УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Кафедра інформаційних технологій та програмної інженерії

КУРСОВА РОБОТА

з Об’єктно-орієнтованого програмування

на тему: Класові типи для роботи з чисельними матрицями та векторами

Студента II курсу _____ групи

спеціальності 121 «Інженерія програмного забезпечення»

Волк А. В.

(прізвище та ініціали)

Керівник _____к.т.н.,

Пашкевич О.П.

Національна шкала

_____

Кількість балів _____ Оцінка: ECTS

_____

м. Івано-Франківськ

2021

_____Університет Короля Данила_____

назва вищого навчального закладу

Кафедра __інформаційних технологій та програмної інженерії_____

Дисципліна _Об’єктоно-орієнтоване програмування_____

Спеціальність 121 «Інженерія програмного забезпечення»

_____

Курс_____II_____ Група _____ІПЗс-2019_____ Семестр _____4_____

ЗАВДАННЯ

на курсовий проект (роботу) студента

_____Волк Артем Віталійович_____

Прізвище, ім’я, по-батькові

1. Тема проекту (роботи) Класові типи для роботи з чисельними матрицями та векторами

   _____

   _____

   _____

2. Строк здачі студентом закінченого проекту (роботи)

_____

3. Вихідні дані до проекту (роботи)

_____

_____

4. Зміст роботи (перелік питань по розділах, які потрібно розробити)

_____

_____

5. Дата видачі завдання

_____

КАЛЕНДАРНИЙ ПЛАН

№ П/П	Назва етапів курсового проекту	Строк виконання	Примітки
1	Ознайомлення та пошук інформації	10.05.2021	
2	Розробка архітектури	11.05.2021	
3	Написання основного класу	15.05.2021	
4	Написання методів та функцій	17.05.2021	
5	Оформлення курсової роботи	21.05.2021	

ЗМІСТ

Вступ

Java (вимовляється *Джава*) — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005.

Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

Постановка задачі

Класові типи для роботи з чисельними матрицями та векторами
1. Створити класовий тип Matrix - двовимірний чисельний масив динамічного типу із змінними розмірами.
2. Створити класовий тип Vector - одновимірний чисельний масив динамічного типу із змінними розмірами.
3. Для конкретизації типів матриці та вектора використати шаблони.
4. Реалізувати компонентні методи:
   - транспонування матриці;
   - множення матриць;
   - складання та віднімання;
   - порівняння матриць;
   - множення матриці на вектор;
   - скалярний добуток векторів;
   - обчислення норми матриці;
   - пошук седлових точок матриці;
   - пошук мінімальних та максимальних елементів.

Функціональне призначення програми

Дана програма є демонстрацією можливостей мови програмування Java та її окремих бібліотек та функцій, для вирішення функціональних задач з алгебри, геометрії, тригонометрії, булевої математики, чисельних матриць та векторів. Программа може бути використана для швидкого і легкого вирішення таких задач як: транспонування матриці, множення матриць, складання та віднімання, порівняння матриць, множення матриці на вектор, скалярний добуток векторів, обчислення норми матриці пошук седлових точок матриці та пошук мінімальних та максимальних елементів.

Поняття про мову програмування Java

Мова програмування Java зародилася в 1991 р. в лабораторіях компанії Sun Microsystems. Розробку проєкту започаткував Джеймс Ґослінґ, сам проєкт мав назву «Green» (Зелений). Створення першої робочої версії, яка мала

назву *«Oak»* (дуб), зайняло 18 місяців. Оскільки виявилось, що ім'я Oak уже використовувалось іншою фірмою, то в результаті тривалих суперечок навколо назви нової мови з-поміж ряду запропонованих було вибрано назву *Java*, у 1995 р. мову було офіційно перейменовано.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. У грудні 1998 р. Sun Microsystems випустила Java 2 (спершу під назвою *J2SE 1.2*), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення корпоративних застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони. У 2006 році в маркетингових цілях версії J2 було перейменовано у Java EE, Java ME та Java SE відповідно.

13 листопада 2006 року Sun випустили більшу частину Java як вільне та відкрите програмне забезпечення згідно з умовами GNU General Public License (GPL). 8 травня 2007 корпорація закінчила процес, в результаті якого всі початкові коди Java були випущенні під GPL, за винятком невеликої частини коду, на який Sun не мала авторського права.

