**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 приложения по анализу и восстановлению расширения файла»

Выполнила

Морошкина Мария Сергеевна

ИБС-22

Проверил:

Старший преподаватель

Цветков Александр Юрьевич

**Оглавление**

[**1.** **Введение** 3](#_Toc153770615)

[**2.** **Основная часть** 5](#_Toc153770616)

[**2.1** **Обзор разработанного приложения** 5](#_Toc153770617)

[**2.2** **Анализ кода программы** 7](#_Toc153770618)

[**2.3** **Блок схема работы приложения** 16](#_Toc153770619)

[**2.4** **Работа программы** 20](#_Toc153770620)

[***Положительная работа приложения*** 20](#_Toc153770621)

[***Отрицательная работа приложения*** 21](#_Toc153770622)

[**2.5** **Использование библиотек** 25](#_Toc153770623)

[**2.6** **UML Диаграмма Прецедентов** 28](#_Toc153770624)

[**2.7** **UML Диаграмма Классов** 29](#_Toc153770625)

[**2.8** **Основные термины** 30](#_Toc153770626)

[**2.9** **Точность и эффективность работы программы** 32](#_Toc153770627)

[**2.10** **Безопасность программы. Производительность** 34](#_Toc153770628)

[**3.** **Выводы** 36](#_Toc153770629)

[**4.** **Источники** 37](#_Toc153770630)

[**5.** **Приложение** 40](#_Toc153770631)

1. **Введение**

Современная цифровая эра неизбежно сопряжена с созданием и использованием различных типов файлов. В процессе работы с компьютером каждый пользователь сталкивается с ситуациями, когда он не может открыть файл из-за отсутствия расширения или его повреждения. Проблема восстановления расширения файла стоит довольно остро и входит в повседневный рабочий процесс для многих людей. В данной курсовой работе предлагается разработка приложения по анализу и восстановлению расширения файла, чтобы помочь пользователям решить подобные проблемы.

**Описание проблемы:**

Существует множество ситуаций, когда пользователи сталкиваются с проблемой востановления расширения файла. Например, при скачивании файла из Интернета, его расширение может быть изменено, что приводит к тому, что операционная система не может определить правильный тип файла и открыть его. Также файл может потерять свое расширение в результате ошибки операционной системы или программного обеспечения.

**Цели и задачи для курсовой работы**

Целью данной курсовой работы является разработка приложения на языке Java, предназначенного для анализа и восстановления расширения файлов. Для достижения поставленной цели были выписаны следующие задачи:

1. Разработка приложения по анализу и восстановлению расширения файла на языке Java.

2. Создание программы, способной определить тип файла по его содержимому.

3. Разработка функционала для восстановления утерянного расширения файла.

4. Сбор информации и создание базы данных типов файлов, их расширений и "магических чисел".

5. Реализация алгоритма для анализа содержимого файлов с использованием базы данных.

6. Интеграция полученного функционала в приложение для восстановления расширения файлов.

7. Тестирование программы на корректность определения типов файлов и восстановления расширений.

8. Оформление отчета о разработанной программе и выполненных этапах работы.

**Основание закладывания логической структуры работы**

Целью создания данной логической структуры отчета является обеспечение последовательного и полного понимания процесса разработки приложения. Разработка данной структуры начинается с введения в основную проблематику и цели, а затем переходит к анализу кода программы, описанию работы приложения с приведением положительных и отрицательных примеров функционирования. Окончание работы представляет собой обзор полученных результатов. Такой подход обеспечивает четкое отслеживание всех этапов работы от зарождения идеи до реализации.

1. **Основная часть**
   1. **Обзор разработанного приложения**

**Описание функциональности приложения**

Основной функционал приложения будет включать следующие возможности:

1. **Анализ типа файла:** Пользователь сможет загрузить файл в приложение и запросить его анализ на основе содержимого. Приложение будет сканировать файл, сравнивая его содержимое с записями в базе данных, и определять его тип.

2. **Восстановление расширения:** В случае, если файл был сохранен без расширения или с неправильным расширением, приложение способно восстановить оригинальное расширение. Для этого будет использоваться информация из базы данных, где каждому типу файла будет соответствовать определенное расширение.

3. **Обработка ошибок:** При работе с файлами и базой данных будут учтены возможные ошибки и исключительные ситуации. Приложение будет обрабатывать ошибки, предоставлять сообщения об ошибках и вести записи журнала событий для удобства отладки.

В итоге, разработанное приложение предоставит пользователям возможность определять тип файлов по содержимому и восстанавливать утерянное или неправильное расширение.

**Использованные технологии и инструменты**

В рамках разработки приложения также были использованы инструменты разработки, такие как Integrated Development Environment (IDE) IntelliJ IDEA. IntelliJ IDEA обладает широким набором функций для разработки и отладки Java-приложений, предоставляя удобный интерфейс и инструменты для повышения производительности и удобства разработчика.

Java Development Kit (JDK): JDK - это основной инструмент для разработки программ на языке Java. Он содержит компилятор Java, который позволяет преобразовывать исходный код в байт-код, понятный для Java виртуальной машины (JVM). Кроме того, в состав JDK входит набор библиотек и инструментов для разработки, отладки и тестирования приложений Java.

* 1. **Анализ кода программы**

Программа состоит из трех классов: основной программы Main, класса MagicBytes и класса HasExtenshion.

