**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Факультет инфокоммуникационных Сетей и систем (иксс)**

**кафедра ЗАЩИЩЕННЫХ СИСТЕМ СВЯЗИ**

**(ЗСС)**

**Курсовая работа**

**По дисциплине**: «Разработка защищенных сетевых приложений»

**Тема**: «Разработка программы по работе с двумя матрицами»

Выполнил

Морозов Дмитрий Константинович

ИБС-22

Проверил:

Старший преподаватель

Цветков Александр Юрьевич

**Оглавление**

[1. Введение 2](#_Toc153745363)

[2. Основная часть 5](#_Toc153745364)

[2.1 Обзор разработанного приложения 5](#_Toc153745365)

[2.2 Анализ кода программы 6](#_Toc153745366)

[2.3 Работа приложения 10](#_Toc153745367)

[2.4 Блок-схема алгоритма приложения 12](#_Toc153745368)

[2.5 Использование библиотек 14](#_Toc153745369)

[2.6 UML Диаграмма Классов 15](#_Toc153745370)

[2.7 UML Диаграмма Прецедентов 15](#_Toc153745371)

[2.9 Безопасность и надежность программы 19](#_Toc153745372)

[2.10 Анализ производительности 23](#_Toc153745373)

[3. Заключение 26](#_Toc153745374)

[4. Список источников 30](#_Toc153745375)

[5. Приложение 31](#_Toc153745376)

# Введение

**Исходная проблема и ее актуальность**

Матрицы - это удобный способ представления и обработки данных, особенно в областях, связанных с линейной алгеброй, статистикой, машинным обучением и другими смежными областями. Однако, работа с матрицами может быть сложной и трудоемкой задачей, особенно при выполнении операций над большими или сложными матрицами.

Программа для работы с матрицами позволяет автоматизировать и упростить процесс работы с матрицами, предоставляя набор функций и операций для выполнения основных операций, таких как сложение, вычитание, умножение, транспонирование, нахождение определителя, обратной матрицы и т. д. Она облегчает работу с матричными данными, позволяет проводить расчеты, а также решать различные задачи, связанные с матрицами.

Актуальность программы для работы с матрицами обусловлена их широким применением в научных и инженерных областях, таких как физика, математика, статистика, экономика, компьютерная графика, разработка игр, приборостроение и многих других. Также, с развитием технологий и появлением больших объемов данных (Big Data), работа с матрицами стала еще более актуальной и востребованной. Программа для работы с матрицами позволяет эффективно и удобно обрабатывать большие объемы данных, что делает ее не только необходимым, но и полезным инструментом для многих специалистов и исследователей.

**Цели и задачи работы**

Цель работы по разработке программы по работе с двумя матрицами - создать эффективную и надежную программу, которая позволит пользователю выполнять различные операции с матрицами, такие как сложение, вычитание, умножение и нахождение определителя для каждой матрицы.

Задачи, которые необходимо выполнить для достижения данной цели:

1. Разработать методы для каждой из операций с матрицами.

2. Написать удобный код на Java для реализации всех методов.

3. Предусмотреть возможность ввода матрицы из файла.

4. Обеспечить проверку корректности вводимых данных, таких как размерность матриц и тип элементов.

5. Оптимизировать программу для улучшения ее производительности.

6. Документировать программу в отчёте, описав алгоритмы и основные функции.

7. Провести тестирование программы и исправить возможные ошибки и недочеты.

В результате успешной работы была создана программа, которая позволит пользователю удобно и быстро работать с матрицами, выполнять различные операции и получать точные результаты.

**Обоснование логической структуры работы**

Логическое обоснование структуры работы над программой для матриц будет основано на следующих принципах:

Матрицы как основные данные программы: Матрицы играют центральную роль в программе, поэтому структура работы должна быть ориентирована на эффективное и удобное взаимодействие с ними. Это может быть достигнуто путем создания отдельного модуля или класса, который будет содержать все необходимые функции и методы для работы с матрицами.

Функции для работы с матрицами: Программа должна содержать основные функции для работы с матрицами, такие как создание матрицы, заполнение ее элементами, вывод на экран, сложение, вычитание, умножение и т.д. Эти функции позволят упростить работу с матрицами и сделать код более читаемым и понятным.

Модульность и расширяемость: Структура работы над программой должна быть модульной, чтобы было легко расширять ее функциональность в будущем. Например, можно добавить новые операции над матрицами или поддержку различных форматов файлов для сохранения и загрузки матриц. Каждая функция или метод должен выполнять одну конкретную задачу, что позволит легко вносить изменения и исправления без влияния на другие части кода.

Обработка ошибок и исключений: При работе с матрицами могут возникать различные ошибки, такие как неправильные размеры матриц для операций или деление на ноль. Поэтому структура работы должна предусматривать обработку исключений и вывод информации об ошибках пользователям.

Тестирование: Программа для работы с матрицами должна содержать тесты, которые позволят проверить правильность ее работы и обнаружить возможные ошибки. Тесты должны охватывать все основные операции и случаи использования, чтобы убедиться, что программа работает корректно.

# 2. Основная часть

## 2.1 Обзор разработанного приложения

**Описание функциональности приложения**

Разработанное приложение представляет собой консольную программу

написанную на языке Java предназначенную для выполнения операций

с матрицами. Основные функции приложения включают в себя:

1. **Чтение из Файлов**: Пользователь может указать путь к файлам, где в текстовом виде описаны матрицы. Приложение затем читает данные из файлов и сохраняет их в двухмерных массивах.
2. **Вывод матриц**: Пользователи могут увидеть матрицы в отформатированном виде внутри консоли.
3. **Операции над матрицами:** Пользователи могут увидеть результат выполнений операций сложения, вычитания, умножения матриц и нахождения определителя.

**Описание использованных технологий и инструментов**

Приложение разработано с использованием следующих технологий и

инструментов:

1. **Язык Программирования Java**: Java обусловлен его широкими возможностями для создания высокопроизводительных и кроссплатформенных приложений.
2. **Стандартные Библиотеки Java**: Для чтения матриц и работы с исключениями из файлов использовались стандартные классы и методы, предоставляемые Java, что обеспечило высокую эффективность и стабильность работы приложения.