Період становлення Java збігся у часі з розквітом міжнародної інформаційної служби World Wide Web.

Об'єктно-орієнтоване програмування

Об'є́ктно-орієнто́ване програмува́ння (*ООП*; іноді *об'єктозорієнто́ване програмува́ння*, *об'єктоорієнто́ване програмува́ння*) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми[4].

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Об'єктно-орієнтоване програмування — це метод програмування, заснований на поданні програми як сукупності взаємодіючих об'єктів, кожен з яких є примірником певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі під час виконання програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

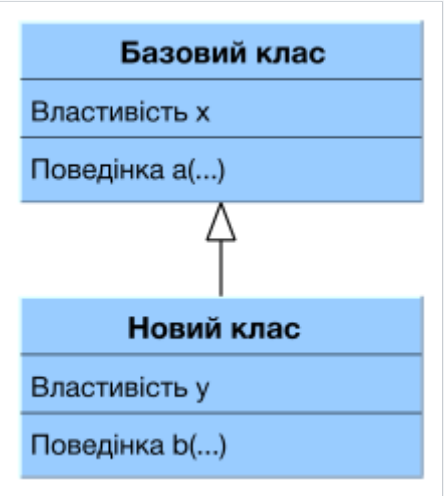
На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

- Все є об'єктами.
- Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, під час якої один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися під час виконання дії.
- Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
- Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.
- У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
- Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично

доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Класові типи для роботи з матрицями та векторами

В об'єктно-орієнтованому програмуванні, клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються *полями даних* (також *властивостями* або *атрибутами*), а члени-функції називають *методами* класу. Створений та ініціалізований екземпляр класу називають *об'єктом* класу. На основі одного класу можна створити безліч *об'єктів*, що відрізнятимуться один від одного своїм *станом* (значеннями полів).



Позначення успадкування класів на UML-діаграмі

В Unified Modeling Language класи зображуються діаграми класів.

Поняття про класи

Поля класу дозволяють вмістити дані про певний реальний об'єкт, а методи здійснювати обробку цих даних. Наприклад, можна створити загальний клас *Людина* з полями *Ім'я* та *Прізвище*, *рік народження*, *професія*,

зарплата. При створенні ж на основі класу конкретного екземпляру, дані поля заповнюються конкретними даними про певну людину. Обробкою цих даних можуть займатися відповідні методи. Наприклад, можна створити метод для обчислення віку людини, тощо.

На основі класів можна створювати підкласи, які успадковують властивості та поведінку батьківських класів. Таким чином можна створити цілу ієрархію класів. Різні мови дещо по різному реалізовують механізм успадкування. Існує множинне та одинарне успадкування. Множинне — це, коли підклас створюється на основі кількох безпосередніх батьків (як то в мові програмування C++). Одинарне успадкування — це коли клас може мати одного безпосереднього батька (мова програмування Java). Надкласи можуть мати свої надкласи, підкласи можуть також бути надкласами для певних класів.

Типи даних

Java є суворо типізованою мовою, кожна змінна та вираз має тип, відомий на етапі компіляції.

Типи даних Java належать до двох категорій: прості (primitive) та вказівникові (reference). До простих типів належить булевий(логічний) тип, числові типи та символьний тип.

Числові типи складаються із цілих типів byte, short, int, long та дійсних типів float, double. Символьний тип представлений типом char. Вказівникові типи складаються із класів, інтерфейсів, масивів. Значенням вказівникового типу є вказівник на об'єкт — екземпляр класу чи масиву. Рядки є об'єктами класу String.

Поняття про матриці

Ма́триця — математичний об'єкт, записаний у вигляді прямокутної таблиці чисел (чи елементів кільця), він допускає операції (додавання, віднімання, множення та множення на скаляр). Зазвичай матриці представляються двовимірними (прямокутними) таблицями. Іноді розглядають багатовимірні матриці або матриці непрямокутної форми. В цій статті вони розглядатися не будуть.

Матриці є корисними для запису даних, що залежать від двох категорій, наприклад: для коефіцієнтів систем лінійних рівнянь та лінійних перетворень.