**Класс MagicBytes**

В данном классе определены следующие методы и переменные:

1. Переменная signatures - это HashMap, которая используется для хранения сигнатур файлов. Ключом является тип файла (например, "gif", "jpg"), а значением - соответствующая ему сигнатура в виде шестнадцатеричного представления.

2. Блок static, который заполняет HashMap signatures сигнатурами для каждого типа файла. Каждая сигнатура представлена в виде строки из шестнадцатеричных значений.

3. Метод readSignature принимает объект FileInputStream в качестве аргумента и читает первые 8 байт из файла. Далее каждый байт преобразуется в шестнадцатеричную строку и добавляется к объекту StringBuilder signature. Между каждым байтом в сигнатуре также добавляется пробел. В конце метод возвращает сигнатуру в виде объекта StringBuilder.

4. Метод detectFileType принимает сигнатуру типа String в качестве аргумента и проверяет, содержит ли она сигнатуры расширений файлов, прописанные в HashMap signatures. Если сигнатура содержит одну из сигнатур, то метод возвращает соответствующий ей тип файла. Если же сигнатура не найдена, метод возвращает null.

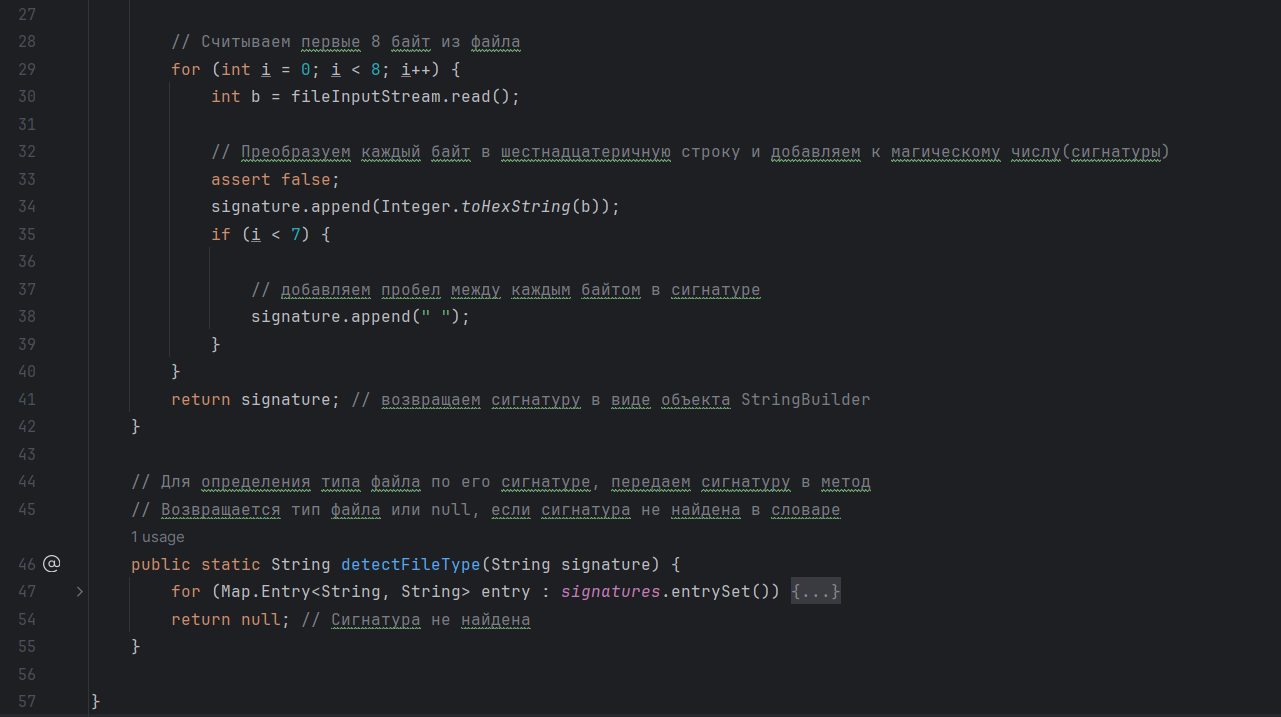
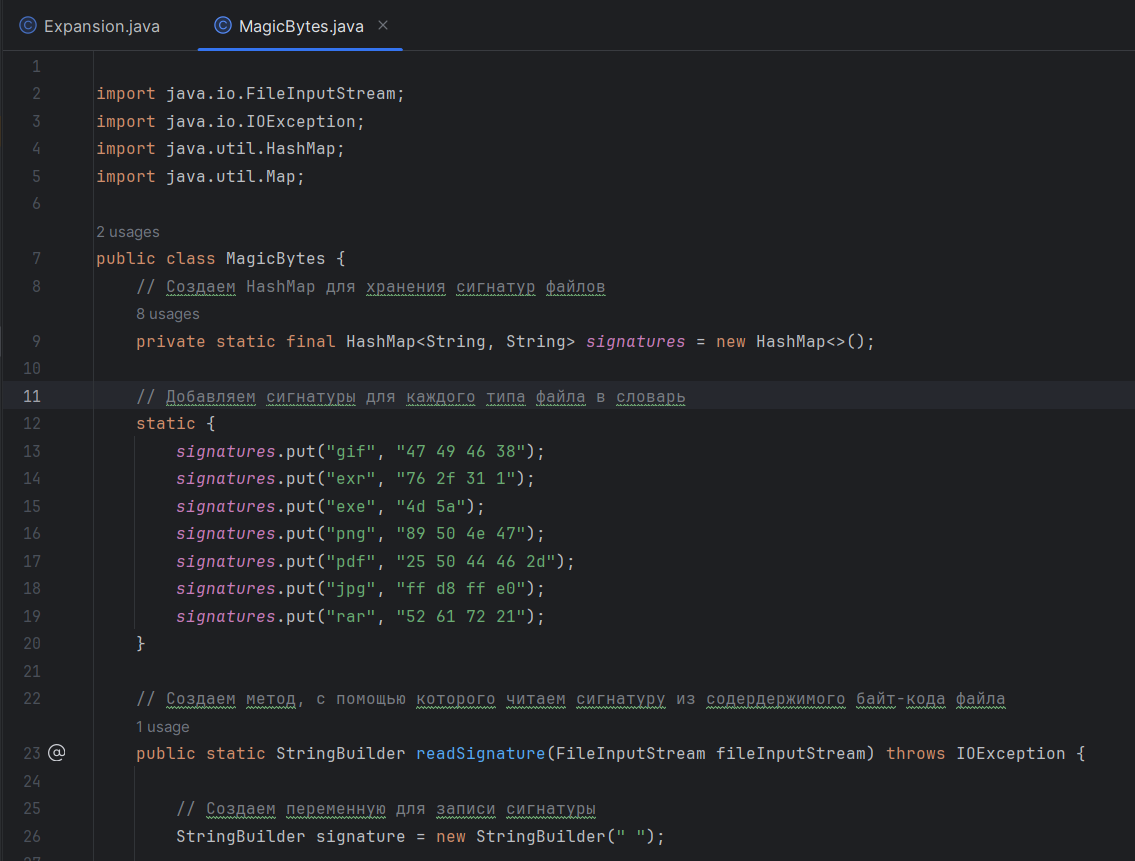


