Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

ПОЯСНЮВАЛЬНА ЗАПИСКА

з курсової роботи з навчальної дисципліни "Системне програмування"

Тема: **“Бібліотека функцій для розвязку задач із лінійної алгебри, комбінаторики та теоріх множин”**

***Допущено до захисту***

***Керівник роботи***

Пустоваров В.І.

”\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

***Допущено до захисту***

***Керівник роботи***

Пустоваров В.І.

”\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

***Допущено до захисту***

***Керівник роботи***

Пустоваров В.І.

”\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

Київ 2014 р.

## ЗАВДАННЯ

на курсову роботу з навчальної дисципліни "Системне програмування"

Тема роботи. Бібліотека функцій для розв’язання задач із лінійної алгебри, теорії множин та комбінаторики у мові програмування Java.

**Функціональні вимоги до програми:**

Бібліотека має мати чотири модулі: функції для роботи із векторами (одновимірними масивами), функції для роботи із матрицями (двохвимірними масивами), функції для роботи із комбінаторикою та функції для роботи із множинами.

Модуль функцій для роботи із векторами повинен містини наступні функції:

* Повернення та встановлення значення вектору
* Арифметичні дії між парами векторів
* Арифметичні дії із елементами вектору
* Перевірка пари векторів на рівність

Модуль функцій для роботи з матрицями:

* Повернення та встановлення значення матриці
* Повернення та встановлення значень рядків, стовпців та елементів матриці
* Перевірки матриці – чи є вона:
  + Нульова, одинична
  + Діагональна, трикутна, східчаста
  + Узгодженою, переставною
  + Симетричною, кососиметричною
* Перевірки на рівність стовпців, рядків, елементів
* Перевірка на рівність пари матриць
* Арифметичні дії із матрицями, їх стовпцями, рядками та елементами
* Натуральний степінь матриці
* Транспонування матриці
* Обчислення визначнику
* Обчислення мінору та алгебричного доповнення до елементу матриці
* Пошук рангу матриці
* Пошук оберненої матриці
* Розв’язання СЛАР

Модуль функцій для роботи із комбінаторикою:

* Розрахунок кількості перестановок, розміщень, розміщень із повтореннями та сполучень
* Пошук всіх перестановок вектору у лексикографічному порядку
* Генератор двійкових векторів у лексикографічному порядку
* Пошук всіх сполучень у лексикографічному порядку

Модуль функцій для роботи із множинами:

* Операції об’єднання, перетину, різниці, симетричної різниці, доповнення

**Вимоги до форматів вхідних і вихідних даних програми:**

Вхідні дані у програмі повинні бути стандартних числових типів Java, або об’єктами класів представлених у роботі. Для операцій із комбінаторикою вхідними даними можуть бути масиви із будь-яким типом даних.

**Вимоги до програмних та апаратних інструментів:**

Бібліотека може використовуватись на будь якому пристрої, що підтримує JVM, як допоміжний ресурс для розв’язання задач.

**Рекомендована література:**

1. Документація Java™ Platform, Standard Edition 7,

http://docs.oracle.com/javase/7/docs/api/

1. Лінійна алгебра та аналітична геометрія. Практикум. (І курс І семестр) / Уклад.: І. В. Алєксєєва, В. О. Гайдей, О. О. Диховичний, Л. Б. Федорова. — К: НТУУ «КПІ», 2011. — 184 с.
2. Методичні вказівки до виконання лабораторних робіт з дисципліни

«Комп’ютерна інженерія». /Укл.: Марковський О.Г, Новотарський М.А.: НТУУ «КПІ»- 2013, 46 с.

Завдання на роботу видав Завдання на роботу прийняв

Керівник роботи Студент групи ІО-24

\_\_\_\_\_\_\_\_\_\_\_\_ Пустоваров В.І. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Репета Я. Г.

(підпис) (підпис студента)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р. ”\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

**Зміст**

Вступ……………………………………………………………………...………..6

1.Теоретичні відомості ……………………………………………………….…..7

1.1.Організація бібліотеки………………………………………………………..7

1.2.Особливості методів…………………………………………………………..7

2.Розробка програми…………………………………………………………...…8

2.1.Алгоритм роботи програми………………………………………………..…8

2.1.1.Алгоритм роботи методу множення матриць…………………………..…8

2.1.2. Алгоритм роботи методу перетину множин…………………………...…9

3. Опис основних класів програмного додатку………………………………..11

4.Інструкція з експлуатації програмного додатку для користувача………….12

5.Тестування ……………………………………………………………………..15

6. Висновок ………………………………………………………………………16

7. Список використаної літератури …………………………………………….17

Додаток А.(код програми) ……………………………………………………...18

ДодатокБ.(детальна інструкція з експлуатації для програміста клієнта)……48

**Вступ**

З кожним роком світ стає більш комп’ютеризованим та роботехнічним. Зараз важко уявити світ без комп’ютерів. З кожним роком з’являються все нові технології та програмне забезпечення, які покращують та полегшують життя людству.

Але незалежно від того, що технологія у собі вміщає, який результат вона видає та у якій сфері діяльності людини застосовується можна впевнено сказати, що для її розв’язання використовувалися масиви числових даних. І це веде до однієї із найпоширеніших проблем: програміст витрачає час на написання алгоритмів обробки масивів числових даних, таким чином він витрачає цей час дарма, оскільки нічому новому не навчається, а пише алгоритми, які він використовує в своїй практиці не вперше.

В даній курсовій роботі я планую розробити програмний модуль із бібліотекою найважливіших функцій по обробці масивів із числовими даними. Таким чином розробник замість наступного написання алгоритму простої обробки зможе просто підключити дану бібліотеку у свій проект та виконати потрібну йому функцію лише за допомогою одно рядка програми.

Після завершення курсової роботи я отримаю базові навички у розробці власної бібліотеки, а також закріплю свої знання із лінійної алгебри, теорії множин та комбінаторики.

Застосування біліотеки можливо для будь-якої програми, яка буде писатися на мові програмування Java.

1. **Теоретичні відомості**

У даній курсовій роботі буде розроблено програму для обробки масивів числових даних. Вона міститиме функціонал відповідно до технічного завдання. Мною була вибрана мова програмування Java, середовище розробки Eclipse. Програма буде відповідати технічному завданню.

**1.1 Організація бібліотеки**

Програма складається із чотирьох класів. Кожен відповідає функції відведені конкретному напрямку. Кожен клас складається із набору методів, для обробки масивів числових даних, таким чином виділити загальний алгоритм розроблюваного програмного модуля є неможливим.

* 1. **Особливості методів**

Методи представлені у програмі можна поділити на три групи:

* Методи, у яких ми працюємо із внутрішнім представлення матриць та векторів, і міняємо їх значення виконуючи дії над ними
* Методи, у яких ми використовуємо внутрішнє значення масиву чи вектору, щоб обрахувати певні величини, що мають до них відношення
* Методи, у яких ми перевіряємо внутрішнє представлення для перевірки та присвоєння певних ознак матриці чи вектору

1. **Розробка програми**

В даному розділі буде детально розглянуто розробку програмного модуля для роботи із задачами з сфери лінійної алгебри, комбінаторики та теорії множин. Будуть представлені приклади алгоритмів деяких важливих методів, та їх програмна реалізація на мові програмування Java. Версія даної програми буде 1.0. Вона буде готова для використання але для максимальної продуктивності роботи із бібліотекою потрібно буде додати до даної бібліотеки методи для розв’язання задач із теорії графів та вищої математики.

**2.1 Алгоритм роботи програми**

Як зазначалося вище бібліотека немає загального алгоритму роботи. Користувач вибирає певний метод із певного класу для розв’язання відповідної задачі. Методи є незалежними між собою. Приведемо приклади двох алгоритмів.

**2.1.1 Алгоритм роботи методу множення матриць**

У даному алгоритмі A, B – вхідні матриці, C – матриця результату. X – висота матриці A, Y – довжина матриці B, Z – довжина матриці A або висота матриці Y.

Код методу:

**double**[][] result = **new** **double**[matrix.length][matr[0].length];

**for** (**int** i = 0; i < result.length; i++) {

**for** (**int** j = 0; j < result[i].length; j++) {

**for** (**int** k = 0; k < matrix[0].length; k++) {

result[i][j] += matrix[i][k] \* matr[k][j].doubleValue();

}

}

}

**return** result;



Рис. 2.1.1.1 – Алгоритм роботи методу множення матриць

**2.1.2 Алгоритм роботи методу перетину множин**

У даному алгоритмі A, B – вхідні вектори, C – вектор результату. X – розмірність A, Y – розмірність B.

Код методу:

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v1.length; i++) {

**for** (**int** j = 0; j < v2.length; j++) {

**if** (v1[i] == v2[j]) {

mass.add(v1[i]);

**break**;

}

}

}

**return** mass.toArray(**new** Double[mass.size()]);



Рис. 2.1.1.2 – Алгоритм роботи методу перетину множин

1. **Опис основних класів програмного додатку**

До складу проекту бібліотеки, орієнтованої на можливість використання в числі інших бібліотек чи програм стандартних функцій входять модулі з наступними вхідними файлами:

* Клас java.util.ArrayList - призначений для створення масивів із зміною довжиною, це стандартний клас Java, тому його опис можна опустити.
* Клас Matrix – призначений для створення об’єкту матриці за допомогою двохвимірного масиву; він містить набір методів для обробки даних, що знаходяться в матриці, для вводу/виводу даних із матриці, для обрахування різних величин, які можна отримати із двохвимірного масиву.
* Клас Vector – призначений для створення об’єкту типу вектор, за допомогою одновимірного масиву даних; містить набір методів для обробки даних, що знаходяться у векторі, для вводу/виводу даних із вектора, для обрахування різних величин, які можна отримати із одновимірного масиву.
* Клас Combinations – призначений для обрахування кількості комбінацій перестановок, сполучень та розміщень із повтореннями та без; також, містить методи для виводу всіх таких комбінацій у лексикографічному порядку.
* Клас Set – призначений для методів операцій із теорії множин, таких як: об’єднання, перетин, різниця, симетрична різниця та доповнення.

1. **Інструкція з експлуатації програмного додатку для користувача**

Для використання розробленої бібліотеки потрібно мати середовище із встановленою JVM та компілятором для мови програмування Java. Розглянемо інструкцію по користуванню на прикладі використання методу множення матрицю.

Спочатку потрібно імпортувати jar-файл «CourseWork.jar», який можна побачити на малюнку.