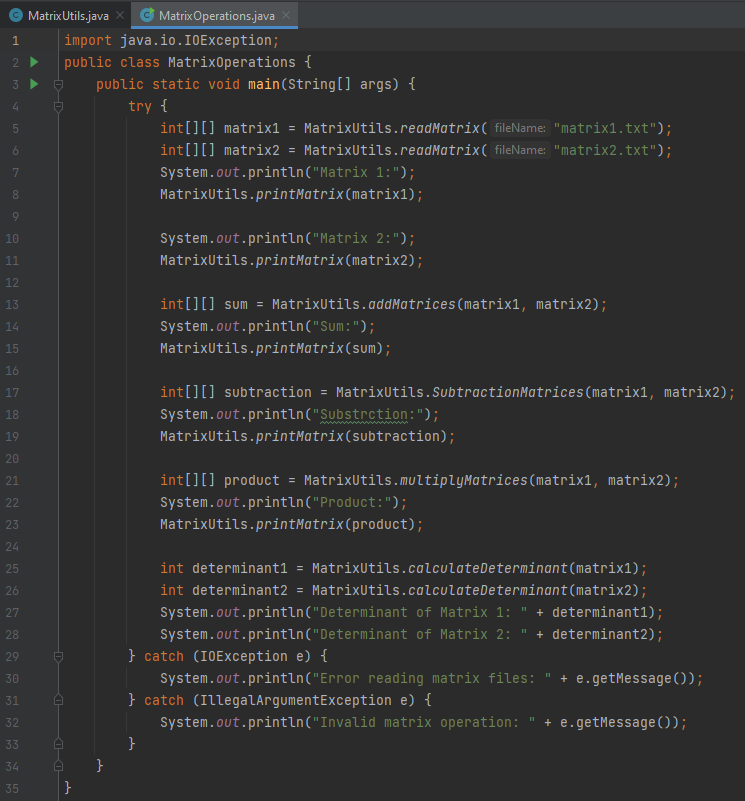
## 2.2 Анализ кода программы

**Обзор основных классов и их функций**

Программа состоит из нескольких ключевых классов, каждый из которых выполняет определенные функции в рамках приложения.

**Класс Main**

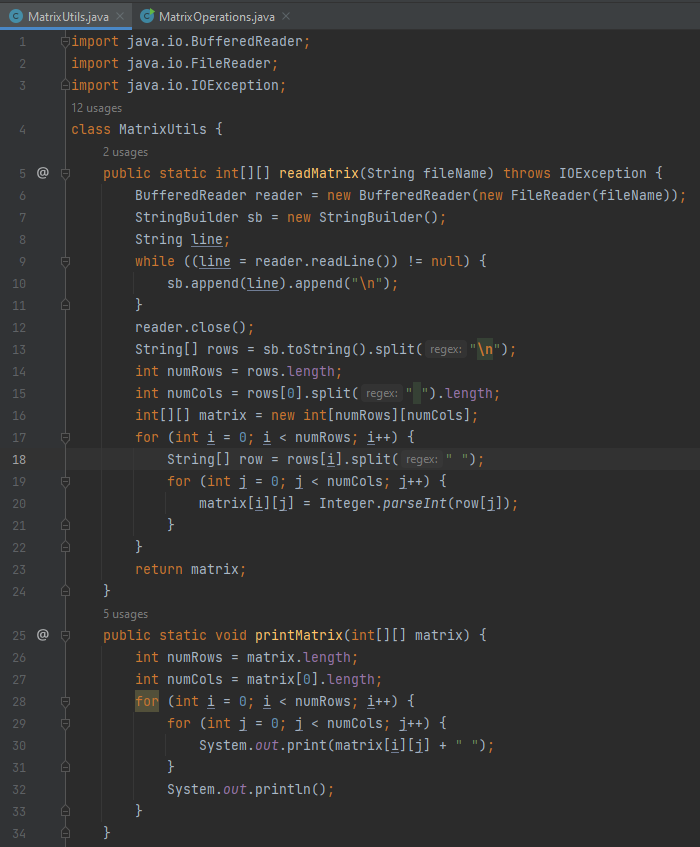
* **Функция**: Является точкой входа в программу. Этот класс отвечает за инициализацию матриц, считанных из файлов и выполнения всех возможных операций и обработки исключений на верхнем уровне.
* **Особенности**: В методе **main** создается 2 двухмерных массива, в которые заносятся данные из файлом с помощью метода класса MatrixUtils. Также здесь присутствует блок **try-catch** для перехвата и обработки исключений, возникающих в процессе работы приложения.