Рисунок №1. Класс MagicBytes

**Класс HasExtenshion**

Внутри него есть один статический метод "hasextension", который принимает на вход строку-имя файла и возвращает булево значение true, если у файла в названии прописано расширение, и false в противном случае.

Внутри метода происходит следующее:

1. Создается переменная "dotindex" типа int и ей присваивается значение индекса последней точки в имени файла с помощью метода "lastindexof" класса String. Этот метод возвращает индекс последнего вхождения указанного символа или подстроки в строке, если такого вхождения нет, то возвращается -1.

2. После этого выполняется проверка, что переменная "dotindex" больше 0 (то есть найдена точка в названии файла) и меньше длины имени файла минус 1 (то есть точка не является самым последним символом в имени файла).

3. Если проверка проходит успешно, то метод возвращает значение true, в противном случае возвращается значение false.

Таким образом, данный класс позволяет определить, содержит ли название файла расширение или нет.

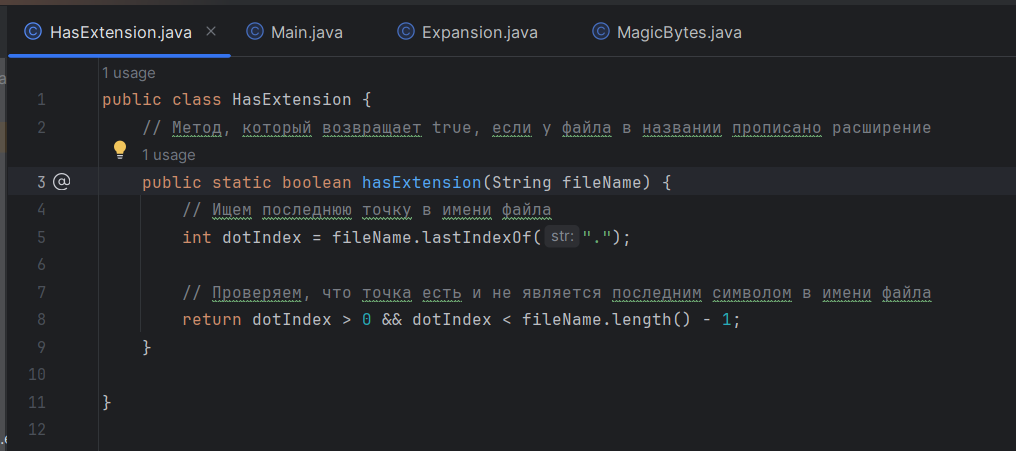
****

Рисунок №2. Класс HasExtenshion

**Класс Expansion**

Работа первого метода expansion данного класса в программе заключается в определении типа файла по его содержимому и восстановлении утерянного расширения. Программа способна распознавать следующие расширения: gif, exr, exe, png, pdf, jpg, rar.

1. Сначала программа выводит сообщение, информирующее о работе программы, а также предупреждение о необходимости использовать символы ASCII кода при вводе пути к файлу из командной строки.

2. Затем создается объект класса Scanner для чтения данных с клавиатуры, и выводится сообщение с просьбой ввести путь к файлу, для которого нужно определить расширение. Введенная строка считывается с помощью метода nextLine() и сохраняется в переменной filePath.

