**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

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

**(СПбГУТ)**

Факультет Инфокоммуникационных сетей и систем

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

**Отчет по курсовой работе**

*Разработка анализатора текстовых данных для определения тематики текста*

Студент:

Шулепов Г.К. ИБС-23

*(Ф.И.О., № группы) (подпись)*

Преподаватель:

асс.к.ЗСС Шариков П.И.

*(уч. степень, уч. звание, Ф.И.О.) (подпись)*

# Оглавление

[Введение 3](#_Toc153744515)

[Глава 1. Методология анализа текста и инструменты реализации 5](#_Toc153744516)

[1.1 Задача анализа темы текста 5](#_Toc153744517)

[1.2 Среда программирования 6](#_Toc153744518)

[1.3 Инструменты для реализации проекта 8](#_Toc153744519)

[Глава 2. Описание программы 12](#_Toc153744520)

[2.1 Метод determine\_theme 12](#_Toc153744521)

[2.2 Метод read\_text\_from\_file 13](#_Toc153744522)

[2.3 Метод get\_file\_extension 14](#_Toc153744523)

[2.4 Методы read\_text\_from\_txt\_file/doc\_file/docx\_file 15](#_Toc153744524)

[2.5 Метод tokenize 16](#_Toc153744525)

[2.6 Метод analyze\_text 17](#_Toc153744526)

[2.7 Метод main 19](#_Toc153744527)

[2.8 Переменные в коде 20](#_Toc153744528)

[Глава 3. Комментарии к работе и завершающие этапы 22](#_Toc153744529)

[3.1 Проверка программы. Создание JAR-файла 22](#_Toc153744530)

[3.2. UML диаграмма 24](#_Toc153744531)

[3.3 Плюсы и минусы программы 26](#_Toc153744532)

[3.4 Обработка исключений в программе 27](#_Toc153744533)

[3.5 Принципы переиспользования кода 28](#_Toc153744534)

[3.6 Работа с GitHub 29](#_Toc153744535)

[Заключение 34](#_Toc153744536)

[Список использованной литературы 35](#_Toc153744537)

# Введение

В современном информационном обществе объем текстовых данных растет в геометрической прогрессии, что представляет собой огромный вызов для их анализа и понимания. Один из важных аспектов в области обработки текста - определение тематики текста. Это является ключевым элементом во многих приложениях, таких как поисковые системы, социальные сети и другие.

Определение тематики текста позволяет эффективно структурировать и категоризировать огромные объемы информации, упрощая последующий анализ и понимание содержания текста. Обработка больших объемов текста человеком часто является трудоемкой и порой невозможной задачей. В связи с этим необходим автоматизированный алгоритм, способный анализировать текстовые документы и автоматически выделять темы из содержания.

Целью данной работы является разработка анализатора текстовых данных, способного автоматически определять тематику текста с высокой точностью. Реализация такого инструмента позволит значительно упростить и ускорить процесс анализа текстов, повысив эффективность работы с большими объемами информации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Разработать методологию анализа текста для определения ключевых слов и выражений, характеризующих тематику текста.
2. Реализовать алгоритм парсинга текста, осуществляющий извлечение слов из входного текста.
3. Сформировать словари ключевых слов для требуемых тематик (медицина, история, программирование, финансы, сети, криптография).
4. Создать механизм подсчета частоты вхождения ключевых слов и визуального отображения этой статистики.
5. Создать механизм обработки исключений для более понятной обратной связи с пользователем.
6. Разработать программу на языке Java, предназначенную для анализа текстовых данных с целью определения их тематики.

Актуальность проведения исследования обусловлена необходимостью эффективного управления информацией в эру больших данных. Разработка автоматизированных методов определения тематики текста позволит существенно улучшить процессы анализа, классификации и категоризации текстовых данных, что имеет важное значение для индустрии информационных технологий, научных исследований и общественного обмена знанием.

Глава 1. Методология анализа текста и инструменты реализации

## 1.1 Задача анализа темы текста

Для определения темы входного текста было решено воспользоваться простым методом анализа текста, основанном на методе «Мешок слов» (Bag of Words). Этот метод представляет текст как неупорядоченный набор слов, игнорируя их порядок и структуру в предложении. Основная идея заключается в том, чтобы создать словарь уникальных слов из текста и затем представить каждый документ (в вашем случае, текст) в виде вектора, где каждый элемент вектора представляет собой частоту встречаемости соответствующего слова из словаря в данном документе.

Процесс работы нашего метода включает в себя следующие шаги:

1. Токенизация: выбранный для анализа текст необходимо подготовить. Токенизация – это процесс разбиения текста на отдельные единицы, называемые токенами. Токен обычно представляет собой одно слово или более крупный фрагмент текста, такой как символ, фраза, или предложение. Текст разделяется по пробелам, удаляются все знаки пунктуации. Задача токенизации заключается в выделении смысловых единиц из текста для последующего анализа.
2. Проверка ключевых слов: каждый полученный токен (одно отдельное слово) будем сравнивать с ключевыми словами из каждой предложенной темы. Все наборы ключевых слов, или же словари, представляют собой 20-25 самых часто-встречающихся слов в каждой теме. По желанию все словари можно расширить, но для достижения поставленной цели выбранного количества слов достаточно.
3. Оценка баллов для каждой темы: для каждого ключевого слова из словаря будем проверять, сколько раз оно встречается в заданном тексте. Полученные значения суммируются для каждой темы. Таким образом, на выходе мы получим структуру, в которой каждая тема имеет свой балл, отражающий, насколько текст соответствует ключевым словам этой темы. Это позволит программе определить, к какой теме текст ближе всего и, таким образом, выделить его тематику для пользователя.

## 1.2 Среда программирования

Для начала разработки программы важно выбрать подходящую среду разработки (IDE), которая предоставит удобный интерфейс и функционал. Существует множество IDE для различных языков программирования, такие как Visual Studio Code, Eclipse IDE и IntelliJ IDEA Community. Однако, мы остановимся на рассмотрении последней – IntelliJ IDEA, одной из наиболее востребованных интегрированных сред разработки (IDE) для языка Java. Наша среда программирования была разработана в 2000 году. Это инструмент для Java-разработчиков, решающий проблемы и повышающий производительность.