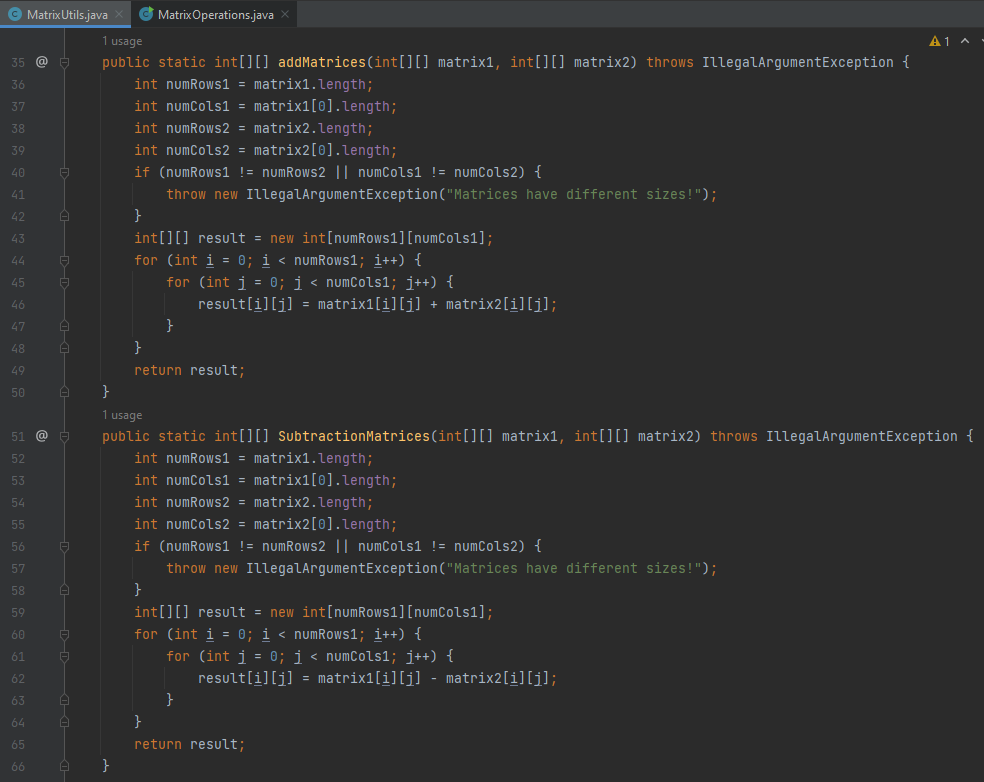
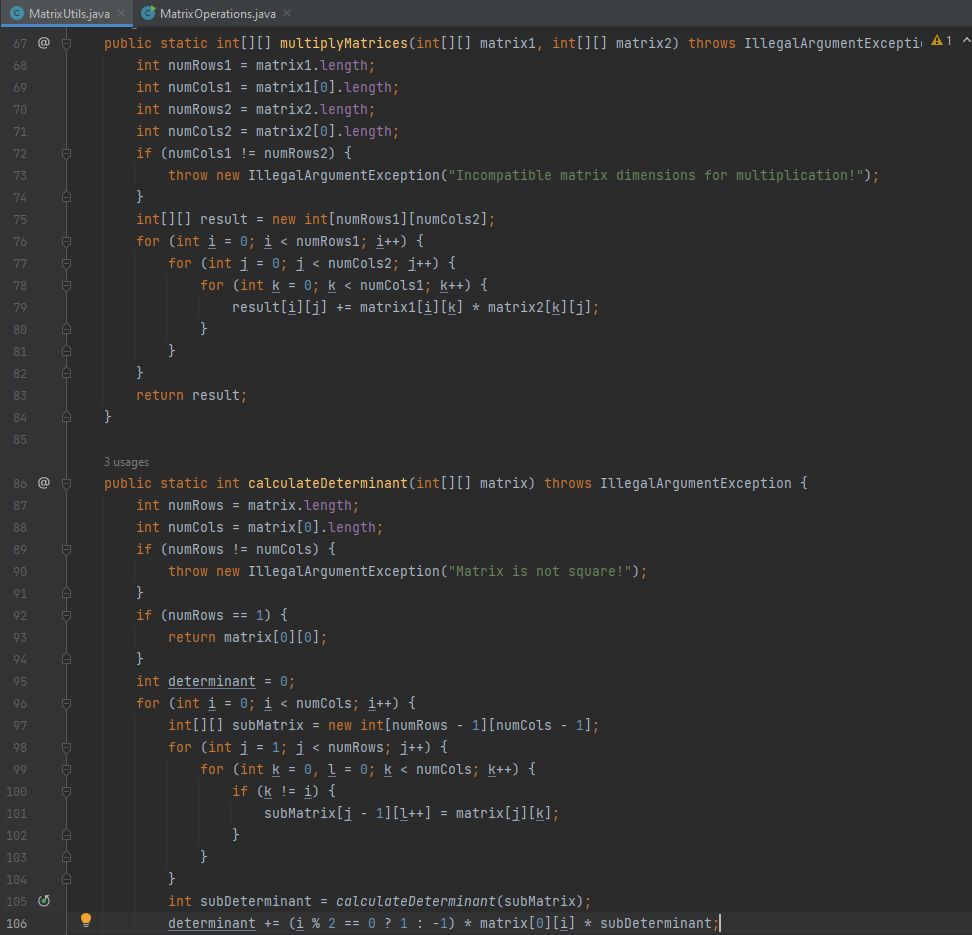


*Класс Main.java*

**Класс MatrixUtils**

* **Функция**: Отвечает за считывание матриц из файлов и операции над ними в формате двухмерных массивов.
* **Особенности**: Класс имеет в себе методы для вывода и выполнения операций сложения, вычитания, умножения матриц и нахождения определителя. Функция readMatrix(String fileName) читает из файла матрицу с помощью BufferedReader и возвращает ее в виде двухмерного массива. Функции addMAtrices, SubstractionMatries, multiplyMatrices совершает вычисления, а именно сложение двух матриц, вычитание двух матриц и произведение двух матриц, соответственно.



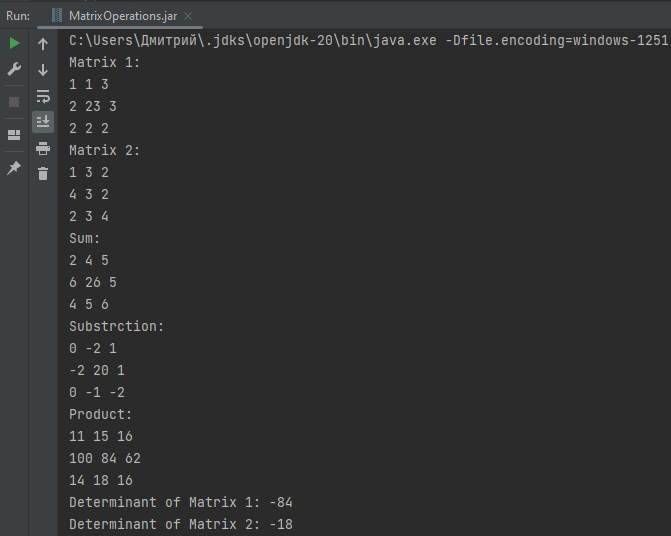


Класс GUI

## 2.3 Работа приложения

***Положительные примеры работы***

* **Сценарий**: Пользователь изначально пишет в созданный файл под названием «matrix1.txt» и «matrix2.txt» значения матриц и компилируем код. После этого мы получаем в консоль сумму, вычитание, умножение наших матриц и их определитель.
* **Ожидаемый результат**: Программа успешно скомпилируется и выведет в консоль выше перечисленные значения.
* **Обработка событий**: Файлы считываются и все значения наших матриц класс MatrixUtils и выполняет все наших вычисления.

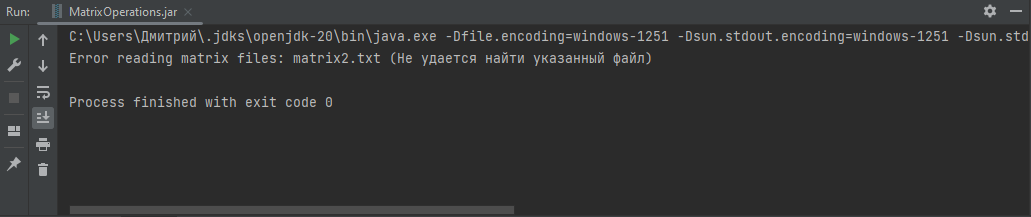


Пример работы программы

***Негативные примеры работы***

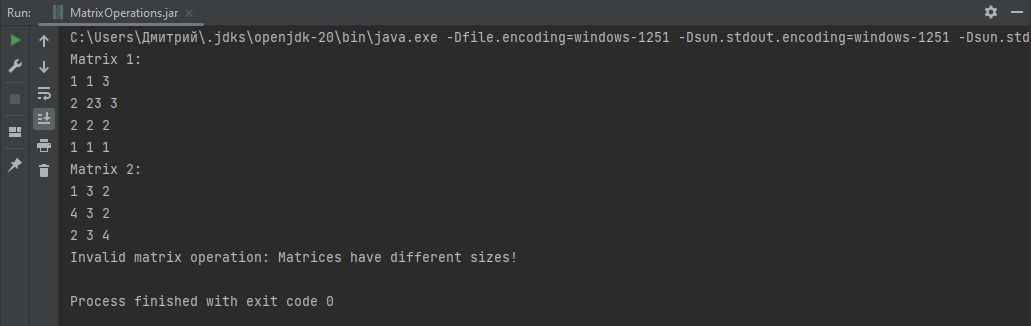
**Примеры обработки ошибок**

* **Сценарий 1**: Пользователь создал файл с расширением txt не так, как указано в коде.
* **Ожидаемый результат**: Вывод ошибки и конкретно, что надо поменять.
* **Обработка событий**: Обработка исключения в классе **MatrixOperations.**