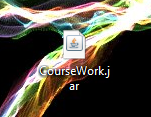


Рис. 4.1 – JarFile

Сам імпорт файлу потрібно зробити відповідно до правил вибраного компілятору. Для Eclipse це можна зробити так:

1) Створити проект у якому буде використовуватися бібліотека.

2) Натиснути правої клавішею миші на створеному проекті і в спливаючому вікні вибрати пункт «Build Path», та підпункт «Configure Build Path…» - Рис. 4.2

3) У спливаючому вікні вибрати вкладку «Libraries» та натиснути кнопку «Add External JARs…» - Рис. 4.3

4) У наступному вікні знайти розроблену бібліотеку на комп’ютері, вибрати її та підтвердити вибір – Рис. 4.4

5) Тепер є доступним імпорт цієї бібліотеки. Для нашої цілі створимо клас із виконавчим методом «main» та імпортуємо в нього клас Matrix із бібліотеки - Рис. 4.5

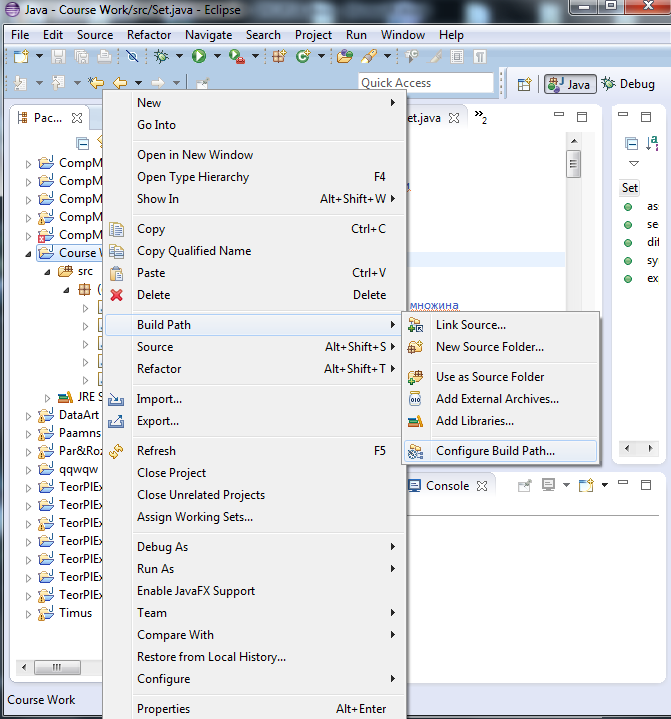


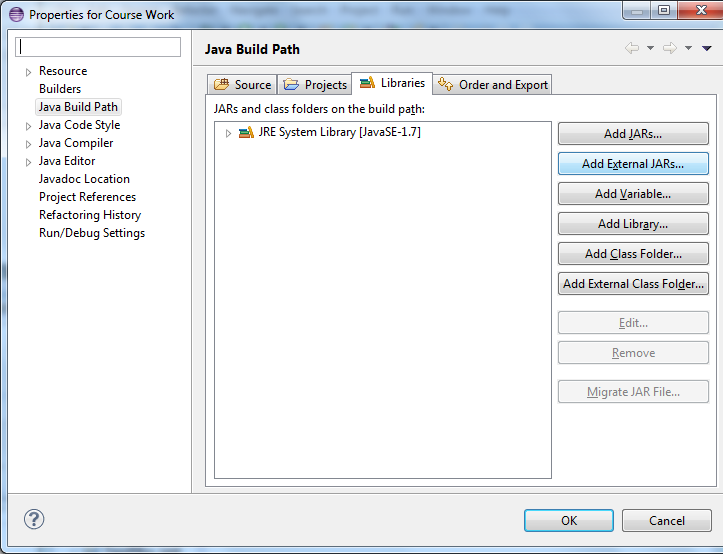
Рис. 4.2 - «Build Path» - «Configure Build Path…»

Рис. 4.3 - «Libreries» - «Add External JARs…»

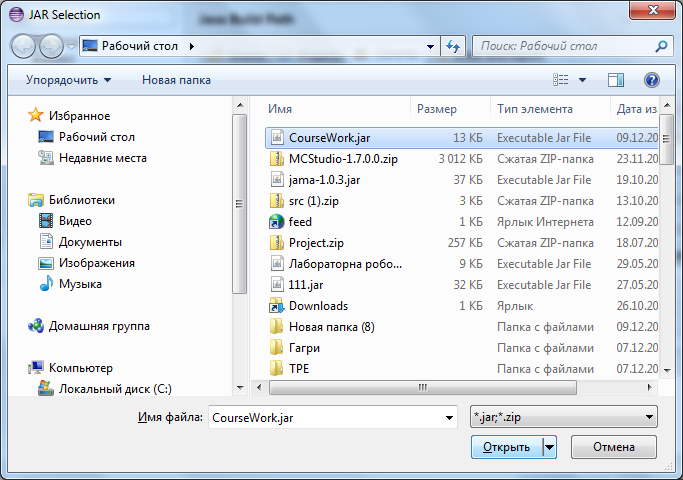


Рис. 4.4 - Пошук бібліотеки

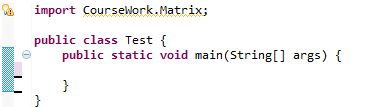


Рис. 4.5 – Використання бібліотеки

6) Далі створимо три масиви, два із якиз ініціалізуємо даними, а третій залишимо, щоб потім у ньому зберегти результат множення. Зробимо це за допомогою таких рядків:

Double[][] matrix1 = { { 1d, 2d, 3d }, { 4d, 5d, 6d },

{ 7d, 8d, 9d } };

Double[][] matrix2 = { { 9d, 8d, 7d }, { 6d, 5d, 4d },

{ 3d, 2d, 1d } };

Double[][] matrix3;

7) Далі створимо із перших двох масивів екземпляри класу Matrix, передавши їх у конструктор класу Matrix. Це можна зробити за допомогою таких рядків:

Matrix<Double> mat1 = **new** Matrix<>(matrix1);

Matrix<Double> mat2 = **new** Matrix<>(matrix2);

8) Далі викличемо метод mulMatrix об’єкту mat1 передавши в параметрі об’єкт mat2 та присвоїмо матриця, яка повертається стореному для цього масиву matrix3. Це можна зробити за допомогою такого рядка:

matrix3 = mat1.mulMatrix(mat2);

Таким чином у масиві matrix3 буде результат множення matrix1 на matrix2.

Повний приклад програми можна побачити у наступному розділі. Де буде показано також, правильність множення матриць.

В даному розділі було створено інструкцію по підключенні бібліотеки до середовища Eclipse. В ній було описано всі деталі які повинен знати користувач, щоб могти її експлуатувати.

1. **Тестування**

Кожен метод провірявся при трьох різних вхідних даних на правильність обрахунку. Використовувався інтернет-ресурс, який дає робить подібні обрахунки. Результати збігалися у всіх методах. Отже представлені алгоритми методів є вірними. Перевірку на множення матриць можна побачити нижче на рисунках.

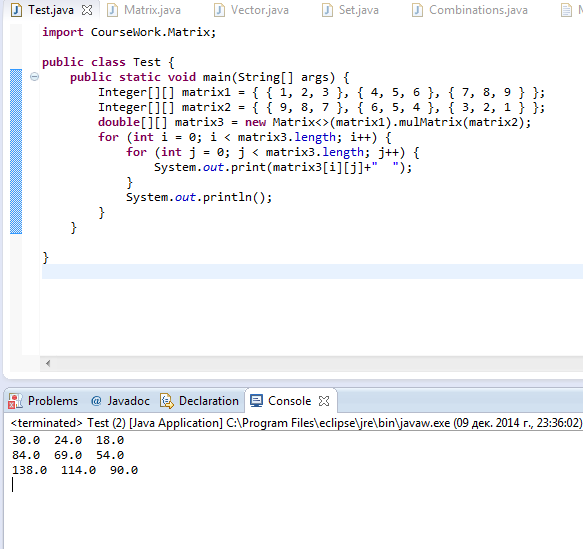


Рис. 5.1 Вивід результату за допомогою розробленої бібліотеки

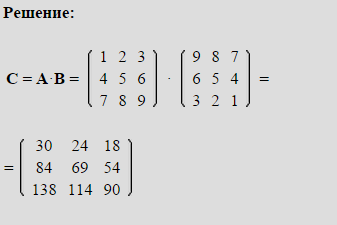


Рис. 5.2 Вивід результату за допомогою інтернет-ресурсу

1. **Висновок**

Задачі обробки масивів числових даних дуже сильно розповсюджені і використовуються практично у кожній програмі. Розроблена бібліотека складає основну базу функцій із лінійної алгебри, комбінаторики та теорії множин. Таким чином будь-який розробник може використовувати дану бібліотеку, і за допомогою не економити свій час. Була розроблена інструкція по експлуатації для користувача, тому проблем із використанням бібліотеки виникати не повинно. Дана бібліотека була розроблена правильно, і видає правильно результати, що підтвердили тести, зокрема, і той що був наведений у розділі «Тестування».

1. **Список використаної літератури**
2. Документація Java™ Platform, Standard Edition 7,

http://docs.oracle.com/javase/7/docs/api/

1. Лінійна алгебра та аналітична геометрія. Практикум. (І курс І семестр) / Уклад.: І. В. Алєксєєва, В. О. Гайдей, О. О. Диховичний, Л. Б. Федорова. — К: НТУУ «КПІ», 2011. — 184 с.
2. Методичні вказівки до виконання лабораторних робіт з дисципліни

«Комп’ютерна інженерія». /Укл.: Марковський О.Г, Новотарський М.А.: НТУУ «КПІ»- 2013, 46 с.