IntelliJ IDEA предоставляет обширный инструментарий для написания кода, включая мощный редактор, автодополнение, быструю навигацию и поддержку Maven и Gradle. Эта среда также интегрирована с системами управления зависимостями Maven и Gradle, что упрощает добавление библиотек в проекты. IntelliJ IDEA предлагает разработчикам ряд преимуществ:

1. Умные подсказки при кодировании: Функционал автодополнения кода и другие возможности, которые повышают производительность и скорость написания кода. Среда программирования выполняет глубокий анализ и создаёт виртуальную карту проект, обнаруживает ошибки и предлагает варианты их исправления, проводил валидацию кода (проверку на соответствие стандартам). Помимо этого, IntelliJ IDEA поддерживает работу со вставками, написанными на других языках программирования.
2. Встроенные инструменты для кодирования: Уже включены в состав среды разработки различные полезные инструменты, такие как терминал, профайлер, инструменты для проведения автоматических тестов и формирования аналитики, которая показывает, какой объем кода протестирован.
3. Обширные интеграции: Множество интеграций с различными внешними системами и сервисами доступны прямо из коробки.
4. Возможности рефакторинга: Разработчики могут легко менять структуру кода без боязни повлиять на существующую функциональность.
5. Эргономичность. В среде можно с легкостью настроить доступ ко всем необходимым и часто используемым функциям. Чтобы сделать программирование комфортным и удобным, в IntelliJ IDEA реализована поддержка дополнительных плагинов, сочетаний горячих клавиш.

Однако, следует отметить некоторые недостатки. IntelliJ IDEA может быть ресурсоемкой, особенно при работе с крупными проектами. Из-за большого количества инструментов и функций новым пользователям может потребоваться некоторое время, чтобы освоиться. Некоторые пользователи считают, что интерфейс может быть сложным и перегруженным. Для получения доступа ко всем функциям этого инструмента разработчика может потребоваться приобрести дорогой премиальный план, что может быть недоступно для некоторых бюджетов.

Чтобы создать новый проект в среде IntelliJ IDEA необходимо на экране запуска выбрать опцию «New Project». Для разработки на Java необходимо наличие в системе установленного пакета Java SDK (JDK), для этого нужно выбрать опцию Download JDK и в появившейся вкладке выбрать тип и версию пакета. В нашем случае вся работа будет выполнена на JDK версии 16.

1.3 Инструменты для реализации проекта

Практически все программы создаются с использованием библиотек. В программировании термин "библиотека" обозначает сборник предопределенных модулей, функций и процедур, предназначенных для повторного использования в различных программах. Библиотеки содержат реализации стандартных алгоритмов, инструменты обработки данных, интерфейсы для работы с разными устройствами и многие другие функции.

Основные причины использования библиотек в программировании:

1. Повторное использование кода: Библиотеки позволяют программистам использовать готовые решения, что экономит время и ресурсы. Вместо написания кода с нуля, разработчик может воспользоваться функциональностью, предоставленной библиотекой.
2. Уменьшение сложности разработки: Библиотеки предоставляют высокоуровневые абстракции, скрывая детали реализации сложных алгоритмов. Это упрощает процесс разработки, поскольку разработчик может использовать готовые компоненты без глубокого понимания их внутренней реализации.
3. Эффективность: Разработка библиотек может быть выполнена опытными программистами с учетом оптимизации и тестирования. Использование готовой библиотеки обеспечивает высокую степень надежности и производительности.

Apache POI (Poor Obfuscation Implementation) — это проект с открытым исходным кодом, предоставляющий библиотеки для обработки документов форматов Microsoft Office, таких как Word (.doc, .docx), Excel (.xls, .xlsx), PowerPoint (.ppt, .pptx), и других связанных форматов. Проект Apache POI ставит своей целью обеспечение разработчиков возможностью создания, изменения и чтения файлов этих форматов, что делает его полезным инструментом для работы с офисными документами в приложениях, написанных на языке Java.

В контексте достижения поставленной цели, библиотека Apache POI используется для чтения текста из документов Microsoft Word (.doc, .docx). Классы, такие как «HWPFDocument» и «XWPFDocument», предоставляют средства для работы с документами в форматах .doc и .docx соответственно, а классы WordExtractor и XWPFWordExtractor используются для извлечения текстовой информации из соответствующих форматов файлов Word.

Включенные в проект библиотеки Apache POI:

* poi-4.0.0.jar (основная библиотека Apache POI для работы с документами формата Microsoft Office).
* poi-examples-4.0.0.jar (библиотека содержит примеры использования Apache POI. Включает в себя готовые примеры кода, демонстрирующие различные возможности и способы использования библиотеки).
* poi-excelant-4.0.0.jar (Apache POI Excel Ant является дополнительным модулем, предоставляющим поддержку Apache Ant для работы с файлами Excel).
* poi-ooxml-4.0.0.jar (Эта библиотека предоставляет поддержку формата Office Open XML (OOXML), который используется в файлах Microsoft Office начиная с версии 2007. Она расширяет функциональность основной библиотеки POI для работы с OOXML-документами).
* poi-ooxml-schemas-4.0.0.jar (Вторая версия библиотеки для работы с OOXML-схемами, предоставляющая классы и методы для эффективного обработки XML-схем в файлах Office Open XML).

«Java.io» — это пакет в языке программирования Java, предоставляющий классы и интерфейсы для ввода-вывода данных (I/O). Он включает в себя множество классов для работы с файлами, потоками данных, сериализацией и другими операциями ввода-вывода. Важнейшие части пакета java.io включают в себя потоки данных(Streams), работу с файлами, сериализацию и работу с процессами.

«Map» — это структура данных, которая содержит набор пар “ключ-значение”. В «Map» каждый ключ уникален, и каждому ключу соответствует только одно значение. По своей структуре данных напоминает словарь, поэтому ее часто так и называют. В то же время, «Map» является интерфейсом, и в стандартном jdk содержит основные реализации: «Hashmap», «LinkedHashMap», «Hashtable», «TreeMap». Самая используемая реализация — «Hashmap», поэтому и будем ее использовать в нашей программе. Она представляет собой коллекцию, которая хранит данные в виде пар "ключ-значение.