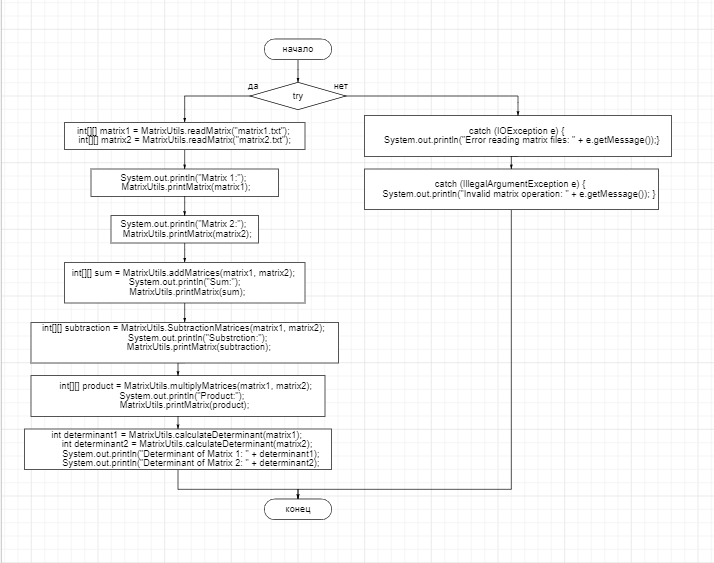
Сценарий 1

* **Сценарий 2**: Ввели некорректно значения в файл.
* **Ожидаемый результат**: Выдаст ошибку, что нельзя сделать то, или иное действие.
* **Обработка событий**:

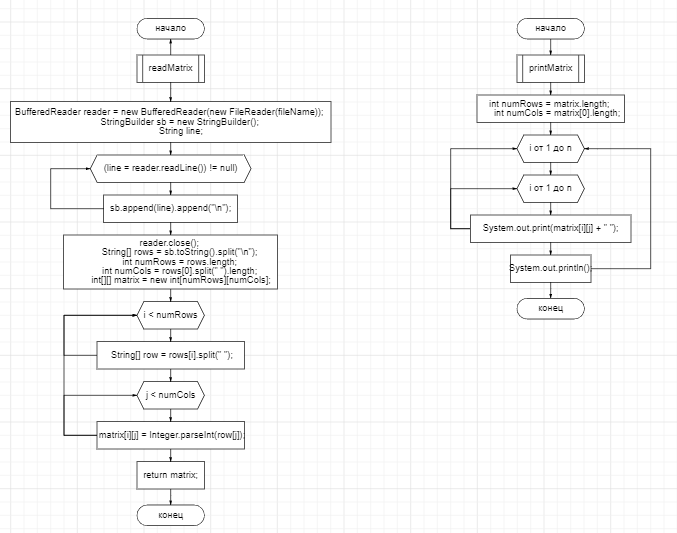


Сценарий 2

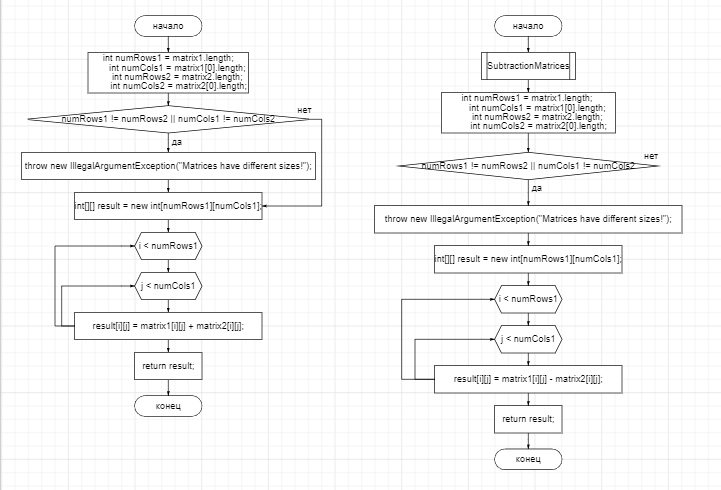
## 2.4 Блок-схема алгоритма приложения



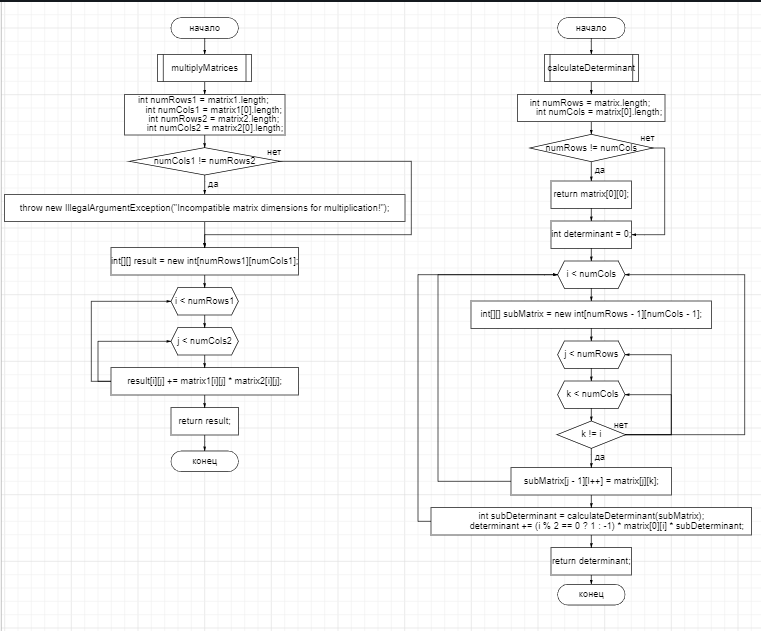
Блок-схема класса MatrixOperations



Блок-схема методов readMatrix и printMatrix в классе MatrixUtils



Блок-схема методов addMatrices и SubtractionMatrices в классе MatrixUtils



Блок-схема методов multiplyMatrices и calculateDeterminant в классе MatrixUtils

## 2.5 Использование библиотек

**Обзор использованных библиотек**

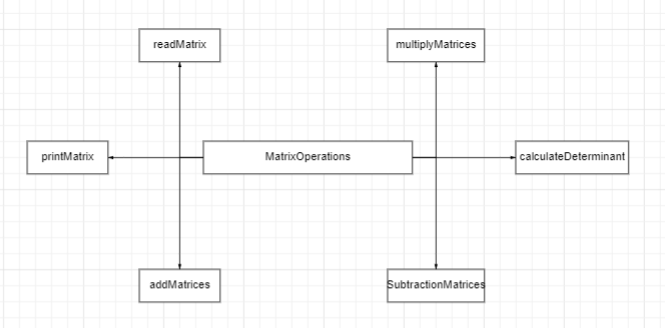
В процессе разработки приложения для работы с матрицами были использованы следующие основные библиотеки и технологии:

1. **Java I/O и NIO**:
   * **Назначение**: Чтение и запись файлов, обработка файловых путей и атрибутов.
   * **Эффективность**: Библиотеки ввода-вывода в Java позволяют эффективно работать с файловой системой, что критически важно для функциональности поиска файлов.

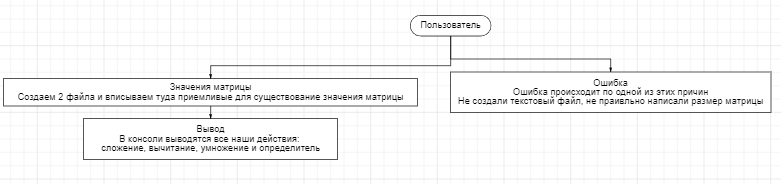
**Почему были выбраны эти библиотеки**

* **Интеграция с Java**: Все перечисленные библиотеки являются частью стандартного набора Java, что обеспечивает гладкую интеграцию и уменьшает вероятность конфликтов и ошибок.
* **Надежность и стабильность**: Эти библиотеки обширно тестировались и используются в многочисленных проектах по всему миру, что гарантирует их надежность и стабильность.
* **Поддержка сообщества**: Благодаря широкому использованию, для этих библиотек существует обширная документация и поддержка со стороны сообщества разработчиков.

## 2.6 UML Диаграмма Классов



## 2.7 UML Диаграмма Прецедентов



**2.8 Основные Концепции и Определения**

В рамках разработки приложения для выполнения операций над матрицами, важно понять ряд ключевых теоретических понятий, которые лежат в основе функциональности программы.

**Операции над матрицами**

* **Матрица:** Структура данных, представляющая собой двумерный массив элементов, расположенных в виде прямоугольной таблицы.
* **Операции над матрицами:**
* Сложение матриц - Происходит поэлементно, при этом соответствующие элементы матриц складываются;
* Умножение матриц: Происходит путем умножения элементов матрицы-множителя на соответствующие элементы другой матрицы и суммированием результатов;
* Определитель матрицы можно найти с помощью различных методов, в зависимости от размера матрицы и доступных инструментов. Вот один из вариантов решения нахождения матрицы:

Метод Гаусса-Жордана:

- Приведите матрицу к ступенчатому виду (используя элементарные преобразования строк), сохраняя все вычисленные коэффициенты.

- Умножьте элементы, находящиеся на главной диагонали, другой ступенчатой матрицы.

- Определитель матрицы будет равен произведению этих элементов и знаку, соответствующему количеству перестановок, которые были сделаны.

* **Вывод матрицы на экран:** Информация о матрице отображается пользователю в консоле.

**Обработка Файлов в Java**

* **Java I/O и NIO**: В Java для работы с файлами используются пакеты I/O (Input/Output) и NIO (New I/O). Они предоставляют классы и интерфейсы для чтения, записи и управления файлами и файловыми системами.
* **Особенности и Применение**: Эти пакеты позволяют эффективно обрабатывать файловые операции, такие как чтение данных из файла или запись в файл, а также получение информации о файлах и директориях.

**Теоретический Обзор Операций Над Матрицами**

Поиск файлов требует применения специализированных алгоритмов. В контексте нашей программы, используется алгоритм, который можно описать следующими шагами:

1. **Чтение файлов** Программа начинает с чтения указанных пользователем файлов для получения данных о матрицах.
2. **Выполнение операций**: Выполняются все реализованные операции над матрицами.
3. **Вывод результатов**: Результаты выполнения операций над матрицами в читаемом человеком виде выводится в консоль.

**Важность Эффективности и Точности**

* **Эффективность**: Учитывая, что операции над матрицами могут выполняться на маломощных компьютерах, эффективность алгоритма критически важна. Она обеспечивает минимальное время отклика и оптимальное использование ресурсов.
* **Точность**: Точность алгоритма вычисления, например, вычисления определителя обеспечивает, что результаты будут надежными и соответствующими заданным пользователем критериям. Это особенно важно в приложениях, где требуется высокая степень точности, например, в космических вычислительных аппаратах.

**Методы Обработки Файлов**

Обработка файлов - один из ключевых компонентов в приложении для выполнения операций с матрицами. Рассмотрим подробнее методы и практики, связанные с этой областью.

**Работа с Файловой Системой в Java**

* **Использование Java I/O и NIO**: В Java, часто используются классы из пакетов java.io и java.nio для выполнения операций чтения и записи файлов. Эти пакеты предлагают разнообразные возможности для работы с файловой системой, включая возможность чтения и записи данных, обработку путей к файлам и управление атрибутами файлов.
* **Буферизированное Чтение и Запись**: Буферизированные потоки, такие как BufferedReader, применяются с целью оптимизировать операции чтения и записи, снижая количество операций ввода-вывода и, следовательно, улучшая эффективность взаимодействия с данными.

**Аспекты Безопасности в Программировании**

При взаимодействии с файлами и данными необходимо уделять внимание аспектам безопасности, чтобы гарантировать защиту информации и предотвращать возможные уязвимости.

**Защита Данных**

* **Контроль Доступа**: Необходимо гарантировать тщательное регулирование и контроль доступа к файлам и операциям с ними.
* **Шифрование**: Иногда необходимо применять шифрование данных, чтобы обеспечить их конфиденциальность.

**Предотвращение Уязвимостей**

* **Обработка Исключений**: Профессиональная обработка исключительных ситуаций помогает избежать аварийного завершения программы при возникновении ошибок.
* **Проверка Ввода**: Корректная обработка исключений способствует предотвращению аварийного завершения программы в случае возникновения ошибок.

