Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

з лабораторної роботи №4

з дисципліни: “Кросплатформні засоби програмування”

на тему: “Виключення”

Виконав:

ст. гр. КІ-305

Ніколенко О.В.

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 розробленої програми.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання.

**Варіант 20**

|  |
| --- |
| y=tg(x)ctg(2x) |

**Лістинг програми:**

**EquationsApp.java**

package KI305.Nikolenko.Lab4;

import java.util.Scanner;

import java.io.\*;

import static java.lang.System.out;

/\*\*

\* Class <code>EquationsApp</code> Implements driver for Equations class

\* @author Oleksandr Nikolenko

\* @version 1.0

\*/

public class EquationsApp {

public static void main(String[] args) {

System.out.println("Success");

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: ");

int x = in.nextInt();

fout.print(eq.calculate(x));

out.print(eq.calculate(x));

}

finally

{

fout.flush();

fout.close();

}

}

catch (CalcException ex)

{

out.print(ex.getMessage());

}

}

catch (FileNotFoundException ex)

{

out.print("Exception reason: Perhaps wrong file path");

}

}

}

**Equations.java**

**package** package KI305.Nikolenko.Lab4;

/\*\*

\* Class <code>Equations</code> implements method for y=tg(x)ctg(2x) expression calculation

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*/

**class** Equations

{

**public** **double** calculate(**int** x) **throws** CalcException

{

**double** y, rad;

rad = x \* Math.***PI*** / 180.0;

**try**

{

y = Math.*tan*(rad) \* ( 1 / Math.*tan*(rad \* 2));

**if** (x==90 || x== -90 || x==0 || x==-180 || x==180 || 2\*x == 90 || 2\*x == -90 || 2\*x == 180 || 2\*x == -180 )

**throw** **new** ArithmeticException();

}

**catch** (ArithmeticException ex)

{

**if** (x==90 || x== -90 || x==-180 || x==180 || 2\*x == 90 || 2\*x == -90 || 2\*x == 180 || 2\*x == -180 )

**throw** **new** CalcException("Exception reason: Illegal value of X for tangent 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;

}

}

**CalcException.java**

**package** package KI305.Nikolenko.Lab4;

/\*\*

\* Class <code>CalcException</code> more precises ArithmeticException

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*/

**class** CalcException **extends** ArithmeticException

{

**public** CalcException(){}

**public** CalcException(String cause)

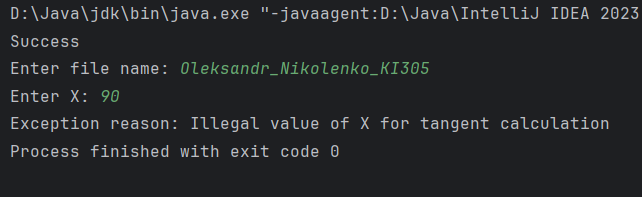
{

**super**(cause);

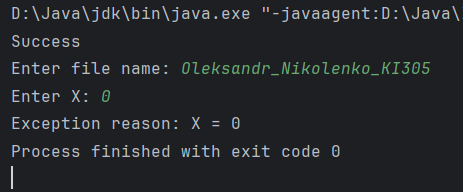
}

}

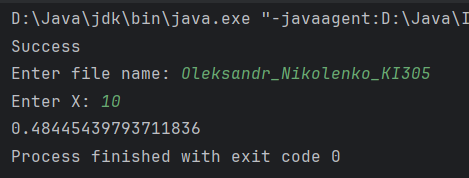
**Результат виконання програми:**



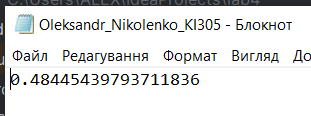
*Рис.1.Виключення для неприпустимого значення для котангеса*



