МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з об'эктно-орієнтованого програмування

(назва дисципліни)

на тему: розв’язання систем алгебраїчних рівнянь точними методами

Студента (ки, ів) 1-го курсу, групи ІП-91

Дзюбака Дем’яна Павловича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник: старший викладач, Головченко Максим Миколайович

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ- 2020 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Об'єктно-орієнтоване програмування

Спеціальність "Інженерія програмного забезпечення"

Курс 1 Група ІП-91 Семестр 2

**ЗАВДАННЯ**

**на курсову роботу студента**

|  |
| --- |
| Дзюбака Дем'яна Павловича |

(прізвище, ім’я, по батькові)

|  |  |
| --- | --- |
| 1. Тема роботи | Розв’язання систем лінійних алгебраїчних рівнянь методами Гауса, Жордана-Гауса та обертань |
|  | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи |  |

|  |  |
| --- | --- |
| 3. Вихідні дані до роботи | Перелік знайдених значень змінних до заданої системи лінійних алгебраїчних рівнянь |
|  | |
|  | |
|  | |

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
| Прийняття СЛАР шляхами зчитування з файлу, автоматичної генерації та введення |
| вручну. ирішення заданої СЛАР методами Гауса, Жордана-Гауса та обертань. Вивантаження |
| знайдених значень змінних заданої системи у файл. |
|  |

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

|  |
| --- |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 6. Дата видачі завдання |  |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Аналіз предметної області |  |  |
| 4. | Проектування архітектури програмної системи |  |  |
| 5. | Розробка сценарію роботи програми |  |  |
| 6. | Узгодження з керівником інтерфейсу користувача |  |  |
| 7. | Розробка програмного забезпечення |  |  |
| 8. | Узгодження з керівником плану тестування |  |  |
| 9. | Тестування програми |  |  |
| 10. | Підготовка пояснювальної записки |  |  |
| 11. | Здача курсової роботи на перевірку |  |  |
| 12. | Захист курсової роботи |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Дзюбак Дем’ян Павлович |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | Головченко Максим Миколайович |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020\_ р.

# Анотація

Пояснювальна записка до курсової роботи: 49 сторінок, 17 рисунки, 8 таблиць, 24 посилання.

Об’єкт дослідження: задача розв’язання системи лінійних алгебраїчних рівнянь.

Мета роботи: дослідження методів розв’язання систем лінійних рівнянь.

В курсовій роботі наявні теоретичні відомості, опис використовуваних алгоритмів, опис програмного забезпечення, інструкція користувача, дослідження програмного забезпечення на обробку помилок та аналіз результатів роботи програми. Код програми знаходиться у Додатку Б.

Вивчено методи вирішення систем лінійних алгебраїчних рівнянь, метод перевірки на єдиність вирішення, інструмент написання графічної частини програм на мові java JavaFX.

Виконана програмна реалізація методів вирішення СЛАР : Гауса, Жордана-Гауса, обертань з підрахунками арифметичних операцій. Реалізований графічний метод СЛАР розмірності 2. Розроблений графічний інтерфейс, що дозволяє вводити вхідну матрицю шляхами випадкової генерації, зчитуванням з файлу або введенням вручну, записувати отриманий результат в обраний файл.

СИСТЕМА ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ, СЛАР, МАТРИЦІ, МЕТОД ГАУСА, МЕТОД ЖОРДАНА-ГАУСА, МЕТОД ОБЕРТАНЬ.

**ЗМІСТ**

[Анотація 4](#_Toc74119605)

[ВСТУП 6](#_Toc74119606)

[1 ПОСТАНОВКА ЗАДАЧІ 7](#_Toc74119607)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc74119608)

[3. ОПИС АЛГОРИТМІВ 13](#_Toc74119609)

[2.1. Загальний алгоритм 13](#_Toc74119610)

[2.2. Алгоритм методу Гауса 14](#_Toc74119611)

[4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 20](#_Toc74119612)

[2.3. Діаграма класів програмного забезпечення 20](#_Toc74119613)

[2.4. Опис методів частин програмного забезпечення 22](#_Toc74119614)

[2.4.1. Стандартні методи 22](#_Toc74119615)

[2.4.2. Користувацькі методи 22](#_Toc74119616)

[5. ІНСТРУКЦІЯ КОРИСТУВАЧА 40](#_Toc74119617)

[5.1 Робота з програмою 40](#_Toc74119618)

[5.2 Формат вхідних та вихідних даних 43](#_Toc74119619)

[5.3 Системні вимоги 44](#_Toc74119620)

[6 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 45](#_Toc74119621)

[6.1. План тестування 45](#_Toc74119622)

[6.2. Приклади тестування 46](#_Toc74119623)

[7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ 49](#_Toc74119624)

[ВИСНОВОК 56](#_Toc74119625)

[ПЕРЕЛІК ПОСИЛАНЬ 57](#_Toc74119626)

[Додаток А Технічне завдання 58](#_Toc74119627)

[Додаток Б Тексти програмного коду 61](#_Toc74119628)

## ВСТУП

Курсова робота призначена для вирішення СЛАР методами Гаусса, Жордана-Гаусса та обертання автоматично.

Ціль роботи – розробити програмне забезпечення типу десктоп, що прийматиме матрицю коефіцієнтів та стовбець вільних членів СЛАР та коректно вирішуватиме її.

Сутність вирішуваної задачі – розробка інтерфейсу, що дозволятиме вводити дані та отримувати результат, який зв’язаний з програмною маделлю, що обраховуватиме СЛАР за отриманими даними та повертатиме результат.

# ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити рішення для заданої СЛАР наступними методами:

а) метод Гаусса;

б) метод Жордана-Гауса;

в) метод обертання;

Вхідними даними для даної роботи є СЛАР, яка задана в матричному вигляді:

де – матриця коефіцієнтів, – вектор шуканих значень (рішення системи),  
 – вектор вільних членів.Програмне забезпечення повинно обробляти матрицю коефіцієнтів та стовпець вільних членів для СЛАР розмірність яких знаходиться в межах від 2 до 8.

Вихідними даними для даної роботи являється сукупність дійсних чисел, що є розв’язками даної системи, які виводяться на екран. Програмне забезпечення повинно видавати розв’язок за умови, якщо для вхідних даних обраний метод сходиться. Якщо це не так, то програма повинна вивести відповідне повідомлення. Якщо розмірність системи не перевищує двох невідомих, то програмне забезпечення повинно виводити графік системи. Якщо система не має розв’язків або їх нескінченна кількість, то програма повинна видати відповідне повідомлення.

# ТЕОРЕТИЧНІ ВІДОМОСТІ

Метод Гаусса:

Нехай вихідна система виглядає наступним чином:

Її можна записати в матричному вигляді , де

, ,

Матриця A називається головною матрицею системи, х – вектор змінних, b - стовпцем вільних членів.

Тоді, відповідно до властивості елементарних перетворень над рядками, основну матрицю цієї системи можна привести до східчастого вигляду (ці ж перетворення потрібно застосовувати до стовпця вільних членів):

|  |  |
| --- | --- |
|  | (2.1) |

де

При цьому будемо вважати, що базисний мінор (ненульовий мінор максимального порядку) основної матриці знаходиться у верхньому лівому куті, тобто в нього входять тільки коефіцієнти при змінних .

Тоді змінні називаються головними змінними. Всі інші називаються вільними.

Якщо хоча б одне число , то розглянута система несумісна, т . е. у неї немає жодного рішення.

Нехай для будь-яких .

Перенесемо вільні змінні за знаки рівності і поділимо кожне з рівнянь системи на свій коефіцієнт при самому лівому з них:

{\displaystyle i=1,\ldots ,r,\quad k=i+1,\ldots ,n.}

|  |  |
| --- | --- |
|  | (2.2) |

Якщо вільним змінним системи (2.2) надавати всі можливі значення і вирішувати нову систему щодо головних невідомих знизу вгору (тобто від нижнього рівняння до верхнього), то ми отримаємо всі рішення цієї СЛАР. Так як ця система отримана шляхом елементарних перетворень над вихідною системою (2.1), то по теоремі про еквівалентність при елементарних перетвореннях системи (2.1) і (2.2) еквівалентні, тобто безлічі їх рішень збігаються.

*Метод Жордана-Гаусса:*

1. Вибирають перший зліва стовпець матриці, в якому є хоч одне відмінне від нуля значення.
2. Якщо саме верхнє число в цьому стовпці нуль, то змінюють всю перший рядок матриці з іншого рядком матриці, де в цій колонці немає нуля.
3. Всі елементи першого рядка ділять на верхній елемент вибраного стовпця.
4. З решти рядків віднімають перший рядок, помножену на перший елемент відповідного рядка, з метою отримати першим елементом кожного рядка (крім першої) нуль.
5. Далі проводять таку ж процедуру з матрицею, що виходить з вихідної матриці після викреслювання першого рядка і першого стовпця.
6. Після повторення цієї процедури n-1 раз отримують верхню трикутну матрицю
7. Вираховують із передостанній рядки останній рядок, помножену на відповідний коефіцієнт, з тим, щоб в передостанньому рядку залишилася тільки 1 на головній діагоналі.
8. Повторюють попередній крок для наступних рядків. В результаті отримують одиничну матрицю і рішення на місці вільного вектора (з ним необхідно проводити всі ті ж перетворення).

*Метод обертань:*

Як і в методі Гаусса, мета прямого ходу перетворень в цьому методі-приведення системи до трикутного виду послідовним обнуленням поддіагональних елементів спочатку першого стовпчика, потім другого і т.д.

|  |  |
| --- | --- |
|  | (2.3) |

Помножити перше рівняння вихідної системи (2.3) на з1, друге на s1 і скласти їх; отриманим рівнянням замінимо перше рівняння системи. Потім перше рівняння вихідної системи множимо на -s1, друге на c1 і результатом їх складання замінимо друге рівняння. Таким чином, перші два рівняння (2.3) замінюються рівняннями :

На параметри с1 та s1 накладемо дві умови :

* умова виключення х1 з другого рівняння і
* умова нормування.

Звідси:

Ці числа можна інтерпретувати як косинус і синус деякого кута (звідси назва метод обертання, кожен крок такого перетворення можна розглядати як обертання розширеної матриці системи в площині онуляюмого індексу).

В результаті перетворень отримаємо систему

Де

.

Далі перше рівняння системи замінюється новим, отриманим складання результатів множення першого і третього рівнянь відповідно на

а третє-рівнянням, отримане при складання результатів множення тих же

де

.

Виконавши перетворення m-1 раз, прийшовши до системи

Вид отриманої системи такий же, як після першого етапу перетворень методом Гаусса. Ця система має наступну властивість: довжина будь-якого вектора-стовпця (евклідова норма) Розширене матриці залишається такою ж, як у вихідній матриці. Отже, при виконанні перетворень не спостерігається зростання елементів.

Далі за цим же алгоритмом перетворюється матриця

і т.д.

В результаті m-1 етапів прямого ходу система буде приведена до трикутного виду.

Знаходження невідомих НЕ відрізняється від зворотного ходу методу Гаусса

# ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено в

таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| Matrix | Розширена матриця системи |
| result | Масив знайдений значень змінних системи |
| permittingElement | Головний елемент(у методі Жордана-Гауса) |
| *c, s* | Допоміжні елементи в методі обертань |

## Загальний алгоритм

1. ПОЧАТОК ;
2. Зчитати розмірність системи;
3. Зчитати матрицю системи та стовпець вільних членів:
   1. Зчитати матрицю системи:
      1. Цикл проходу по всіх рядках матриці системи (:
         1. Цикл проходу всіх стовпцях матриці системи   
            (:
            1. ЯКЩО поточний елемент матриці – вірно записане число, ТО записати його в відповідну комірку *matrix*. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 8;
   2. Зчитати стовпець вільних членів:
      1. Цикл проходу по всіх елементах стовпця вільного членів:
         1. Якщо поточний елемент стовпцю вільних членів – вірно записане число, ТО записати його в відповідну комірку *free*. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 8;
4. ЯКЩО обраний метод Гауса, ТО обробити дані згідно алгоритму методу Гауса (пункт 3.2);
5. ЯКЩО обраний метод Жордана-Гауса, ТО обробити дані згідно алгоритму методу Жордана-Гауса (пункт 3.3);
6. ЯКЩО обраний метод обертань, ТО обробити дані згідно алгоритму методу обертань(пункт 3.4);
7. ЯКЩО обраний метод сходиться для вхідних даних, ТО:
   1. ЯКЩО обрана система на дві невідомих, ТО побудувати та вивести графік системи;
   2. Вивести рішення системи;
   3. Записати систему та її рішення у файл;
8. КІНЕЦЬ.

## Алгоритм методу Гауса

1. ПОЧАТОК;
2. Шляхом елементарних перетворень знайти ранг розширеної та нерозширеної матриць:
   1. ЯКЩО ранг розширеної матриці більший за ранг основної ТО:

Вивести повідомлення : «Матриця несумісна, розв’язків не має»;

Перейти до пункту 7;

ІНАКШЕ ЯКЩО ранг розширеної матриці менший за ранг основної ТО :

Вивести повідомлення : «Матриця сумісна і має безліч розв’язків», перейти до пункту 7;

ІНАКШЕ

Продовжити;

1. Приведення матриці до канонічного (діагонального) вигляду (**stableMatrix**):
   * 1. Поділити елементи першого рядку на його перший елемент
     2. ЦИКЛ проходу по всіх рядках матриці (*Matrix*):
        1. ЦИКЛ проходу по всіх елементах поточного рядка у матриці (*Matrix*):
        2. Відняти відповідні елементи першого рядка, домножені на значення першого елемента в поточному рядку від елементів останнього;
2. Зворотній хід:
   1. Цикл проходу по всіх рядках матриці(*Matrix*), починаючи з останнього:
      1. Розділити поточний рядок на перше ненульове значення;
      2. Отримане значення в стовбці вільних членів помножити на усі елементи наступного по циклу рядка, крім першого ненульового;
3. Ініціювати масив чисел (*result*) довжиною кількості рядків матриці(*Matrix*);
4. Цикл проходу по індексам масиву (*result*) :
   1. Присвоїти значення відповідного елемента по індексу стовпця вільних членів елементу масиву (*result*);
5. Кінець.