## 2.9 Безопасность и надежность программы

**Важность Безопасности в Современных Приложениях**

В современных приложениях безопасность программного обеспечения становится критически важной. В случае приложения для работы с матрицами, безопасность обеспечивает защиту данных пользователя и обеспечивает стабильную работу системы.

**Основные Угрозы**

* **Несанкционированный доступ**: Несанкционированный доступ к файлам может привести к утечке конфиденциальной информации.
* **Исполнение вредоносного кода**: Исполнение небезопасного кода может привести к компрометации системы пользователя.
* **Повреждение данных**: Неправильная обработка файлов может привести к их случайному повреждению или потере.

**Меры по Обеспечению Безопасности**

**Обработка Исключений**

* **Предотвращение "Падения" Приложения**: Корректная обработка исключений важна для предотвращения внезапных сбоев приложения, особенно при работе с файлами.
* **Логирование Ошибок**: Логирование ошибок помогает в диагностике проблем и улучшении надежности программы.

**Валидация Ввода**

* **Проверка Данных Пользователя**: Валидация входных данных пользователя помогает предотвратить атаки, такие как инъекции кода или пути.
* **Ограничение Доступа к Файлам**: Проверка корректности путей файлов для исключения ошибок, связанных с недоступными или несуществующими файлами.

**Использование Надежных Библиотек**

* **Стандартные Библиотеки Java**: Использование проверенных и надежных библиотек Java уменьшает риск безопасностных уязвимостей, связанных с обработкой данных.

**Значение Надежности в Приложениях**

Надежность программного обеспечения — это не только обеспечение стабильной и безошибочной работы, но и уверенность пользователя в том, что приложение будет корректно функционировать в различных условиях и сценариях использования.

**Резервное Копирование и Восстановление**

* **Сохранение Состояния**: Реализация функций сохранения состояния и настроек приложения помогает в быстром восстановлении работы после сбоев.

**Тестирование и Отладка**

* **Широкое Тестирование**: Регулярное и всестороннее тестирование программы на предмет ошибок и уязвимостей является ключевым элементом обеспечения её надежности.

**Шифрование Данных**

описанные практики особенно важны в случаях, когда приложение обрабатывает чувствительные данные, такие как пароли, личная информация пользователей или финансовые данные. Использование проверенных и надежных библиотек Java для шифрования данных помогает предотвратить утечку информации и защищает конфиденциальность пользователей.

Одним из популярных инструментов для шифрования данных в Java является Java Cryptography Extension (JCE). JCE предоставляет набор классов и методов для реализации различных алгоритмов шифрования, таких как AES, RSA и других.

При использовании JCE можно гарантировать, что методы шифрования работают правильно и безопасно, поскольку библиотека прошла многочисленные проверки и тестирования на уязвимости. Это снижает риск возникновения уязвимостей, связанных с обработкой данных, таких как возможность перехвата информации или взлома системы.

В дополнение к шифрованию данных, также важно регулярно обновлять использованные библиотеки и инструменты с целью получения последних обновлений безопасности и исправления уязвимостей. Постоянное обновление кода и проверка его на предмет возможных уязвимостей помогут улучшить надежность приложения и обезопасить обработку данных.

В целом, использование проверенных и надежных библиотек Java позволяет разработчикам повысить надежность приложения и снизить риск возникновения безопасностных уязвимостей, связанных с обработкой данных. Это позволяет пользователям быть уверенными в том, что их данные защищены и приложение работает правильно в любых условиях.

**Обновления и исправления**

• Регулярные обновления: Регулярное обновление программного обеспечения помогает исправлять обнаруженные уязвимости и улучшать его функциональность.

• Быстрое реагирование на уязвимости: Способность быстро выпускать патчи и обновления в ответ на новые угрозы повышает надежность приложения.

Обеспечение надежности

Стабильность приложения

• Иммунитет к ошибкам: При разработке приложения необходимо предусмотреть возможность возникновения ошибок (как пользовательских, так и системных) и защитить его от критических сбоев.

**Робастность Приложения**

* **Устойчивость к Ошибкам**: Разработка приложения должна учитывать возможность возникновения ошибок (как пользовательских, так и системных) и обеспечивать устойчивость к ним, предотвращая критические сбои.
* **Адаптивность к Различным Условиям**: Приложение должно быть способно адаптироваться к различным операционным системам и условиям использования.

**Процедуры Восстановления**

* **Механизмы Восстановления**: В случае сбоев или ошибок важно иметь механизмы для восстановления работы приложения без потери данных пользователя.
* **Резервное Копирование**: Предоставление функций резервного копирования помогает предотвратить потерю данных в случае сбоев или повреждений.

## 2.10 Анализ производительности

**Введение в Анализ Производительности**

Необходимо провести анализ производительности приложения для определения его эффективности и выявления областей, где можно внести потенциальные улучшения. В случае приложения для работы с матрицами, производительность играет ключевую роль, особенно при обработке больших объемов данных.

**Ключевые Параметры Производительности**

**Оценка производительности:**

- Мы можем измерить время отклика, которое требуется приложению для выполнения операций, таких как поиск файлов, чтение файлов и вывод данных. Это позволит нам оценить скорость работы приложения.

- Мы можем оптимизировать алгоритмы, используемые в приложении. Например, замена алгоритмов на более эффективные может заметно ускорить работу приложения. В частности, мы можем использовать более эффективные алгоритмы для нахождения определителя в классе MatrixUtils.

Оценка использования ресурсов:

- Нам важно анализировать, сколько оперативной памяти использует приложение, чтобы определить его эффективность. Особое внимание нужно уделить эффективному управлению памятью при обработке больших файлов.

- Мы должны также наблюдать за нагрузкой на процессор во время работы приложения, чтобы выявить возможные узкие места в его производительности.

**Исследование производительности приложения**

**Практические эксперименты**

• Тестирование с различными объемами данных: Проверка производительности приложения при работе с матрицами разных размеров, от маленьких до очень больших. Это позволило оценить, как изменение объема данных влияет на скорость и эффективность поиска.

• Анализ использования сценариев: Исследование различных сценариев использования, включая обработку некорректных матриц, помогло оценить общую производительность приложения.

**Результаты и наблюдения**

• Время отклика: Было замечено, что время отклика увеличивается по мере увеличения размера матрицы, что указывает на необходимость оптимизации алгоритма поиска.