*Рис.2.Виключення при значенні нуля*

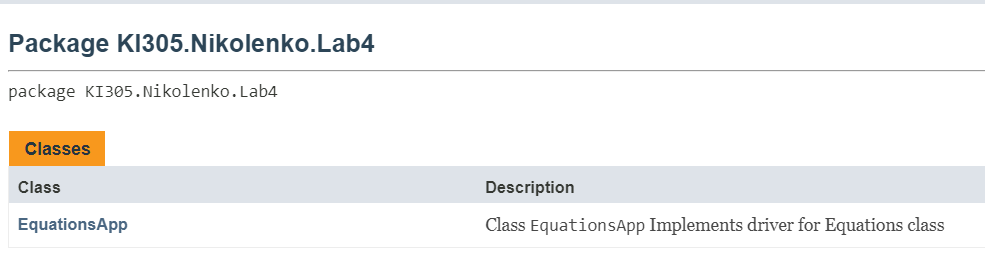
**

*Рис.3. Результат виконання програми*

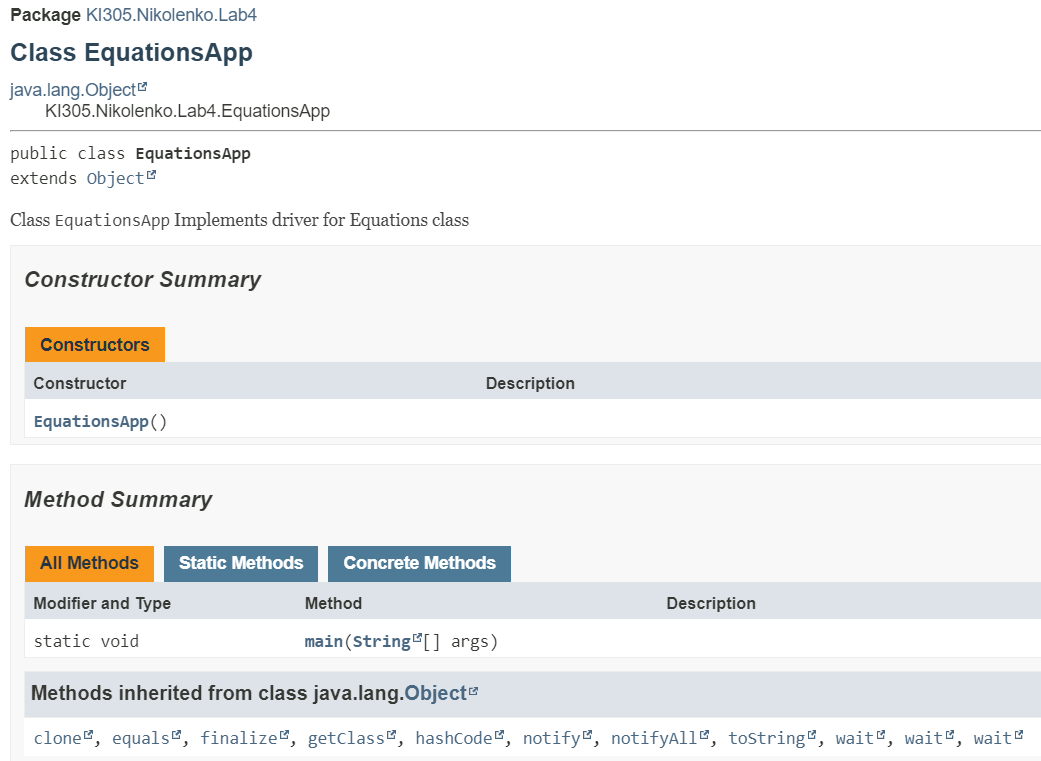
**

*Рис.4.Результат виконання програми у файлі*

**Згенерована документація**

****

*Рис.5.Вміст вкладки Package*

**

*Рис.6.Вміст вкладки Сlass(EquationsApp)*

**Відповіді на 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.

1. **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

1. **Яка ієрархія виключень використовується у мові Java?**

Всі виключення в мові Java поділяються на контрольовані і неконтрольовані та спадкуються від суперкласу Throwable. Безпосередньо від цього суперкласу спадкуються 2 класи Error і Exception.

Ієрархія класів, що спадкує клас Error, описує внутрішні помилки і ситуації, що пов’язані з браком ресурсів у системі підтримки виконання програм. Жоден об’єкт цього типу самостійно згенерувати неможна. При виникненні внутрішньої помилки можна лише відобразити повідомлення користувачу та спробувати коректно завершити виконання програми. Такі помилки є нечастими.

Ієрархія класів, що спадкує клас Exception поділяється на клас RuntimeException та інші. Виключення типу RuntimeException виникають внаслідок помилок програмування. Всі інші помилки є наслідком непередбачених подій, що виникають під час виконання коректної програми, наприклад, помилок вводу/виводу.

1. **Як створити власний клас виключень?**

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.

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

Приклад оголошення методу, що може генерувати виключення:

public int loadData(String fName) throws EOFException, MalformedURLException

{

…

}

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

Оголошення всіх можливих виключень, які може генерувати метод, є поганим стилем програмування. Оголошувати слід лише всі контрольовані виключення. Якщо цього не зробити, то компілятор видасть повідомлення про помилку. Якщо метод оголошує, що він може генерувати виключення певного класу, то він може також генерувати виключення і його підкласів.

1. **Як згенерувати контрольоване виключення?**

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:throw new IOException();

IOException ex = new IOException();

throw ex;

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

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

}

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

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

}

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

Інструкція finally застосовується після інструкцій try-catch в тому випадку, коли незалежно від того, чи перехоплюється виняток, чи не перехоплюється, чи програма прининяє своє виконання, потрібно щоб виконався певний програмний код

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

}

finally { /\* код в інструкції finally виконається незалежно від того, чи було перехоплено помилку, чи ні \*/

програмний\_ код

}

**Висновок:**

На цій лабораторній роботі я оволодів навиками використання механізму виключень при написанні програм мовою Java.