Метод Жордана-Гауса :

1. ПОЧАТОК;
2. Шляхом елементарних перетворень знайти ранг розширеної та нерозширеної матриць:
   1. ЯКЩО ранг розширеної матриці більший за ранг основної ТО:

Вивести повідомлення : «Матриця несумісна, розв’язків не має»,

Перейти до пункту 4;

ІНАКШЕ ЯКЩО ранг розширеної матриці менший за ранг основної ТО :

Вивести повідомлення : «Матриця сумісна і має безліч розв’язків», перейти до пункту 4;

ІНАКШЕ:

Продовжити;

1. Приведеня основної матриці до одиничного:
   1. Цикл(1) проходу по всіх рядках матриці(*Matrix*):
      1. Ініціювати чисельну змінну (*permittingElement*) ;
      2. Надати їй значення елементу поточного рядка з індексом стовпця поточного рядка(елемент головної діагоналі) ;
      3. Цикл проходу по всіх елементах поточного рядка матриці(*Matrix*):
         1. Розділити елемент поточного індекса по стовбцю на (*permittingElement*);
      4. Надати змінній (*permittingElement*) значення елементу головної діагоналі поточного рядка;
      5. Цикл проходу по елементах рядка заданої матриці (*Matrix*)(починаючи з індексу, на один більший за поточний індекс рядка):
         1. Цикл(2) походу по стовбцях матриці (*Matrix*):
            1. Якщо індекс циклу(2) не дорівнює індексу циклу(1);

Помножити (*permittingElement*) на елемент матриці (*Matrix*)по індексам :[поточний індекс циклу по елементах рядка][поточний індекс циклу(2)] та відняти від отриманого результату добуток елементів по індексам :[поточний індекс циклу(1)][ поточний індекс циклу по елементах рядка], [поточний індекс циклу(2)][ поточний індекс циклу(1)];

Елементу матриці (*Matrix*) по індексам: [поточний індекс циклу по елементах рядка][почтовий індекс циклу(2)] присвоїти отримане значення;

* + 1. Онулювати значення ствобця поточного індекса циклу(1) крім елементу головної діагоналі;
    2. Ініціювати чисельний масив(*result*) довжиною кількості змінних системи;
    3. Цикл проходу по масиву(*result*):
       1. Присвоїти значення відповідного елемента по індексу стовпця вільних членів елементу масиву (*result*);

1. КІНЕЦЬ.

Метод обертань:

1. ПОЧАТОК
2. Шляхом елементарних перетворень знайти ранг розширеної та нерозширеної матриць:

2.1 ЯКЩО ранг розширеної матриці більший за ранг основної ТО:

Вивести повідомлення : «Матриця несумісна, розв’язків не має»,

Перейти до пункту 4;

ІНАКШЕ ЯКЩО ранг розширеної матриці менший за ранг основної ТО :

Вивести повідомлення : «Матриця сумісна і має безліч розв’язків», перейти до пункту 4;

ІНАКШЕ:

Продовжити;

1. Ітерація(1) по рівняннях матриці(matrix):

3.1 Ітерація(2) по рівняннях матриці(matrix), починаючи з рівняння, на один нижче, від поточного рівняння ітерації(1);

3.1.1 Надати змінним *c,s* значень коефіцієнта головної діагоналі матриці поточного рівняння ітерації(1) та коефіцієнта поточного рівняння ітерації(2), що знаходиться під першим відповідно;

3.1.2 Ітерація(3) по стовпцях матриці;

3.1.2.1 Надати коефіцієнту поточного стовпця ітерації(3) поточного рівняння ітерації(1) значення суми добутків коефіцієнта *с*  на цей самий елемент та коефіцієнта *s* на коефіцієнт поточного стовпця ітерації(2) поточного рівняння ітерації(1);

3.1.2.2 Надати коефіцієнту поточного стовпця ітерації(3) поточного рівняння ітерації(2) значення різниці добутків коефіцієнта *s*  на цей самий елемент та коефіцієнта *с* на коефіцієнт поточного стовпця ітерації(3) поточного рівняння ітерації(1)(до надання йому нового значення);

1. Зворотній хід:
   1. Цикл проходу по всіх рядках матриці(*Matrix*), починаючи з останнього:

4.1.1Розділити поточний рядок на перше ненульове значення;

4.1.2 Отримане значення в стовбці вільних членів помножити на усі елементи наступного по циклу рядка, крім першого ненульового;

1. Ініціювати масив чисел (*result*) довжиною кількості змінних матриці(*Matrix*);
2. Цикл проходу по індексам масиву (*result*) :
   1. Присвоїти значення відповідного елемента по індексу стовпця вільних членів елементу масиву (*result*);
3. Кінець.

# ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Діаграма класів програмного забезпечення

Програма буде мати 5 користувацьких(View, Model, ChartSolve, Coodrdinates, MathFunctionBD) та один стандартний (Application). Клас View відповідає за все, що має відношення до графічного інтерфейсу програми. Model виконує генерацію, зчитування, перевірку, обробку та обчислення матриць. ChartSolve відповідає за побудову графіка системи з 2-ма змінними та двома рівняннями. Coodrdinates та MathFunctionBD – внутрішні допоміжні класи об’єкту ChartSolve. Application – абстрактний клас, з реалізації якого програма запускається програма(генерується та виводиться на екран інтерфейс).

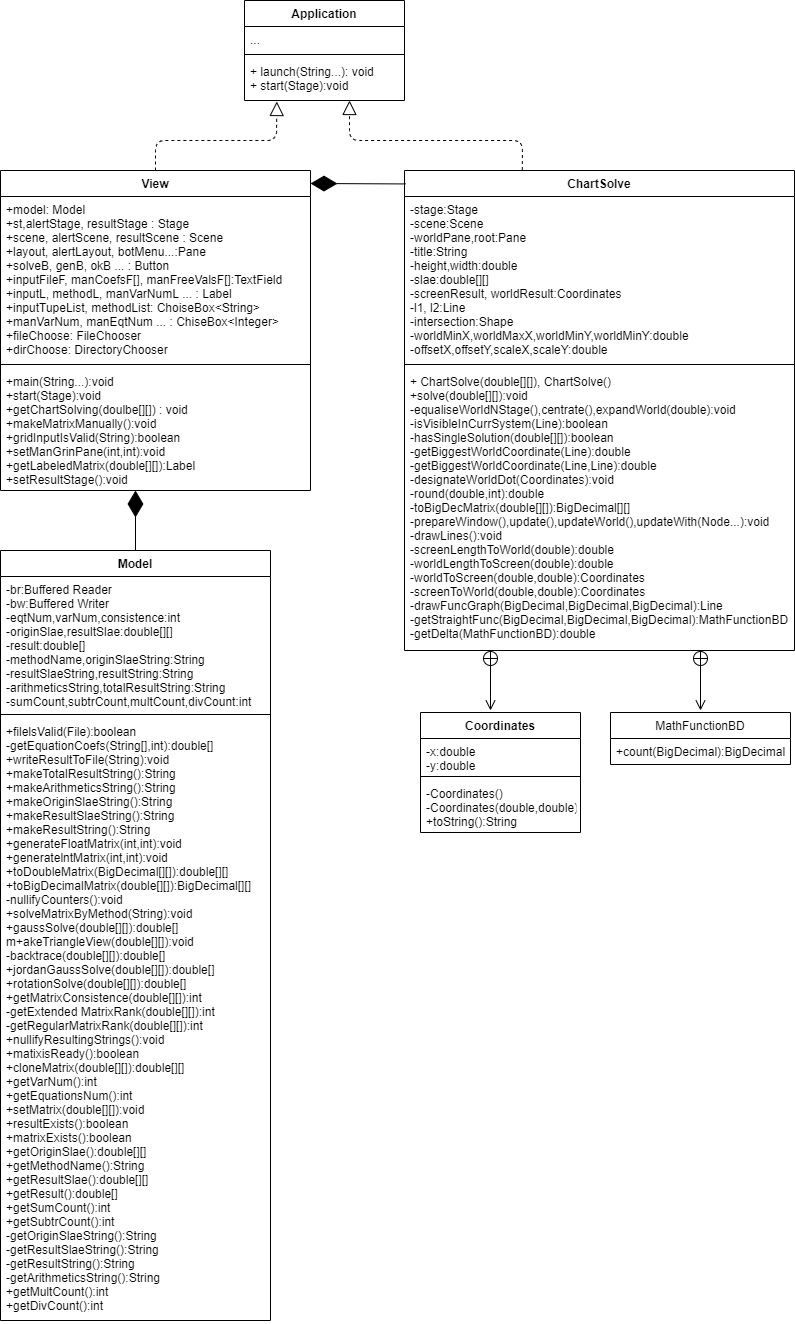
На рисунку 4.1 наведено діаграму відношень класів програми:

Рисунок 4.1 – Діаграма класів

## Опис методів частин програмного забезпечення

### Стандартні методи

У таблиці 1.1 наведені стандартні методі, що відповідають за запуск інтерфейсу програми

– Стандартніметоди

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | Application | start | Формування інтерфейсної частини програми | Stage | void | Applicatoin.java |

### Користувацькі методи

У таблиці 1.2 наведені користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | View | start | Формування інтерфейсної частини програми | Stage | void | View.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 2 | View | getChartSolving | Графіче рішення слар 2 на 2 | double[][] | void | View.java |
| 3 | View | makeMatrixManually | Передача в модель матриці, введеної вручну | - | void | View.java |
| 4 | View | gridInputIsValid | Перевірка на корректність введених вручнукоефіцієнтів матриці | String | boolean | View.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | | Заголовний файл |
| 5 | View | setManualGridPane | Задання розмірності матриці для введення коефіцієнтів вручну | int, int | void | | View.java |
| 6 | View | getLabeledMatrix | Оформлення матриці у вигляді елементу графічного інтерфейсу | Label | double[][] | | View.java |
| 7 | View | setResultStage | Передача повного результату обчислень у вікно результату | - | | void | View.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 8 | Model | fileIsValid | Перевірка обраного файлу вхідної матриці на коректність | File | boolean | Model.java |
| 9 | Model | getEquationCoefs | Зі зчитаного рядка вхідного файлу повернути масив з коефіцієнтами | String, int | double[] | Model.java |
| 10 | Model | writeResultToFile | Запис результату в файл по вказаному шляху | String | void | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 11 | Model | makeTotalResultString | Представити результат у вигляді рядка | - | String | Model.java |
| 12 | Model | makeArithmeticsString | Представити підрахунок кількості арифметичних операцій у вигляді рядка | - | String | Model.java |
| 13 | Model | makeOriginSlaeString | Представити вхідну матрицю у вигляді рядка | - | String | Model.java |
| 14 | Model | makeResultSlaeString | Представити результативну матрицю у вигляді рядка | - | String | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 15 | Model | makeResultString | Представити результат обчислень матриці у вигляді рядка | - | String | Model.java |
| 16 | Model | generateFloatMatrix | Згенерувати матрицю дійсних значень | int,int | void | Model.java |
| 17 | Model | generateIntMatrix | Згенерувати матрицю дійсних значень | int,int | void | Model.java |
| 18 | Model | toDoubleMatrix | Переформатувати тип матриці з BigDecimal в double | BigDecimal[][] | double[][] | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 19 | Model | toBigDecimalMatrix | Переформатувати тип матриці з double в BigDecimal | double[][] | BigDecimal [][] | Model.java |
| 20 | Model | nullifyCounters | Онулювати підрахунок кількості арифметичних операцій | - | void | Model.java |
| 21 | Model | solveMatrixByMethod | Вирішити слар вказаним методом | String | void | Model.java |
| 22 | Model | gaussSolve | Обчислює слар методом Гаусса | с | double[] | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 23 | Model | makeTriangleView | Шляхом елементарних перетворень зводить слар до трикутного вигляду | double[][] | void | Model.java |
| 24 | Model | backtrace | З слар трикутного вигляду повертає вирахувані значення змінних | double[][] | double[] | Model.java |
| 25 | Model | jordanGaussSolve | Обчислює слар методом Жордана-Гаусса | double[][] | double[] | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 26 | Model | rotationSolve | Обчислює слар методом обертань | double[][] | double[] | Model.java |
| 27 | Model | getMatrixConsistence | Повертає -1 якщо матриця нусумістна, 0, якщо має єдине рішення, 1, якщо має безліч | double[][] | int | Model.java |
| 28 | Model | getExtendedMatrixRank | Повертає ранг розширеної матриці переданої слар | double[][] | int | Model.java |
| 29 | Model | getRegularMatrixRank | Повертає ранг стандартної матриці переданої слар | double[][] | int | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 30 | Model | nullifyResultingStrings | Стерти результати у вигляді рядків | - | void | Model.java |
| 31 | Model | matixIsReady | Повертає ІСТИНУ, якщо вхідна матриця не має значення NULL | - | boolean | Model.java |
| 32 | Model | cloneMatrix | Клонує передану матрицю | double[][] | double[][] | Model.java |
| 33 | Model | getVarNum | Повертає кількість змінних сучасної матриці | - | int | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 34 | Model | getEquationsNum | Повертає кількість  рівнянь сучасної матриці | - | int | Model.java |
| 35 | Model | setMatrix | Встановлює значення матриці | double[][] | void | Model.java |
| 36 | Model | resultExists | Повертає ІСТИНУ, якщо масив результату не має значення NULL | - | boolean | Model.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 37 | Model | getOriginSlae | Повертає вхідну матрицю | - | double[][] | Model.java |
| 38 | ChartSolve | ChartSolve | Створює об’єкт з вказаною слар | double[][] | ChartSolve | ChartSolve.java |
| 39 | ChartSolve | main | Завантажує аргументи командної строки до классу Application | String[] | void | ChartSolve.java |
| 40 | ChartSolve | start | Генерує інтерфейсну частину частину аплікації побудови графіка | Stage | void | ChartSolve.java |
| 41 | ChartSolve | solve | Виконання головних процесів побудови та виводу графіка | double[][] | void | ChartSolve.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 42 | ChartSolve | equaliseWorldNStage | Зрівнює розмір створеної системи координат з розмірами екрана | - | void | ChartSolve.java |
| 43 | ChartSolve | centrate | Встановлює параметри здвигу рівними центру екрана | - | void | ChartSolve.java |
| 44 | ChartSolve | isVisiblelnCurrSystem | Перевірка чи влазить передана лінія в сучасну розмірність системи | Line | boolean | ChartSolve.java |
| 45 | ChartSolve | expandWorld | Розширює розмірність системи на задану величину | double | void | ChartSolve.java |
| 46 | ChartSolve | hasSingleSolution | Повертає ІСТИНУ, якщо слар має єдине рішення | double[][] | boolean | ChartSolve.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 47 | ChartSolve | getBiggestWorldCoordinate | Повертає найвіддаленіше від розмірності системи значення координати переданої лінії | Line | double | ChartSolve.java |
| 48 | ChartSolve | getBiggestWorldCoordinate | Повертає найвіддаленіше від розмірності системи значення координати однієї з переданих ліній | Line. Line | double | ChartSolve.java |
| 49 | ChartSolve | designateWorldDot | Помічає задану точку червоними линіями | Coordinates | void | ChartSolve.java |
| 50 | ChartSolve | round | Округлює задане дійсне до заданої кількості знаків після крапки | double, int | double | ChartSolve.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 51 | ChartSolve | prepareWindow | Готує елементи вікна до виведення на екран | - | void | ChartSolve.java |
| 52 | ChartSolve | update | Оновлює систему на екрані згідно зміненим параметрам | - | void | ChartSolve.java |
| 53 | ChartSolve | updateWorld: | Очищае все з екрану і наново перемальовує пусту систему координат | - | void | ChartSolve.java |
| 54 | ChartSolve | updateWith | Оновлює контейнер системи каардинат заданими елементами | Node... | void | ChartSolve.java |
| 55 | ChartSolve | drawLines | Малює пусту систему координат | - | void | ChartSolve.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 56 | ChartSolve | screenLengthToWorld | Конвертує довжину з системи координат вікна у відносну систему координат | double | double | ChartSolve.java |
| 57 | ChartSolve | worldLengthToScreen | Конвертує довжину з відносної системи координат у систему вікна | double | double | ChartSolve.java |
| 58 | ChartSolve | worldToScreen | Конвертує значення положення точки з відносної системи координат у систему вікна | Coordinates | Coordinates | ChartSolve.java |

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 59 | ChartSolve | screenToWorld | Конвертує значення положення точки з системи координат вікна у відносну систему координат | Coordinates | Coordinates | ChartSolve.java |
| 60 | ChartSolve | drawFuncGraph | Повертає лінію, намальовану згідно переданим коефіцієнтам при x, y та вільному відповідно | BigDecimal, BigDecimal, BigDecimal | Line | ChartSolve.java |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 61 | ChartSolve | getStraightFunc | Генерує функцію, згідно  переданим коефіцієнтам при x, y та вільному відповідно | BigDecimal, BigDecimal, BigDecimal | MathFunctionBD | ChartSolve.java |
| 62 | ChartSolve | getDelta | Повертає значення зміни функції | MathFunctionBD | double | ChartSolve.java |

# ІНСТРУКЦІЯ КОРИСТУВАЧА

## Робота з програмою

Після запуску программи, відкривається її головне вікно (Рисунок 5.1).