• Использование ресурсов: Приложение показало умеренное потребление ресурсов, однако при обработке очень больших матриц наблюдалось увеличение использования памяти.Методы и Инструменты для Оптимизации Производительности

**Профилирование Приложения**

**•** Использование Профайлеров: Применение инструментов, таких как VisualVM или JProfiler, для профилирования приложения, может помочь выявить места с наибольшим влиянием на производительность, такие как избыточные операции ввода-вывода или неэффективное использование памяти.

• Анализ Результатов: На основе полученных данных от профилирования можно определить, какие части кода требуют оптимизации. Например, если обнаружено, что процесс нахождения определителя в классе MatrixUtils занимает больше времени, чем ожидалось, это может указывать на необходимость улучшения алгоритма поиска

**Оптимизация Алгоритмов**

Оптимизация алгоритмов является важным аспектом для повышения эффективности программного кода. Например, использование многопоточности при работе с файлами и матрицами может значительно улучшить производительность. Другой способ оптимизации - избегание лишних и повторяющихся операций, особенно в циклах и при обработке данных.

Также важной частью оптимизации является эффективное использование ресурсов. Правильное управление памятью, например, через правильное управление объектами и предотвращение утечек памяти, может повысить общую производительность и стабильность приложения. Балансировка нагрузки на процессор также может улучшить отклик приложения и уменьшить его влияние на общую производительность системы.

**Выводы и Дальнейшие Шаги**

* **Необходимость Оптимизации**: Анализ подтвердил, что хотя приложение работает стабильно, существует потенциал для улучшения производительности, особенно при обработке больших объемов данных.
* **Планирование Улучшений**: В качестве следующих шагов рекомендуется провести детальную оптимизацию ключевых компонентов приложения, особенно тех, которые оказывают наибольшее влияние на производительность.

# Заключение

**Итоги Работы**

В результате разработки программы для работы с двумя матрицами были реализованы функции сложения, вычитания, умножения матриц и расчета определителя матрицы. Элементы матриц были заданы двумя файлами, выбранными пользователями, а результаты операций выводились на экран.

Программа предоставляет удобный способ для работы с матрицами, позволяя пользователю использовать ввод из файлов и получать результаты непосредственно на экране. Такой подход обеспечивает удобство и гибкость в использовании программы.

Реализация функций сложения, вычитания, умножения матриц и расчета определителя матрицы позволяет пользователю получать разнообразные результаты, в зависимости от потребностей исследований или задач.

Таким образом, разработанная программа представляет собой полезный инструмент для работы с матрицами, предоставляя все необходимые возможности для выполнения операций и удобный способ для просмотра результатов.

**Основные Достижения**

1. **Реализация Пользовательского Интерфейса (GUI)**: Использование стандартной библиотеки для ввода-вывода позволило создать интуитивно понятный и функциональный интерфейс.
2. **Эффективное выполнение операций**: Интеграция алгоритмов для работы над матрицами обеспечила эффективное и точное определение результатов.
3. **Гибкость Управления Матрицами**: Реализованный функционал управления файлами с матрицами обеспечивает гибкость в вводе различных данных.
4. **Обработка Исключений и Пользовательский Опыт**: Внимание к обработке исключений и предоставлению четких сообщений об ошибках повысило удобство использования приложения и его надежность.

**///Выводы и Направления для Дальнейшего Развития**

* **Эффективность и Надежность**: Приложение успешно выполняет свои задачи, обладая высокой эффективностью и надежностью в работе.
* **Расширение Функционала**: В перспективе, приложение может быть расширено путем добавления новых возможностей, таких как поддержка более сложных операций или интеграция с внешними сервисами и базами данных.
* **Оптимизация Производительности**: Существует потенциал для дальнейшей оптимизации производительности, особенно при работе с большими объемами данных.
* **Улучшение Пользовательского Интерфейса**: Интерфейс может быть улучшен для обеспечения удобного и интуитивно понятного взаимодействия с пользователем.

**Оценка Соответствия Разработанного Приложения Поставленным Целям:**

**Цели Проекта**

* **Разработка ПО для выполнения операций над матрицами**: Центральной задачей было создать приложение, позволяющее пользователю эффективно выполнять операции над матрицами по заданным матрицам в указанных файлах.

**Достижения**

* **Функциональность**: Реализованный класс MatrixUtilsуспешно выполняет чтение матриц, используя метод **readMatrix**.
* **Управление методами**: Класс MatrixUtils предоставляет функциональность для сложения, вычитания, умножения и вычисления определителя матриц. Это является ключевым достижением, соответствующим первоначальной цели проекта.

**Недостатки и Ограничения**

* **Ограниченная функциональность консоли**: Хотя пользовательский интерфейс, созданный с помощью консоли, интуитивно понятен, он может быть улучшен с точки зрения дизайна и пользовательского опыта.
* **Обработка исключений**: Несмотря на наличие базовой обработки ошибок, дополнительные меры безопасности и более детальная обработка исключений могли бы повысить надежность системы.

**Критический Анализ Выполненной Работы**

**Анализ Кода и Архитектуры**

* **Структурирование и Читаемость Кода**: Код программы хорошо структурирован и разделен на классы с четко определенными функциями. Однако, возможно улучшение в части комментирования и документирования кода для облегчения понимания и поддержки.
* **Эффективность Алгоритмов**: Алгоритмы, используемые в **MatrixUtils**, эффективны, но могут быть оптимизированы для улучшения производительности, особенно при работе с большими объемами данных.

**Тестирование и Надежность**

* **Покрытие Тестами**: Хотя приложение было протестировано на функциональность, полное покрытие тестами всех аспектов программы не было достигнуто. Особое внимание следует уделить тестированию исключительных ситуаций и граничных условий.
* **Обработка Ошибок**: Текущая система обработки ошибок обеспечивает базовый уровень защиты, но её можно расширить, включив более детальное логирование и обратную связь пользователю.

**Выводы и Пути Улучшения**

* **Оценка Общей Эффективности**: В целом, приложение успешно решает поставленную задачу выполнения операций над матрицами. Однако, существует потенциал для дальнейших улучшений как в функциональности, так и в надежности программы.
* **Направления Развития**: В будущем, можно расширить функциональность приложения, добавив более сложные операции и улучшенный пользовательский интерфейс. Также важно сосредоточиться на улучшении надежности и безопасности программы, включая более тщательное тестирование и обработку ошибок.