Основные методы интерфейса «Map»:

put(key, value): Добавляет в карту новую пару ключ-значение. Если ключ уже существует в карте, то значение, связанное с этим ключом, обновляется.

get(Object key): Возвращает значение, связанное с указанным ключом. Если ключ не найден, возвращается null.

remove(Object key): Удаляет пару ключ-значение, связанную с указанным ключом.

keySet(): Возвращает множество всех ключей в карте.

«Scanner» — это класс в Java, который предоставляет удобные методы для чтения данных из ввода. Он считывает данные из указанного источника, распознает эту информацию и обрабатывает нужным образом. Scanner может работать с различными источниками данных, такими как строки, файлы и консоль, что делает его гибким инструментом.

Основные методы и функциональность класса «Scanner»:

Scanner(InputStream source) - создает объект Scanner для сканирования ввода из указанного потока ввода (например, System.in для консоли).

Scanner(File source) - создает объект Scanner для сканирования содержимого указанного файла.

Scanner(String source) - создает объект Scanner для сканирования содержимого указанной строки.

next() - считывает и возвращает следующее слово (до разделителя по умолчанию, который является пробелом).

hasNext() - возвращает true, если есть еще данные для сканирования.

close() - закрывает сканер, освобождая связанные ресурсы.

Глава 2. Описание программы

## 2.1 Метод determine\_theme

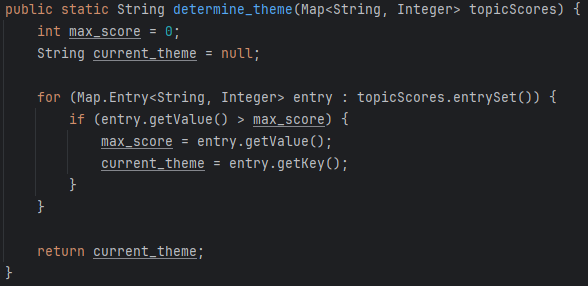


Рис. 1. Метод determine\_theme в коде.

Метод «determine\_theme» принимает карту «topicScores», где ключи представляют собой темы, а значения - оценки каждой темы на основе анализа текста. Он определяет тему текста, выбирая тему с наивысшей оценкой.

Метод проходит по каждой записи в карте «topicScores», сравнивает оценку темы с текущей максимальной оценкой. Если оценка текущей темы больше, чем текущая максимальная оценка, то обновляются переменные «max\_score», переменная для хранения максимальной оценки темы, и «current\_theme», переменная для хранения названия темы с максимальной оценкой.. После прохода по всем записям в карте, метод «determine\_theme» возвращает тему с наивысшей оценкой.

## 2.2 Метод read\_text\_from\_file



Рис. 2. Метод read\_text\_from\_file в коде.

Cоздается объект «File», представляющий файл по указанному пути. Затем, благодаря методу exists() для объекта «File» проверяется, существует ли файл. Метод «Exists()» возвращает «true», если файл или директория с указанным путем существует, и «false», если файла или директории нет. Если файл не существует, выбрасывается исключение «FileNotFoundException» с сообщением об ошибке.

Метод «get\_file\_extension» используется для получения расширения файла из его пути. В зависимости от распознанного расширения файла вызывается соответствующий метод чтения файла. Если расширение - "txt", вызывается метод «read\_text\_from\_txt\_file», «doc» - метод «read\_text\_from\_doc\_file» и «docx» - метод «read\_text\_from\_docx\_file». Если же расширение не поддерживается, выбрасывается исключение «UnsupportedOperationException».

## 2.3 Метод get\_file\_extension

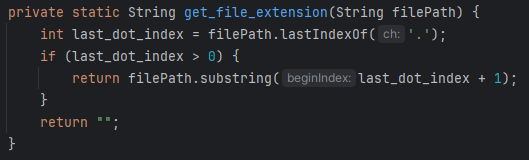


Рис. 3. Метод get\_file\_extenstion в коде.

Метод «get\_file\_extension» является вспомогательным методом, который принимает путь к файлу и возвращает его расширение. Метод «lastIndexOf» в Java применяется к строкам и массивам символов (char). Он используется для поиска последнего вхождения указанного символа или подстроки в строке или массиве символов. Метод возвращает индекс последнего вхождения символа или подстроки. В нашей программе метод «lastIndexOf» вызывается для строки «filePath» с аргументом '.'. Он возвращает индекс последнего вхождения точки в строку или -1, если точка не найдена.

Если точка найдена, и она не первый символ в строке, то метод возвращает подстроку, начиная с символа после последней точки до конца строки. Это и есть расширение файла. Если точка не найдена или она первый символ в строке, метод возвращает пустую строку. Таким образом, метод «get\_file\_extension» извлекает расширение файла из его пути, обеспечивая удобный способ получения информации о формате файла.

## 2.4 Методы read\_text\_from\_txt\_file/doc\_file/docx\_file

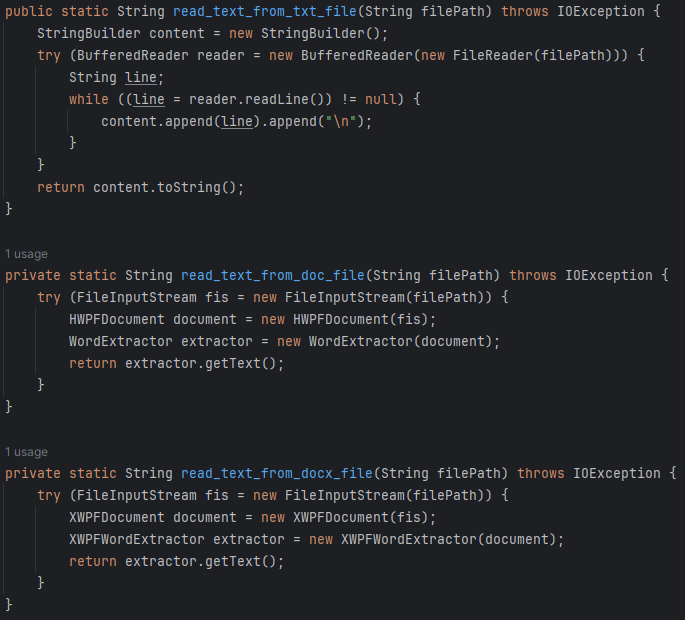


Рис. 4. Методы read\_text\_from\_txt\_file/doc\_file/docx\_file в коде.