**Додаток А. (код програми)**

**package** CourseWork;

**import** java.util.ArrayList;

/\*\*

\* Клас роботи із комбінаторикою

\* **@author** Sluva

\*

\*/

**public** **class** Combinations {

/\*\*

\* Обраховує кількість перестановок із N елементів

\* **@param** n - кількість елементів у комбінації

\* **@return** Кількість перестановок

\*/

**public** **int** permutationAmount(**int** n) {

**return** factorial(n);

}

/\*\*

\* Обраховує кількість розміщень із N елементів по M елементів

\* **@param** n - кількість доступних унікальних елементів

\* **@param** m - кількість елементів у комбінації

\* **@return** Кількість розміщень

\*/

**public** **int** distributionAmount(**int** n, **int** m) {

**return** factorial(n) / factorial(n - m);

}

/\*\*

\* Обраховує кількість сполучень із N елементів по M

\* **@param** n - кількість доступних унікальних елементів

\* **@param** m - кількість елементів у комбінації

\* **@return** кількість сполучень

\*/

**public** **int** combinationAmount(**int** n, **int** m) {

**return** factorial(n) / (factorial(n - m) \* factorial(m));

}

/\*\*

\* Обраховує кількість розміщень з повтореннями із N елементів

\* **@param** n - кількість доступних унікальних елементів

\* **@param** m - кількість елементів в комбінації

\* **@return** Кількість розміщень з повтореннями

\*/

**public** **int** distributionWithRepetitionAmount(**int** n, **int** m) {

**int** s = 1;

**for** (**int** i = 0; i < m; i++) {

s \*= n;

}

**return** s;

}

/\*\*

\* Знаходить всі перестановки із вектору

\* **@param** vector - початковий вектор

\* **@return** Масив об'єктів із масивами перестановок

\*/

**public** <V> Object[] permutation(V[] vector) {

**int**[] v1 = **new** **int**[vector.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = i;

}

ArrayList<V[]> mass = **new** ArrayList<>();

mass.add(vector);

**while** (**true**) {

**int** k = -1;

**for** (**int** i = v1.length - 2; i >= 0; i--) {

**if** (v1[i] < v1[i + 1]) {

k = i;

**break**;

}

}

**if** (k == -1)

**break**;

**int** m = -1;

**for** (**int** i = v1.length - 1; i > k; i--) {

**if** (v1[k] < v1[i]) {

m = i;

**break**;

}

}

**int** l = v1[k];

v1[k] = v1[m];

v1[m] = l;

V p = vector[k];

vector[k] = vector[m];

vector[m] = p;

**int**[] massint = v1.clone();

V[] massV = vector.clone();

**for** (**int** i = k + 1; i < massV.length; i++) {

v1[i] = massint[massV.length + k - i];

vector[i] = massV[massV.length + k - i];

}

}

**return** mass.toArray();

}

/\*\*

\* Генератор двійкових векторів довжини N

\* **@param** n - довжина вектору

\* **@return** Масив двійкових векторів

\*/

**public** Integer[] binaryVector(**int** n) {

ArrayList<Integer[]> result = **new** ArrayList<>();

Integer[] mass = **new** Integer[n];

result.add(mass);

**for** (**int** i = 0; i < Math.*pow*(2, n) - 1; i++) {

addBinary(mass, n);

result.add(mass);

}

**return** result.toArray(**new** Integer[result.size()]);

}

/\*\*

\* Пошук всіх сполучень із заданого вектору і заданої довжини

\* **@param** mass - початковий вектор

\* **@param** n - довжина комбінації

\* **@return** Масив векторів сполучень

\*/

**public** Object[] combination(Object[] mass, **int** n) {

Integer[] v = **new** Integer[n];

**for** (**int** i = 0; i < v.length; i++) {

v[i] = i;

}

ArrayList<Integer[]> all = **new** ArrayList<>();

all.add(v);

**for** (**int** i = 0; i < combinationAmount(n, mass.length); i++) {

addComb(v, n);

all.add(v);

}

**return** all.toArray();

}

/\*\*

\* Пошук наступного сполучення із заданого і заміна ним поточного

\* **@param** mass - вхідне сполучення

\* **@param** n - елемент сполучення, який змінюється

\*/

**private** **void** addComb(Integer[] mass, **int** n) {

**if** (mass[n - 1] < n)

mass[n - 1]++;

**else**

addComb(mass, n - 1);

}

/\*\*

\* Пошук наступного двійкого вектору із заданого та заміна ним поточного

\* **@param** mass - вхідний двійковий вектор

\* **@param** n - елемент вектору, який підлягає зміні

\*/

**private** **void** addBinary(Integer[] mass, **int** n) {

**if** (mass[n] == 0)

mass[n] = 1;

**else** {

mass[n] = 0;

addBinary(mass, n - 1);

}

}

/\*\*

\* Обчислення факторіалу числа

\* **@param** n - вхідне число

\* **@return** Значення факторіалу

\*/

**private** **int** factorial(**int** n) {

**if** (n == 1)

**return** 1;

**return** factorial(n - 1) \* n;

}

}

**package** CourseWork;

/\*\*

\* Клас обробки данних типу "Матриця"

\* **@author** Sluva

\*

\* **@param** <T> - тип данних у матриці

\*/

**public** **class** Matrix<T **extends** Number> {

/\*\*

\* Поле зберігання поточного значення елементів матриці

\*/

**private** **final** **double**[][] matrix;

/\*\*

\* Конструктор класу

\* **@param** matrix - двовимірний масив числових данних

\*/

**public** Matrix(T matrix[][]) {

**this**.matrix = **new** **double**[matrix.length][matrix[0].length];

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**this**.matrix[i][j] = matrix[i][j].doubleValue();

}

}

}

/\*\*

\* Повертає елемент матриці

\* **@param** x - номер рядка

\* **@param** y - номер стовпця

\* **@return** Елемент матриці із координатами масиву x,y

\*/

**public** **double** getValueElement(**int** x, **int** y) {

**return** matrix[x][y];

}

/\*\*

\* Повертає стовпець матриці

\* **@param** column - індекс стовпця

\* **@return** Стовпець матриці із індексом column

\*/

**public** **double**[] getColumn(**int** column) {

**double**[] vector = **new** **double**[matrix.length];

**for** (**int** i = 0; i < matrix.length; i++) {

vector[i] = matrix[column][i];

}

**return** vector;

}

/\*\*

\* Повертає рядок матриці

\* **@param** row - індекс рядка

\* **@return** Рядок матриці із індексом row

\*/

**public** **double**[] getRow(**int** row) {

**return** matrix[row].clone();

}

/\*\*

\* Поветрає матрицю

\* **@return** Матриця, яка підлягає обробці методами

\*/

**public** **double**[][] getMatrix() {

**return** matrix;

}

/\*\*

\* Перевіряє чи матриця нульова

\* **@return** true - матриця нульова, false - не нульова

\*/

**public** **boolean** isZeroMatrix() {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matrix[i][j] != 0)

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє чи матриця квадратна

\* **@return** true - матриця квадратна, false - не квадратна

\*/

**public** **boolean** isSquareMatrix() {

**return** matrix.length == matrix[0].length;

}

/\*\*

\* Перевіряє чи матриця нижня трикутна

\* **@return** true - матриця нижня трикутна, false - не нижня трикутна

\*/

**public** **boolean** isLowerTriangularMatrix() {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = i + 1; j < matrix[i].length; j++) {

**if** (matrix[i][j] != 0)

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє чи матриця верхня трикутна

\* **@return** true - матриця верхня трикутна, false - не верхня трикутна

\*/

**public** **boolean** isUpperTriangularMatrix() {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < i; j++) {

**if** (matrix[i][j] != 0)

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє чи матриця діагональна

\* **@return** true - матриця діагональна, false - не діагональна

\*/

**public** **boolean** isDiagonalMatrix() {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matrix[i][j] != 0 && i != j)

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє чи матриця одинична

\* **@return** true - матриця одинична, false - не одинична

\*/

**public** **boolean** isUnitaryMatrix() {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matrix[i][j] != 0 && i != j)

**return** **false**;

**if** (matrix[i][j] != 1 && i == j)

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє рівність стовпців

\* **@param** column1 - індекс першого стовпця

\* **@param** column2 - індекс другого стовпця

\* **@return** true - стовпці рівні, false - не рівні

\*/

**public** **boolean** isEqualityColumns(**int** column1, **int** column2) {

**for** (**int** i = 0; i < matrix.length; i++) {

**if** (matrix[i][column1] != matrix[i][column2])

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Перевіряє рівність рядків

\* **@param** row1 - індекс першого рядка

\* **@param** row2 - індекс другого рядка

\* **@return** true - рядки рівні, false - не рівні

\*/

**public** **boolean** isEqualityRows(**int** row1, **int** row2) {

**for** (**int** i = 0; i < matrix[row1].length; i++) {

**if** (matrix[row1][i] != matrix[row2][i])

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Додає до стовпця інший стовпець

\* **@param** column1 - індекс цільового стовпця

\* **@param** column2 - індекс стовпця, що додають

\*/

**public** **void** addColumns(**int** column1, **int** column2) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column1] += matrix[i][column2];

}

}

/\*\*

\* Додає до стовпця вектор

\* **@param** column - індекс цільового стовпця

\* **@param** vect - одновимірний масив типу Vector

\*/

**public** **void** addColumns(**int** column, Vector<?> vect) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column] += vect.getVector()[i];

}

}

/\*\*

\* Додає до стовпця одновимірний масив

\* **@param** column - індекс цільового стовпця

\* **@param** vector - одновимірний числовий масив \*/

**public** <V **extends** Number> **void** addColumns(**int** column, V[] vector) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column] += vector[i].doubleValue();

}

}

/\*\*

\* Додає до рядка інший рядок

\* **@param** row1 - індекс цільового рядка

\* **@param** row2 - індекс рядка, що додають

\*/

**public** **void** addRows(**int** row1, **int** row2) {

**for** (**int** i = 0; i < matrix[row1].length; i++) {

matrix[row1][i] += matrix[row2][i];

}

}

/\*\*

\* Додає до рядка вектор

\* **@param** row - індекс цільового рядка

\* **@param** vect - одновимірний масив типу Vector

\*/

**public** **void** addRows(**int** row, Vector<?> vect) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[row][i] += vect.getVector()[i];

}

}

/\*\*

\* Додає до рядка одновимірний числовий масив

\* **@param** row - індекс цільового рядка

\* **@param** vector - одновимірний числовий масив

\*/

**public** <V **extends** Number> **void** addRows(**int** row, V[] vector) {

**for** (**int** i = 0; i < matrix[row].length; i++) {

matrix[row][i] += vector[i].doubleValue();

}

}

/\*\*

\* Віднімає від стовпця інший стовпець

\* **@param** column1 - індекс цільового стовпця

\* **@param** column2 - індекс стовпця, що віднімають

\*/

**public** **void** subColumns(**int** column1, **int** column2) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column1] -= matrix[i][column2];

}

}

/\*\*

\* Віднімає від стовпця вектор

\* **@param** column - індекс цільового стовпця

\* **@param** vect - одновимірний масив типу Vector

\*/

**public** **void** subColumns(**int** column, Vector<?> vect) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column] -= vect.getVector()[i];

}

}

/\*\*

\* Віднімає від стовпця одновимірний масив

\* **@param** column - індекс цільового стовпця

\* **@param** vector - одновимірний числовий масив

\*/

**public** <V **extends** Number> **void** subColumns(**int** column, V[] vector) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column] -= vector[i].doubleValue();