Приклад 1.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Приклад 2.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$

— Горизонтальні лінії в матриці звуть **рядками**, вертикальні — **стовпчиками** або **стовпцями.**

— Елемент матриці A, що знаходиться на перетині i-го рядка з j-им стовпчиком, називають i,j-им елементом або (i,j)-им елементом A.

Розмір матриці визначає кількість рядків і стовпців, які вона містить. Матрицю із m рядками і n стовпцями називають матрицею $m \times n$ або m-на-n матрицею, а самі m і n називають розмірами матриці.

Матриці, які мають лише один рядок називаються *векторами-рядками*, а ті що мають один стовпець називаються *векторами-стовпцями*. Матриця з однаковою кількістю рядків і стовпців називається *квадратною матрицею*. Матриця із нескінченною кількістю рядків або стовпців (або їх обох) називається *нескінченною матрицею*. У деякому контексті, наприклад, в комп'ютерних програмах, іноді зручно розглядати таку матрицю, що не містить рядків або стовпців, що називається *порожньою матрицею*.

Приклад 3. Додавання матриць.

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{bmatrix}$$

Приклад 4. Скалярний добуток

$$2\begin{bmatrix} 1 & 8 & -3 \\ 4 & -2 & 5 \end{bmatrix} = \begin{bmatrix} 2 \times 1 & 2 \times 8 & 2 \times -3 \\ 2 \times 4 & 2 \times -2 & 2 \times 5 \end{bmatrix} = \begin{bmatrix} 2 & 16 & -6 \\ 8 & -4 & 10 \end{bmatrix}$$

Приклад 5. Множення матриць

$$\begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} (1 \times 3 + 0 \times 2 + 2 \times 1) & (1 \times 1 + 0 \times 1 + 2 \times 0) \\ (-1 \times 3 + 3 \times 2 + 1 \times 1) & (-1 \times 1 + 3 \times 1 + 1 \times 0) \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 4 & 2 \end{bmatrix}$$

Приклад 6. Транспонування матриці

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -6 & 7 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 \\ 2 & -6 \\ 3 & 7 \end{bmatrix}$$

Поняття про вектори

Вектор (від лат. *vector*, «той, що несе») — у найпростішому випадку математичний об'єкт, який характеризується величиною і напрямком. Наприклад, у геометрії і в природничих науках вектор є спрямований відрізок прямої в евклідовому просторі (або на площині).

Приклади: радіус-вектор, швидкість, момент сили. Якщо в просторі задана система координат, то вектор однозначно задається набором своїх координат. Тому в математиці, інформатиці та інших науках упорядкований набір чисел часто теж називають вектором. У більш загальному сенсі вектор у математиці розглядається як елемент деякого векторного (лінійного) простору.

Є одним з основоположних понять лінійної алгебри. При використанні найбільш загального означення векторами виявляються практично всі досліджувані в лінійній алгебрі об'єкти, зокрема матриці, тензори, однак, за наявності в навколишньому контексті цих об'єктів, під вектором мають на увазі відповідно вектор-рядок або вектор-стовпець, тензор першого рангу. Властивості операцій над векторами вивчаються у векторному численні.

Приклад 1

$\langle a_1, a_2, \ldots, a_n \rangle,\ (a_1, a_2, \ldots, a_n), \{a_1, a_2, \ldots, a_n\}$

Приклад 2

$\bar{a},\ \vec{a}, \mathbf{a}, \mathfrak{A},\ \mathfrak{a}.$

Приклад 3. Сумма векторів

$\vec{a} + \vec{b}$

Generics

Узагальнення в Java (англ. *generics*) — це можливість узагальненого програмування, що була додана у мову програмування Java в 2004 році як частина стандартної платформи J2SE 5.0. Узагальнення дають можливість створювати типи або методи таким чином, щоб вони могли оперувати різноманітними типами даних, при цьому на етапі компіляції для типів забезпечується відповідний механізм безпеки. Так, наприклад, метод може приймати параметри типу String або Integer і повертати різноманітні типи без реалізації програмістом кількох різних методів. Після появи узагальнень, ряд класів платформи Java були перероблені під їх використання. Так узагальнення реалізовані в колекціях класів Java (Java Collections Framework), які можуть одночасно зберігати та оперувати різноманітними типами даних. До їх появи для досягнення згаданих цілей програміст використовував надкласи тих типів з якими необхідно працювати та коли це було необхідно здійснювалося перетворення типів,

проте при такому підході легко допуститися цілого ряду помилок, які виявляються лише під час виконання програми.