Все три метода предназначены для чтения содержимого заданного текстового файла. Сначала рассмотрим обработку файлов с расширением .txt.

Параметр «String filePath» представляет собой путь к текстовому файлу, который необходимо прочитать. Создаем объект BufferedReader, который обеспечивает буферизированное чтение символов из файла. «FileReader» используется для чтения символов из файла в буфер. В цикле «while» считываются строки из файла с использованием «readLine()». Каждая строка добавляется к объекту «StringBuilder», который представляет собой изменяемую последовательность символов. В качестве результата возвращается текст, собранный из строк файла в виде строки.

Методы «read\_text\_from\_doc\_file» и «read\_text\_from\_docx\_file» используются для чтения текста из файлов формата .doc и .docx соответственно. Оба метода используют библиотеку Apache POI. Объект «FileInputStream» используется для создания потока ввода из файла, а затем объекты «HWPFDocument» и «XWPFDocument» представляют документы в соответствующих форматах. «WordExtractor» и «XWPFWordExtractor» служат для извлечения текста из этих документов.

## 2.5 Метод tokenize

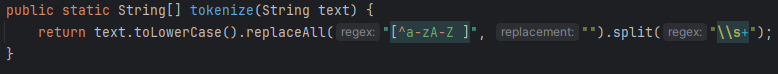


Рис. 5. Метод tokenize.

Этот метод выполняет основную операцию - токенизацию. Строка «text» приводится к нижнему регистру с помощью метода «toLowerCase()». Это делается для того, чтобы предотвратить учет регистра при последующей обработке.

С использованием регулярного выражения «[^a-zA-Z ]», удаляются все символы, кроме букв английского алфавита и пробелов. Таким образом, в строке остаются только слова. Метод «split("\\s+")» разделяет строку на массив строк, используя пробелы как разделители. Таким образом, каждое слово становится элементом массива.

Результатом работы метода является массив строк, представляющих токены (слова) текста после приведения к нижнему регистру и удаления лишних символов.

## 2.6 Метод analyze\_text

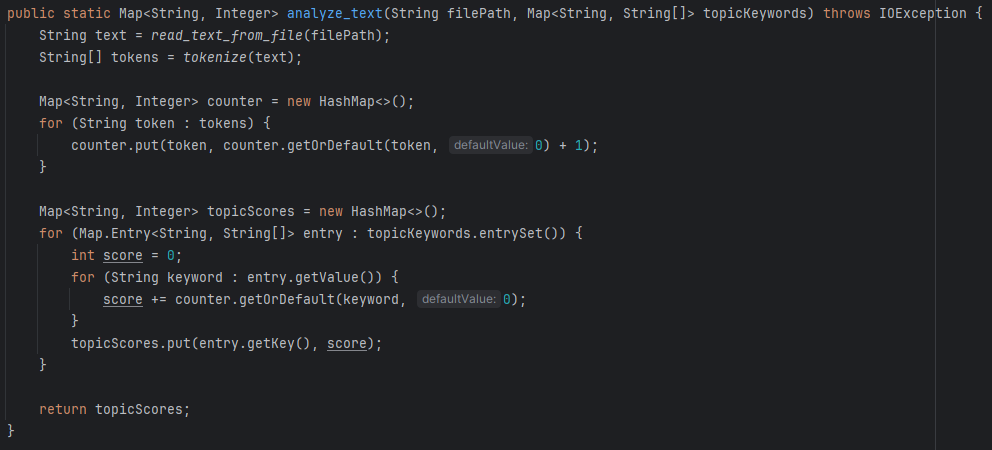


Рис. 6. Метод analyze\_text в коде.

Этот метод, ключевой в работе программы, выполняет анализ текста с использованием заданных ключевых слов для каждой темы и возвращает результат в виде словаря, где ключ - название темы, а значение - количество вхождений ключевых слов для данной темы в текст.

Текст из файла считывается с использованием метода «read\_text\_from\_file», который возвращает содержимое файла в виде строки. Полученный текст разбивается на токены (слова) с использованием метода «tokenize», создается словарь «counter», в котором каждое слово из токенов представлено в виде ключа, а соответствующее значение - количество вхождений этого слова в текст.

Для каждой темы из переданного словаря «topicKeywords» вычисляется оценка, основываясь на количестве вхождений ключевых слов для данной темы в текст. Логика обработки ключевых слов:

1. Map<String, Integer> topicScores = new HashMap<>(). Создаем карту для хранения оценок для каждой темы.
2. for (Map.Entry<String, String[]> entry : topicKeywords.entrySet()). Проходимся по каждой теме и ее ключевым словам.
3. int score = 0. Инициализируем переменную для подсчета баллов для текущей темы.
4. for (String keyword : entry.getValue()) { score += counter.getOrDefault(keyword, 0); }. Для каждого ключевого слова в теме добавляем частоту этого слова из текста к общему баллу.
5. topicScores.put(entry.getKey(), score). Сохраняем общий балл для текущей темы в карту оценок (в словаре «topicScores»).

На выходе получаем словарь, где ключ - название темы, а значение - оценка соответствующей темы на основе вхождений ключевых слов. Также, метод может выбросить исключение типа «IOException» для обработки возможных ошибок ввода-вывода при чтении текстового файла.