3. Далее создается объект класса File с указанным путем и проверяется, точно ли это путь к файлу. Если да, программа выполняет следующую проверку: существует ли файл. Если файл существует, программа открывает его для чтения с помощью класса FileInputStream. Затем считывается магическое число (сигнатура) с помощью метода readSignature и записывается в строковую переменную signature.

4. После этого файл закрывается, и с помощью метода detectFileType из класса MagicBytes определяется тип файла, используя полученную сигнатуру. Если тип файла определен, программа выводит сообщение с указанием его типа.

5. Затем программа проверяет, есть ли уже расширение у имени файла. Если расширение отсутствует, то программа получает путь к директории файла, добавляет расширение к названию файла, создает новый объект класса File для нового имени файла с тем же путем к директории и переименовывает файл с использованием метода renameTo().

6. Если переименование прошло успешно, то программа выводит сообщение об успешном переименовании файла. В противном случае выводится сообщение о неудаче.

7. Если у файла уже присутствует расширение, то выводится сообщение о том, что изменений не производилось.

8. Если тип файла не удалось определить, программа выводит сообщение с напоминанием о доступных расширениях.

9. Если файл не найден, программа выводит сообщение о проверке правильности написания названия файла.

10. Если указанный путь не указывает на файл, то выводится сообщение о том, что пользователь указал путь к директории.

Второй метод yesOrNo принимает в качестве аргумента строку continuationOfWork и возвращает значение типа boolean.

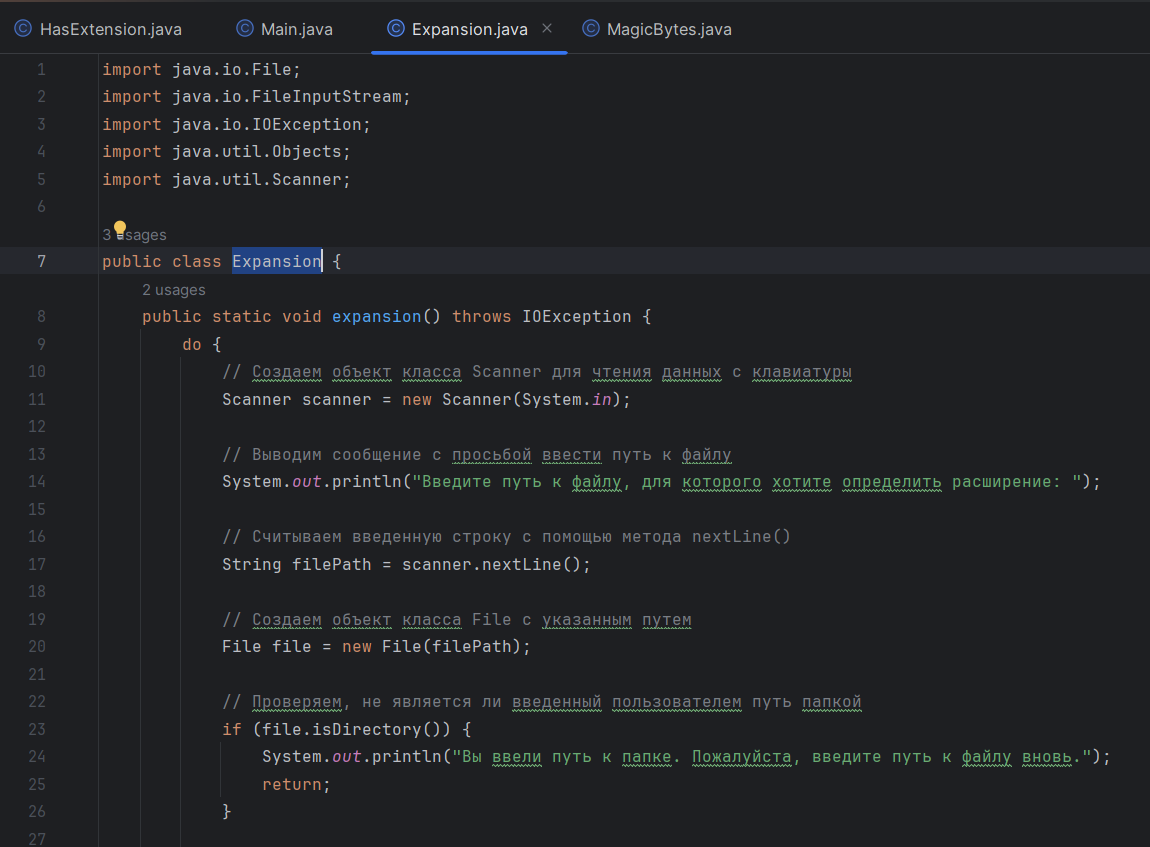
Метод выполняет следующие действия:

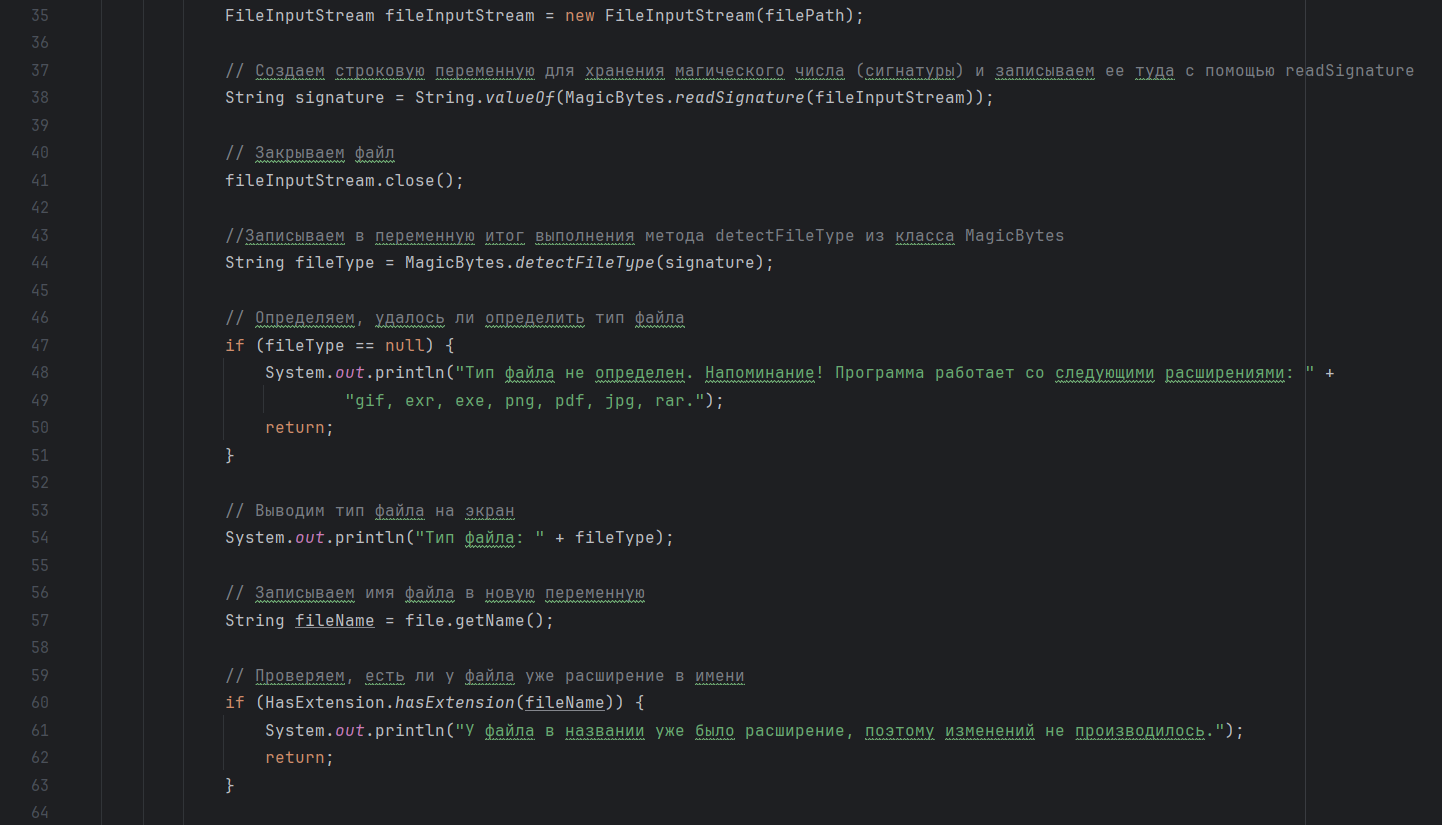
1. Проверяет, равна ли переданная строка continuationOfWork значению "Да". Если да, то вызывается метод expansion() из класса Expansion. Затем метод возвращает значение true.

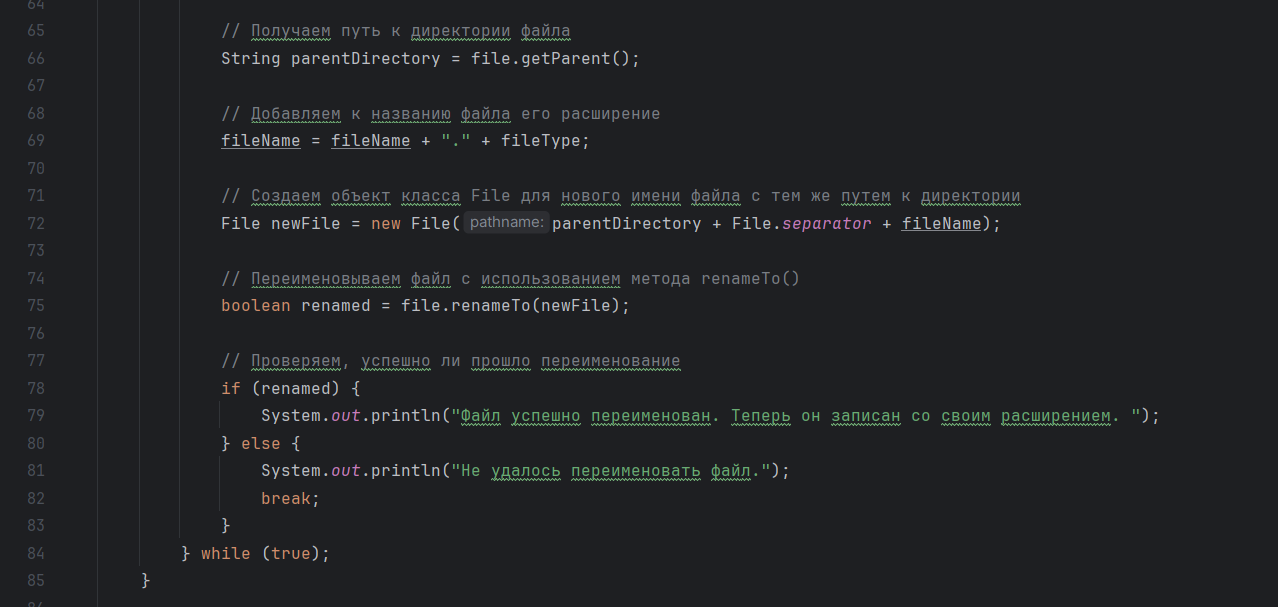
2. Если строка continuationOfWork равна "Нет", выводится сообщение о завершении работы программы, а метод возвращает значение false.