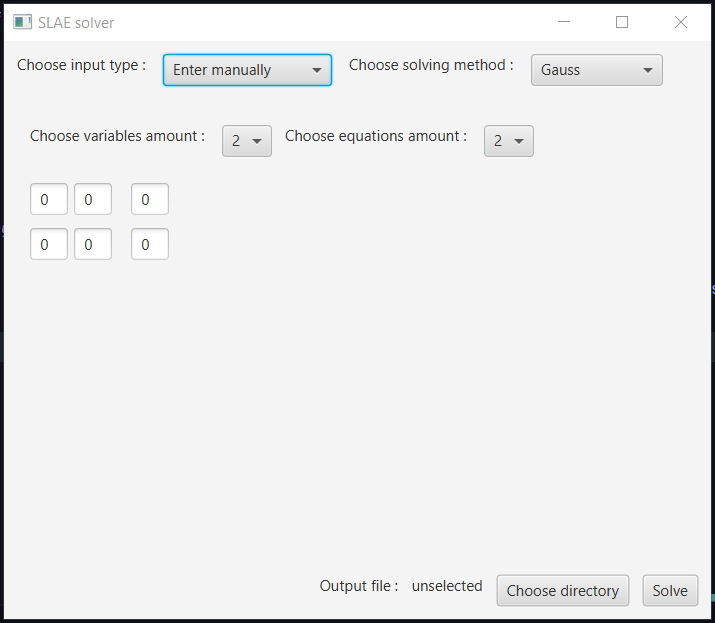


Рисунок 5.1 – Головне вікно програми

Користувач може ввести матрицю дійсних, або цілих чисел заданого розміру, або змінити його у Дроп-даун меню після лейблів «Сhoose variables amount:» та «Сhoose equations amount:» (Рисунок 5.2):

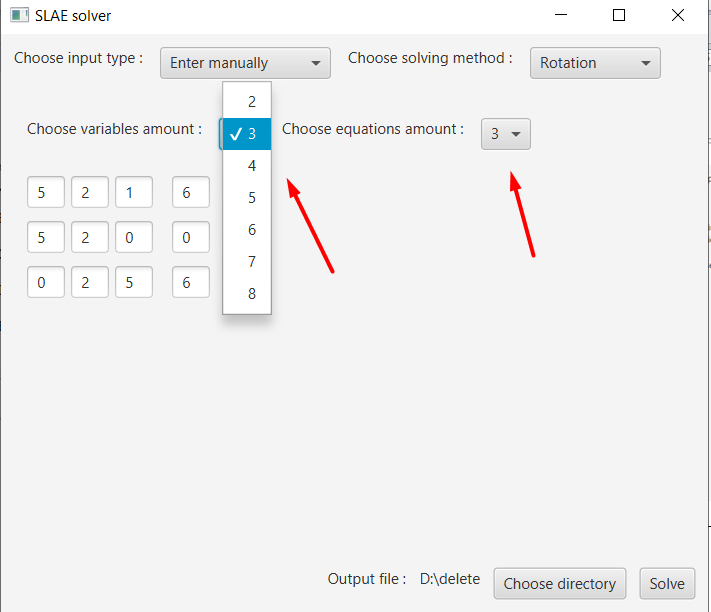


Рисунок 5.2 – Вибір необхідного розміру системи та введення її коефіцієнтів

Також користувач може обрати тип введення вхідних даних та метод, яким буде вирішено СЛАР (Рисунок 5.3):

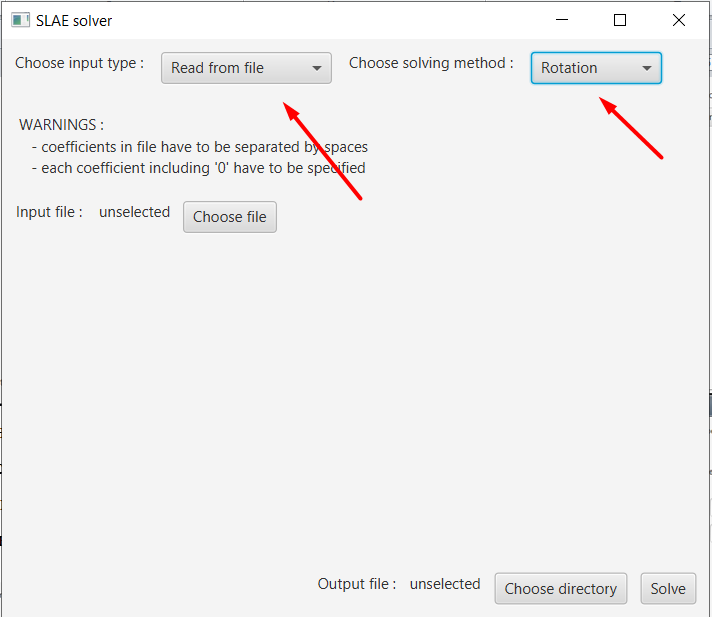


Рисунок 5.3 – Обрання типу введення вхідних даних та методу вирішення СЛАР

Обираючи метод зчитування з файлу слід обрати вхідний файл, після чого, якщо він не містить некоректних для роботи програми символів, задана в ньому матриця виводиться на екран(Рисунок 5.4):

|  |  |
| --- | --- |
|  |  |

Рисунок 5.4 – обрання вхідного файлу та вивід матриці з нього на екран

Також можна обрати метод випадкової генерації коефіцієнтів матриці. Обравши кількість рівнянь і змінних, слід натиснути одну з двох кнопок «Generate integer num», або «Generate float num» за бажанням. Вони згенерують матрицю з відповідними коефіцієнтами та виведуть її на екран(Рисунок 5.5):

|  |  |
| --- | --- |
|  |  |
|  |  |

Рисунок 5.5 – генерація матриці та виведення її на екран

Після обрання типу введення, методу та кількості змінних користувач може вказати папку, у який хоче записати отриманий результат. Це не обов’язково(Рисунок 5.6):

|  |  |
| --- | --- |
|  |  |

Рисунок 5.4 – обрання вихідного файлу

Після цього натискається кнопка «Solve». Після натиску матриця перевіряється на сумісніть, єдність рішення та якщо матриця сумусні та має єдине рішення, наекран виводиться вікно із шляхом до вихідного-файлу, якщо він був вказаний, вхідна матриця, матриця, отримана в результаті розрахунків, результат та кількість виконаних арифметичних операцій, виконаних обраним методом. Якщо вказані кількість рівнянь і змінних співпадають та дорівнює 2, на екран виводиться графічне вирішення заданої слар(Рисунок 5.7).

|  |  |
| --- | --- |
|  |  |

Рисунок 5.7 – Результати вирішення СЛАР

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається СЛАР у матричному вигляді, тобто задається за допомогою матриці системи та стовпця вільних членів, числа яких дійсні з точністю не більше, ніж 2 знака після коми (якщо точність більша, то програма автоматично округлить їх за математичними правилами до 2-х знаків після коми).

Результатом виконання програми є розв’язок зданої СЛАР, який видається у вигляді таблиці кожне число якої записане з точністю до 2-х знаків після коми або повідомлення, що дана система не має розв’язків або не сходиться для обраного методу.

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 5.1.

Таблиця 5.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.5.2 або вище | |

# ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## План тестування

Наведемо план тестування програмного забезпечення. Та проведемо тести над основним функціоналом програми.

1. Тестування правильності введених значень.
   1. Тестування при введенні некоректних символів.
2. Тестування коректної роботи при введені систем, що не мають коренів.
   1. Тестування роботи методу Гаусса на несиметричній слар.
   2. Тестування роботи методу Гаусса на несумісній слар.
3. Тестування коректності роботи методів Гаусса, Жордана- Гаусса та обертань.
   1. Перевірка коректності роботи методу Гаусса.
   2. Перевірка коректності роботи методу Жордана-Гаусса.
   3. Перевірка коректності роботи методу обертань.
4. Тестування коректності роботи методів Гаусса, Жордана- Гаусса та обертань з дробовими коефіцієнтами.
   1. Перевірка правильності результатів.
5. Тестування побудови графіків.

## Приклади тестування

Протестуємо основний функціонал програми на коректність обробки некоректних вхідних даних, правильність роботи методів, коректність побудови графіків(Таблиці 1.1 – 1.3).

Таблиця 1.1 ‑ Приклад роботи програми при введенні некоректних символів

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 2 3 b 6 S 4 6 f f y 9 17 |
| Схема проведення тесту | Поелементне заповнення матриці коефіцієнтів |
| Очікуваний результат | Повідомлення про помилку  формату даних |
| Стан програми після проведення випробувань | Видано помилку «Матриця має містити лише додатні та від’ємні натуральні числа та десяткові дроби , використовуючи крапку для розділення цілої та дробової частини числа» |

Таблиця 6.2 – Приклад роботи програми при введенні несиметричної системи

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 9 10 6 4 0 7 1 9 |

Продовження таблиці 6.2

|  |  |
| --- | --- |
| Схема проведення тесту | Генерація несиметричної матриці коефіцієнтів |
| Очікуваний результат | Повідомлення про помилку  «Система має безліч розв’язків» |
| Стан програми після проведення випробувань | Видано помилку «Система має безліч розв’язків» |

Таблиця 6.3 – Приклад роботи програми про введенні несумісної матриці

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення несумісніх систем |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 2 3 5 4 6 8 |
| Схема проведення тесту | Поелементне заповнення матриці коефіцієнтів |
| Очікуваний результат | Повідомлення про помилку сумісності матриці |
| Стан програми після проведення випробувань | Видано помилку «Матриця несумісна» |

Тестування коректності роботи методів Гаусса, Жордана- Гаусса та обертань:

* 1. Перевірка коректності роботи методу Гаусса:

Пройдено

* 1. Перевірка коректності роботи методу Жордана-Гаусса:

Пройдено

* 1. Перевірка коректності роботи методу обертань:

Пройдено

Тестування коректності роботи методів Гаусса, Жордана- Гаусса та обертань з дробовими коефіцієнтами.

Програма розрахована на числа з дробовими коефіцієнтами, тому всі методи пройшли тест успішно.

Тестування коректності побудови графіків.

Програма будує графіки згідно рівнянням СЛАР

Висновок: програма пройшла базове тестування.

# АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для розв’язання СЛАР наступними методами: Гауса, Жордана-Гауса та обертань.

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, ретельно провіряються на валідність і лише потім подаються на обробку програмі.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаюся MS Excel:

а)Метод Гауса.

Результат виконання методу Гауса наведено на рисунку 7.1:

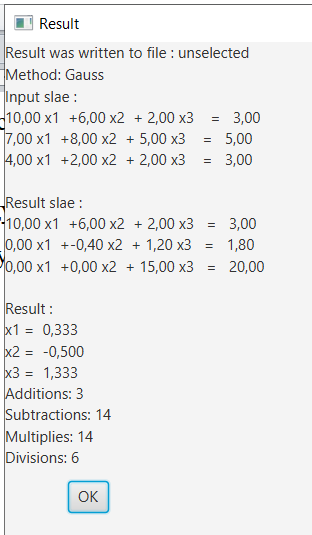


Рисунок 7.1 – Результат виконання методу Якобі

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.2), то даний метод працює вірно.

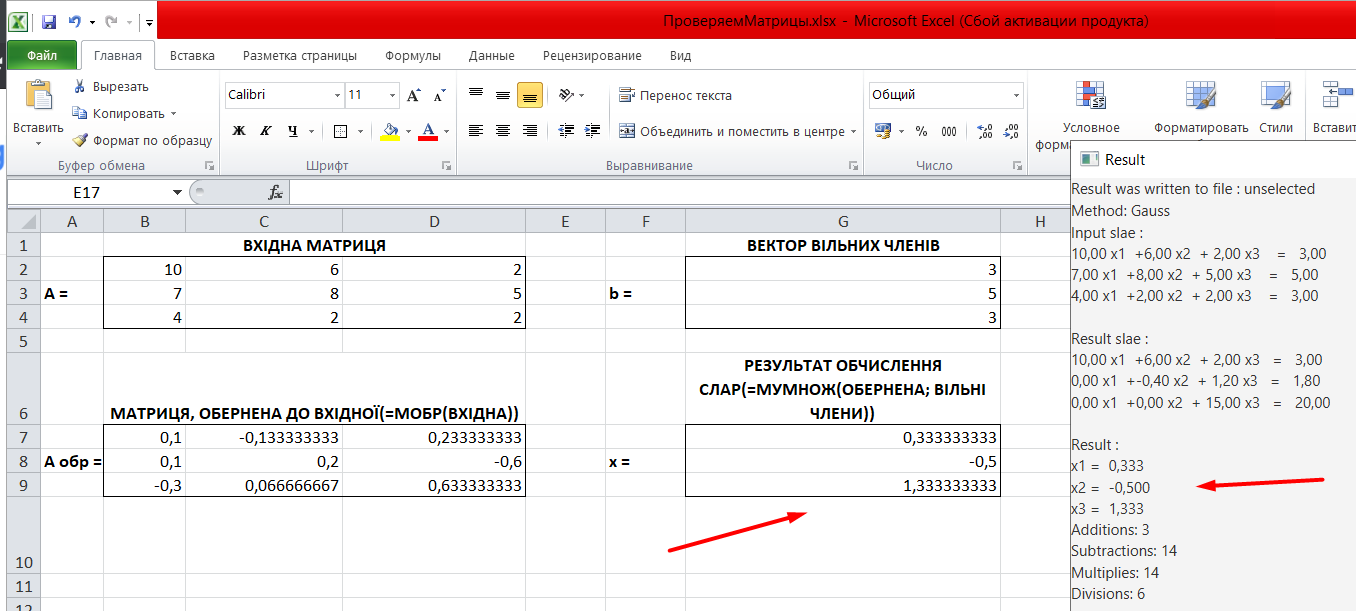


Рисунок 7.2 – Перевірка методу Гауса в MS Excel 2010

б) Метод Жордана-Гауса.