}

}

/\*\*

\* Віднімає від рядка інший рядок

\* **@param** row1 - індекс цільового рядка

\* **@param** row2 - індекс рядка, що віднімають

\*/

**public** **void** subRows(**int** row1, **int** row2) {

**for** (**int** i = 0; i < matrix[row1].length; i++) {

matrix[row1][i] -= matrix[row2][i];

}

}

/\*\*

\* Віднімає від рядка вектор

\* **@param** row - індекс цільового рядка

\* **@param** vect - одновимірний масив типу Vector

\*/

**public** **void** subRows(**int** row, Vector<?> vect) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[row][i] -= vect.getVector()[i];

}

}

/\*\*

\* Віднімає від рядка одновимірний числовий масив

\* **@param** row - індекс цільового рядка

\* **@param** vector - одновимірний числовий масив

\*/

**public** <V **extends** Number> **void** subRows(**int** row, V[] vector) {

**for** (**int** i = 0; i < matrix[row].length; i++) {

matrix[row][i] -= vector[i].doubleValue();

}

}

/\*\*

\* Додає до елемента матриці інший елемент

\* **@param** row1 - індекс рядка цільового елементу

\* **@param** column1 - індекс стовпця цільового елементу

\* **@param** row2 - індекс рядка елементу, який додають

\* **@param** column2 - індекс стовпця елементу, який додають

\*/

**public** **void** addElements(**int** row1, **int** column1, **int** row2, **int** column2) {

matrix[row1][column1] += matrix[row2][column2];

}

/\*\*

\* Додає до елемента матриці числове значення

\* **@param** row - індекс рядка цільового елементу

\* **@param** column - індекс стовпця цільового елементу

\* **@param** value - число, яке додають

\*/

**public** <V **extends** Number> **void** addElement(**int** row, **int** column, V value) {

matrix[row][column] += value.doubleValue();

}

/\*\*

\* Віднімає від елемента матриці інший елемент

\* **@param** row1 - індекс рядка цільового елементу

\* **@param** column1 - індекс стовпця цільового елементу

\* **@param** row2 - індекс рядка елементу, який віднімають

\* **@param** column2 - індекс стовпця елементу, який віднімають

\*/

**public** **void** subElements(**int** row1, **int** column1, **int** row2, **int** column2) {

matrix[row1][column1] -= matrix[row2][column2];

}

/\*\*

\* Віднімає від елемента матриці числове значення

\* **@param** row - індекс рядка цільового елементу

\* **@param** column - індекс стовпця цільового елементу

\* **@param** value - число, яке віднімають

\*/

**public** <V **extends** Number> **void** subElements(**int** row, **int** column, V value) {

matrix[row][column] -= value.doubleValue();

}

/\*\*

\* Множення стовпця матриці на число

\* **@param** column - індекс стовпця

\* **@param** value - число, на яке множать

\*/

**public** <V **extends** Number> **void** mulColumnOnNumber(**int** column, V value) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][column] \*= value.doubleValue();

}

}

/\*\*

\* Множення рядка матриці на число

\* **@param** row - індекс рядка

\* **@param** value - число, на яке множать

\*/

**public** <V **extends** Number> **void** mulRowOnNumber(**int** row, V value) {

**for** (**int** i = 0; i < matrix.length; i++) {

matrix[i][row] \*= value.doubleValue();

}

}

/\*\*

\* Перевіряє матриці на рівність

\* **@param** matr - матриця типу Matrix

\* **@return** true - матриці рівні, false - не рівні

\*/

**public** **boolean** isEquality(Matrix<?> matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matr.getMatrix()[i][j] != matrix[i][j])

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевіряє матриці на рівність

\* **@param** matr - матриця типу двовимірного числового масиву

\* **@return** true - матриці рівні, false - не рівні

\*/

**public** <V **extends** Number> **boolean** isEquality(V[][] matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matr[i][j].doubleValue() != matrix[i][j])

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Додає до матриці іншу матрицю

\* **@param** matr - матриця типу Matrix, яку додають

\*/

**public** **void** addMatrix(Matrix<?> matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

matrix[i][j] += matr.getMatrix()[i][j];

}

}

}

/\*\*

\* Додає до матриці іншу матрицю

\* **@param** matr - матриця типу числового двовимірного масиву, яку додають

\*/

**public** <V **extends** Number> **void** addMatrix(V[][] matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

matrix[i][j] += matr[i][j].doubleValue();

}

}

}

/\*\*

\* Віднімає від матриці іншу матрицю

\* **@param** matr - матриця типу Matrix, яку віднімають

\*/

**public** **void** subMatrix(Matrix<?> matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

matrix[i][j] -= matr.getMatrix()[i][j];

}

}

}

/\*\*

\* Віднімає від матриці іншу матрицю

\* **@param** matr - матриця типу числового двовимірного масиву, яку віднімають

\*/

**public** <V **extends** Number> **void** subMatrix(V[][] matr) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

matrix[i][j] -= matr[i][j].doubleValue();

}

}

}

/\*\*

\* Множення матриці на число

\* **@param** value - число, на яке множать

\*/

**public** <V **extends** Number> **void** mulMatrixOnNumber(V value) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

matrix[i][j] \*= value.doubleValue();

}

}

}

/\*\*

\* Перевірка чи є матриці узгоджені

\* **@param** matr - матриця типу числового двовимірного масиву

\* **@return** true - матриці узгоджені, false - матриці не узгоджені

\*/

**public** <V **extends** Number> **boolean** isCoordinatedMatrix(V[][] matr) {

**return** matrix[0].length == matr.length;

}

/\*\*

\* Перевірка чи є матриці узгоджені

\* **@param** matr - матриця типу Matrix

\* **@return** true - матриці узгоджені, false - матриці не узгоджені

\*/

**public** **boolean** isCoordinatedMatrix(Matrix<?> matr) {

**return** matrix[0].length == matr.getMatrix().length;

}

/\*\*

\* Множення матриць

\* **@param** matr - матриця типу Matrix, на яку множать

\* **@return** двовимірний числовий масив - результат множення

\*/

**public** **double**[][] mulMatrix(Matrix<?> matr) {

**double**[][] result = **new** **double**[matrix.length][matr.getMatrix()[0].length];

**for** (**int** i = 0; i < result.length; i++) {

**for** (**int** j = 0; j < result[i].length; j++) {

**for** (**int** k = 0; k < matrix[0].length; k++) {

result[i][j] += matrix[i][k] \* matr.getMatrix()[k][j];

}

}

}

**return** result;

}

/\*\*

\* Множення матриць

\* **@param** matr - матриця типу числового двовимірного масиву, на яку множать

\* **@return** двовимірний числовий масив - результат множення

\*/

**public** <V **extends** Number> **double**[][] mulMatrix(V[][] matr) {

**double**[][] result = **new** **double**[matrix.length][matr[0].length];

**for** (**int** i = 0; i < result.length; i++) {

**for** (**int** j = 0; j < result[i].length; j++) {

**for** (**int** k = 0; k < matrix[0].length; k++) {

result[i][j] += matrix[i][k] \* matr[k][j].doubleValue();

}

}

}

**return** result;

}

/\*\*

\* Перевіряє чи матриці переставні

\* **@param** matr - матриця типу Matrix

\* **@return** true - матриці переставні, false - матриці не переставні

\*/

**public** **boolean** isCommutingMatrix(Matrix<?> matr) {

**double**[][] result1 = **new** **double**[matrix.length][matr.getMatrix()[0].length];

**double**[][] result2 = **new** **double**[matr.getMatrix().length][matrix[0].length];

**for** (**int** i = 0; i < result1.length; i++) {

**for** (**int** j = 0; j < result1[i].length; j++) {

**for** (**int** k = 0; k < matrix[0].length; k++) {

result1[i][j] += matrix[i][k] \* matr.getMatrix()[k][j];

result2[i][j] += matr.getMatrix()[i][k] \* matrix[k][j];

**if** (result1 != result2)

**return** **false**;

}

}

}

**return** **true**;

}

/\*\*

\* Натуральний степінь матриці

\* **@param** n - степінь, до якого потрібно піднести матрицю

\*/

**public** **void** powerMatrix(**int** n) {

**double**[][] result = matrix.clone();

**for** (**int** l = 0; l < n; l++) {

**for** (**int** i = 0; i < result.length; i++) {

**for** (**int** j = 0; j < result.length; j++) {

**for** (**int** k = 0; k < matrix[0].length; k++) {

matrix[i][j] += result[i][k] \* result[k][j];

}

}

}

}

}

/\*\*

\* Транспонування матриці

\* **@return** Транспонована матриця

\*/

**public** **double**[][] transpositionMatrix() {

**double**[][] mat = **new** **double**[matrix[0].length][matrix.length];

**for** (**int** i = 0; i < mat.length; i++) {

**for** (**int** j = 0; j < mat[0].length; j++) {

mat[i][j] = matrix[j][i];

}

}

**return** mat;

}

/\*\*

\* Перевірка на симетричність

\* **@return** true - матриця симетрична, false - матриця не симетрична

\*/

**public** **boolean** isSummetric() {

**double**[][] matr = transpositionMatrix();

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix.length; j++) {

**if** (matrix[i][j] != matr[i][j])

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Перевірка на кососиметричність

\* **@return** true - матриця кососиметрична, false - матриця не кососиметрична

\*/

**public** **boolean** isSkewSummetric() {

**double**[][] matr = transpositionMatrix();

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix.length; j++) {

**if** (matrix[i][j] != -matr[i][j])

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Обчислення визначнику матриці

\* **@param** matr матриця, визначник якої потрібно визначити

\* **@return** Числове значення визначника

\*/

**public** **double** determinantMatrix(**double**[][] matr) {

**if** (matr.length == 1)

**return** matr[0][0];

**double**[][] dopMat = **new** **double**[matr.length - 1][matr.length - 1];

**double** s = 0;

**for** (**int** i = 0; i < matr.length; i++) {

**double**[][] matrixRz = partMatrix(matr);

System.*arraycopy*(matrixRz, 0, dopMat, 0, i);

**if** (i != matr.length - 1)

System.*arraycopy*(matrixRz, i + 1, dopMat, i, matr.length - i

- 1);

**if** (i % 2 == 0)

s += matr[i][0] \* determinantMatrix(dopMat);

**else**

s -= matr[i][0] \* determinantMatrix(dopMat);

}

**return** s;

}

/\*\*

\* Повертає частину матриці

\* **@param** matr - матриця, частину якої потрібно знайти

\* **@return** Частина матриці

\*/

**private** **double**[][] partMatrix(**double**[][] matr) {

**double**[][] dopMatrix = **new** **double**[matr.length][matr.length - 1];

**for** (**int** i = 0; i < matr.length; i++) {

System.*arraycopy*(matr[i], 1, dopMatrix[i], 0, dopMatrix[i].length);