3. В противном случае, если переданная строка continuationOfWork не является ни "Да", ни "Нет", выводится сообщение о некорректном вводе, а метод возвращает значение true.

Этот метод используется для определения продолжения работы программы в зависимости от ввода пользователя. Если пользователь вводит "Да", то программа выполняет метод expansion() и продолжает работу. Если пользователь вводит "Нет", то программа выводит сообщение о завершении работы и прекращает выполнение. В случае некорректного ввода программа просит пользователя ввести значение еще раз.







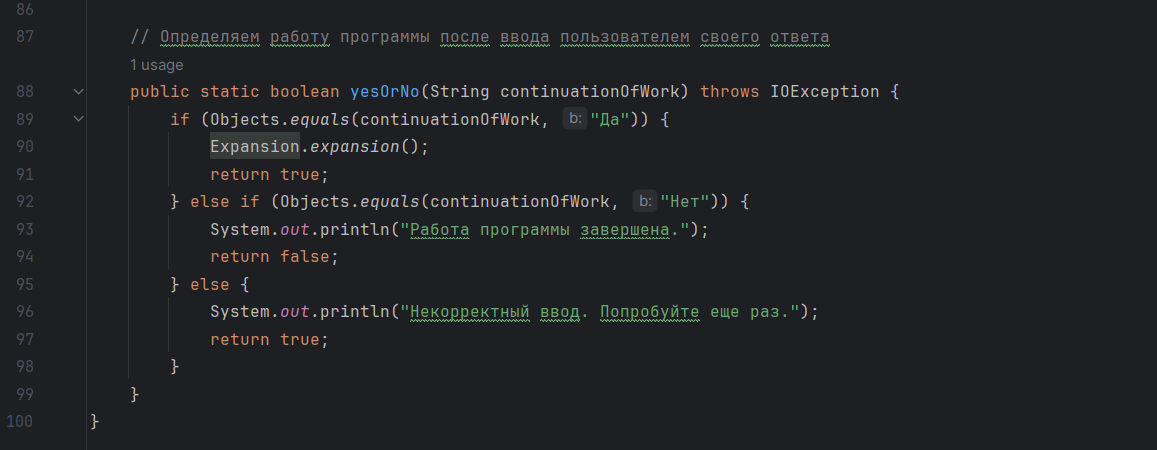


Рисунок №3. Класс Expansion

**Класс Main**

Сначала в коде импортируются два класса: "java.io.IOException" для работы с исключениями и "java.util.Scanner" для чтения данных с клавиатуры.

Затем создается класс "main", который содержит метод "main" - точку входа в программу. В объявлении метода указывается возможность генерирования исключения "IOException".

Внутри метода "main" выводится сообщение с информацией о работе программы при помощи метода "println" из класса "System.out". В данном сообщении описывается функционал программы и особенности ее использования.

Затем создается объект класса "Scanner" с именем "scanner", используя конструктор класса и передавая в него объект "System.in", который представляет стандартный ввод данных с клавиатуры.

Далее создается переменная "continuationofwork" типа "String", в которую будет записываться ответ пользователя.

Затем вызывается метод "expansion.expansion()", который выполняет основную работу программы.

Далее инициализируется переменная "answer" типа "boolean" со значением "true".

Затем используется цикл "do-while", который будет повторяться, пока значение переменной "answer" равно "true". Внутри цикла выводится вопрос пользователю о том, нужно ли запустить программу вновь, после чего считывается введенная строка с помощью метода "nextLine()" объекта "scanner". Затем с помощью метода "yesOrNo()" класса "expansion" определяется дальнейшая работа программы на основе введенного ответа пользователя.

Таким образом, программа будет работать до тех пор, пока пользователь не введет ответ "No".