Результат виконання методу Жордана-Гауса наведено на рисунку 7.3:

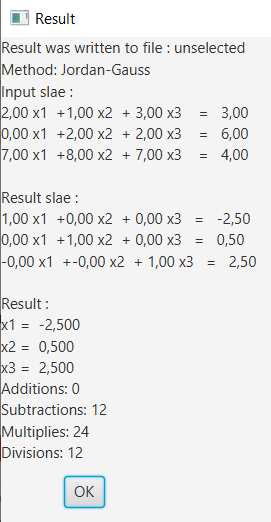


Рисунок 7.3 – результат роботи методу Жордана-Гауса

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.4), то даний метод працює вірно.

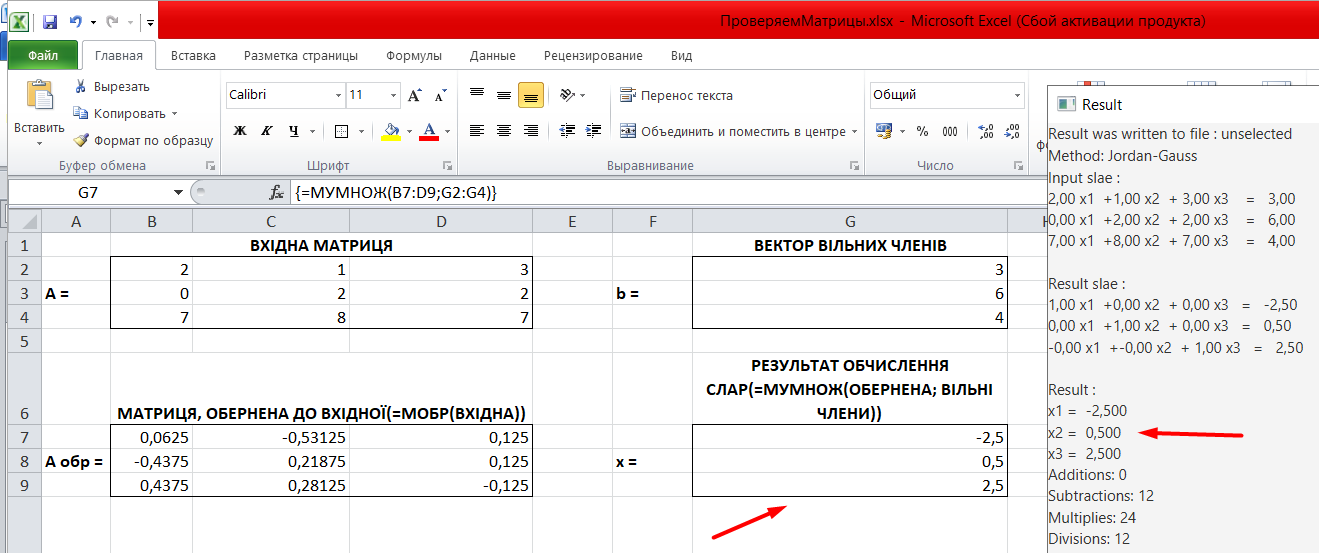


Рисунок 7.4 – Перевірка методу Жордана-Гауса в MS Excel 2010

в) Метод обертань.

Результат виконання методу обертань наведено на рисунку 7.5:

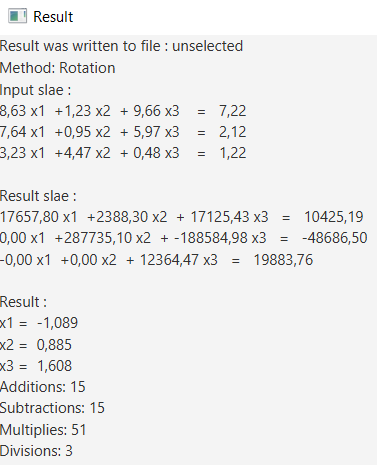


Рисунок 7.5 – результат роботи методу обертань

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.4), то даний метод працює вірно.

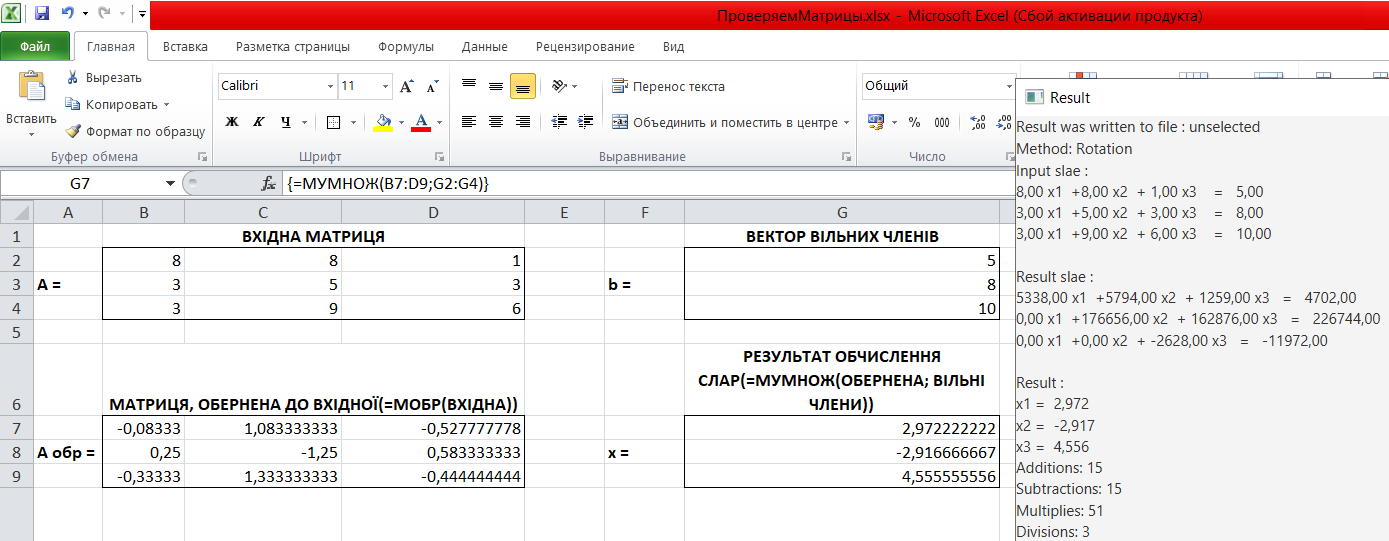


Рисунок 7.6 – Перевірка методу обертань в MS Excel 2010

Для проведення тестування ефективності програми випадково генерувалися матриці в розмірності від двох до восьми.

Результати тестування ефективності алгоритмів розв’язання СЛАР наведено в таблиці 7.1:

Таблиця 7.1 – Тестування ефективності методів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розмірність системи | Параметри тестування | Методи | | |
| Гауса | Жордана-Гауса | Обертань |
| 2 | Кількість елементарних операцій додавання | 1 | 0 | 4 |
| Кількість елементарних операцій віднімання | 11 | 9 | 11 |
| Кількість елементарних операцій множення | 10 | 12 | 19 |
| Кількість елементарних операцій ділення | 5 | 8 | 4 |
| 4 | Кількість елементарних операцій додавання | 6 | 0 | 36 |
| Кількість елементарних операцій віднімання | 30 | 30 | 34 |

Продовження таблиці 7.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Кількість елементарних операцій множення | 32 | 60 | 126 |
| Кількість елементарних операцій ділення | 10 | 20 | 4 |
| 6 | Кількість елементарних операцій додавання | 15 | 0 | 120 |
| Кількість елементарних операцій віднімання | 91 | 105 | 111 |
| Кількість елементарних операцій множення | 100 | 210 | 435 |
| Кількість елементарних операцій ділення | 21 | 42 | 6 |
| 8 | Кількість елементарних операцій додавання | 28 | 0 | 280 |
| Кількість елементарних операцій віднімання | 204 | 252 | 260 |
| Кількість елементарних операцій множення | 224 | 504 | 1036 |
| Кількість елементарних операцій ділення | 36 | 72 | 8 |

Візуалізація результатів таблиці 7.1 наведено на рисунках 7.1 – 7.3:

Рисунок 7.1 – Графік залежності кількості арифметичних операцій методу Гауса від розміру вхідної слар

Рисунок 7.2 - Графік залежності кількості арифметичних операцій методу Жордана-Гауса від розміру вхідної слар

Рисунок 7.3 - Графік залежності кількості арифметичних операцій методу обертань від розміру вхідної слар

За результатами тестування можна зробити такі висновки:

а) Всі розглянуті методи дозволяють знаходити розв’язки СЛАР розмірності в межах від 2-х до 8-ми.

б) Складність всіх розглянутих методів є квадратичною, тобто –  
 , де – розмір СЛАР.

в) З розглянутих методів найоптимальнішим для практичного використання є метод Гауса, оскільки він виконується найшвидше.

## ВИСНОВОК

Протягом написання курсової роботи було вивчено:

а) Методи вирішення слар : Гауса, Жордана-Гауса, обертань

б) Методи роботи з інструментом написання графічного інтерфейсу мовою java JavaFX.

Реалізовано програмне забезпечення, що приймає на вхід матрицю коефіцієнтів СЛАР та вирішує її будь-яким з вказаних методів. Реалізовано графічне вирішення системи у випадку, якщо матриця має єдине рішення та розмірність 2 .

Завдяки аналізу результатів роботи методів було виведено, що метод Гауса є найефективнішим серед трьох вказаних, оскільки має найменший приріст кількості елементарих арифметичних операцій. Якість обробки вхідних СЛАР програмою було перевірено завдяки програмі високої надійності Excel 2010.

## ПЕРЕЛІК ПОСИЛАНЬ

Метод Гаусса: <https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0>

Метод Жордана-Гаусса: <https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0_%E2%80%94_%D0%96%D0%BE%D1%80%D0%B4%D0%B0%D0%BD%D0%B0>

Метод обертань: <https://works.doklad.ru/view/qYgaul2bTZI.html>

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

Керівник Головченко Максим Михайлович

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_ р.

Виконавець:

Студент Дзюбак Дем'ян Павлович

«19» березня 2020 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: Розв'язання СЛАР точними методами

з дисципліни:

«Об'єктно-орієнтоване програмування»

Київ 2020

* 1. *Мета*: Метою курсової роботи є розробка програмного забезпечення, що буде знаходити рішення для заданої Системи Лінійних Алгебраїчних Рівнянь(СЛАР) методами Гаусса, Жордана-Гауса та обертання.
  2. *Дата початку роботи*: «19» березня 2020 р.
  3. *Дата закінчення роботи*: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2020 р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Генерувати, зчитувати з файлу або мати можливість приймати матрицю коефіцієнтів СЛАР шляхом введення вручну в вигляді Ax = b де – матриця коефіцієнтів, – вектор шуканих значень (рішення системи), b – вектор вільних членів;
* Обробляти матрицю коефіцієнтів та стовпець вільних членів для СЛАР розмірність яких знаходиться в межах від 2 до 8;
* Виводити у вікні графічне вирішення системи у випадку, якщо кількість змінних та рівнянь дорівнює 2;
* Можливість обирати метод вирішення СЛАУ з заданих;
* Завантажувати результат роботи в задану папку або файл;
* Виводити у вікні кількість операцій додавання, віднімання, множення та ділення;
* Записувати результат обчислень у вказаний файл.

1. Нефункціональні вимоги:

* Тип – десктоп;
* Мати графічний інтерфейс, що дозволяє вводити відомі коефіцієнти та вільні члени СЛАР в окремі комірки в відповідних місцях(відомі коефіцієнти біля відповідних невідомих, вільні члени після знаку «=») і отримувати результат її розв’язку;
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 95 - Розробка технічної документації.

