custom error message.
     *
```

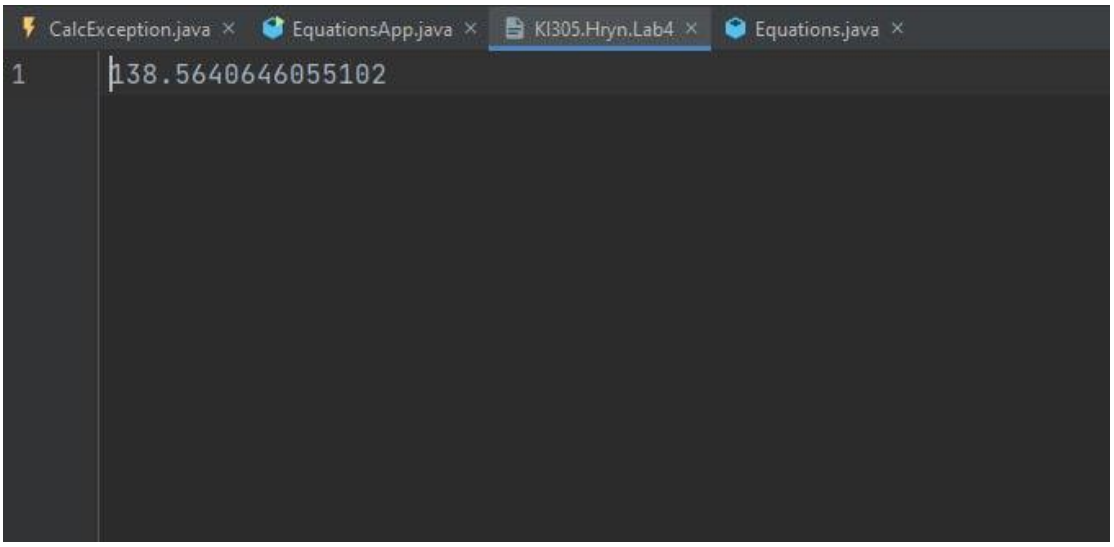
```
     * @param cause A string describing the cause of the exception.
     */
    public CalcException(String cause)
    {
        super(cause);
    }
}
```

Class EquationsApp:

```
package KI305.Hryn.Lab4;

import java.util.Scanner;
import java.io.*;
import static java.lang.System.out;

/**
 * The EquationsApp class is a command-line application for calculating and writing equation results to a file.
 */
public class EquationsApp {

    /**
     * The main method of the application.
     *
     * @param args Command-line arguments (not used in this application).
     */
    public static void main(String[] args)
    {
        try
        {
            out.print("Enter file name: ");
            Scanner in = new Scanner(System.in);
            String fName = in.nextLine();
            PrintWriter fout = new PrintWriter(new File(fName));
            try
            {
                try
                {
                    Equations eq = new Equations();
                    out.print("Enter X: ");
                    fout.print(eq.calculate(in.nextInt()));
                }
                finally
                {
                    fout.flush();
                    fout.close();
                }
            }
            catch (CalcException ex)
            {
                out.print(ex.getMessage());
```

```
        }
    }
    catch (FileNotFoundException ex)
    {
        out.print("Exception reason: Perhaps wrong file path");
    }
  }
}
```

Результат роботи програми:



Package KI305.Hryn.Lab4

package KI305.Hryn.Lab4

Classes

Class	Description
Equations	The Equations class provides methods for calculating mathematical equations.
EquationsApp	The EquationsApp class is a command-line application for calculating and writing equation results to a file.

Package KI305.Hryn.Lab4

Class Equations

java.lang.Object
KI305.Hryn.Lab4.Equations

public class **Equations**
extends Object

The Equations class provides methods for calculating mathematical equations.

Constructor Summary

Constructors

Constructor	Description
Equations()	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
double	calculate(int x)	Calculates the value of a mathematical equation based on the input parameter.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

Equations

public Equations()

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
double	calculate(int x)	Calculates the value of a mathematical equation based on the input parameter.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

Equations

public Equations()

Method Details

calculate

public double calculate(int x)
throws KI305.Hryn.Lab4.CalcException

Calculates the value of a mathematical equation based on the input parameter.

Parameters:
x - The input value in degrees.

Returns:
The result of the equation.

Throws:
KI305.Hryn.Lab4.CalcException - If an exception occurs during calculation.

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package KI305.Hryn.Lab4

Class EquationsApp

java.lang.Object
KI305.Hryn.Lab4.EquationsApp