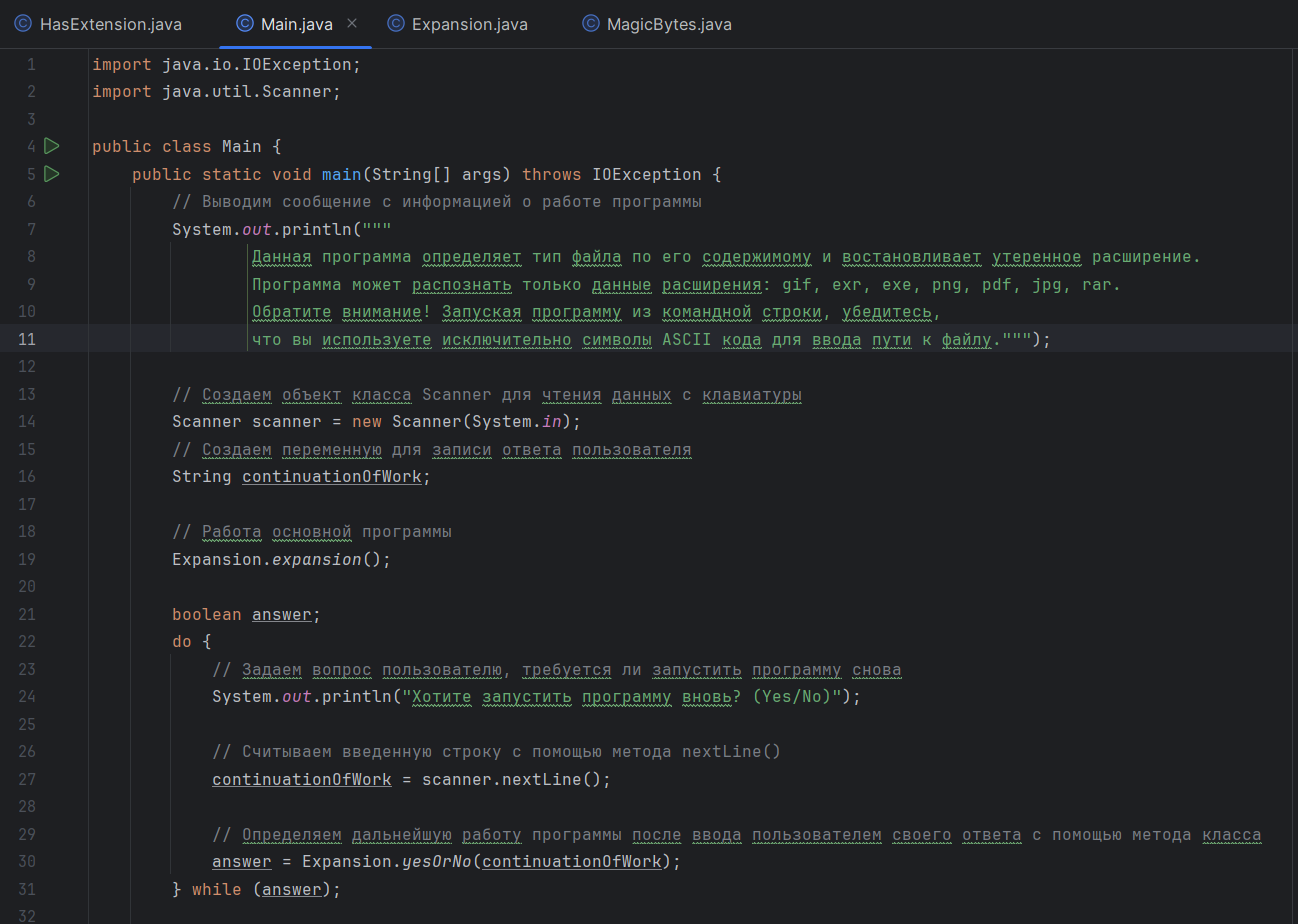


Рисунок № 4. Класс Main

* 1. **Блок схема работы приложения**

Чтобы наглядно представить, как работает приложение, необходимо разработать блок-схему алгоритма. Это позволит визуализировать поток выполнения программы и взаимодействие между ее компонентами. Блок-схема является полезным инструментом для понимания логики работы приложения и может использоваться в качестве отправной точки для дальнейших улучшений и отладки.