Наступний блок Java коду демонструє проблему, що виникає, коли узагальнення не застосовуються. Спочатку створюється список: оголошується ArrayList, методи якого працюють з даними типу Object, який є суперкласом для усіх класів Java. Тобто ArrayList може працювати з будь-яким типом даних. Далі додаємо рядок тексту типу String до списку. І зрештою, пробуємо одержати доданий рядок привівши його до типу Integer.

```
List v = new ArrayList();

 v.add("test");

 Integer i = (Integer)v.get(0);
```

Шаблони Java

Шаблони проектування програмного забезпечення (англ. *software design patterns*) — ефектні способи вирішення задач проєктування програмного забезпечення. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення.

Типи шаблонів GOF

- Основні шаблони
- Твірні шаблони
- Структурні шаблони

- Шаблони поведінки
- Шаблони паралельних операцій

Також існує інша група шаблонів проектування, що отримала назву GRASP — General Responsibility Assignment Software Patterns. Опис цих шаблонів наводить Крэг Ларман у своїй книзі. Шаблони GRASP формулюють найбільш базові принципи розподілу обов'язків між типами.

Параметризовані класи

Узагальнення – це механізм побудови програмного коду для деякого типу з довільним іменем з метою його подальшого конвертування (перетворення) у будь-який конкретний посилальний тип. Реалізацію конвертування з узагальненого типу в деякий конкретний здійснює компілятор.

Узагальнення можуть бути застосовані до класів, інтерфейсів чи методів. Якщо клас, інтерфейс чи метод оперує деяким узагальненим типом T, то цей клас (інтерфейс, метод) називається узагальненим. Тип, що отримує узагальнений клас в якості параметру називається параметризованим

типом. Ім’я параметризованого типу можна задавати будь-яким (T, Type, TT і т.д.).

Переваги застосування узагальнень

Використання узагальнень в мові Java дає наступні переваги:

- забезпечується компактність програмного коду;
- завдяки узагальненням усі операції приведення типів виконуються автоматично і неявно. Повторно використовуваний код обробляється більш легше та безпечніше;
- узагальнення забезпечують типову безпеку типів на відміну від використання посилань на тип Object. Як відомо, у мові Java усі класи чи інтерфейси є підкласами класу Object. Це означає, що з допомогою посилання на клас Object можна оперувати різнотипними об’єктами. Однак, такий спосіб не забезпечує типової безпеки типів.

Функції, що викликаються в програмі

- public class Matrix <T extends Number> - функція основного інтерфейсу програми. Створення параметризованого класу Matrix с абстрактним типом даних T і наслідуванням інтерфейсу Number.
- public T get - геттер-функція, приймаюча елемент двовимірного масиву матриці з типом T.
- public T set - сеттер-функція, визначающа елемент двовимірного масиву матриці з типом даних T.

- public String toString() - функція-шаблон, змінююча абстрактний тип даних масиву Matrix на символьний тип String.
- public int Transponse - функція транспонування матриці
- public Matrix<Double> multiply - метод функції, яка дозволяє виконувати множення матриці на матрицю
- private Double calculateMultiplyElement - метод функції, яка дозволяє винконувати множення елементів всередені матриці;
- public <E extends Number> Matrix <Double> multiplyScalar - метод функції, яка дозволяє знаходити скалярний добуток при множені матриць.