## 2.7 Метод main

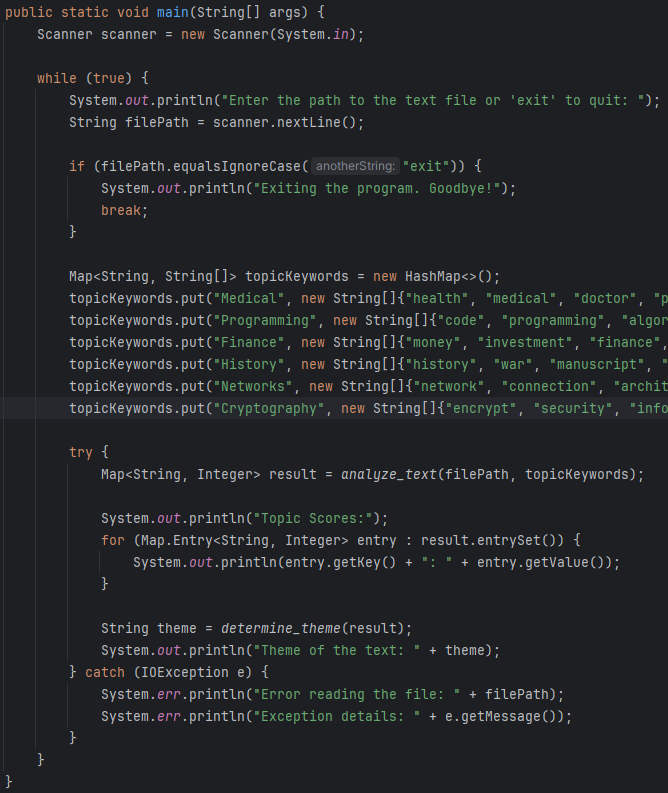


Рис. 7. Метод main в коде.

Этот метод представляет собой основную логику программы и взаимодействие с пользователем через командную строку.Первым шагом является создание объекта «Scanner» для считывания ввода пользователя из консоли.Далее инициализируется бесконечный цикл «while», который выполняется до тех пор, пока пользователь не введет команду «exit».Пользователю предлагается ввести путь к текстовому файлу. Если же введена команда «exit», программа завершает выполнение.

Получив отличный от команды «exit» ввод, программа создает карту для хранения тем и связанных с ними ключевых слов. Далее описываются словари для каждой из шести тем, которые содержат в себе начальные формы наиболее частых для каждой темы слов – это и есть наши ключевые слова, с которыми будет сравниваться каждый элемент мешка слов из рассматриваемого файла.

Вызываем метод «analyze\_text», описанный выше, проводим анализ текста и возвращаем результат в виде словаря оценок для каждой темы. Результат выводится в консоль в формате «Тема: баллы», например: Medical: 5.Если при чтении файла возникает ошибка (IOException), программа выводит сообщение, содержащее информацию об ошибке, в консоль.

Метод «main» является точкой входа в программу, где осуществляется ввод, обработка и вывод результатов работы анализатора текста.

## **2.8 Переменные в коде**

В коде программы используются следующие переменные:

* topicScores: карта, представляющая оценки тем.
* max\_score: целочисленная переменная для хранения максимальной оценки.
* current\_theme: строковая переменная для хранения текущей темы с максимальной оценкой.
* filePath: строковая переменная, представляющая путь к файлу.
* file: объект класса File, представляющий файл.
* file\_extenstion: строковая переменная, представляющая расширение файла.
* last\_dot\_index: целочисленная переменная для хранения индекса последней точки в пути.
* text: строковая переменная, представляющая текст для токенизации.
* topicKeywords: карта, представляющая ключевые слова для каждой темы.
* scanner: объект класса Scanner для ввода данных с консоли.
* result: карта, представляющая оценки тем после анализа текста.
* theme: строковая переменная, представляющая тему текста.

Глава 3. Комментарии к работе и завершающие этапы

## 3.1 Проверка программы. Создание JAR-файла

Для работы с программой у пользователя должен иметься документ, содержащий текст на английском языке. Рассмотрим пример работы программы в среде IntelliJ IDEA с текстовым документом с расширением .docx на медицинскую тему:

«The doctor assessed the patient's overall health during the routine checkup.

The patient's health improved significantly after following the doctor's advice.

The doctor prescribed a personalized health plan for the patient.

Regular checkups with the doctor are essential for maintaining good health.

The patient's positive attitude played a crucial role in their health recovery».

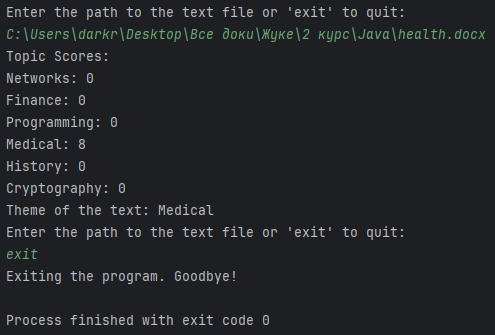


Рис. 8. Пример результата работы программы.

Данный текст содержит в себе 8 слов, которые есть в словаре на медицинскую тематику, и ни одного слова из словарей на другие темы, программа работает корректно, а результат работы программы соответствует ожидаемому результату.

Произведём сборку JAR-файла и проверим его работоспособность при запуске через консоль Windows. IntelliJ IDEA дает возможность быстро создавать исполняемый JAR-файл программы, содержащий модули со всеми зависимостями. Для сборки JAR-файла необходимо:

Настроить артефакты, перейдя в верхнее меню среды IntelliJ IDEA и выбрав опцию «File» -> "Project Structure". В открывшемся окне выбираем «Artifacts», нажимаем на кнопку «+» и выбираем «JAR» -> "From modules with dependencies". Указываем основной класс, который будет использоваться при запуске JAR-файла, в нашем случае это класс «TextAnalyzer». Применяем внесенные изменения в структуре нашего проекта и нажимаем «ОК», чтобы закрыть окно настроек артефактов.

Далее в верхнем меню выбираем опцию «Build» и в выпадающем меню выбираем опцию «Build Artifacts», где выбираем созданный нами артефакт.

После завершения сборки JAR-файла, он появляется в директории проекта, и обычно расположен в папке «out» или «target».