* 1. *Стадії та етапи розробки*:

1. Об'єктно-орієнтований аналіз предметної області задачі (до\_\_.\_\_.2020 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до \_\_.\_\_.2020 р.)
3. Розробка програмного забезпечення (до \_\_.\_\_.2020 р.)
4. Тестування розробленої програми (до \_\_.\_\_.2020 р.)
5. Розробка пояснювальної записки (до \_\_.\_\_.2020 р.).
6. Захист курсової роботи (до \_\_.\_\_.2020 р.).
   1. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

Додаток Б Тексти програмного коду

*студента групи ІП-91 І курсу*

*Дзюбака Д. П.*

(Обсяг програми (документа), арк., Кб)

*21 арк, 103 Кб*

(Вид носія даних)

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі вирішення СЛАР*

*public class* View *extends* Application {  
 *//поле моделі, в якій проводятсья вся робота над СЛАР* Model model;  
 *//поля вікон програм, контейнерів їх елементів, самих елементів графічного інтерфейсу* Stage st, alertStage, resultStage;  
 Scene scene, alertScene, resultScene;  
 BorderPane layout, alertLayout;  
 HBox topMenu, manChosePane, botMenu, genGenPane, genSlaeParamPane, readReadPane;  
 VBox readPane, manPane, genPane, alertPane, resultPane, genChoosePane;  
 AnchorPane buttonPane;  
 GridPane manFieldsPane;  
 Button solveB, genGenDoubleB, okB, genGenIntB, okRB, chooseInputFileB,chooseOutputDirB;  
 TextField inputFileF, manCoefsF[][], manFreeValsF[];  
 Label inputL, methodL, manVarNumL, readWarnL, genVarNumL, enterPathL, generateMatrixL, generatedMatrixL,  
 chooseFileFirstlyL, incorrectInputL, resultWrittenL, inputFieldEmptyL, fileDoesntExistL,  
 readMatrixL, inconsistentMatrixL, infiniteSolutionsL, invalidInputL, inputMatrixL,  
 genEqtNumL,manEqtNumL, inputFileL, outputFileL, inputFilePathL, outputFilePathL,  
 totalResultL;  
 ChoiceBox<String> inputTypeList, methodList;  
 ChoiceBox<Integer> manVarNumCB, genVarNumCB, genEqtNumCB, manEqtNumCB;  
 FileChooser fileChoose;  
 DirectoryChooser dirChoose;  
  
 *//Метод, що запускає програму  
 public static void* main(String[] args) {  
 *launch*(args);  
 }  
  
 *//метод, що починає роботу поргами, на вхід подається об'єкт вікна програми  
 //Повністю оформлюємо сторінки інтерфейсу* @Override  
 *public void* start(Stage stage) *throws* Exception {  
 model = *new* Model();  
 st = stage;  
  
 st.setTitle("SLAE solver");  
 st.setWidth(580);  
 st.setHeight(500);  
  
 *//оформлення частини інтерфейсу, що буде супроводжувати всі сторінки  
 //general menu* inputL = *new* Label("Choose input type : ");  
 methodL = *new* Label(" Choose solving method : ");  
 inputTypeList = *new* ChoiceBox<>();  
 inputTypeList.getItems().addAll("Generate randomly", "Read from file", "Enter manually");  
 inputTypeList.setValue("Enter manually");  
 inputTypeList.getSelectionModel().selectedItemProperty().addListener((v, oldVal, newVal) -> {  
 *if* (newVal == "Read from file") {  
 layout.setCenter(readPane);  
 scene.setRoot(layout);  
 } *else if* (newVal == "Generate randomly") {  
 layout.setCenter(genPane);  
 scene.setRoot(layout);  
 } *else if* (newVal == "Enter manually") {  
 layout.setCenter(manPane);  
 scene.setRoot(layout);  
 }  
 });  
 methodList = *new* ChoiceBox<>();  
 methodList.getItems().addAll("Gauss", "Jordan-Gauss", "Rotation");  
 methodList.setValue("Gauss");  
 outputFileL = *new* Label("Output file :");  
 outputFilePathL = *new* Label("unselected");  
 topMenu = *new* HBox(10);  
 topMenu.getChildren().addAll(inputL, inputTypeList, methodL, methodList);  
 topMenu.setPadding(*new* Insets(10, 10, 10, 10));  
 totalResultL = *new* Label();  
  
 chooseOutputDirB = *new* Button("Choose directory");  
 chooseOutputDirB.setOnAction(ae ->{  
 dirChoose = *new* DirectoryChooser();  
 File outputDir = dirChoose.showDialog(*null*);  
  
 *if*(outputDir != *null*)  
 outputFilePathL.setText(outputDir.getAbsolutePath());  
 });  
  
 solveB = *new* Button("Solve");  
 solveB.setOnAction(actionEvent -> {  
 *try* {  
 model.nullifyResultingStrings();  
 *if* (inputTypeList.getValue() == "Generate randomly" && !model.matrixIsReady()) {  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, generateMatrixL);  
 alertStage.show();  
 } *else if* (inputTypeList.getValue() == "Enter manually")   
 makeMatrixManually();  
  
  
 model.solveMatrixByMethod(methodList.getValue());  
 *if* (model.getVarNum() == 2 && model.getEquationsNum() == 2)  
 getChartSolving(model.getOriginSlae());  
 *if* (model.resultExists()) {  
 totalResultL.setText(model.makeTotalResultString());  
 setResultStage();  
 *if* (!outputFilePathL.getText().equals("unselected"))  
 model.writeResultToFile(outputFilePathL.getText());  
  
 resultWrittenL.setText("Result was written to file : " + outputFilePathL.getText());  
 resultPane.getChildren().remove(1);  
 resultPane.getChildren().add(1, totalResultL);  
 resultStage.show();  
 }  
  
 } *catch* (InconsistentMatrixException e) {  
 inconsistentMatrixL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, inconsistentMatrixL);  
 alertStage.setMinWidth(inconsistentMatrixL.getLayoutBounds().getWidth());  
 alertStage.show();  
 } *catch* (InfiniteSolutionsAmountException e) {  
 infiniteSolutionsL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, infiniteSolutionsL);  
 alertStage.setMinWidth(infiniteSolutionsL.getLayoutBounds().getWidth());  
 alertStage.show();  
 } *catch* (InvalidInputException e) {  
 invalidInputL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, invalidInputL);  
 alertStage.setMinWidth(invalidInputL.getLayoutBounds().getWidth());  
 alertStage.show();  
 }  
 });  
 botMenu = *new* HBox(10);  
 botMenu.getChildren().addAll(outputFileL, outputFilePathL, chooseOutputDirB, solveB);  
 buttonPane = *new* AnchorPane();  
 buttonPane.getChildren().addAll(botMenu);  
 AnchorPane.*setRightAnchor*(botMenu, 10d);  
 AnchorPane.*setBottomAnchor*(botMenu, 10d);  
 layout = *new* BorderPane();  
 layout.setTop(topMenu);  
 layout.setBottom(buttonPane);  
 scene = *new* Scene(layout);  
  
 *//Оформлення вікна помилок  
 // alertBox* alertStage = *new* Stage();  
 alertStage.initModality(Modality.*APPLICATION\_MODAL*);  
 alertStage.setTitle("Alert box");  
 alertStage.setHeight( 160);  
 alertStage.setWidth(240 );  
 enterPathL = *new* Label("Enter valid input file path!");  
 enterPathL.setPadding(*new* Insets(10, 10, 10, 10));  
 chooseFileFirstlyL = *new* Label("Enter valid output file path!");  
 incorrectInputL = *new* Label("Input matrix file must contain only " +  
 "\ndecimal or float(using '.') values " +  
 "\nfollowed by '-' if needed");  
 inputFieldEmptyL = *new* Label("Input file field is empty");  
 fileDoesntExistL = *new* Label("Specified file doesn't exist");  
 generateMatrixL = *new* Label("Generate matrix firstly!");  
 inconsistentMatrixL = *new* Label("Matrix is inconsistent");  
 infiniteSolutionsL = *new* Label("Matrix has infinite solutions amount");  
 invalidInputL = *new* Label("Input matrix characters must contain only " +  
 "\ndecimal or floating point(using '.') numbers " +  
 "\nfollowed by '-' if need");  
 okB = *new* Button("OK");  
 okB.setOnAction(ae -> {  
 alertStage.close();  
 });  
 alertPane = *new* VBox();  
 alertPane.getChildren().add(0, enterPathL);  
 alertPane.getChildren().add(1, okB);  
 VBox.*setMargin*(okB, *new* Insets(10, 50, 10, 10));  
 alertLayout = *new* BorderPane();  
 alertLayout.setCenter(alertPane);  
 alertScene = *new* Scene(alertLayout);  
 alertStage.setScene(alertScene);  
  
 *// оформлення сторінки генерації матриці  
 // "generate" pane* genVarNumL = *new* Label("Set variables amount : ");  
 genVarNumCB = *new* ChoiceBox<>();  
 genVarNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 genVarNumCB.setValue(2);  
 genEqtNumL = *new* Label("Set equations amount : ");  
 genEqtNumCB = *new* ChoiceBox<>();  
 genEqtNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 genEqtNumCB.setValue(2);  
 genGenDoubleB = *new* Button("Generate float num");  
 genGenDoubleB.setOnAction(ae -> {  
 model.generateFloatMatrix(genVarNumCB.getValue(), genEqtNumCB.getValue());  
 genPane.getChildren().remove(generatedMatrixL);  
 generatedMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 genPane.getChildren().add(generatedMatrixL);  
 layout.setCenter(genPane);  
 });  
 genGenIntB = *new* Button("Generate integer num");  
 genGenIntB.setOnAction(ae -> {  
 model.generateIntMatrix(genVarNumCB.getValue(), genEqtNumCB.getValue());  
 genPane.getChildren().remove(generatedMatrixL);  
 generatedMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 genPane.getChildren().add(generatedMatrixL);  
 layout.setCenter(genPane);  
 });  
 generatedMatrixL = *new* Label();  
 VBox.*setMargin*(generatedMatrixL, *new* Insets(10,10,10,10));  
 genGenPane = *new* HBox();  
 genSlaeParamPane = *new* HBox();  
 genChoosePane = *new* VBox();  
 genChoosePane.setPadding(*new* Insets(10));  
 genSlaeParamPane.getChildren().addAll(genVarNumL, genVarNumCB, genEqtNumL, genEqtNumCB);  
 genGenPane.getChildren().addAll(genGenDoubleB, genGenIntB);  
  
 genChoosePane.getChildren().addAll(genSlaeParamPane, genGenPane);  
 genPane = *new* VBox();  
 genPane.getChildren().addAll(genChoosePane, generatedMatrixL);  
  
 *//оформлення сторінки зчитування матриці з файлу  
 // "readPane"* readWarnL = *new* Label("WARNINGS :\n - coefficients in file have to be separated by spaces\n" +  
 " - each coefficient including '0' have to be specified");  
  
 fileChoose = *new* FileChooser();  
 chooseInputFileB = *new* Button("Choose file");  
 chooseInputFileB.setOnAction(ae -> {  
 File inputF = fileChoose.showOpenDialog(*null*);  
  
 *if*(inputF != *null*)  
 *if*(model.fileIsValid(inputF)){  
 readPane.getChildren().removeAll(readMatrixL);  
 readMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 readPane.getChildren().add(readMatrixL);  
 readReadPane.getChildren().remove(1);  
 inputFilePathL.setText(inputF.getAbsolutePath());  
 readReadPane.getChildren().add(1, inputFilePathL);  
 solveB.setDisable(*false*);  
  
 } *else* {  
 *if*(!solveB.isDisabled())  
 solveB.setDisable(*true*);  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, incorrectInputL);  
 alertStage.show();  
 }  
  
  
 });  
 readWarnL.setPadding(*new* Insets(10));  
 inputFileF = *new* TextField();  
 inputFileF.setPromptText("Enter input SLAE matrix file path : ");  
 inputFileL = *new* Label("Input file : ");  
 inputFilePathL = *new* Label("unselected");  
 readMatrixL = *new* Label("");  
  
 readReadPane = *new* HBox();  
 readReadPane.setPadding(*new* Insets(3, 3, 3, 3));  
  
 readReadPane.getChildren().addAll(inputFileL,inputFilePathL, chooseInputFileB);  
 HBox.*setMargin*(inputFileL, *new* Insets(5,5,5,5));  
 HBox.*setMargin*(inputFilePathL, *new* Insets(5,5,5,5));  
 HBox.*setMargin*(chooseInputFileB, *new* Insets(5,5,5,5));  
 readPane = *new* VBox();  
 readPane.setPadding(*new* Insets(3, 3, 3, 3));  
 readPane.getChildren().addAll(readWarnL, readReadPane,readMatrixL);  
  
 *//оформлення сторінки введення коефіцієнтів матриці вручну  
 //"manual" choose pane* manVarNumL = *new* Label("Choose variables amount : ");  
 manVarNumCB = *new* ChoiceBox<>();  
 manVarNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 manVarNumCB.setValue(2);  
 manEqtNumL = *new* Label("Choose equations amount : ");  
 manEqtNumCB = *new* ChoiceBox<>();  
 manEqtNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 manEqtNumCB.setValue(2);  
 manVarNumCB.getSelectionModel().selectedItemProperty().addListener((v, oldval, newVal) -> {  
 manPane.getChildren().remove(manFieldsPane);  
 manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(newVal, manEqtNumCB.getValue());  
 manPane.getChildren().add(manFieldsPane);  
 layout.setCenter(manPane);  
 });  
 manEqtNumCB.getSelectionModel().selectedItemProperty().addListener((v, oldval, newVal) -> {  
 manPane.getChildren().remove(manFieldsPane);  
 manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(manVarNumCB.getValue(),newVal);  
 manPane.getChildren().add(manFieldsPane);  
 layout.setCenter(manPane);  
 });  
 manChosePane = *new* HBox(10);  
 manChosePane.setPadding(*new* Insets(10, 10, 10, 10));  
 manChosePane.getChildren().addAll(manVarNumL, manVarNumCB, manEqtNumL, manEqtNumCB);  
 *//"manual" elements pane* manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(manVarNumCB.getValue(), genEqtNumCB.getValue());  
 manPane = *new* VBox();  
 manPane.setPadding(*new* Insets(10));  
 manPane.getChildren().addAll(manChosePane, manFieldsPane);  
 layout.setCenter(manPane);  
  
 st.setScene(scene);  
 st.show();  
 }  
  
 *//Метод виводить на укран вікно з графічним рішенням вхідної СЛАР розмірності 2  
 public void* getChartSolving(*double*[][] matrix) {  
 *if* (matrix.length != 2 & matrix[0].length != 3)  
 *throw new* RuntimeException("Matrix has less or more than two variables to solve it by chart");  
 *else* {  
 ChartSolve cs = *new* ChartSolve(matrix);  
 cs.start(*new* Stage());  
 }  
 }  
  
 *//перевіряє коефіцієнти, введені користувачем вручну та записує іх до матриці моделі  
 public void* makeMatrixManually() {  
 *int* length = manCoefsF.length;  
 *int* width = manCoefsF[0].length+1;  
 *double*[][] matrix = *new double*[length][width];  
  
 String var;  
 *for* (*int* i = 0; i < length; i++)  
 *for* (*int* j = 0; j < width; j++) {  
 *if* (j < width-1 && gridInputIsValid(var = manCoefsF[i][j].getText().trim()))  
 matrix[i][j] = Double.*parseDouble*(var);  
 *else if*(j == width-1 && gridInputIsValid(var = manFreeValsF[i].getText().trim()))  
 matrix[i][j] = Double.*parseDouble*(var);  
 *else  
 throw new* InvalidInputException("Matrix grid must contain only decimal or float value followed by '-' if needed");  
 }  
 model.setMatrix(matrix);  
 }  
  
 *//Повертає ТАК, якщо вхідний рядок містить корректні для обробки програмою символи  
 private boolean* gridInputIsValid(String s) {  
 *return* s.matches("-?\\d+([.,]\\d+)?");  
 }  
  