}

**return** dopMatrix;

}

/\*\*

\* Обчислення мінору матриці

\* **@param** row - індекс рядка елементу, мінор якого потрібно визначити

\* **@param** column - індекс стовпця елементу, мінор якого потрібно визначити

\* **@return** Числове значення мінору елементу

\*/

**public** **double** supplementingMinorInMatrix(**int** row, **int** column) {

**double**[][] matr = **new** **double**[matrix.length - 1][matrix[0].length - 1];

**int** a = 0;

**int** b = 0;

**for** (**int** i = 0; i < matrix.length; i++) {

b = 0;

**if** (i != column) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (j != row) {

matr[a][b] = matrix[i][j];

b++;

}

}

a++;

}

}

**return** determinantMatrix(matr);

}

/\*\*

\* Обчислення алгебричного доповнення

\* **@param** row - індекс рядка елементу, алгебричне доповнення якого потрібно визначити

\* **@param** column - індекс стовпця елементу, алгебричне доповнення якого потрібно визначити

\* **@return** Числове значення алгебричного доповнення елементу

\*/

**public** **double** algebraicComplement(**int** row, **int** column) {

**if** ((row + column) % 2 == 0)

**return** supplementingMinorInMatrix(row, column);

**else**

**return** -supplementingMinorInMatrix(row, column);

}

/\*\*

\* Перевірка матриці на виродженість

\* **@return** true - матриця вироджена, false - не вироджена

\*/

**public** **boolean** isSingularMatrix() {

**return** determinantMatrix(matrix) == 0;

}

/\*\*

\* Пошук оберненої матриці

\* **@return** Обернена матриця

\*/

**public** **double**[][] turnedMatrix() {

Double[][] matr = **new** Double[matrix.length][matrix[0].length];

**for** (**int** i = 0; i < matr.length; i++) {

**for** (**int** j = 0; j < matr[i].length; j++) {

matr[j][i] = algebraicComplement(i, j);

}

}

Matrix<?> mat = **new** Matrix<>(matr);

mat.mulMatrixOnNumber(1 / determinantMatrix(getMatrix()));

**return** mat.getMatrix();

}

/\*\*

\* Обчислення рангу матриці

\* **@return** Ранг матриці

\*/

**public** **int** rankMatrix() {

**double**[][] matr = matrix.clone();

**int** column = 0;

**int** row = 0;

**int** k = -1;

**while** (column != matr[0].length) {

**for** (**int** i = row; i < matr.length; i++) {

**if** (matr[i][column] != 0) {

**for** (**int** j = column; j < matr[0].length; j++) {

**double** swap = matr[0][j];

matr[0][j] = matr[i][j];

matr[i][j] = swap;

}

**break**;

}

k = i;

}

**if** (k == matr.length - 1) {

column++;

**continue**;

}

**for** (**int** i = row + 1; i < matr.length; i++) {

**double** s = matr[i][column] / matr[row][column];

**for** (**int** j = column; j < matr[0].length; j++) {

matr[i][j] -= s \* matr[row][j];

}

}

column++;

row++;

}

**int** h = 0;

**for** (**int** i = matr.length - 1; i >= 0; i--) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < matr[i].length; j++) {

**if** (matr[i][j] != 0)

mark = **false**;

}

**if** (mark) {

h++;

**break**;

}

}

**return** matr.length - h;

}

/\*\*

\* Пошук розв'язків СЛАР

\* **@return** Матриця із розв'язками

\*/

**public** **double**[] SLAR() {

**double** M;

**double**[][] matr = matrix.clone();

**int** dimension = matrix.length;

**for** (**int** l = 0; l < dimension - 1; l++) {

**if** (matr[l][l] == 0) {

**for** (**int** i = l + 1; i < dimension; i++) {

**if** (matr[i][l] != 0) {

changeLines(l, i, matr);

**break**;

}

}

}

multiplyingRow(l, 1 / matr[l][l], matr);

**for** (**int** u = l + 1; u < dimension; u++) {

M = matr[u][l];

multiplyingRowSubtract(l, u, M, matr);

}

}

**for** (**int** i = dimension - 1; i >= 0; i--) {

multiplyingRow(i, 1 / matr[i][i], matr);

**for** (**int** j = 0; j < i; j++) {

multiplyingRowSubtract(i, j, matr[j][i], matr);

}

}

**double**[] answer = **new** **double**[matr.length];

**for** (**int** i = 0; i < answer.length; i++) {

answer[i] = matr[i][matr[i].length - 1];

}

**return** answer;

}

/\*\*

\* Зміна рядків місцями

\* **@param** a - індекс першого рядка

\* **@param** b - індекс другого рядка

\* **@param** matr - матриця, яка потребує зміни

\*/

**private** **void** changeLines(**int** a, **int** b, **double**[][] matr) {

**double** help[] = matr[a];

matr[a] = matr[b];

matr[b] = help;

}

/\*\*

\* Множення рядка на число

\* **@param** a - індекс рядка

\* **@param** x - число, на яке потрібно помножити

\* **@param** matr - матриця, яка потребує в операції

\*/

**private** **void** multiplyingRow(**int** a, **double** x, **double**[][] matr) {

**for** (**int** i = 0; i < matr.length + 1; i++) {

matr[a][i] = matr[a][i] \* x;

}

}

/\*\*

\* Віднімання від рядка іншого рядка помноженого на число

\* **@param** a - індекс рядка, який віднімають

\* **@param** b - індекс цільового рядка

\* **@param** x - число, на яке домножають

\* **@param** matr - матриця, яка потребує в дії

\*/

**private** **void** multiplyingRowSubtract(**int** a, **int** b, **double** x, **double**[][] matr) {

**for** (**int** i = 0; i < matr.length + 1; i++) {

matr[b][i] = matr[b][i] - matr[a][i] \* x;

}

}

}

**package** CourseWork;

**import** java.util.ArrayList;

/\*\*

\* Клас операцій над множинами

\* **@author** Sluva

\*

\*/

**public** **class** Set {

/\*\*

\* Операція об'єднання

\* **@param** vector1 - перша множина

\* **@param** vector2 - друга множина

\* **@return** Множина об'єднання вхідних множин

\*/

**public** <T **extends** Number, V **extends** Number> Double[] association(

T[] vector1, V[] vector2) {

**double**[] v1 = **new** **double**[vector1.length];

**double**[] v2 = **new** **double**[vector2.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = vector1[i].doubleValue();

}

**for** (**int** i = 0; i < v2.length; i++) {

v2[i] = vector2[i].doubleValue();

}

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v1.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < mass.size(); j++) {

**if** (v1[i] == mass.get(j)) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v1[i]);

}

**for** (**int** i = 0; i < v2.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < mass.size(); j++) {

**if** (v2[i] == mass.get(j)) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v2[i]);

}

**return** mass.toArray(**new** Double[mass.size()]);

}

/\*\*

\* Операція перетину

\* **@param** vector1 - перша множина

\* **@param** vector2 - друга множина

\* **@return** Множина перетину вхідних множин

\*/

**public** <T **extends** Number, V **extends** Number> Double[] section(T[] vector1,

V[] vector2) {

**double**[] v1 = **new** **double**[vector1.length];

**double**[] v2 = **new** **double**[vector2.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = vector1[i].doubleValue();

}

**for** (**int** i = 0; i < v2.length; i++) {

v2[i] = vector2[i].doubleValue();

}

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v1.length; i++) {

**for** (**int** j = 0; j < v2.length; j++) {

**if** (v1[i] == v2[j]) {

mass.add(v1[i]);

**break**;

}

}

}

**return** mass.toArray(**new** Double[mass.size()]);

}

/\*\*

\* Операція різниці

\* **@param** vector1 - перша множина

\* **@param** vector2 - друга множина

\* **@return** Множина різниці вхідних множин

\*/

**public** <T **extends** Number, V **extends** Number> Double[] difference(

T[] vector1, V[] vector2) {

**double**[] v1 = **new** **double**[vector1.length];

**double**[] v2 = **new** **double**[vector2.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = vector1[i].doubleValue();

}

**for** (**int** i = 0; i < v2.length; i++) {

v2[i] = vector2[i].doubleValue();

}

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v1.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < v2.length; j++) {

**if** (v1[i] == v2[j]) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v1[i]);

}

**return** mass.toArray(**new** Double[mass.size()]);

}

/\*\*

\* Операція симетричної різниці

\* **@param** vector1 - перша множина

\* **@param** vector2 - друга множина

\* **@return** Множина симетричної різниці вхідних множин

\*/

**public** <T **extends** Number, V **extends** Number> Double[] symmetricalDifference(

T[] vector1, V[] vector2) {

**double**[] v1 = **new** **double**[vector1.length];

**double**[] v2 = **new** **double**[vector2.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = vector1[i].doubleValue();

}

**for** (**int** i = 0; i < v2.length; i++) {

v2[i] = vector2[i].doubleValue();

}

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v1.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < v2.length; j++) {

**if** (v1[i] == v2[j]) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v1[i]);

}

**for** (**int** i = 0; i < v2.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < v1.length; j++) {

**if** (v2[i] == v1[j]) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v2[i]);

}

**return** mass.toArray(**new** Double[mass.size()]);

}

/\*\*

\* Операція доповнення

\* **@param** vector1 - множина

\* **@param** vector2 - універсальна множина

\* **@return** Множина значень доповнення до вхідної множини

\*/

**public** <T **extends** Number, V **extends** Number> Double[] explement(T[] vector1,

V[] vector2) {

**double**[] v1 = **new** **double**[vector1.length];

**double**[] v2 = **new** **double**[vector2.length];

**for** (**int** i = 0; i < v1.length; i++) {

v1[i] = vector1[i].doubleValue();

}

**for** (**int** i = 0; i < v2.length; i++) {

v2[i] = vector2[i].doubleValue();

}

ArrayList<Double> mass = **new** ArrayList<>();

**for** (**int** i = 0; i < v2.length; i++) {

**boolean** mark = **true**;

**for** (**int** j = 0; j < v1.length; j++) {

**if** (v2[i] == v1[j]) {

mark = **false**;

**break**;

}

}

**if** (mark)

mass.add(v2[i]);

}

**return** mass.toArray(**new** Double[mass.size()]);

}

}

**package** CourseWork;

/\*\*

\* Клас обробки даних типу "Вектор"