Необходимо проверить работоспособность собранного файла, и для этого мы воспользуемся командной строкой Windows. Запускаем систему от имени администратора и переходим в папку с JAR-файлом. Для этого в командной строке вводим «cd» (изменение каталога) и путь до папки с нашим проектом. Следующим шагом является запуск исполнения JAR-файла, и для него вводим «java -jar» и наименование JAR-файла. Наш файл успешно запустился:

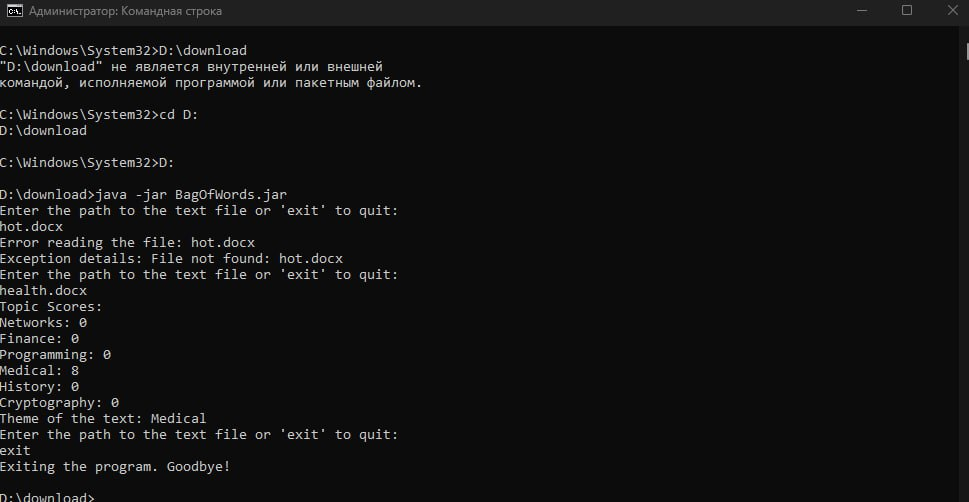


Рис. 9. Пример работы программы через командную строку Windows.

## 3.2. UML диаграмма

UML (Unified Modeling Language) диаграмма в IntelliJ IDEA — это визуальное представление структуры и поведения системы или программы с использованием стандартных графических элементов. IntelliJ IDEA предоставляет инструменты для построения и просмотра UML диаграмм, чтобы облегчить понимание и визуализацию архитектуры проекта.

При помощи UML можно описать такие структурные аспекты как:

1. Диаграмма классов (Показывает классы, их атрибуты, методы и взаимосвязи между ними).
2. Диаграмма компонентов (Описывает компоненты системы и их взаимосвязи).
3. Диаграмма объектов (Представляет конкретные объекты системы и связи между ними).
4. Поведенческие аспекты (Последовательность показывает порядок взаимодействия между объектами во времени, время описывает изменения внутреннего состояния объекта с течением времени., использование представляет, как акторы взаимодействуют с системой).
5. Архитектурные аспекты (Показывает, как компоненты системы развертываются на физическом оборудовании, группирует элементы системы в пакеты и показывает зависимости между ними).

Для создания UML диаграммы нашего кода необходимо: Открыть окно Project, перейдя в правую часть среды IntelliJ IDEA, где находится вкладка «Project». Здесь выбираем классы или пакеты, для которых мы хотим построить UML-диаграмму. Правой кнопкой мыши нажимаем на выбранном классе или пакете, в контекстном меню выбираем «Diagrams», и затем выбираем необходимый тип диаграммы. В результате получаем готовую UML-диаграмму:

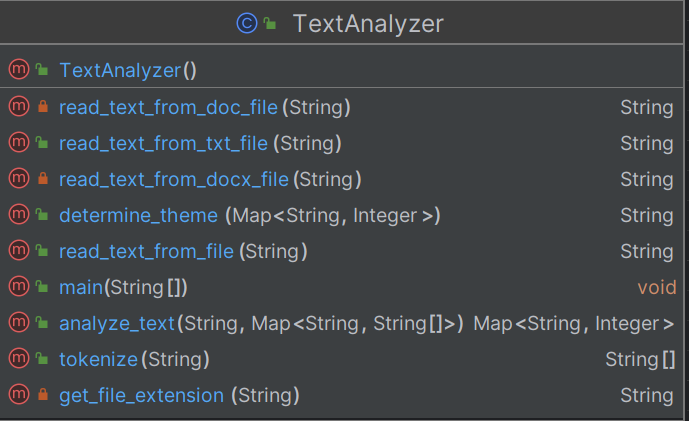


Рис. 10. UML-диаграмма.

## 3.3 Плюсы и минусы программы

Плюсы программы:

1. Многозадачность. Разработанная программа способна обрабатывать текстовые данные по запросу пользователя, позволяя многократно вводить пути к файлам и проводить анализ текстов.
2. Гибкость форматов. Программа поддерживает обработку популярных форматов текстовых файлов: .txt, .doc, .docx.
3. Простота использования. Интерфейс ввода и вывода представляет собой простой и интуитивно понятный способ взаимодействия программа. При неправильном вводе, инструмент обработки исключений укажет пользователю природу ошибки и предложит повторно провзаимодействовать с программой.
4. Расширяемость. Разработанная программа открыта для добавления и изменения тем, ключевых слов для анализа без необходимости изменять основной код.

Минусы программы:

1. Ограниченность метода анализа. Наш метод основан на простом метода анализа Bag of Words, что может привести к потере некоторой информации в тексте.
2. Отсутствие механизма лемматизации текста. После процесса токенизации, программа начинает работу с разделенными словами и не учитывает формы слов, отличные от начальной. Это связанно со сложностью механизма лемматизации лингвистическими особенностями английских слов.
3. Ограниченность по ключевым словам. Анализ основан на ключевых словах, исключая более сложные методы машинного обучения, что может уменьшить точность определения темы.
4. Ограниченность определяемых тем. Программа определяет только одну тему с наивысшим баллом, что может быть недостаточным для текстов, которые охватывают несколько тем.
5. Неудобство ввода путей. В программе нет автоматического определения путей к файлам; пользователь должен вручную вводить пути.