Текст програми

Class Main

```
public class Main {

  public static void main(String[] args) {

      Vector<Double> testVector = new Vector<>(1.5, 2.3, 3.5, 4.6, 5.2);

      System.out.printf("first: %f\n", testVector.front());

      System.out.printf("last: %f\n", testVector.back());

      System.out.printf("third: %f\n", testVector.get(2));

      System.out.printf("size: %d\n", testVector.size());

      testVector.push_back(7.3);

      System.out.printf("push_back 7: %s\n", testVector.toString());

      testVector.remove(3);

      System.out.printf("remove element at index 3: %s\n",
testVector.toString());

      testVector.insert(3, 11.9);

      System.out.printf("insert 11 at 3: %s\n", testVector.toString());
```

```
System.out.println("=====Matrix=====");
        Matrix<Float> mtx1 = new Matrix<>(2,3);
        mtx1.set(0.0f, 0, 0);
        mtx1.set(0.1f, 0, 1);
        mtx1.set(0.2f, 0, 2);
        mtx1.set(1.0f, 1, 0);
        mtx1.set(1.1f, 1, 1);
        mtx1.set(1.2f, 1, 2);
        System.out.println("mtx1: " + mtx1.toString());
        Matrix<Float> mtx2 = new Matrix<>(2,3);
        mtx2.set(2.0f, 0, 0);
        mtx2.set(2.1f, 0, 1);
        mtx2.set(2.2f, 0, 2);
        mtx2.set(3.0f, 1, 0);
        mtx2.set(3.1f, 1, 1);
        mtx2.set(3.2f, 1, 2);
        System.out.println("mtx2: " + mtx2.toString());
        System.out.println("mtx1 + mtx2: " + mtx1.add(mtx2).toString());
```

```
System.out.println("mtx1 - mtx2: " + mtx1.substract(mtx2).toString());

Matrix<Integer> mtx3 = new Matrix<>(2,3);

mtx3.set(1, 0, 0);

mtx3.set(2, 0, 1);

mtx3.set(3, 0, 2);

mtx3.set(4, 1, 0);

mtx3.set(5, 1, 1);

mtx3.set(6, 1, 2);

System.out.println("mtx3: " + mtx3.toString());

Matrix<Integer> mtx4 = new Matrix<>(3,2);

mtx4.set(7, 0, 0);

mtx4.set(8, 0, 1);

mtx4.set(9, 1, 0);

mtx4.set(10, 1, 1);

mtx4.set(11, 2, 0);

mtx4.set(12, 2, 1);

System.out.println("mtx4: " + mtx4.toString());

System.out.println("mtx3 * mtx4: " + mtx3.multiply(mtx4).toString());
```

```
    }
}
```

Class Matrix

```
public class Matrix<T extends Number> {
    private Object[][] data;

    public Matrix(int r, int c) {
        data = new Object[r][c];
    }

    public T get(int r, int c) {
        if (data.length > r && data[0].length > c) {
            return (T) data[r][c];
        }
        return null;
    }
```

```
        public Vector<T> getRow(int r) {
            if (r >= data.length) return null;
            Vector<T> row = new Vector<>(data[0].length);
            for (int i = 0; i < row.size(); i++) {
                row.set(i, (T) data[r][i]);
            }
            return row;
        }

public Vector<T> getColumn(int c) {
   if (c >= data[0].length) return null;
   Vector<T> col = new Vector<>(data.length);
   for (int i = 0; i < col.size(); i++) {
      col.set(i, (T) data[i][c]);
```

```
    }
    return col;
}


public void set(T element, int r, int c) {
    if (data.length > r && data[0].length > c) {
        data[r][c] = element;
    }
}

public int[] size() {
    return new int[]{data.length, data[0].length};
}

public <E extends Number> Matrix<Double> multiplyScalar(E scalar) {
    int[] thisSize = size();
    Matrix<Double> result = new Matrix<>(thisSize[0], thisSize[1]);
    for (int r = 0; r < data.length; r++) {
```

```
        for (int c = 0; c < data[0].length; c++) {
            T thisEl = (T) data[r][c];
            Double resultingEl = thisEl.doubleValue() * scalar.doubleValue();
            result.set(resultingEl, r, c);
        }
    }
    return result;
}
public Matrix<Double> multiply(Matrix<T> other) {
    if (other == null) return null;
    int[] otherSize = other.size();
    int[] thisSize = size();
    if (thisSize[1] != otherSize[0]) return null;

    Matrix<Double> result = new Matrix<>(thisSize[0], otherSize[1]);

    for (int r = 0; r < result.size()[0]; r++) {
        for (int c = 0; c < result.size()[1]; c++) {
```

```
            Double resultEl = calculateMultiplyElement(getRow(r),
other.getColumn(c));
            result.set(resultEl, r, c);
        }
    }
    return result;
}

private Double calculateMultiplyElement(Vector<T> rowI, Vector<T> colJ) {
    Double result = 0.0;
    for (int i = 0; i < rowI.size(); i++) {
        result += rowI.get(i).doubleValue() * colJ.get(i).doubleValue();
    }
    return result;
}
public Matrix<Double> substract(Matrix<T> other) {
        return this.add(other.multiplyScalar(-1));
    }
```

```
public <E extends Number> Matrix<Double> add(Matrix<E> other) {
    if (other == null) return null;
    int[] otherSize = other.size();
    int[] thisSize = size();
    if (otherSize[0] != thisSize[0] || otherSize[1] != thisSize[1]) return null;
    Matrix<Double> result = new Matrix<>(thisSize[0], thisSize[1]);
    for (int r = 0; r < data.length; r++) {
        for (int c = 0; c < data[0].length; c++) {
            T thisEl = (T) data[r][c];
            T otherEl = (T) other.get(r, c);
            Double resultingEl = thisEl.doubleValue() + otherEl.doubleValue();
            result.set(resultingEl, r, c);
        }
    }
    return result;
}

public boolean equals(Matrix<T> other) {
    if (other == null) return false;
```

```
    int[] otherSize = other.size();
    int[] thisSize = size();
    if (otherSize[0] != thisSize[0] || otherSize[1] != thisSize[1]) return false;
    for (int r = 0; r < data.length; r++) {
        for (int c = 0; c < data[0].length; c++) {
            if (!data[r][c].equals(other.get(r, c))) return false;
        }
    }
    return true;
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[\n");
    for (Object[] row : data) {
        sb.append("    [ ");
        for (Object o : row) {
            sb.append(o.toString()).append(" ");
```

```
            }
            sb.append("]\n");
        }
        sb.append("]");
        return sb.toString();
    }
}
```