 *// Оформлює сітку елементів, в які можна вводити вхідні коефіцієнти СЛАР заданої розмірності  
 public void* setManualGridPane(*int* varNum, *int* eqtNum) {  
 manCoefsF = *new* TextField[eqtNum][varNum];  
 manFreeValsF = *new* TextField[eqtNum];  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 *for* (*int* j = 0; j < varNum; j++) {  
 manCoefsF[i][j] = *new* TextField("0");  
 manCoefsF[i][j].setMaxSize(30, 30);  
 manFieldsPane.getChildren().add(manCoefsF[i][j]);  
 GridPane.*setConstraints*(manCoefsF[i][j], j, i);  
 *if* (j + 2 == varNum+1) {  
 manFreeValsF[i] = *new* TextField("0");  
 manFreeValsF[i].setMaxSize(30, 30);  
 manFieldsPane.getChildren().add(manFreeValsF[i]);  
 GridPane.*setConstraints*(manFreeValsF[i], j + 1, i);  
 GridPane.*setMargin*(manFreeValsF[i], *new* Insets(0, 10, 0, 10));  
 }  
 }  
 }  
 }  
  
 *//Представляє вхідну матрицю у вигляді лейблу  
 //Зручно для відображення на інтерфейсі  
 public static* Label getLabeledMatrix(*double*[][] matrix) {  
 Label label = *new* Label("");  
 *if* (matrix[0].length == 3) {  
 *for* (*int* i = 0; i < matrix.length; i++)  
 label.setText(label.getText() + matrix[i][0] + " x" + (i+1) + " " + (matrix[i][1] >= 0 ? "+" + matrix[i][1] : matrix[i][1])  
 + " y" + (i+1) + " = " + matrix[i][2] + "\n");  
  
 }*else  
 for* (*int* i = 0; i < matrix.length; i++) {  
 *for* (*int* j = 0; j < matrix[0].length; j++)  
 *if* (j < matrix[0].length - 1) {  
 *int* index = j + 1;  
 *if* (j < matrix[0].length - 2)  
 label.setText(label.getText() + String.*format*(" %.2f x" + index + " + ", matrix[i][j]));  
 *else* label.setText(label.getText() + String.*format*(" %.2f x" + index, matrix[i][j]));  
 } *else* label.setText(label.getText() + String.*format*(" = %.2f " + "%n", matrix[i][j]));  
 }  
 *return* label;  
 }  
  
 *//Готує вікно результату до заповнення  
 private void* setResultStage() {  
 resultStage = *new* Stage();  
 resultStage.setTitle("Result");  
 resultStage.setHeight(500);  
 resultStage.setWidth(500);  
 resultWrittenL = *new* Label();  
 okRB = *new* Button("OK");  
 okRB.setOnAction(ae -> {  
 resultStage.close();  
 });  
 resultPane = *new* VBox();  
 resultPane.getChildren().addAll(resultWrittenL, totalResultL, okRB);  
 VBox.*setMargin*(okRB, *new* Insets(10, 50, 10, 50));  
 resultScene = *new* Scene(resultPane);  
 resultStage.setScene(resultScene);  
 }  
}

*public class* Model {  
 *// зчитувач та записувач у файли  
 private* BufferedReader br;  
 *private* BufferedWriter bw;  
 *//вхідна СЛАР та її розмірність  
 private double*[][] originSlae;  
 *private int* eqtNum;  
 *private int* varNum;  
 *//значення сумісності та єдиності рішення СЛАР, результативна СЛАР та масив результату  
 private int* consistence;  
 *private double*[][] resultSlae;  
 *private double*[] result;  
 *//вихідні та вихідні дані у вигляді рядків для зручного відображення у файлі та інтерфейсі  
 private* String methodName;  
 *private* String originSlaeString, resultSlaeString, resultString, arithmeticsString, totalResultString;  
  
 *//лічильники елементарних арифметичних операцій  
 private static int sumCount* = 0, *subtrCount* = 0, *multCount* = 0, *divCount* = 0;  
  
 *//поветає ХИБНЕ, якщо файл містить некорректні символи  
 //Якщо файл задовільний - записує матрицю з нього до поля класу  
 public boolean* fileIsValid(File input) {  
 *List*<String[]> tempList = *new* LinkedList<>();  
 *try* {  
 br = *new* BufferedReader(*new* FileReader(input));  
 *while* (br.ready())  
 tempList.add(br.readLine().split(" +"));  
 br.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
  
 *int* maxWidth = 0;  
 *for* (*int* i = 0; i < tempList.size(); i++)  
 *if*(tempList.get(i)[0].isEmpty() | tempList.get(i)[0].isBlank())  
 tempList.remove(i);  
 *else* {  
 *for* (String s:tempList.get(i)) {  
 *if*(!s.matches("-?\\d+([.,]\\d+)?"))  
 *return false*;  
 *else if* (s.contains(","))  
 s.replace(',', '.');  
 }  
 *if* (tempList.get(i).length > maxWidth)  
 maxWidth = tempList.get(i).length;  
  
 }  
 *int* eqtNum = 2, eqtWidth = 2;  
 *if* (tempList.size() > eqtNum)  
 eqtNum = tempList.size();  
 *if* (maxWidth > eqtWidth)  
 eqtWidth = maxWidth;  
  
 originSlae = *new double*[eqtNum][eqtWidth];  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 originSlae[i] = getEquationCoefs(tempList.get(i), eqtWidth);  
 }  
 *return true*;  
 }  
  
 *//з вхідного масиву коефіцієнтів у вигляді масиву рядків генерує масив дробових значень величиною рівною другому параметру  
 private double*[] getEquationCoefs(String[] s, *int* eqtWidth) {  
 *double*[] coefs = *new double*[eqtWidth];  
 *for* (*int* i = 0; i < eqtWidth; i++)  
 *if* (i < s.length && !s[i].isEmpty() & !s[i].isBlank())  
 coefs[i] = Double.*parseDouble*(s[i]);  
 *else* coefs[i] = 0;  
 *return* coefs;  
 }  
  
 *//записує результат у файл, що знаходиться у вказаній папці  
 public void* writeResultToFile(String outPath) {  
 File outF = *new* File(outPath);  
 *if* (outF.isDirectory()) {  
 outF = *new* File(outPath + "\\" + (outF.listFiles().length+1) + ".txt");  
 *try* {  
 outF.createNewFile();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *try* {  
 bw = *new* BufferedWriter(*new* FileWriter(outF));  
 bw.write(totalResultString);  
 bw.flush();  
 bw.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *// повертає шлях до вихідного файлу, назву обраного методу вирішення СЛАР, вхідну, вихідну матриці,  
 // результат та кількість зроблених арфиметичних операцій у вигляді одного рядка  
 public* String makeTotalResultString() {  
 *return* totalResultString =  
 "Method: "+ methodName + "\n" +  
 "Input slae :\n" +  
 makeOriginSlaeString() +  
 "\nResult slae : \n" +  
 makeResultSlaeString() +  
 "\nResult :\n" +  
 makeResultString()+  
 makeArithmeticsString();  
 }  
  
 *//повертає кількість викоинаних арифметичних операцій у вигляді одного рядка  
 private* String makeArithmeticsString() {  
 *return* arithmeticsString = "Additions: " + *sumCount* + "\n" +  
 "Subtractions: " + *subtrCount* + "\n" +  
 "Multiplies: " + *multCount* + "\n" +  
 "Divisions: " + *divCount* + "\n";  
 }  
  
 *//повертає вхідну матрицю у вигляді одного рядка  
 private* String makeOriginSlaeString() {  
 *if* (originSlae[0].length == 3) {  
 *for* (*int* i = 0; i < originSlae.length; i++) {  
 originSlaeString += originSlae[i][0] + " x" + (i + 1) + " " + (originSlae[i][1] >= 0 ? "+" + originSlae[i][1] : originSlae[i][1])  
 + " y" + (i + 1) + " = " + originSlae[i][2] + "\n";  
 }  
 } *else* {  
 *for* (*int* i = 0; i < originSlae.length; i++)  
 *for* (*int* j = 0; j < originSlae[0].length; j++) {  
 *int* index = j + 1;  
 *if* (j < originSlae[0].length - 1) {  
 *if* (j < originSlae[0].length - 2) {  
 originSlaeString += String.*format*("%.2f x" + index + (originSlae[i][j + 1] >= 0 ? " +" : " "), originSlae[i][j]);  
 } *else* {  
 originSlaeString += String.*format*(" %.2f x" + index + " ", originSlae[i][j]);  
 }  
 } *else* {  
 originSlaeString += String.*format*(" = %.2f " + "%n", originSlae[i][j]);  
 }  
 }  
 }  
 *return* originSlaeString;  
 }  
  
 *//повертає вихідну матрицю у вигляді одного рядка  
 private* String makeResultSlaeString() {  
 *for* (*int* i = 0; i < resultSlae.length; i++)  
 *for* (*int* j = 0; j < resultSlae[0].length; j++) {  
 *int* index = j + 1;  
 *if* (j < resultSlae[0].length - 1)  
 *if* (j < resultSlae[0].length - 2)  
 resultSlaeString += String.*format*("%.2f x" + index + (originSlae[i][j + 1] >= 0 ? " +" : " "), resultSlae[i][j]);  
 *else* resultSlaeString += String.*format*(" %.2f x" + index, resultSlae[i][j]);  
 *else* resultSlaeString += String.*format*(" = %.2f " + "%n", resultSlae[i][j]);  
  
 }  
 *return* resultSlaeString;  
 }  
 *//повертає результат у вигляді одного рядка  
 private* String makeResultString() {  
 *if* (result.length == 2)  
 resultString += String.*format*("x" + " = " + " %.3f \n", result[0]) + String.*format*("y" + " = " + " %.3f \n", result[1]);  
 *else  
 for* (*int* i = 0; i < result.length; i++) {  
 *int* index = i + 1;  
 resultString += String.*format*("x" + index + " = " + " %.3f \n", result[i]);  
 }  
 *return* resultString;  
  
 }  
  
 *//генерує та поміщає в поле класу СЛАР з дробовими значеннями заданими велечинами  
 public void* generateFloatMatrix(*int* varNum, *int* eqtNum) {  
 *double*[][] matrix = *new double*[eqtNum][varNum + 1];  
 *for* (*int* i = 0; i < eqtNum; i++)  
 *for* (*int* j = 0; j < varNum + 1; j++)  
 matrix[i][j] = Math.*random*() \* 10;  
 *this*.originSlae = matrix;  
 *this*.varNum = varNum;  
 *this*.eqtNum = eqtNum;  
 }  
  
 *//генерує та поміщає в поле класу СЛАР з цілими значеннями заданими велечинами  
 public void* generateIntMatrix(*int* varNum, *int* eqtNum) {  
 *double*[][] matrix = *new double*[eqtNum][varNum + 1];  
 *for* (*int* i = 0; i < eqtNum; i++)  
 *for* (*int* j = 0; j < varNum + 1; j++)  
 matrix[i][j] = Math.*round*(Math.*random*() \* 10);  
 *this*.originSlae = matrix;  
 *this*.varNum = varNum;  
 *this*.eqtNum = eqtNum;  
 }  
  
 *//конвертує матрицю типу BigDecimal до типу double  
 public static double*[][] toDoubleMatrix(BigDecimal[][] m) {  
 *double*[][] res = *new double*[m.length][m[0].length];  
 *for* (*int* i = 0; i < m.length; i++)  
 *for* (*int* j = 0; j < m[0].length; j++)  
 res[i][j] = m[i][j].doubleValue();  
  
 *return* res;  
 }  
  
 *//конвертує матрицю типу double до типу BigDecimal  
 public static* BigDecimal[][] toBigDecimalMatrix(*double*[][] m) {  
 BigDecimal[][] res = *new* BigDecimal[m.length][m[0].length];  
 *for* (*int* i = 0; i < m.length; i++)  
 *for* (*int* j = 0; j < m[0].length; j++)  
 res[i][j] = BigDecimal.*valueOf*(m[i][j]);  
  
 *return* res;  
 }  
 *// онулює значення лічильників арифметичних опреаций  
 private void* nullifyCounters() {  
 *sumCount* = 0;  
 *subtrCount* = 0;  
 *multCount* = 0;  
 *divCount* = 0;  
 }  
  
 *//перевіряє матрицю з поля класу на єдиність рішення, вирішує заданим методом та  
 // записує результат у поле результату класу  
 public void* solveMatrixByMethod(String methodName) {  
 *if* (matrixIsReady()) {  
 *this*.eqtNum = originSlae.length;  
 *this*.varNum = originSlae[0].length - 1;  
 *if* ((*this*.consistence = *getMatrixConsistence*(*cloneMatrix*(originSlae))) == -1) {  
 *throw new* InconsistentMatrixException("Matrix is inconsistent");  
 } *else if* (*this*.consistence == 1)  
 *throw new* InfiniteSolutionsAmountException("Matrix has infinite solutions amount");  
 }  
 *this*.methodName = methodName;  
 nullifyCounters();  
 *if* (methodName == "Gauss") {  
 result = gaussSolve(*cloneMatrix*(originSlae));  
 }  
 *else if* (methodName == "Jordan-Gauss") {  
 result = jordanGaussSolve(*cloneMatrix*(originSlae));  
 }  
 *else if* (methodName == "Rotation") {  
 result = rotationSolve(*cloneMatrix*(originSlae));  
 }  
 }  
 *public double*[] gaussSolve(*double*[][] matrix) {  
 *makeTriangleView*(matrix);  
 *return* backtrace(matrix);  
 }  
  
 *//зводить вхідну матрицю до трикутного вигляду  
 public static void* makeTriangleView(*double*[][] matrix) {  
 *int* eqtNum = matrix.length;  
 *int* eqtWidth = matrix[0].length - 1;  
  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 *int* max = i;  
 *for* (*int* j = i+1; j < eqtNum; j++)  
 *if* (Math.*abs*(matrix[j][i]) < Math.*abs*(matrix[max][j]))  
 max = j;  
  
 *double*[] temp = matrix[i];  
 matrix[i] = matrix[max];  
 matrix[max] = temp;  
  
 *for* (*int* k = i+1; k < eqtNum; k++) {  
 *if* (matrix[i][i] == 0) {  
 *addOneToEachElement*(matrix[i]);  
 *sumCount* += matrix[i].length;  
 }  
 *double* alfa = matrix[k][i] / matrix[i][i];  
 *divCount*++;  
 matrix[k][eqtWidth] -= alfa \* matrix[i][eqtWidth];  
 *multCount*++;  
 *subtrCount*++;  
  
 *for* (*int* j = i; j < eqtNum; j++) {  
 matrix[k][j] -= alfa\*matrix[i][j];  
 *multCount*++;  
 *subtrCount*++;  
 }  
 }  
 }  
 }  
  
 *//додає до кожного елемента вхідного масиву 1  
 private static void* addOneToEachElement(*double*[] arr) {  
 *for* (*int* i = 0; i < arr.length; i++) {  
 arr[i] += 1;  
 }  
 }  
  