В целом, программа представляет собой простое средство для анализа текстовых данных, но она ограничена своим методом анализа и подходит для простых сценариев. Для более точного и гибкого анализа, возможно, потребуется использование более сложных методов и инструментов машинного обучения.

## 3.4 Обработка исключений в программе

В разработанной программе реализована обработка исключений в методах «read\_text\_from\_file» и «main» с использованием блока «try-catch».

При чтении программа может не распознать заданный файл, его расширение, или же формат файла просто не поддерживается. Если возникает ошибка, управление передается в блок «catch», где это исключение обрабатывается.

Аналогично, в методе «main» может возникнуть ошибка ввода пользователем или другие неожиданные ошибки, которые обрабатывает блок «catch» и позволяет продолжить работу программы без аварийных выходов.

## 3.5 Принципы переиспользования кода

Принципы переиспользования кода — это набор рекомендаций и стратегий, направленных на создание такого программного кода, который можно легко использовать в различных частях приложения или в разных проектах. Эти принципы способствуют уменьшению дублирования кода, повышению читаемости, облегчению сопровождения и улучшению общей структуры программного продукта.

1. DRY (Don't Repeat Yourself): принцип гласит, что каждая часть знаний или системы должна иметь единственное, безусловное представление внутри системы.
2. KISS (Keep It Simple, Stupid): Принцип простоты. Чем проще и понятнее код, тем легче его переиспользовать.
3. Composition Over Inheritance: Предпочтение композиции над наследованием. Вместо того чтобы создавать сложные иерархии классов, рекомендуется использовать композицию объектов.
4. Single Responsibility Principle (SRP): Принцип единственной ответственности. Каждый класс должен иметь только одну причину для изменения. Это способствует созданию более маленьких, независимых и переиспользуемых компонентов.
5. Open/Closed Principle (OCP): Принцип открытости/закрытости. Классы должны быть открыты для расширения, но закрыты для изменения. Это означает, что вы можете добавлять новую функциональность без изменения существующего кода.
6. Dependency Inversion Principle (DIP): Принцип инверсии зависимостей. Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба типа модулей должны зависеть от абстракций.

В нашей программе реализованы принципы SRP (Класс TextAnalyzer выполняет отдельные задачи: считывание текста из файла, токенизацию, анализ и выявление темы. Это способствует созданию модульных и независимых компонентов), OCP (Код анализа текста и определения темы написан таким образом, что его можно расширить для добавления новых тем, не изменяя существующий код. Новые темы могут быть добавлены путем расширения topicKeywords в методе main) и DIP (Метод analyzeText зависит от абстракции (Map<String, String[]>), а не от конкретной реализации. Это позволяет легко изменять или расширять набор ключевых слов для тем без изменения кода самого метода).

## 3.6 Работа с GitHub

GitHub — это веб-платформа для хостинга и совместной разработки программного обеспечения с использованием системы контроля версий Git.

Git — это распределенная система контроля версий, которая используется для отслеживания изменений в исходном коде и управления совместной разработкой программного обеспечения. Эта система позволяет разработчикам вести работу над проектами, отслеживать изменения, создавать ветки для новых функциональностей или исправлений ошибок, а также эффективно объединять изменения от нескольких участников.

На GitHub разработчики могут хранить свой код в репозиториях, вносить изменения, предлагать свои ветки (branches), создавать запросы на изменения (pull requests), следить за изменениями в коде, а также обсуждать и рецензировать код.

Для понимания работы веб-платформы GitHub, будет нелишним разобраться в её ключевых аспектах:

1. Репозитории (Repositories). Это место для хранения кода проекта. Репозиторий может быть публичным (доступным для всех) или приватным (доступным только для определенных пользователей). Он обычно содержит:

* Исходный код: Файлы, которые составляют программу или проект.
* Конфигурационные файлы: например, файлы настроек для развертывания приложения, настройки среды разработки и другие параметры.
* Документация: Описание проекта, инструкции по установке, руководства пользователя и другая документация.
* История изменений: Репозиторий отслеживает изменения, сделанные в коде и других файлах, и сохраняет их историю. Это позволяет возвращаться к предыдущим версиям, сравнивать изменения и отслеживать, кто и когда вносил изменения.

Репозитории могут храниться локально на компьютере разработчика или на удаленном сервере, таком как GitHub, GitLab или Bitbucket, где они доступны для коллаборации и обмена между членами команды. Создание репозитория позволяет упорядочить и структурировать разработку проекта, обеспечивая централизованное управление и контроль за версиями кода и других файлов.

1. Контроль версий (Version Control). GitHub использует систему контроля версий Git, что позволяет отслеживать изменения в коде, создавать ветки для экспериментов и объединять изменения в основной код.
2. Ветвление (Branching). Разработчики могут создавать ветки для изоляции изменений, проведения экспериментов и создания новых функций, а затем объединять их с основной веткой.
3. Предложение изменений (Pull Requests). Разработчики создают запрос на включение, предлагая изменения в основной код. Это позволяет провести на платформе обсуждение изменений, их рецензирование и объединение с основной веткой.
4. Проблемы и задачи (Issues). GitHub предоставляет инструментарий для отслеживания ошибок, задач и обсуждения вопросов, связанных с проектом.

Преимущества платформы GitHub:

1. Широкое распространение: GitHub широко признан и применяется в разнообразных открытых и коммерческих проектах, обеспечивая прозрачность и доступность для совместной работы.
2. Открытость и доступность: Большинство репозиториев на GitHub открыты для общего просмотра, что позволяет многим людям изучать код, предлагать изменения и активно участвовать в проектах.
3. Инструменты управления проектами: GitHub предоставляет возможности создания задач, отслеживания ошибок, обсуждения изменений в коде и другие инструменты для эффективного управления проектами.
4. Коллаборация и форкинг проектов: Пользователи могут создавать свои версии проектов (форкать) и предлагать свои изменения через pull-реквесты, что упрощает процессы совместной работы и рецензирования кода.
5. Интеграция с CI/CD: GitHub интегрируется с различными инструментами Continuous Integration (CI) и Continuous Deployment (CD), что позволяет автоматизировать процессы тестирования и развертывания.