Class Vector

```
import java.util.Arrays;

public class Vector<T extends Number> {
    private Object[] data;
    public Vector(int size) {
        data = new Object[size];
    }
```

```
public Vector(T ... data) {
    this.data = new Object[data.length];
    for (int i = 0; i < data.length; i++) {
        this.data[i] = data[i];
    }
}
public T get(int index) {
    return (T) data[index];
}
public int size() {
    return data.length;
}
public T front() {
    return data.length == 0 ? null : (T) data[0];
}
public T back() {
    return data.length == 0 ? null : (T) data[data.length - 1];
}
public void set(int index, T element) {
```

```
    if (index >= 0 && index < data.length) {
        data[index] = element;
    }
}
public void insert(int index, T element) {
    Object[] bigger = new Object[data.length + 1];
    for (int i = 0; i < index; i++) {
        bigger[i] = data[i];
    }
    bigger[index] = element;
    for (int i = index; i < data.length; i++) {
        bigger[i + 1] = data[i];
    }
    data = bigger;
}
public void push_back(T element) {
    Object[] bigger = new Object[data.length + 1];
    for (int i = 0; i < data.length; i++) {
        bigger[i] = data[i];
```

```
    }
    bigger[bigger.length - 1] = element;
    data = bigger;
}
public void erase(int index) {
    if (index < data.length) {
        data[index] = null;
    }
}

public void remove(int index) {
    erase(index);
    Object[] smaller = new Object[data.length - 1];
    int newi = 0;
    for (int i = 0; i < data.length; i++) {
        if (i != index) {
            smaller[newi++] = data[i];
        }
    }
```

```
    data = smaller;
}
public int find(T element) {
    for (int i = 0; i < data.length; i++) {
        if (data[i].equals(element)) return i;
    }
    return -1;
}
public int accumulateAsInt() {
    int sum = 0;
    for(int i = 0; i < data.length; i++) {
        sum += ((T) data[i]).intValue();
    }
    return sum;
}
public boolean equals(Vector<T> candidate) {
    if (candidate == null) return false;
    if (candidate.size() != data.length) return false;
    for (int i = 0; i < data.length; i++) {
```

```
            if (!data[i].equals(candidate.get(i))) return false;
        }
        return true;
    }
    public void swap(Vector<T> other, int index) {
        if (other.size() > index && data.length > index) {
            T buffer = other.get(index);
            other.set(index, (T) data[index]);
            data[index] = buffer;
        }
    }

    @Override
    public String toString() {
        return "Vector{" +
                "data=" + Arrays.toString(data) +
```