# Список источников

1. Что такое невырожденная матрица. Ссылка: https://uchet-jkh.ru/i/cto-takoe-nevyrozdennaya-matrica/
2. Кэти Сьерра и Берт Бейтс - Изучаем Java Ссылка: https://books.google.ie/books/about/Head\_First\_Java.html?id=nzhxR1spWEYC&source=kp\_cover&redir\_esc=y.
3. Allen B. Downey & Chris Mayfield “Think Java. How to Think Like a Computer Scientist” – бумажная версия.
4. Эффективное вычисление определителя на языке Java Ссылка: <https://codeforces.com/blog/entry/19750?locale=ru>
5. Изучение Java с нуля Ссылка: https://youtube.com/playlist?list=PLDyJYA6aTY1lT614ixLYq48har7EnCXpk&si=5eVQojrxQxNUeqks
6. Effective Java Addison-Wesley Professional Ссылка: <https://books.google.ie/books/about/Effective_Java.html?id=ka2VUBqHiWkC&source=kp_cover&redir_esc=y>
7. Data Structures and Algorithms in Java Ссылка: <https://www.amazon.com/Data-Structures-Algorithms-Java-2nd/dp/0672324539>

# Приложение

Листинг программы:

Класс MatrixOperations

import java.io.IOException;  
public class MatrixOperations {  
 public static void main(String[] args) {  
 try {  
 int[][] matrix1 = MatrixUtils.*readMatrix*("matrix1.txt");  
 int[][] matrix2 = MatrixUtils.*readMatrix*("matrix2.txt");  
 System.*out*.println("Matrix 1:");  
 MatrixUtils.*printMatrix*(matrix1);  
  
 System.*out*.println("Matrix 2:");  
 MatrixUtils.*printMatrix*(matrix2);  
  
 int[][] sum = MatrixUtils.*addMatrices*(matrix1, matrix2);  
 System.*out*.println("Sum:");  
 MatrixUtils.*printMatrix*(sum);  
  
 int[][] subtraction = MatrixUtils.*SubtractionMatrices*(matrix1, matrix2);  
 System.*out*.println("Substrction:");  
 MatrixUtils.*printMatrix*(subtraction);  
  
 int[][] product = MatrixUtils.*multiplyMatrices*(matrix1, matrix2);  
 System.*out*.println("Product:");  
 MatrixUtils.*printMatrix*(product);  
  
 int determinant1 = MatrixUtils.*calculateDeterminant*(matrix1);  
 int determinant2 = MatrixUtils.*calculateDeterminant*(matrix2);  
 System.*out*.println("Determinant of Matrix 1: " + determinant1);  
 System.*out*.println("Determinant of Matrix 2: " + determinant2);  
 } catch (IOException e) {  
 System.*out*.println("Error reading matrix files: " + e.getMessage());  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Invalid matrix operation: " + e.getMessage());  
 }  
 }  
}

Класс MatrixUtils

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
class MatrixUtils {  
 public static int[][] readMatrix(String fileName) throws IOException {  
 BufferedReader reader = new BufferedReader(new FileReader(fileName));  
 StringBuilder sb = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 sb.append(line).append("\n");  
 }  
 reader.close();  
 String[] rows = sb.toString().split("\n");  
 int numRows = rows.length;  
 int numCols = rows[0].split(" ").length;  
 int[][] matrix = new int[numRows][numCols];  
 for (int i = 0; i < numRows; i++) {  
 String[] row = rows[i].split(" ");  
 for (int j = 0; j < numCols; j++) {  
 matrix[i][j] = Integer.*parseInt*(row[j]);  
 }  
 }  
 return matrix;  
 }  
 public static void printMatrix(int[][] matrix) {  
 int numRows = matrix.length;  
 int numCols = matrix[0].length;  
 for (int i = 0; i < numRows; i++) {  
 for (int j = 0; j < numCols; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 public static int[][] addMatrices(int[][] matrix1, int[][] matrix2) throws IllegalArgumentException {  
 int numRows1 = matrix1.length;  
 int numCols1 = matrix1[0].length;  
 int numRows2 = matrix2.length;  
 int numCols2 = matrix2[0].length;  
 if (numRows1 != numRows2 || numCols1 != numCols2) {  
 throw new IllegalArgumentException("Matrices have different sizes!");  
 }  
 int[][] result = new int[numRows1][numCols1];  
 for (int i = 0; i < numRows1; i++) {  
 for (int j = 0; j < numCols1; j++) {  
 result[i][j] = matrix1[i][j] + matrix2[i][j];  
 }  
 }  
 return result;  
 }  
 public static int[][] SubtractionMatrices(int[][] matrix1, int[][] matrix2) throws IllegalArgumentException {  
 int numRows1 = matrix1.length;  
 int numCols1 = matrix1[0].length;  
 int numRows2 = matrix2.length;  
 int numCols2 = matrix2[0].length;  
 if (numRows1 != numRows2 || numCols1 != numCols2) {  
 throw new IllegalArgumentException("Matrices have different sizes!");  
 }  
 int[][] result = new int[numRows1][numCols1];  
 for (int i = 0; i < numRows1; i++) {  
 for (int j = 0; j < numCols1; j++) {  
 result[i][j] = matrix1[i][j] - matrix2[i][j];  
 }  
 }  
 return result;  
 }  
 public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) throws IllegalArgumentException {  
 int numRows1 = matrix1.length;  
 int numCols1 = matrix1[0].length;  
 int numRows2 = matrix2.length;  
 int numCols2 = matrix2[0].length;  
 if (numCols1 != numRows2) {  
 throw new IllegalArgumentException("Incompatible matrix dimensions for multiplication!");  
 }  
 int[][] result = new int[numRows1][numCols2];  
 for (int i = 0; i < numRows1; i++) {  
 for (int j = 0; j < numCols2; j++) {  
 for (int k = 0; k < numCols1; k++) {  
 result[i][j] += matrix1[i][k] \* matrix2[k][j];  
 }  
 }  
 }  
 return result;  
 }  
  
 public static int calculateDeterminant(int[][] matrix) throws IllegalArgumentException {  
 int numRows = matrix.length;  
 int numCols = matrix[0].length;  
 if (numRows != numCols) {  
 throw new IllegalArgumentException("Matrix is not square!");  
 }  
 if (numRows == 1) {  
 return matrix[0][0];  
 }  
 int determinant = 0;  
 for (int i = 0; i < numCols; i++) {  
 int[][] subMatrix = new int[numRows - 1][numCols - 1];  
 for (int j = 1; j < numRows; j++) {  
 for (int k = 0, l = 0; k < numCols; k++) {  
 if (k != i) {  
 subMatrix[j - 1][l++] = matrix[j][k];  
 }  
 }  
 }  
 int subDeterminant = *calculateDeterminant*(subMatrix);  
 determinant += (i % 2 == 0 ? 1 : -1) \* matrix[0][i] \* subDeterminant;  
 }  
 return determinant;  
 }  
}

**Ссылка на код (GitHub):**