\* **@author** Sluva

\*

\* **@param** <T> - тип даних у векторі

\*/

**public** **class** Vector<T **extends** Number> {

/\*\*

\* Поле зберігання значення вектору

\*/

**private** **final** **double**[] vector;

/\*\*

\* Конструктор класу

\* **@param** vector - одновимірний масив числових даних

\*/

**public** Vector(T vector[]) {

**this**.vector = **new** **double**[vector.length];

**for** (**int** i = 0; i < vector.length; i++) {

**this**.vector[i] = vector[i].doubleValue();

}

}

/\*\*

\* Повертає значення вектору

\* **@return** Поточне значення вектору

\*/

**public** **double**[] getVector() {

**return** vector;

}

/\*\*

\* Перевірка чи є вектор нульовим

\* **@return** true - вектор нульвоий, false - ні

\*/

**public** **boolean** isZeroVector() {

**for** (**int** i = 0; i < vector.length; i++) {

**if** (vector[i] != 0)

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Додавання до вектору класу іншого вектору

\* **@param** vect - вектор типу Vector

\*/

**public** **void** addVectors(Vector<?> vect) {

**for** (**int** i = 0; i < vector.length; i++) {

vector[i] += vect.getVector()[i];

}

}

/\*\*

\* Додавання до вектору класу іншого вектору

\* **@param** vect - вектор типу числового масиву

\*/

**public** <V **extends** Number> **void** addVectors(V[] vect) {

**for** (**int** i = 0; i < vector.length; i++) {

vector[i] += vect[i].doubleValue();

}

}

/\*\*

\* Віднімання від вектору класу іншого вектору

\* **@param** vect - вектор типу Vector

\*/

**public** **void** subVectors(Vector<?> vect) {

**for** (**int** i = 0; i < vector.length; i++) {

vector[i] -= vect.getVector()[i];

}

}

/\*\*

\* Віднімання від вектору класу іншого вектору

\* **@param** vect - вектор типу числового масиву

\*/

**public** <V **extends** Number> **void** subVectors(V[] vect) {

**for** (**int** i = 0; i < vector.length; i++) {

vector[i] -= vect[i].doubleValue();

}

}

/\*\*

\* Додавання до елементу вектора іншого елементу

\* **@param** x1 - індекс цільового елементу

\* **@param** x2 - індекс елемету, який додають

\*/

**public** **void** addElements(**int** x1, **int** x2) {

vector[x1] += vector[x2];

}

/\*\*

\* Віднімання ід елементу вектора іншого елементу

\* **@param** x1 - індекс цільового елементу

\* **@param** x2 - індекс елемету, який віднімають

\*/

**public** **void** subElements(**int** x1, **int** x2) {

vector[x1] -= vector[x2];

}

/\*\*

\* Додавання до елементу вектора числа

\* **@param** x - індекс цільового елементу

\* **@param** value - число, яке додають

\*/

**public** <V **extends** Number> **void** addElement(**int** x, V value) {

vector[x] += value.doubleValue();

}

/\*\*

\* Віднімання від елементу вектора числа

\* **@param** x - індекс цільового елементу

\* **@param** value - число, яке віднімають

\*/

**public** <V **extends** Number> **void** subElement(**int** x, V value) {

vector[x] -= value.doubleValue();

}

/\*\*

\* Множення вектора на число

\* **@param** value - число, на яке множать

\*/

**public** <V **extends** Number> **void** mulVectorOnNumber(V value) {

**for** (**int** i = 0; i < vector.length; i++) {

vector[i] \*= value.doubleValue();

}

}

/\*\*

\* Перевірка вектора на рівність іншому вектору

\* **@param** vect - вектор типу Vector

\* **@return** true - вектори рівні, false - не рівні

\*/

**public** **boolean** isEquality(Vector<?> vect) {

**for** (**int** i = 0; i < vector.length; i++) {

**if** (vect.getVector()[i] != vector[i])

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Перевірка вектора на рівність іншому вектору

\* **@param** vect - вектор типу масиву числових данних

\* **@return** true - вектори рівні, false - не рівні

\*/

**public** <V **extends** Number> **boolean** isEquality(T[] vect) {

**for** (**int** i = 0; i < vector.length; i++) {

**if** (vect[i].doubleValue() != vector[i])

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Множення вектора на інший вектор

\* **@param** vect - вектор-множник типу Vector

\* **@return** число - результат множення

\*/

**public** **double** mulVectors(Vector<?> vect) {

**double** result = 0;

**for** (**int** i = 0; i < vector.length; i++) {

result += vector[i] \* vect.getVector()[i];

}

**return** result;

}

/\*\*

\* Множення вектора на інший вектор

\* **@param** vect - вектор-множник типу масиву числових данних

\* **@return** число - результат множення

\*/

**public** <V **extends** Number> **double** mulVectors(V[] vect) {

**double** result = 0;

**for** (**int** i = 0; i < vector.length; i++) {

result += vector[i] \* vect[i].doubleValue();

}

**return** result;

}

}

**Додаток Б. (детальна інструкція з експлуатації для програміста клієнта)**

## Class Matrix<T extends java.lang.Number>

* java.lang.Object
  + Matrix<T>
* Type Parameters:

T - - тип данних у матриці

public class Matrix<T extends java.lang.Number>

extends java.lang.Object

Клас обробки данних типу "Матриця"

* + ***Constructor Summary***

|  |
| --- |
|  |
| **Constructor and Description** |
| Matrix(**T**[][] matrix)  Конструктор класу |

* + ***Method Summary***

|  |  |
| --- | --- |
|  | |
| **Modifier and Type** | **Method and Description** |
| void | addColumns(int column1, int column2)  Додає до стовпця інший стовпець |
| <V extends java.lang.Number> void | addColumns(int column, V[] vector)  Додає до стовпця одновимірний масив |
| void | addColumns(int column, **Vector**<?> vect)  Додає до стовпця вектор |
| <V extends java.lang.Number> void | addElement(int row, int column, V value)  Додає до елемента матриці числове значення |
| void | addElements(int row1, int column1, int row2, int column2)  Додає до елемента матриці інший елемент |
| void | addMatrix(**Matrix**<?> matr)  Додає до матриці іншу матрицю |
| <V extends java.lang.Number> void | addMatrix(V[][] matr)  Додає до матриці іншу матрицю |
| void | addRows(int row1, int row2)  Додає до рядка інший рядок |
| <V extends java.lang.Number> void | addRows(int row, V[] vector)  Додає до рядка одновимірний числовий масив |
| void | addRows(int row, **Vector**<?> vect)  Додає до рядка вектор |
| double | algebraicComplement(int row, int column)  Обчислення алгебричного доповнення |
| double | determinantMatrix(double[][] matr)  Обчислення визначнику матриці |
| double[] | getColumn(int column)  Повертає стовпець матриці |
| double[][] | getMatrix()  Поветрає матрицю |
| double[] | getRow(int row)  Повертає рядок матриці |
| double | getValueElement(int x, int y)  Повертає елемент матриці |
| boolean | isCommutingMatrix(**Matrix**<?> matr)  Перевіряє чи матриці переставні |
| boolean | isCoordinatedMatrix(**Matrix**<?> matr)  Перевірка чи є матриці узгоджені |
| <V extends java.lang.Number> boolean | isCoordinatedMatrix(V[][] matr)  Перевірка чи є матриці узгоджені |
| boolean | isDiagonalMatrix()  Перевіряє чи матриця діагональна |
| boolean | isEquality(**Matrix**<?> matr)  Перевіряє матриці на рівність |
| <V extends java.lang.Number> boolean | isEquality(V[][] matr)  Перевіряє матриці на рівність |
| boolean | isEqualityColumns(int column1, int column2)  Перевіряє рівність стовпців |
| boolean | isEqualityRows(int row1, int row2)  Перевіряє рівність рядків |
| boolean | isLowerTriangularMatrix()  Перевіряє чи матриця нижня трикутна |
| boolean | isSingularMatrix()  Перевірка матриці на виродженість |
| boolean | isSkewSummetric()  Перевірка на кососиметричність |
| boolean | isSquareMatrix()  Перевіряє чи матриця квадратна |
| boolean | isSummetric()  Перевірка на симетричність |
| boolean | isUnitaryMatrix()  Перевіряє чи матриця одинична |
| boolean | isUpperTriangularMatrix()  Перевіряє чи матриця верхня трикутна |
| boolean | isZeroMatrix()  Перевіряє чи матриця нульова |
| <V extends java.lang.Number> void | mulColumnOnNumber(int column, V value)  Множення стовпця матриці на число |
| double[][] | mulMatrix(**Matrix**<?> matr)  Множення матриць |
| <V extends java.lang.Number> double[][] | mulMatrix(V[][] matr)  Множення матриць |
| <V extends java.lang.Number> void | mulMatrixOnNumber(V value)  Множення матриці на число |
| <V extends java.lang.Number> void | mulRowOnNumber(int row, V value)  Множення рядка матриці на число |
| void | powerMatrix(int n)  Натуральний степінь матриці |
| int | rankMatrix()  Обчислення рангу матриці |
| double[] | SLAR()  Пошук розв'язків СЛАР |
| void | subColumns(int column1, int column2)  Віднімає від стовпця інший стовпець |
| <V extends java.lang.Number> void | subColumns(int column, V[] vector)  Віднімає від стовпця одновимірний масив |
| void | subColumns(int column, **Vector**<?> vect)  Віднімає від стовпця вектор |
| void | subElements(int row1, int column1, int row2, int column2)  Віднімає від елемента матриці інший елемент |
| <V extends java.lang.Number> void | subElements(int row, int column, V value)  Віднімає від елемента матриці числове значення |
| void | subMatrix(**Matrix**<?> matr)  Віднімає від матриці іншу матрицю |
| <V extends java.lang.Number> void | subMatrix(V[][] matr)  Віднімає від матриці іншу матрицю |
| void | subRows(int row1, int row2)  Віднімає від рядка інший рядок |
| <V extends java.lang.Number> void | subRows(int row, V[] vector)  Віднімає від рядка одновимірний числовий масив |
| void | subRows(int row, **Vector**<?> vect)  Віднімає від рядка вектор |
| double | supplementingMinorInMatrix(int row, int column)  Обчислення мінору матриці |
| double[][] | transpositionMatrix()  Транспонування матриці |
| double[][] | turnedMatrix()  Пошук оберненої матриці |

* + - **Methods inherited from class java.lang.Object**

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

* + ***Constructor Detail***
    - **Matrix**

public Matrix(T[][] matrix)

Конструктор класу

Parameters:

matrix - - двовимірний масив числових данних

* + ***Method Detail***
    - **getValueElement**
    - public double getValueElement(int x,

int y)

Повертає елемент матриці

Parameters:

x - - номер рядка

y - - номер стовпця

Returns:

Елемент матриці із координатами масиву x,y

* + - **getColumn**

public double[] getColumn(int column)

Повертає стовпець матриці

Parameters:

column - - індекс стовпця

Returns:

Стовпець матриці із індексом column

* + - **getRow**

public double[] getRow(int row)

Повертає рядок матриці

Parameters:

row - - індекс рядка

Returns:

Рядок матриці із індексом row

* + - **getMatrix**

public double[][] getMatrix()

Поветрає матрицю

Returns:

Матриця, яка підлягає обробці методами

* + - **isZeroMatrix**

public boolean isZeroMatrix()

Перевіряє чи матриця нульова

Returns:

true - матриця нульова, false - не нульова

* + - **isSquareMatrix**

public boolean isSquareMatrix()

Перевіряє чи матриця квадратна

Returns:

true - матриця квадратна, false - не квадратна

* + - **isLowerTriangularMatrix**

public boolean isLowerTriangularMatrix()

Перевіряє чи матриця нижня трикутна

Returns:

true - матриця нижня трикутна, false - не нижня трикутна

* + - **isUpperTriangularMatrix**

public boolean isUpperTriangularMatrix()

Перевіряє чи матриця верхня трикутна

Returns:

true - матриця верхня трикутна, false - не верхня трикутна

* + - **isDiagonalMatrix**

public boolean isDiagonalMatrix()

Перевіряє чи матриця діагональна

Returns:

true - матриця діагональна, false - не діагональна

* + - **isUnitaryMatrix**

public boolean isUnitaryMatrix()

Перевіряє чи матриця одинична

Returns:

true - матриця одинична, false - не одинична

* + - **isEqualityColumns**
    - public boolean isEqualityColumns(int column1,

int column2)

Перевіряє рівність стовпців

Parameters:

column1 - - індекс першого стовпця

column2 - - індекс другого стовпця

Returns:

true - стовпці рівні, false - не рівні

* + - **isEqualityRows**
    - public boolean isEqualityRows(int row1,

int row2)

Перевіряє рівність рядків

Parameters:

row1 - - індекс першого рядка

row2 - - індекс другого рядка

Returns:

true - рядки рівні, false - не рівні

* + - **addColumns**
    - public void addColumns(int column1,

int column2)

Додає до стовпця інший стовпець

Parameters:

column1 - - індекс цільового стовпця

column2 - - індекс стовпця, що додають

* + - **addColumns**
    - public void addColumns(int column,

Vector<?> vect)

Додає до стовпця вектор

Parameters:

column - - індекс цільового стовпця

vect - - одновимірний масив типу Vector

* + - **addColumns**
    - public <V extends java.lang.Number> void addColumns(int column,

V[] vector)

Додає до стовпця одновимірний масив

Parameters:

column - - індекс цільового стовпця

vector - - одновимірний числовий масив

* + - **addRows**
    - public void addRows(int row1,

int row2)

Додає до рядка інший рядок

Parameters:

row1 - - індекс цільового рядка

row2 - - індекс рядка, що додають

* + - **addRows**
    - public void addRows(int row,

Vector<?> vect)

Додає до рядка вектор

Parameters:

row - - індекс цільового рядка

vect - - одновимірний масив типу Vector

* + - **addRows**
    - public <V extends java.lang.Number> void addRows(int row,

V[] vector)

Додає до рядка одновимірний числовий масив

Parameters:

row - - індекс цільового рядка

vector - - одновимірний числовий масив

* + - **subColumns**
    - public void subColumns(int column1,

int column2)

Віднімає від стовпця інший стовпець

Parameters:

column1 - - індекс цільового стовпця

column2 - - індекс стовпця, що віднімають

* + - **subColumns**
    - public void subColumns(int column,

Vector<?> vect)

Віднімає від стовпця вектор

Parameters:

column - - індекс цільового стовпця

vect - - одновимірний масив типу Vector

* + - **subColumns**
    - public <V extends java.lang.Number> void subColumns(int column,

V[] vector)

Віднімає від стовпця одновимірний масив

Parameters:

column - - індекс цільового стовпця

vector - - одновимірний числовий масив

* + - **subRows**
    - public void subRows(int row1,

int row2)

Віднімає від рядка інший рядок

Parameters:

row1 - - індекс цільового рядка

row2 - - індекс рядка, що віднімають

* + - **subRows**
    - public void subRows(int row,

Vector<?> vect)

Віднімає від рядка вектор

Parameters:

row - - індекс цільового рядка

vect - - одновимірний масив типу Vector

* + - **subRows**
    - public <V extends java.lang.Number> void subRows(int row,

V[] vector)

Віднімає від рядка одновимірний числовий масив

Parameters:

row - - індекс цільового рядка

vector - - одновимірний числовий масив

* + - **addElements**
    - public void addElements(int row1,
    - int column1,
    - int row2,

int column2)

Додає до елемента матриці інший елемент

Parameters:

row1 - - індекс рядка цільового елементу

column1 - - індекс стовпця цільового елементу

row2 - - індекс рядка елементу, який додають

column2 - - індекс стовпця елементу, який додають

* + - **addElement**
    - public <V extends java.lang.Number> void addElement(int row,
    - int column,

V value)

Додає до елемента матриці числове значення

Parameters:

row - - індекс рядка цільового елементу

column - - індекс стовпця цільового елементу

value - - число, яке додають

* + - **subElements**
    - public void subElements(int row1,
    - int column1,
    - int row2,

int column2)

Віднімає від елемента матриці інший елемент

Parameters:

row1 - - індекс рядка цільового елементу

column1 - - індекс стовпця цільового елементу

row2 - - індекс рядка елементу, який віднімають

column2 - - індекс стовпця елементу, який віднімають

* + - **subElements**
    - public <V extends java.lang.Number> void subElements(int row,
    - int column,

V value)

Віднімає від елемента матриці числове значення

Parameters:

row - - індекс рядка цільового елементу

column - - індекс стовпця цільового елементу

value - - число, яке віднімають

* + - **mulColumnOnNumber**
    - public <V extends java.lang.Number> void mulColumnOnNumber(int column,

V value)

Множення стовпця матриці на число

Parameters:

column - - індекс стовпця

value - - число, на яке множать

* + - **mulRowOnNumber**
    - public <V extends java.lang.Number> void mulRowOnNumber(int row,

V value)

Множення рядка матриці на число

Parameters:

row - - індекс рядка

value - - число, на яке множать

* + - **isEquality**

public boolean isEquality(Matrix<?> matr)

Перевіряє матриці на рівність

Parameters:

matr - - матриця типу Matrix

Returns:

true - матриці рівні, false - не рівні

* + - **isEquality**

public <V extends java.lang.Number> boolean isEquality(V[][] matr)

Перевіряє матриці на рівність

Parameters:

matr - - матриця типу двовимірного числового масиву

Returns:

true - матриці рівні, false - не рівні

* + - **addMatrix**

public void addMatrix(Matrix<?> matr)

Додає до матриці іншу матрицю

Parameters:

matr - - матриця типу Matrix, яку додають

* + - **addMatrix**

public <V extends java.lang.Number> void addMatrix(V[][] matr)

Додає до матриці іншу матрицю

Parameters:

matr - - матриця типу числового двовимірного масиву, яку додають

* + - **subMatrix**

public void subMatrix(Matrix<?> matr)

Віднімає від матриці іншу матрицю

Parameters:

matr - - матриця типу Matrix, яку віднімають

* + - **subMatrix**

public <V extends java.lang.Number> void subMatrix(V[][] matr)

Віднімає від матриці іншу матрицю

Parameters:

matr - - матриця типу числового двовимірного масиву, яку віднімають

* + - **mulMatrixOnNumber**

public <V extends java.lang.Number> void mulMatrixOnNumber(V value)

Множення матриці на число

Parameters:

value - - число, на яке множать

* + - **isCoordinatedMatrix**

public <V extends java.lang.Number> boolean isCoordinatedMatrix(V[][] matr)

Перевірка чи є матриці узгоджені

Parameters:

matr - - матриця типу числового двовимірного масиву

Returns:

true - матриці узгоджені, false - матриці не узгоджені

* + - **isCoordinatedMatrix**

public boolean isCoordinatedMatrix(Matrix<?> matr)

Перевірка чи є матриці узгоджені

Parameters:

matr - - матриця типу Matrix

Returns:

true - матриці узгоджені, false - матриці не узгоджені

* + - **mulMatrix**

public double[][] mulMatrix(Matrix<?> matr)

Множення матриць

Parameters:

matr - - матриця типу Matrix, на яку множать

Returns:

двовимірний числовий масив - результат множення

* + - **mulMatrix**

public <V extends java.lang.Number> double[][] mulMatrix(V[][] matr)

Множення матриць

Parameters:

matr - - матриця типу числового двовимірного масиву, на яку множать

Returns:

двовимірний числовий масив - результат множення

* + - **isCommutingMatrix**

public boolean isCommutingMatrix(Matrix<?> matr)

Перевіряє чи матриці переставні

Parameters:

matr - - матриця типу Matrix

Returns:

true - матриці переставні, false - матриці не переставні

* + - **powerMatrix**

public void powerMatrix(int n)

Натуральний степінь матриці

Parameters:

n - - степінь, до якого потрібно піднести матрицю

* + - **transpositionMatrix**

public double[][] transpositionMatrix()

Транспонування матриці

Returns:

Транспонована матриця

* + - **isSummetric**

public boolean isSummetric()

Перевірка на симетричність

Returns:

true - матриця симетрична, false - матриця не симетрична

* + - **isSkewSummetric**

public boolean isSkewSummetric()

Перевірка на кососиметричність

Returns:

true - матриця кососиметрична, false - матриця не кососиметрична

* + - **determinantMatrix**

public double determinantMatrix(double[][] matr)

Обчислення визначнику матриці

Parameters:

matr - матриця, визначник якої потрібно визначити

Returns:

Числове значення визначника

* + - **supplementingMinorInMatrix**
    - public double supplementingMinorInMatrix(int row,

int column)

Обчислення мінору матриці

Parameters:

row - - індекс рядка елементу, мінор якого потрібно визначити

column - - індекс стовпця елементу, мінор якого потрібно визначити

Returns:

Числове значення мінору елементу

* + - **algebraicComplement**
    - public double algebraicComplement(int row,

int column)

Обчислення алгебричного доповнення

Parameters:

row - - індекс рядка елементу, алгебричне доповнення якого потрібно визначити

column - - індекс стовпця елементу, алгебричне доповнення якого потрібно визначити

Returns:

Числове значення алгебричного доповнення елементу

* + - **isSingularMatrix**

public boolean isSingularMatrix()

Перевірка матриці на виродженість

Returns:

true - матриця вироджена, false - не вироджена

* + - **turnedMatrix**

public double[][] turnedMatrix()

Пошук оберненої матриці

Returns:

Обернена матриця

* + - **rankMatrix**

public int rankMatrix()

Обчислення рангу матриці

Returns:

Ранг матриці

* + - **SLAR**

public double[] SLAR()

Пошук розв'язків СЛАР

Returns:

Матриця із розв'язками

## Class Vector<T extends java.lang.Number>

* java.lang.Object
  + Vector<T>
* Type Parameters:

T - - тип даних у векторі

public class Vector<T extends java.lang.Number>

extends java.lang.Object

Клас обробки даних типу "Вектор"

* + ***Constructor Summary***

|  |
| --- |
|  |
| **Constructor and Description** |
| Vector(**T**[] vector)  Конструктор класу |

* + ***Method Summary***

|  |  |
| --- | --- |
|  | |
| **Modifier and Type** | **Method and Description** |
| <V extends java.lang.Number> void | addElement(int x, V value)  Додавання до елементу вектора числа |
| void | addElements(int x1, int x2)  Додавання до елементу вектора іншого елементу |
| <V extends java.lang.Number> void | addVectors(V[] vect)  Додавання до вектору класу іншого вектору |
| void | addVectors(**Vector**<?> vect)  Додавання до вектору класу іншого вектору |
| double[] | getVector()  Повертає значення вектору |
| <V extends java.lang.Number> boolean | isEquality(**T**[] vect)  Перевірка вектора на рівність іншому вектору |
| boolean | isEquality(**Vector**<?> vect)  Перевірка вектора на рівність іншому вектору |
| boolean | isZeroVector()  Перевірка чи є вектор нульовим |
| <V extends java.lang.Number> void | mulVectorOnNumber(V value)  Множення вектора на число |
| <V extends java.lang.Number> double | mulVectors(V[] vect)  Множення вектора на інший вектор |
| double | mulVectors(**Vector**<?> vect)  Множення вектора на інший вектор |
| <V extends java.lang.Number> void | subElement(int x, V value)  Віднімання від елементу вектора числа |
| void | subElements(int x1, int x2)  Віднімання ід елементу вектора іншого елементу |
| <V extends java.lang.Number> void | subVectors(V[] vect)  Віднімання від вектору класу іншого вектору |
| void | subVectors(**Vector**<?> vect)  Віднімання від вектору класу іншого вектору |

* + - **Methods inherited from class java.lang.Object**

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

* + ***Constructor Detail***
    - **Vector**

public Vector(T[] vector)

Конструктор класу

Parameters:

vector - - одновимірний масив числових даних

* + ***Method Detail***
    - **getVector**

public double[] getVector()

Повертає значення вектору

Returns:

Поточне значення вектору

* + - **isZeroVector**

public boolean isZeroVector()

Перевірка чи є вектор нульовим

Returns:

true - вектор нульвоий, false - ні

* + - **addVectors**

public void addVectors(Vector<?> vect)

Додавання до вектору класу іншого вектору

Parameters:

vect - - вектор типу Vector

* + - **addVectors**

public <V extends java.lang.Number> void addVectors(V[] vect)

Додавання до вектору класу іншого вектору

Parameters:

vect - - вектор типу числового масиву

* + - **subVectors**

public void subVectors(Vector<?> vect)

Віднімання від вектору класу іншого вектору

Parameters:

vect - - вектор типу Vector

* + - **subVectors**

public <V extends java.lang.Number> void subVectors(V[] vect)

Віднімання від вектору класу іншого вектору

Parameters:

vect - - вектор типу числового масиву

* + - **addElements**
    - public void addElements(int x1,

int x2)

Додавання до елементу вектора іншого елементу

Parameters:

x1 - - індекс цільового елементу

x2 - - індекс елемету, який додають

* + - **subElements**
    - public void subElements(int x1,

int x2)

Віднімання ід елементу вектора іншого елементу

Parameters:

x1 - - індекс цільового елементу

x2 - - індекс елемету, який віднімають

* + - **addElement**
    - public <V extends java.lang.Number> void addElement(int x,

V value)

Додавання до елементу вектора числа

Parameters:

x - - індекс цільового елементу

value - - число, яке додають

* + - **subElement**
    - public <V extends java.lang.Number> void subElement(int x,

V value)

Віднімання від елементу вектора числа

Parameters:

x - - індекс цільового елементу

value - - число, яке віднімають

* + - **mulVectorOnNumber**

public <V extends java.lang.Number> void mulVectorOnNumber(V value)

Множення вектора на число

Parameters:

value - - число, на яке множать

* + - **isEquality**

public boolean isEquality(Vector<?> vect)

Перевірка вектора на рівність іншому вектору

Parameters:

vect - - вектор типу Vector

Returns:

true - вектори рівні, false - не рівні

* + - **isEquality**

public <V extends java.lang.Number> boolean isEquality(T[] vect)

Перевірка вектора на рівність іншому вектору

Parameters:

vect - - вектор типу масиву числових данних

Returns:

true - вектори рівні, false - не рівні

* + - **mulVectors**

public double mulVectors(Vector<?> vect)

Множення вектора на інший вектор

Parameters:

vect - - вектор-множник типу Vector

Returns:

число - результат множення

* + - **mulVectors**

public <V extends java.lang.Number> double mulVectors(V[] vect)

Множення вектора на інший вектор

Parameters:

vect - - вектор-множник типу масиву числових данних

Returns:

число - результат множення

## Class Set

* java.lang.Object
  + Set

public class Set

extends java.lang.Object

Клас операцій над множинами

* + ***Constructor Summary***

|  |
| --- |
|  |
| **Constructor and Description** |
| Set() |

* + ***Method Summary***

|  |  |
| --- | --- |
|  | |
| **Modifier and Type** | **Method and Description** |
| <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] | association(T[] vector1, V[] vector2)  Операція об'єднання |
| <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] | difference(T[] vector1, V[] vector2)  Операція різниці |
| <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] | explement(T[] vector1, V[] vector2)  Операція доповнення |
| <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] | section(T[] vector1, V[] vector2)  Операція перетину |
| <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] | symmetricalDifference(T[] vector1, V[] vector2)  Операція симетричної різниці |

* + - **Methods inherited from class java.lang.Object**

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

* + ***Constructor Detail***
    - **Set**

public Set()

* + ***Method Detail***
    - **association**
    - public <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] association(T[] vector1,

V[] vector2)

Операція об'єднання

Parameters:

vector1 - - перша множина

vector2 - - друга множина

Returns:

Множина об'єднання вхідних множин

* + - **section**
    - public <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] section(T[] vector1,

V[] vector2)

Операція перетину

Parameters:

vector1 - - перша множина

vector2 - - друга множина

Returns:

Множина перетину вхідних множин

* + - **difference**
    - public <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] difference(T[] vector1,

V[] vector2)

Операція різниці

Parameters:

vector1 - - перша множина

vector2 - - друга множина

Returns:

Множина різниці вхідних множин

* + - **symmetricalDifference**
    - public <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] symmetricalDifference(T[] vector1,

V[] vector2)

Операція симетричної різниці

Parameters:

vector1 - - перша множина

vector2 - - друга множина

Returns:

Множина симетричної різниці вхідних множин

* + - **explement**
    - public <T extends java.lang.Number,V extends java.lang.Number> java.lang.Double[] explement(T[] vector1,

V[] vector2)

Операція доповнення

Parameters:

vector1 - - множина

vector2 - - універсальна множина

Returns:

Множина значень доповнення до вхідної множини

## Class Combinations

* java.lang.Object
  + Combinations

public class Combinations

extends java.lang.Object

Клас роботи із комбінаторикою

* + ***Constructor Summary***

|  |
| --- |
|  |
| **Constructor and Description** |
| Combinations() |

* + ***Method Summary***

|  |  |
| --- | --- |
|  | |
| **Modifier and Type** | **Method and Description** |
| java.lang.Integer[] | binaryVector(int n)  Генератор двійкових векторів довжини N |
| java.lang.Object[] | combination(java.lang.Object[] mass, int n)  Пошук всіх сполучень із заданого вектору і заданої довжини |
| int | combinationAmount(int n, int m)  Обраховує кількість сполучень із N елементів по M |
| int | distributionAmount(int n, int m)  Обраховує кількість розміщень із N елементів по M елементів |
| int | distributionWithRepetitionAmount(int n, int m)  Обраховує кількість розміщень з повтореннями із N елементів |
| <V> java.lang.Object[] | permutation(V[] vector)  Знаходить всі перестановки із вектору |
| int | permutationAmount(int n)  Обраховує кількість перестановок із N елементів |

* + - **Methods inherited from class java.lang.Object**

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

* + ***Constructor Detail***
    - **Combinations**

public Combinations()

* + ***Method Detail***
    - **permutationAmount**

public int permutationAmount(int n)

Обраховує кількість перестановок із N елементів

Parameters:

n - - кількість елементів у комбінації

Returns:

Кількість перестановок

* + - **distributionAmount**
    - public int distributionAmount(int n,

int m)

Обраховує кількість розміщень із N елементів по M елементів

Parameters:

n - - кількість доступних унікальних елементів

m - - кількість елементів у комбінації

Returns:

Кількість розміщень

* + - **combinationAmount**
    - public int combinationAmount(int n,

int m)

Обраховує кількість сполучень із N елементів по M

Parameters:

n - - кількість доступних унікальних елементів

m - - кількість елементів у комбінації

Returns:

кількість сполучень

* + - **distributionWithRepetitionAmount**
    - public int distributionWithRepetitionAmount(int n,

int m)

Обраховує кількість розміщень з повтореннями із N елементів

Parameters:

n - - кількість доступних унікальних елементів

m - - кількість елементів в комбінації

Returns:

Кількість розміщень з повтореннями

* + - **permutation**

public <V> java.lang.Object[] permutation(V[] vector)

Знаходить всі перестановки із вектору

Parameters:

vector - - початковий вектор

Returns:

Масив об'єктів із масивами перестановок

* + - **binaryVector**

public java.lang.Integer[] binaryVector(int n)

Генератор двійкових векторів довжини N

Parameters:

n - - довжина вектору

Returns:

Масив двійкових векторів

* + - **combination**
    - public java.lang.Object[] combination(java.lang.Object[] mass,

int n)

Пошук всіх сполучень із заданого вектору і заданої довжини

Parameters:

mass - - початковий вектор

n - - довжина комбінації

Returns:

Масив векторів сполучень