Приклад виконання програми

first: 1.500000

last: 5.200000

```
third: 3.500000

size: 5

push_back 7: Vector{data=[1.5, 2.3, 3.5, 4.6, 5.2, 7.3]}

remove element at index 3: Vector{data=[1.5, 2.3, 3.5, 5.2, 7.3]}

insert 11 at 3: Vector{data=[1.5, 2.3, 3.5, 11.9, 5.2, 7.3]}

=====Matrix=====

mtx1: [

   [ 0.0 0.1 0.2 ]

   [ 1.0 1.1 1.2 ]

]

mtx2: [

   [ 2.0 2.1 2.2 ]

   [ 3.0 3.1 3.2 ]

]

mtx1 + mtx2: [

   [ 2.0 2.1999999061226845 2.400000050663948 ]

   [ 4.0 4.199999928474426 4.400000095367432 ]

]

mtx1 - mtx2: [
```

```
    [ -2.0 -1.9999999031424522 -2.0000000447034836 ]
    [ -2.0 -1.9999998807907104 -2.0 ]
]
mtx3: [
    [ 1 2 3 ]
    [ 4 5 6 ]
]
mtx4: [
    [ 7 8 ]
    [ 9 10 ]
    [ 11 12 ]
]
mtx3 * mtx4: [
    [ 58.0 64.0 ]
    [ 139.0 154.0 ]
]
```

Висновки

Java так використані бібліотеки підтримує основні принципи об'єктно-орієнтованого програмування - інкапсуляцію, поліморфізм і множинне спадкоємство, а також нововведені специфікації і ключові слова в стандарті мови.

Програми базовані на мові Java володіють широкими можливостями доступу до виконання математичних та алгебраїчних операцій. Оскільки методи і массиви призначені не лише для зберігання, але і для вибору і обробки інформації, одним з найважливіших аспектів їх використання є створення запитів до них.

За допомогою двомірних та одновимірних масивів, у мові Java можливо створювати та виконувати різноманітні алгебраїчні операції, у тому числі конструювання и задання нових чисельних матриць та векторів, виконання операцій над ними, редагування та видалення математичних даних.

Використання Java для розробки програм для математичних обчислень має наступні переваги:

Швидкодія. Швидкість роботи програм на Java не поступається программам виконанним на інших обєктно-оріентованних мовах, хоча програмісти отримали в свої руки нові можливості і нові засоби.

Масштабованість. На мові Java розробляють програми для самих різних платформ і систем.

Можливість роботи на низькому рівні з пам'яттю, адресами, портами. (Що, при необережному використанні, може легко перетворитися на недолік.)

Можливість створення узагальнених алгоритмів для різних типів даних, їх спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Вікіпедія — загальнодоступна вільна багатомовна онлайн-енциклопедія. https://uk.wikipedia.org/wiki
2. “Java. Эффективное программирование” Д.Блох (видавництво “Лорі”).
3. «Специальный справочник Java» Борис Карпов, Татьяна Баранова
4. “ Java. Полное руководство. Том 1 ” Герберт Шилдт, 2019.
5. Ресурси порталу https://metanit.com
6. Д.М. Харт. «Системное програмирование в среде Windows (3-е издание)». К.:Вильямс, 2005.
7. Б. Еккер “Філософія Java”
8. Б. Берд “Java для чайников”
9. https://metanit.com/java/tutorial/3.1.php
10. https://metanit.com/java/tutorial/3.14.php
11. https://metanit.com/java/tutorial/3.4.php
12. https://metanit.com/java/tutorial/3.6.php
13. https://www.bestprog.net/ru/2018/06/04/classes-part-1-the-use-of-classes-in-java-programs-class-and-class-object-definition_ru/
14. https://uk.wikibooks.org/wiki/%D0%9E%D1%81%D0%B2%D0%BE%D1%8E%D1%94%D0%BC%D0%BE_Java/%D0%92%D1%81%D1%82%D1%83%D0%BF_%D0%B2_%D0%B

A%D0%BB%D0%B0%D1%81%D0%B8_%D1%82%D0%B0_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8

15. https://javarush.ru/groups/posts/1949-znakomstvo-s-klassami-napisanie-sobstvennihkh-klassov-konstruktorih

16. Д. С. Ерман “Java ++”

17. “Виразний JavaScript” Т. Норман