 *//повертає масив результату з раніше передбачуваної трикутної матриці  
 private double*[] backtrace(*double*[][] matrix) {  
 *int* length = matrix.length;  
 *int* width = matrix[0].length;  
 *double*[] x = *new double*[length];  
  
 *for* (*int* i = length-1; i >= 0; i--) {  
 *double* sum = 0;  
 *for* (*int* j = i+1; j < length; j++) {  
 sum += matrix[i][j] \* x[j];  
 *multCount*++;  
 *sumCount*++;  
 }  
 *//matrix[i][width-1]  
 //vector[i]* x[i] = (matrix[i][width-1] - sum)/matrix[i][i];  
 *subtrCount*++;  
 *divCount*++;  
 }  
 *this*.resultSlae = matrix;  
 *return* x;  
 }  
  
 *//вирішує передану слар методом Жордана-Гауса, записує результат у поле класу та повертає його  
 public double*[] jordanGaussSolve(*double*[][] matrix) {  
 *int* n = matrix.length;  
 *if* ((matrix[0].length - matrix.length) != 1)  
 *throw new* RuntimeException("Coefficient matrix is not square!");  
 *if* (matrix[0][0] == 0) {  
 *for* (*int* i = 0; i < varNum; i++) {  
 matrix[0][i] += 1;  
 }  
 }  
 *for* (*int* i = 0; i < n; i++) {  
 *double* permittingElement = matrix[i][i];  
  
 *for* (*int* j = 0; j < n+1; j++) { *//making permitting element equal to 1* matrix[i][j] /= permittingElement;  
 *divCount*++;  
 }  
  
 permittingElement = matrix[i][i];  
  
 *for* (*int* j = i+1; j < matrix[0].length; j++)  
 *for* (*int* k = 0; k < n; k++)  
 *if* (k != i) {  
 matrix[k][j] = permittingElement \* matrix[k][j] - matrix[i][j] \* matrix[k][i];  
 *multCount* += 2;  
 *subtrCount*++;  
 }  
  
 *for* (*int* j = 0; j < n; j++)  
 *if* (i != j)  
 matrix[j][i] = 0;  
 }  
  
 *double*[] x = *new double*[n];  
 *for* (*int* j = 0; j < n; j++)  
 x[j] = matrix[j][n];  
 *this*.resultSlae = matrix;  
  
 *return* x;  
 }  
  
 *//вирішує передану слар методом обертань, записує результат у поле класу та повертає його  
 public double*[] rotationSolve(*double*[][] matrix) {  
 *double* c,s, mik;  
 *for* (*int* i = 0 ; i < matrix.length; i++) {  
 *for* (*int* j = i+1; j < matrix.length; j++) {  
 c = matrix[i][i];  
 s = matrix[j][i];  
 *for* (*int* k = 0; k < matrix[0].length; k++) {  
 mik = matrix[i][k];  
 matrix[i][k] = c \* matrix[i][k] + s \* matrix[j][k];  
 matrix[j][k] = c \* matrix[j][k] - s \* mik;  
 *multCount* += 4;  
 *sumCount*++;  
 *subtrCount*++;  
 }  
 }  
 }  
  
 *double* sum;  
 *double*[] x = *new double*[varNum];  
 *for* (*int* i = varNum-1; i >= 0; i--) {  
 sum = 0;  
 *for* (*int* j = i+1; j < eqtNum; j++) {  
 sum += matrix[i][j] \* x[j];  
 *multCount*++;  
 *sumCount*++;  
 }  
 x[i] = (matrix[i][varNum] - sum)/matrix[i][i];  
  
 *subtrCount*++;  
 *divCount*++;  
 }  
 *this*.resultSlae = matrix;  
  
 *return* x;  
 }  
  
 *//повертає -1, якщо матриця несумісна, 0, якщо має єдине значення, 1, якщо безліч  
 public static int* getMatrixConsistence(*double*[][] m) {  
 *makeTriangleView*(m);  
 *int* extendedMatrixRank = *getExtendedMatrixRank*(m);  
 *if*(extendedMatrixRank < m[0].length-1)  
 *return* 1;  
 *return* Integer.*compare*(*getRegularMatrixRank*(m), extendedMatrixRank);  
 }  
  
 *//повертає значення рангу розширеної вхідної матриці  
 private static int* getExtendedMatrixRank(*double*[][] m) {  
 *int* nonZeroRawCounter = 0;  
 *for* (*int* i = 0; i < m.length; i++) {  
 *for* (*int* j = 0; j < m[0].length; j++) {  
 *if*(m[i][j] != 0){  
 nonZeroRawCounter++;  
 j = m[0].length;  
 }  
 }  
 }  
 *return* nonZeroRawCounter;  
 }  
  
 *//повертає значення рангу стандартної частини вхідної матриці  
 private static int* getRegularMatrixRank(*double*[][] m) {  
 *int* nonZeroRawCounter = 0;  
 *for* (*int* i = 0; i < m.length; i++) {  
 *for* (*int* j = 0; j < m[0].length-1; j++) {  
 *if*(m[i][j] != 0){  
 nonZeroRawCounter++;  
 j = m[0].length;  
 }  
 }  
 }  
 *return* nonZeroRawCounter;  
 }  
  
 *//очищає всі результати у вигляді рядків  
 public void* nullifyResultingStrings() {  
 originSlaeString = "";  
 resultSlaeString = "";  
 resultString = "";  
 arithmeticsString = "";  
 totalResultString = "";  
 }  
  
 *//повертає ТАК, якщо поле вхідної матриці класу не дорівнює NULL  
 public boolean* matrixIsReady() {  
 *return this*.originSlae != *null*;  
 }  
  
 *//поветає копію вхідної матриці, що дохволяє проводити над нею зміни не втрачаючи початкового стану  
 public static double*[][] cloneMatrix(*double*[][] matrix) {  
 *double*[][] result = *new double*[matrix.length][matrix[0].length];  
 *for* (*int* i = 0; i < matrix.length; i++)  
 *for* (*int* j = 0; j < matrix[0].length; j++)  
 result[i][j] = matrix[i][j];  
 *return* result;  
 }  
  
 *// поветрає значення кількості змінних СЛАР  
 public int* getVarNum() {  
 *return* varNum;  
 }  
  
 *// поветрає значення кількості рівнянь СЛАР  
 public int* getEquationsNum() {  
 *return* eqtNum;  
 }  
  
 *//передає вхідну матрицю та її размірність до відповіних полів класу  
 public void* setMatrix(*double*[][] matrix) {  
 *this*.originSlae = matrix;  
 *this*.varNum = matrix[0].length - 1;  
 *this*.eqtNum = matrix.length;  
 }  
  
 *//повертає ТАК, якщо маси результату не дорівнює null  
 public boolean* resultExists() {  
 *return* result != *null*;  
 }  
  
 *//повертає вхідну матрицю  
 public double*[][] getOriginSlae() {  
 *return* originSlae;  
 }  
  
}

*public class* ChartSolve *extends* Application {  
 *private* Stage stage; *// Поле вікна програми  
 private* Scene scene; *// Поле контейнера особливого вигляду вікна програми  
 private* StackPane root; *// Контейнер елементів вікна програми  
 private* Pane worldPane; *// Контейнер елементів вікна програми  
 private* String title = "Chart Solve"; *//заголовок вікна  
 //розмірність вікна  
 private double* height =720;  
 *private double* width = 720;  
  
 *//Вхідні та результативні дані  
 private double*[][] slae;  
 *private* Coordinates screenResult, worldResult;  
  
 *//Головні графічні елементи(прямі та точка їх перетину)  
 private* Line l1;  
 *private* Line l2;  
 *private* Shape intersection;  
  
 *//розмірність відносної системи координат(далі - ВСК)  
 private double* worldMinX = -10;  
 *private double* worldMaxX = 10;  
 *private double* worldMinY = -10;  
 *private double* worldMaxY = 10;  
  
 *//здвиг ВСК від лівого верхнього кута вікна  
 private double* offsetX = 0;  
 *private double* offsetY = 0;  
  
 *//масштаб ВСК  
 private double* scaleX = 1.0;  
 *private double* scaleY = 1.0;  
  
 *// метод запуску програми  
 public static void* main(String[] args) {  
 *launch*(args);  
 }  
  
 *//Метод початку роботи програми  
 //На вхід передається об'єкт вікна, в якому буде відображатися графічна частина програми  
 //Тут програма готує основні елемнти та параметри вікна, задає початкові здвиг та масштаб ВСК* @Override  
 *public void* start(Stage stage) {  
 *this*.stage = stage;  
 prepareWindow();  
 scaleX = 10;  
 scaleY = 10;  
  
 offsetX = -width/2/scaleX;  
 offsetY = -height/2/scaleY;  
 update();  
 }  
  
 *//Конструктор, що приймає на вхід вирішувану матрицю  
 //Саме його ми і використовуємо  
 public* ChartSolve(*double*[][] matrix) {  
 *this*.slae = matrix;  
 }  
  
 *public* ChartSolve() {  
  
 }  
  
 *// основний метод побудови графіків  
 //На вхід подається передана в конструкторі матриця  
 // Тут будуються графіки функцій, за необхідності розширюється ВСК, після чого графіки перебудовуются.  
 // Якщо графіки мають спільну точну, вона помічається червоними лініями  
 // Все, що зображено на ВСК розтягується по розмірності вікна  
 private void* solve(*double*[][] m) {  
 BigDecimal[][] matrix = Model.*toBigDecimalMatrix*(Model.*cloneMatrix*(m));  
  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 *if* (hasSingleSolution(Model.*toDoubleMatrix*(matrix))) {  
 intersection = Line.*intersect*(l1, l2);  
 *double* maxCoordinate = getBiggestWorldCoordinate(l1, l2);  
  
 *while* (intersection.getBoundsInParent().isEmpty()) {  
 expandWorld(maxCoordinate);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 intersection = Line.*intersect*(l1, l2);  
 }  
 screenResult = *new* Coordinates(intersection.getBoundsInParent().getCenterX(), intersection.getBoundsInParent().getCenterY());  
 worldResult = screenToWorld(screenResult);  
 *if* (worldMaxX - Math.*abs*(worldResult.x) < 3 || worldMaxY - Math.*abs*(worldResult.y) < 3) {  
 expandWorld(worldMaxX \* 0.3);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 intersection = Line.*intersect*(l1, l2);  
 screenResult = *new* Coordinates(intersection.getBoundsInParent().getCenterX(), intersection.getBoundsInParent().getCenterY());  
 worldResult = screenToWorld(screenResult);  
 }  
  
 updateWorld();  
 equaliseWorldNStage();  
 } *else* {  
 *double* maxCoordinate = getBiggestWorldCoordinate(l1, l2);  
 *while* (!isVisibleInCurrSystem(l1) || !isVisibleInCurrSystem(l2)) {  
 expandWorld(maxCoordinate);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 }  
 updateWorld();  
 }  
  
 l1.setStroke(Color.*BLUE*);  
 l1.setStrokeWidth(2);  
 l2.setStroke(Color.*GREEN*);  
 l2.setStrokeWidth(2);  
  
 updateWith(l1,l2);  
 *if* (hasSingleSolution(Model.*toDoubleMatrix*(matrix)))  
 designateWorldDot(worldResult);  
 }  
  
 *//Масштабує ВСК до розміру вікна та поміщає її центр в центр екрана  
 private void* equaliseWorldNStage() {  
 scaleX = 0.49 \* width / worldMaxX;  
 scaleY = 0.49 \* height / worldMaxY;  
 centrate();  
 }  
  
 *//поміщає центр ВСК в центр екрана  
 private void* centrate() {  
 offsetX = -width/2/scaleX;  
 offsetY = -height/2/scaleY;  
 }  
  
 *// повертає ТАК, якщо вхідна лінія знаходиться в межах розмірності сучасної ВСК  
 private boolean* isVisibleInCurrSystem(Line l) {  
 *double* visBorderX = worldMaxX \* 0.9;  
 *double* visBorderY = worldMaxY \* 0.9;  
  
 Coordinates screenRectCoord = worldToScreen(-visBorderX, visBorderY);  
 *double* rectScreenWidth = visBorderX \* 2 \* scaleX;  
 *double* rectScreenHeight = visBorderY \* 2 \* scaleY;  
  
 Rectangle visibleBorder = *new* Rectangle(rectScreenWidth,rectScreenHeight);  
 visibleBorder.setX(screenRectCoord.x);  
 visibleBorder.setY(screenRectCoord.y);  
 visibleBorder.setWidth(rectScreenWidth);  
 visibleBorder.setHeight(rectScreenHeight);  
 *return* !Line.*intersect*(visibleBorder, l).getBoundsInParent().isEmpty();  
 }  
  
 *//Розщирює розмірність ВСК на вказану величину  
 private void* expandWorld(*double* addSize){  
 worldMaxX += addSize;  
 worldMinX -= addSize;  
 worldMaxY += addSize;  
 worldMinY -= addSize;  
 }  
  
 *// Повертає ТАК, якщо вхідна матриця має єдине рішення  
 private boolean* hasSingleSolution(*double*[][] m) {  
 *return* Model.*getMatrixConsistence*(Model.*cloneMatrix*(m)) == 0;  
 }  
  
  
 *// Повертає значення найбільш віддаленої від початку ВСК координати, що належить передаваній лінії  
 private double* getBiggestWorldCoordinate(Line l) {  
 Coordinates s = screenToWorld(l.getStartX(), l.getStartY());  
 Coordinates e = screenToWorld(l.getEndX(), l.getEndY());  
 *double* max = s.x;  
 *double* num;  
 *if*((num = s.y) > max)  
 max = num;  
 *if*((num = e.x) > max)  
 max = num;  
 *if*((num = e.y) > max)  
 max = num;  
 *return* max;  
 }  
  
 *// Повертає значення найбільш віддаленої від початку ВСК координати, та що є найбільшою серед передаваних лінінй  
 private double* getBiggestWorldCoordinate(Line l1, Line l2) {  
 *double* l1c = getBiggestWorldCoordinate(l1);  
 *double* l2c = getBiggestWorldCoordinate(l2);  
 *return* Math.*max*(l1c, l2c);  
 }  
  