```
public class EquationsApp
extends Object
```

The EquationsApp class is a command-line application for calculating and writing equation results to a file.

Constructor Summary

Constructors

Constructor	Description
EquationsApp()	

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	main(String[] args)	The main method of the application.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

EquationsApp

```
public EquationsApp()
```

Контрольні запитання

1. ***Дайте визначення терміну «виключення».***
   Виключення – це механізм мови Java, що забезпечує негайну передачу керування блоку коду опрацювання критичних помилок при їх виникненні уникаючи процесу розкручування стеку.
2. ***У яких ситуаціях використання виключень є виправданим?***
   Генерація виключень застосовується при:
   • помилках введення, наприклад, при введенні назви неіснуючого файлу або Інтернет адреси з подальшим зверненням до цих ресурсів, що призводить до генерації помилки системним програмним забезпеченням;
   • збоях обладнання;
   • помилках, що пов’язані з фізичними обмеженнями комп’ютерної системи, наприклад, при заповненні оперативної пам’яті або жорсткого диску;
   • помилках програмування, наприклад, при некоректній роботі методу, читанні елементів порожнього стеку, виходу за межі масиву тощо.
3. ***Яка ієрархія виключень використовується у мові Java?***

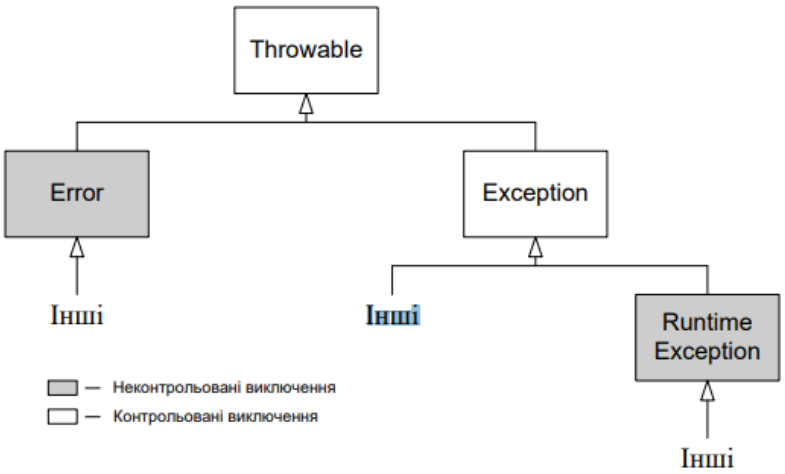


Рис. 1. Ієрархія класів виключень.

4. *Як створити власний клас виключень?*

Створення власних класів виключень

Як правило, власні класи контрольованих виключень використовуються для конкретизації виключних ситуацій, що генеруються стандартними класами контрольованих виключень, з метою їх точнішого опрацювання. Для створення власного класу контрольованих виключень необхідно обов'язково успадкувати один з існуючих класів контрольованих виключень та розширити його новою функціональністю. Найчастіше власні класи оснащують конструктором по замовчуванню та конструктором, що приймає детальний опис ситуації, яка призвела до генерації виключення. Для відображення опису помилкової ситуації можна використати метод `toString()` класу `Throwable`. Для цього необхідно викликати відповідний конструктор класу, що розширяється. Після цього створений клас можна застосовувати для генерації виключень.

Приклад власного класу виключень:

```
class FileFormatException extends IOException
{
      public FileFormatException()
      {}

      public FileFormatException(String message)
      {
            /*
            Даний виклик конструктора суперкласу дозволяє
            використовувати метод toString() класу Throwable
            */
            super(message);
      }
}
```

5. *Який синтаксис оголошення методів, що можуть генерувати виключення?*

Приклад оголошення методу, що може генерувати виключення:

```
public int loadData(String fName) throws EOFException, MalformedURLException
{
      …
}
```

6. *Які виключення слід вказувати у заголовках методів і коли?*

Виключення можуть генеруватися лише методами. Якщо метод може генерувати виключення певного класу, то назву цього класу слід вказати в заголовку методу після 5 ключового слова throws. Якщо метод може генерувати кілька видів виключень, то всі вони перелічуються через кому.

7. *Як згенерувати контрольоване виключення?*

Генерація контрольованих виключень

Генерація контрольованих виключень відбувається за допомогою ключового слова `throw` після якого необхідно вказати об'єкт класу виключення який і є власне виключенням, що генерує метод. Це можна зробити двома шляхами, використовуючи іменовані або анонімні об'єкти:

```
1. throw new IOException();

2. IOException ex = new IOException();
   throw ex;
```

Деякі класи контрольованих виключень мають конструктори, що приймають рядок з описом причини виникнення виключення. Такі конструктори корисно застосовувати для полегшення пошуку місця виникнення помилки під час виконання програми.

8. *Розкрийте призначення та особливості роботи блоку try.*

Основне призначення блоку try – це спроба виконати деяку частину коду, в якому може виникнути помилка.

Щоб виділити код, який потрібно відслідковувати на виникнення винятків, використовують ключове слово **try**. Після слова **try** пишеться блок, в якому розміщується програмний код, де можливе виникнення винятку(помилки виконання):

```
try {
 програмний_код
}
```

9. *Розкрийте призначення та особливості роботи блоку catch.*

Основне призначення блоку cath – це продовження виконання програми при виникненні помилки. В інших мовах програмування при виникненні виключної ситуації програма одразу завершується. Для одного блоку try може існувати довільна кількість блоків cath, оскільки один і той самий код, що розміщений в блоці try може генерувати різні види виключних ситуацій. Найбільш спеціалізовані блоки обробки виключних ситуацій повинні йти першими.

Після інструкції **try** обов'язково має бути інструкція перехоплення винятків, яка позначається ключовим словом **catch**. Після слова **catch** в дужках (параметри) вказується перехоплений об'єкт винятку, який був створений у блоці **try**. Після параметрів пишеться блок коду, який має виконатися при перехопленні винятку:

```
try { /* всередині цього блоку пишеться програмний код, який буде відслідковуватися на виникнення винятків */
 програмний_код
}
catch (об'єкт_винятку) { /* всередині цього блоку пишеться код, який має виконатися при перехоленні вказаного в параметрах об'єкту винятку */
 програмний_ код
}
```

10. *Розкрийте призначення та особливості роботи блоку finally.*

Блок **finally** виконується після try-catch незалежно від того, чи виник виняток. Це необов’язковий блок, але якщо немає блоку catch, то блок finally необхідний.
У цьому блоці можна, наприклад, закрити файл, який відкрито в блоці try, як у наведеному нижче коді.

```
import java.io.FileWriter;
import java.io.IOException;

public class FinallyTest {
    public static void main(String[] args) {
        FileWriter writer = null;
        try {
            writer = new FileWriter("out.txt");
            writer.write("Writing to the file!");
            System.out.println("Файл записаний вдало.");
        } catch (IOException e) {
            System.out.println("Помилка запису у файл.");
            e.printStackTrace();
        } finally {
            if ( writer != null ){
                try{
                  writer.close();
                } catch (IOException e) {
```

```
                    System. out.println("Помилка закриття файла.");
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Висновок: оволоділа навиками використання механізму виключень при написанні програм мовою Java.