Недостатки платформы GitHub:

1. Ограничения в использовании приватных репозиториев на GitHub: Бесплатные аккаунты GitHub сталкиваются с ограничениями по количеству приватных репозиториев и их размеру, что может оказаться недостаточным для нужд коммерческих проектов.
2. Необходимость постоянного интернет-соединения: Для работы с GitHub требуется постоянное подключение к интернету, что может стать неудобным в условиях ограниченной связи.
3. Проблемы безопасности: Поскольку многие репозитории открыты для общего просмотра, существует риск утечки конфиденциальной информации, если неадекватно управлять доступом к репозиториям.
4. Ограниченные возможности в бесплатном тарифе: Некоторые расширенные функции, такие как защищенные ветки и дополнительные инструменты управления проектами, доступны только в платных версиях GitHub.
5. Зависимость от единственного поставщика услуг: Использование одной платформы для хостинга репозиториев может создать зависимость от GitHub, что в определенных ситуациях может представлять определенные риски.

Для того, чтобы загрузить проект с нашим кодом на GitHub, необходимо загрузить его в репозиторий этой платформы. Для этого нужно выполнить следующие несложные действия:

Первым делом, необходимо зайти на сайт GitHub и создать учетную запись. Далее на главной странице GitHub необходимо нажать на кнопку «+» -> «New repository» и создать новый репозиторий, где будут храниться наши файлы.

Теперь нужно загрузить необходимые файлы в наш пустой репозиторий. Для этого на странице репозитория выбираем опцию «Upload files» и в открывшейся вкладке выбираем файл на нашем устройстве, который нужно загрузить на сайт. Подтверждаем, что нужные файлы были загружены и сохраняем изменения при помощи опции «Commit changes».

На этом загрузка нашего файла завершена, и он успешно добавлен в репозиторий платформы GitHub.

Заключение

В ходе выполнения работы по разработке анализатора текстовых данных для определения тематики текста были рассмотрены и реализованы различные аспекты обработки естественного языка с использованием языка программирования Java и среды программирования IntelliJ IDEA. Программа, разработанная на основе методов достижения поставленных задач, способна взаимодействовать с пользователем, анализировать входные текстовые данные и определять их тематику на основе метода «Мешок слов» и словарей ключевых слов.

По ходу разработки программы, были освоены и использованы основные библиотеки, такие как Apache POI для работы с файлами форматов .doc и .docx, и структуры данных, такие как Map, HashMap и Scanner, для эффективного чтения, хранения и обработки информации.

В программе были использованы принципы переиспользования кода, что способствует его читаемости и облегчает последующее развитие этого проекта, а также была реализована и освоена обработка исключение для обеспечения более корректной работы программы и взаимодействия с пользователем.

В заключение работы можно сказать, что созданный анализатор текстовых данных работает корректно и представляет собой надежный инструмент для определения тематики текста. Программа может быть дополнительно усовершенствована и расширена для более точных результатов анализа и учета специфических особенностей различных текстовых данных.

# Список использованной литературы

1. Научные Статьи.Ру. Простыми словами о методах обработки текстов и анализа данных в Natural Language Processing. URL: https://nauchniestati.ru/spravka/metody-obrabotki-tekstov-i-analiza-dannyh-v-natural-language-processing/
2. Medium. Bag of Words in Machine Learning. URL: https://medium.com/swlh/bag-of-words-in-machine-learning-13c1de579d2d
3. Хабр. Три подхода к анализу текстов. URL: https://habr.com/ru/articles/670566/
4. Server Space. Шпаргалка по CMD командам в Windows. URL: https://nauchniestati.ru/spravka/metody-obrabotki-tekstov-i-analiza-dannyh-v-natural-language-processing/
5. Uchet-jkh.ru. Как открыть jar файл с помощью Java. URL: https://uchet-jkh.ru/i/kak-otkryt-jar-fail-s-pomoshhyu-java/
6. JetBrains. Compile and build applications with IntelliJ IDEA. URL: https://www.jetbrains.com/help/idea/compiling-applications.html
7. GeekBrains. Подключение библиотек в Java. URL: https://gb.ru/blog/podklyuchenie-bibliotek-v-java/
8. David’s Blog. Apache POI A Comprehensive Guide to Interacting with Microsoft Office Files in Java. URL: https://friendlyuser.github.io/posts/tech/2023/Apache\_POI\_A\_Comprehensive\_Guide\_to\_Interacting\_with\_Microsoft\_Office\_Files\_in\_Java/
9. Medium. How to Read Microsoft Documents in Java using Apache POI. URL: https://medium.com/sstudio/read-microsoft-documents-in-java-3f9b731253d3
10. Skillbox. Библиотеки в программировании. URL: https://skillbox.ru/media/code/biblioteki\_v\_programmirovanii/
11. JavaRush. IO vs NIO. URL: https://javarush.com/quests/lectures/jru.module1.lecture41
12. JavaRush. Массивы в Java. URL: https://javarush.com/groups/posts/massivy-java
13. JavaRush. HashMap в Java. URL: https://javarush.com/quests/lectures/jru.module1.lecture41
14. JavaRush. Подробный разбор класса HashMap. URL: https://javarush.com/groups/posts/2496-podrobnihy-razbor-klassa-hashmap
15. JavaRush. Класс Scanner в Java. URL: https://javarush.com/groups/posts/klass-scanner
16. JavaRush. Исключения в Java: перехват и обработка. URL: https://javarush.com/groups/posts/1943-iskljuchenija-perekhvat-i-obrabotka
17. Хабр. Механизмы повторного исполнения кода. URL: https://javarush.com/quests/lectures/jru.module1.lecture41
18. Skillbox. Что такое GitHub и как им пользоваться. URL: https://skillbox.ru/media/code/chto-takoe-github-i-kak-im-polzovatsya/
19. Medium. Знакомство с Git и GitHub. URL: https://medium.com/nuances-of-programming/знакомство-с-git-и-github-руководство-для-начинающих-54ea2567d76c