 *//Вказана точка на ВСК відмічається червоними лініями та велечинами своїх координат  
 private void* designateWorldDot(Coordinates worldDot) {  
 Coordinates horStart = worldToScreen(*new* Coordinates(worldDot.x,0));  
 Coordinates horEnd = worldToScreen(*new* Coordinates(worldDot.x, worldDot.y));  
 Coordinates vertStart = worldToScreen(*new* Coordinates(0, worldDot.y));  
 Coordinates vertEnd = worldToScreen(*new* Coordinates(worldDot.x, worldDot.y));  
 Line horizontal = *new* Line(horStart.x, horStart.y, horEnd.x, horEnd.y);  
 Line vertical = *new* Line(vertStart.x, vertStart.y, vertEnd.x, vertEnd.y);  
 Coordinates xStringLoc = worldToScreen(*new* Coordinates(worldDot.x, 0));  
 Coordinates yStringLoc = worldToScreen(*new* Coordinates(0, worldDot.y));  
 Text xString = *new* Text(Double.*toString*(round(worldDot.x,2)));  
 *double* stringWidth = xString.getLayoutBounds().getWidth();  
 *double* stringHeight = xString.getLayoutBounds().getHeight();  
 xString.setStroke(Color.*RED*);  
 xString.setX(xStringLoc.x - stringWidth/2);  
  
 *if*(worldDot.y > 0)  
 xString.setY(xStringLoc.y + stringHeight);  
 *else* xString.setY(xStringLoc.y - 5);  
  
 *if*(worldDot.x > 0 && worldLengthToScreen(worldMaxX - worldDot.x) <= stringWidth)  
 xString.setX(xStringLoc.x - stringWidth - 3);  
 *else if*(worldDot.x < 0 && worldLengthToScreen(worldMinX - worldDot.x) <= stringWidth)  
 xString.setX(xStringLoc.x + 3);  
  
 Text yString = *new* Text(Double.*toString*(round(worldDot.y,2)));  
 yString.setStroke(Color.*RED*);  
 stringWidth = yString.getLayoutBounds().getWidth();  
 stringHeight = yString.getLayoutBounds().getHeight();  
 *if* (worldDot.x > 0)  
 yString.setX(yStringLoc.x - stringWidth - 5);  
 *else* yString.setX(yStringLoc.x + 3);  
  
 yString.setY(yStringLoc.y + stringHeight /3);  
 *if*(worldDot.y > 0 && worldLengthToScreen(worldMaxY - worldDot.y) <= stringHeight)  
 yString.setY(yStringLoc.y + stringHeight + 3);  
 *else if*(worldDot.y < 0 && worldLengthToScreen(worldMinY - worldDot.y) <= stringHeight)  
 yString.setY(yStringLoc.y - 3);  
  
 horizontal.setStroke(Color.*RED*);  
 vertical.setStroke(Color.*RED*);  
 horizontal.setStrokeWidth(2);  
 vertical.setStrokeWidth(2);  
 worldPane.getChildren().addAll(horizontal, vertical, xString, yString);  
 }  
  
 *// округлє передане дробове значення до вказаної кількості знаків після коми  
 private double* round(*double* value, *int* places) {  
 *if* (places < 0) *throw new* IllegalArgumentException();  
 BigDecimal bd = BigDecimal.*valueOf*(value);  
 bd = bd.setScale(places, RoundingMode.*HALF\_UP*);  
 *return* bd.doubleValue();  
 }  
  
 *//заповнює головне вікно необхідними для відображення графіки елементами  
 private void* prepareWindow() {  
 root = *new* StackPane();  
 scene = *new* Scene(root, width, height);  
 worldPane = *new* Pane();  
 root.getChildren().add(worldPane);  
 stage.setTitle(title);  
 stage.setScene(scene);  
 stage.show();  
 }  
  
  *//очищає все, що знаходиться на ВСК на перемальовує осі координат  
 private void* updateWorld() {  
 worldPane.getChildren().clear();  
 drawLines();  
 }  
  
 *//очищає все, що було намальовано на ВСК та наново перемальовує графіки функцій згідно стану програми  
 private void* update() {  
 updateWorld();  
 solve(slae);  
 }  
  
 *//додає до контейнеру ВСК(далі - КВСК) вказані елемети  
 private void* updateWith(Node... n) {  
 worldPane.getChildren().addAll(n);  
 }  
  
 *//додає до КВСК осі координат  
 private void* drawLines() {  
 *int* smallStrWidth = 1;  
 *int* bigStrWidth = 3;  
 *double* dashSize = worldMaxX/30;  
 *int* gap = (*int*)worldMaxX/10;  
 Line l = *null*;  
 Text xAxisName = *new* Text();  
 Text yAxisName = *new* Text();  
 Text dashNum;  
 xAxisName.setFont(Font.*font*(18));  
 yAxisName.setFont(Font.*font*(18));  
 Coordinates screenS, screenE, axisNameScreenLoc, dashNumLoc;  
 equaliseWorldNStage();  
 *for* (*int* x =(*int*) worldMinX; x <= worldMaxX; x++) {  
 *if* (x != 0 & x%gap == 0) {  
 screenS = worldToScreen(x, -dashSize);  
 screenE = worldToScreen(x, dashSize);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(smallStrWidth);  
 dashNum = *new* Text(Integer.*toString*(x));  
 dashNum.setStrokeWidth(1);  
 dashNum.setFont(*new* Font(10));  
 dashNum.setStroke(Color.*GREY*);  
 dashNumLoc = worldToScreen(x - screenLengthToWorld(dashNum.getLayoutBounds().getWidth()/2),  
 -dashSize-screenLengthToWorld(dashNum.getLayoutBounds().getHeight()));  
 dashNum.setX(dashNumLoc.x);  
 dashNum.setY(dashNumLoc.y);  
 worldPane.getChildren().addAll(l, dashNum);  
 } *else if* (x == 0){  
 screenS = worldToScreen(x, worldMinY);  
 screenE = worldToScreen(x, worldMaxY);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(bigStrWidth);  
 xAxisName.setText("X");  
 axisNameScreenLoc = worldToScreen(worldMaxX,0);  
 xAxisName.setX(axisNameScreenLoc.x - xAxisName.getLayoutBounds().getWidth() - 3);  
 xAxisName.setY(axisNameScreenLoc.y + xAxisName.getLayoutBounds().getHeight());  
 worldPane.getChildren().add(xAxisName);  
 worldPane.getChildren().add(l);  
 }  
 }  
 *for* (*int* y = (*int*)worldMinY; y <= worldMaxY; y++) {  
 *if* (y != 0 & y%gap == 0) {  
 screenS = worldToScreen(-dashSize,y);  
 screenE = worldToScreen(dashSize, y);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(smallStrWidth);  
 dashNum = *new* Text(Integer.*toString*(y));  
 dashNum.setStrokeWidth(1);  
 dashNum.setFont(*new* Font(10));  
 dashNum.setStroke(Color.*GREY*);  
 dashNumLoc = worldToScreen(-dashSize - screenLengthToWorld(dashNum.getLayoutBounds().getWidth()) - 0.3,  
 y - screenLengthToWorld(dashNum.getLayoutBounds().getHeight()/2));  
 dashNum.setX(dashNumLoc.x);  
 dashNum.setY(dashNumLoc.y);  
 worldPane.getChildren().addAll(l, dashNum);  
 } *else if* (y == 0){  
 screenS = worldToScreen(worldMinX, y);  
 screenE = worldToScreen(worldMaxX, y);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(bigStrWidth);  
 yAxisName.setText("Y");  
 axisNameScreenLoc = worldToScreen(0, worldMaxY);  
 yAxisName.setX(axisNameScreenLoc.x+4);  
 yAxisName.setY(axisNameScreenLoc.y + yAxisName.getLayoutBounds().getHeight()-5);  
 worldPane.getChildren().add(l);  
 worldPane.getChildren().add(yAxisName);  
  
 }  
  
 }  
  
 }  
  
 *//Конвертує вхідне значення довжини з системи координат вікна(далі - СКВ) до ВСК  
 private double* screenLengthToWorld(*double* length) {  
 *return* length / scaleX;  
 }  
  
 *//Конвертує вхідне значення довжини з ВСК до СКВ  
 private double* worldLengthToScreen(*double* length) {  
 *return* length \* scaleX;  
 }  
  
 *//конвертує вхідне значення координат за ВСК до СКВ  
 private* Coordinates worldToScreen(*double* wx, *double* wy) {  
 *return new* Coordinates(  
 worldXToScreen(wx),  
 worldYToScreen(wy)  
 );  
 }  
 *//конвертує вхідне значення координати "у" за ВСК до СКВ  
 private double* worldYToScreen(*double* wy) {  
 BigDecimal wyBD = BigDecimal.*valueOf*(wy);  
 BigDecimal scaleYBD = BigDecimal.*valueOf*(scaleY);  
 BigDecimal offsetYBD = BigDecimal.*valueOf*(offsetY);  
 *return* wyBD.negate().subtract(offsetYBD).multiply(scaleYBD).doubleValue();  
 }  
 *//конвертує вхідне значення координати "х" за ВСК до СКВ  
 private double* worldXToScreen(*double* wx) {  
 BigDecimal wxBD = BigDecimal.*valueOf*(wx);  
 BigDecimal scaleXBD = BigDecimal.*valueOf*(scaleX);  
 BigDecimal offsetXBD = BigDecimal.*valueOf*(offsetX);  
 *return* wxBD.subtract(offsetXBD).multiply(scaleXBD).doubleValue();  
 }  
  
 *//конвертує вхідне значення координат за ВСК до СКВ  
 private* Coordinates worldToScreen(Coordinates world) {  
 *return new* Coordinates(  
 worldXToScreen(world.x),  
 worldYToScreen(world.y)  
 );  
 }  
 *//конвертує вхідне значення координати "х" з СКВ до ВСК  
 private double* screenXToWorld(*double* sx) {  
 BigDecimal sxBD = BigDecimal.*valueOf*(sx);  
 BigDecimal scaleXBD = BigDecimal.*valueOf*(scaleX);  
 BigDecimal offsetXBD = BigDecimal.*valueOf*(offsetX);  
 *return* sxBD.divide(scaleXBD, RoundingMode.*HALF\_EVEN*).add(offsetXBD).doubleValue();  
 }  
 *//конвертує вхідне значення координати "у" з СКВ до ВСК  
 private double* screenYToWorld(*double* sy) {  
 BigDecimal syBD = BigDecimal.*valueOf*(sy);  
 BigDecimal scaleYBD = BigDecimal.*valueOf*(scaleY);  
 BigDecimal offsetYBD = BigDecimal.*valueOf*(offsetY);  
 *return* syBD.divide(scaleYBD, RoundingMode.*HALF\_EVEN*).add(offsetYBD).negate().doubleValue();  
 }  
  
 *//конвертує вхідне значення координат з СКВ до ВСК  
 private* Coordinates screenToWorld(*double* sx, *double* sy) {  
 *return new* Coordinates(  
 screenXToWorld(sx),  
 screenYToWorld(sy)  
 );  
 }  
 *//конвертує вхідне значення координат з СКВ до ВСК  
 private* Coordinates screenToWorld(Coordinates screen) {  
 *return new* Coordinates(  
 screenXToWorld(screen.x),  
 screenYToWorld(screen.y)  
 );  
 }  
  
 *// Повертає лінію графіка функції, згенеровану згідно передаваним значенням коефіцієнтів при "х" та "у" та вільному члену  
 private* Line drawFuncGraph(BigDecimal xCoef, BigDecimal yCoef, BigDecimal freeCoef){  
 *if*(yCoef.doubleValue() == 0 & xCoef.doubleValue() == 0)  
 *throw new* RuntimeException("Cant draw graph");  
 Coordinates worldStart = *new* Coordinates();  
 Coordinates worldEnd = *new* Coordinates();  
 Coordinates screenStart, screenEnd;  
 BigDecimal wMinYBD = BigDecimal.*valueOf*(worldMinY);  
 BigDecimal wMaxYBD = BigDecimal.*valueOf*(worldMaxY);  
 BigDecimal wMinXBD = BigDecimal.*valueOf*(worldMinX);  
 BigDecimal wMaxXBD = BigDecimal.*valueOf*(worldMaxX);  
 *MathFunctionBD* f = getStraightFunc(xCoef, yCoef, freeCoef);  
  
 *if* (f != *null* && Math.*abs*(getDelta(f)) >= 1) { *//yCoef != 0 & func with xVariable doesnt fit in height size  
 if* ((f = getStraightFunc(yCoef, xCoef, freeCoef)) != *null*) { *//xCoef != 0* worldStart.x = f.count(wMinYBD).doubleValue();  
 worldStart.y = worldMinY;  
 worldEnd.x = f.count(wMaxYBD).doubleValue();  
 worldEnd.y = worldMaxY;  
 } *else* { *//xCoef == 0  
 double* yVal = freeCoef.divide(yCoef).doubleValue();  
 worldStart.x = worldMinX;  
 worldStart.y = yVal;  
 worldEnd.x = worldMaxX;  
 worldEnd.y = yVal;  
 }  
 } *else if*(f != *null*){  
 worldStart.x = worldMinX;  
 worldStart.y = f.count(wMinXBD).doubleValue();  
 worldEnd.x = worldMaxX;  
 worldEnd.y = f.count(wMaxXBD).doubleValue();  
 } *else if*(xCoef.doubleValue() != 0){  
 *double* x = freeCoef.divide(xCoef, 2, RoundingMode.*HALF\_UP*).doubleValue();  
 worldStart.x = x;  
 worldStart.y = worldMinY;  
 worldEnd.x = x;  
 worldEnd.y = worldMaxY;  
 }  
 screenStart = worldToScreen(worldStart);  
 screenEnd = worldToScreen(worldEnd);  
 *return new* Line(screenStart.x, screenStart.y, screenEnd.x, screenEnd.y);  
 }  
  
 *//повертає математичну функцію, згенеровану згідно передаваним значенням коефіцієнтів при "х","у" та вільному члену  
 private MathFunctionBD* getStraightFunc(BigDecimal varCoef, BigDecimal denominator, BigDecimal freeVal){  
 *if* (denominator.doubleValue() == 0)  
 *return null*;  
 *return* var -> (freeVal.subtract(varCoef.multiply(var))).divide(denominator, 2, RoundingMode.*HALF\_UP*);  
 }  
  
 *// повертає значення зміни вхідної математичної функції  
 private double* getDelta(*MathFunctionBD* f) {  
 *return* f.count(BigDecimal.*valueOf*(2))  
 .subtract(f.count(BigDecimal.*valueOf*(1)))  
 .doubleValue();  
 }  
  
 *//функціональний інтерфейс математичної функції  
 interface MathFunctionBD* {  
 BigDecimal count(BigDecimal num);  
 }  
  
 *//класс координат  
 private class* Coordinates{  
 *//значення координат  
 private double* x;  
 *private double* y;  
  
 *private* Coordinates() { }  
  
 *private* Coordinates(*double* x, *double* y) {  
 *this*.x = x;  
 *this*.y = y;  
 }  
  
 *public* String toString() {  
 *return* "Coordinates:" + x + " | " + y;  
 }  
 }  
}