**Метод hasExtension класса HasExtension**

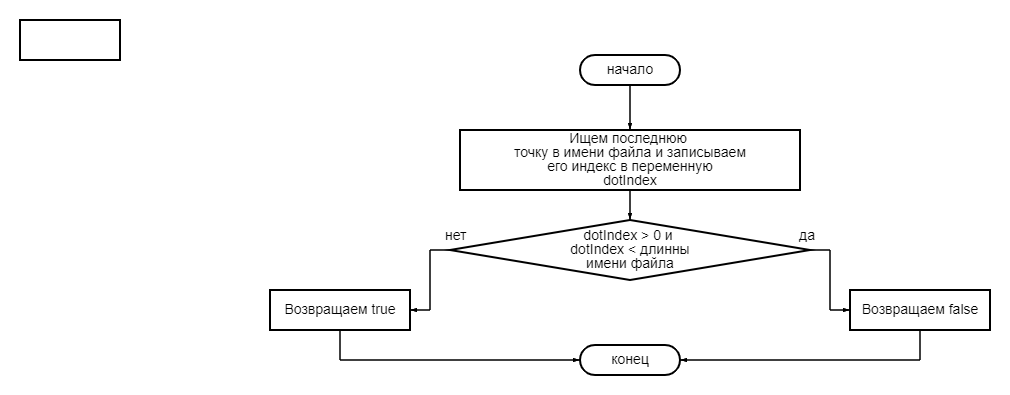


Рисунок №5. Метод hasExtension класса HasExtension

**Метод readSignature класса MagicBytes**

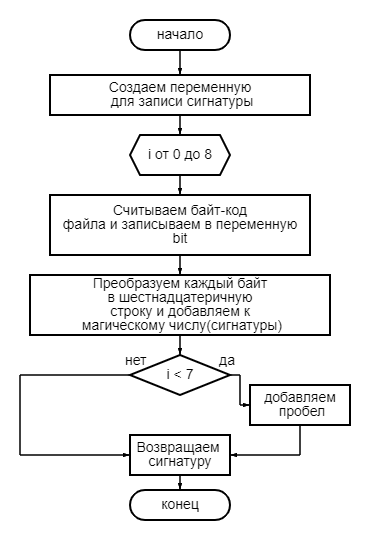


Рисунок №6. Метод readSignature класса MagicBytes

**Метод detectFileType класса MagicBytes**

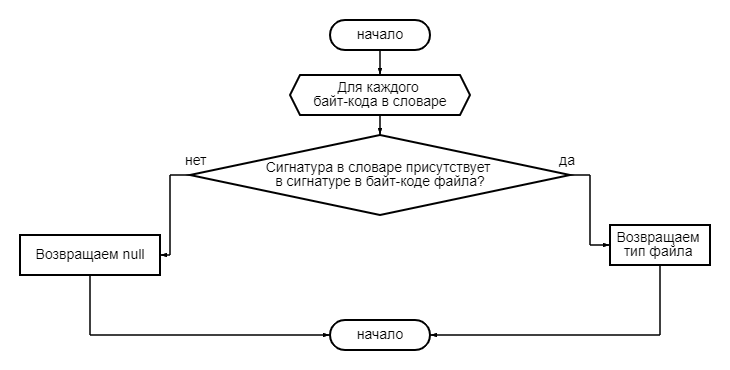
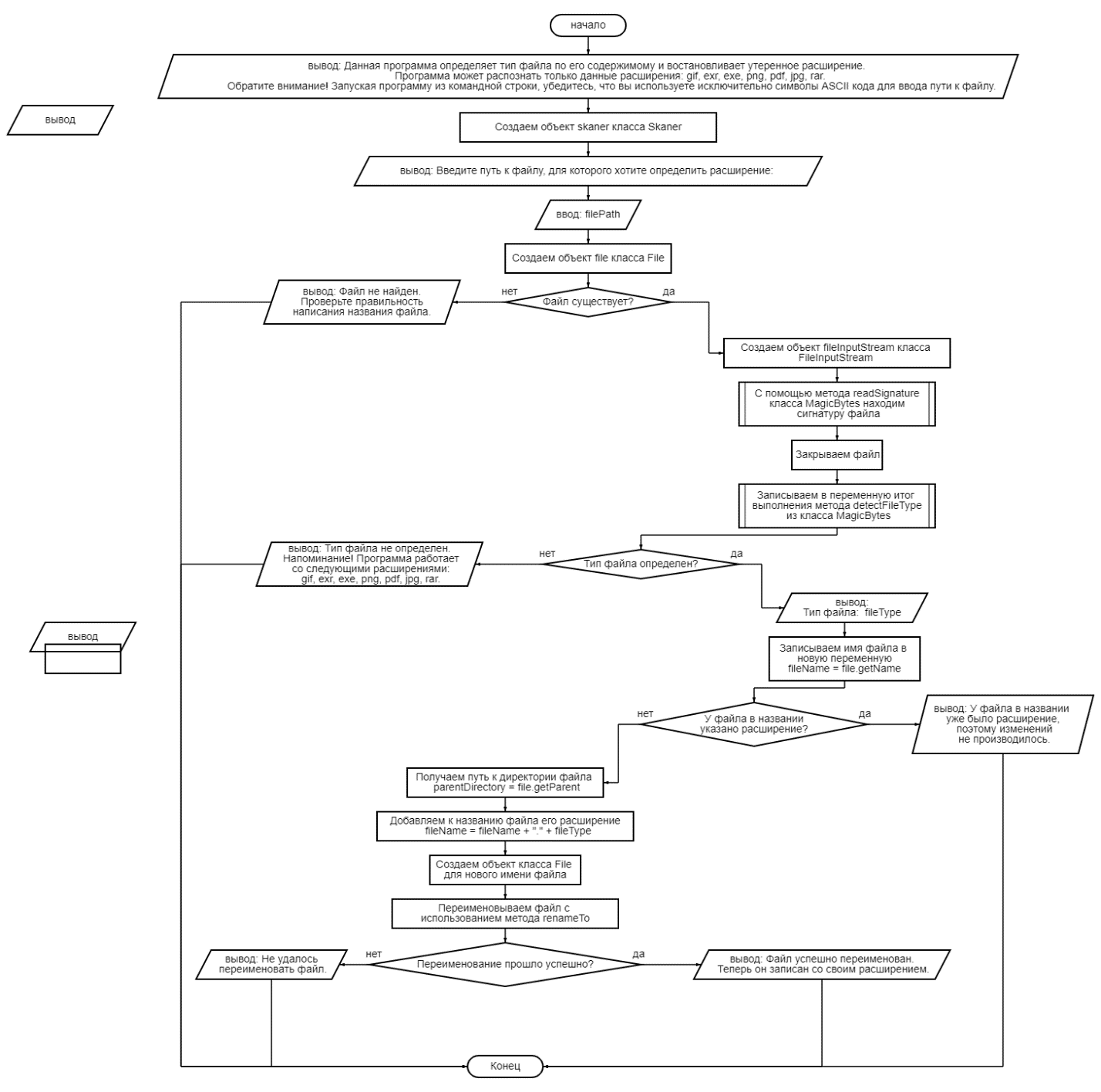
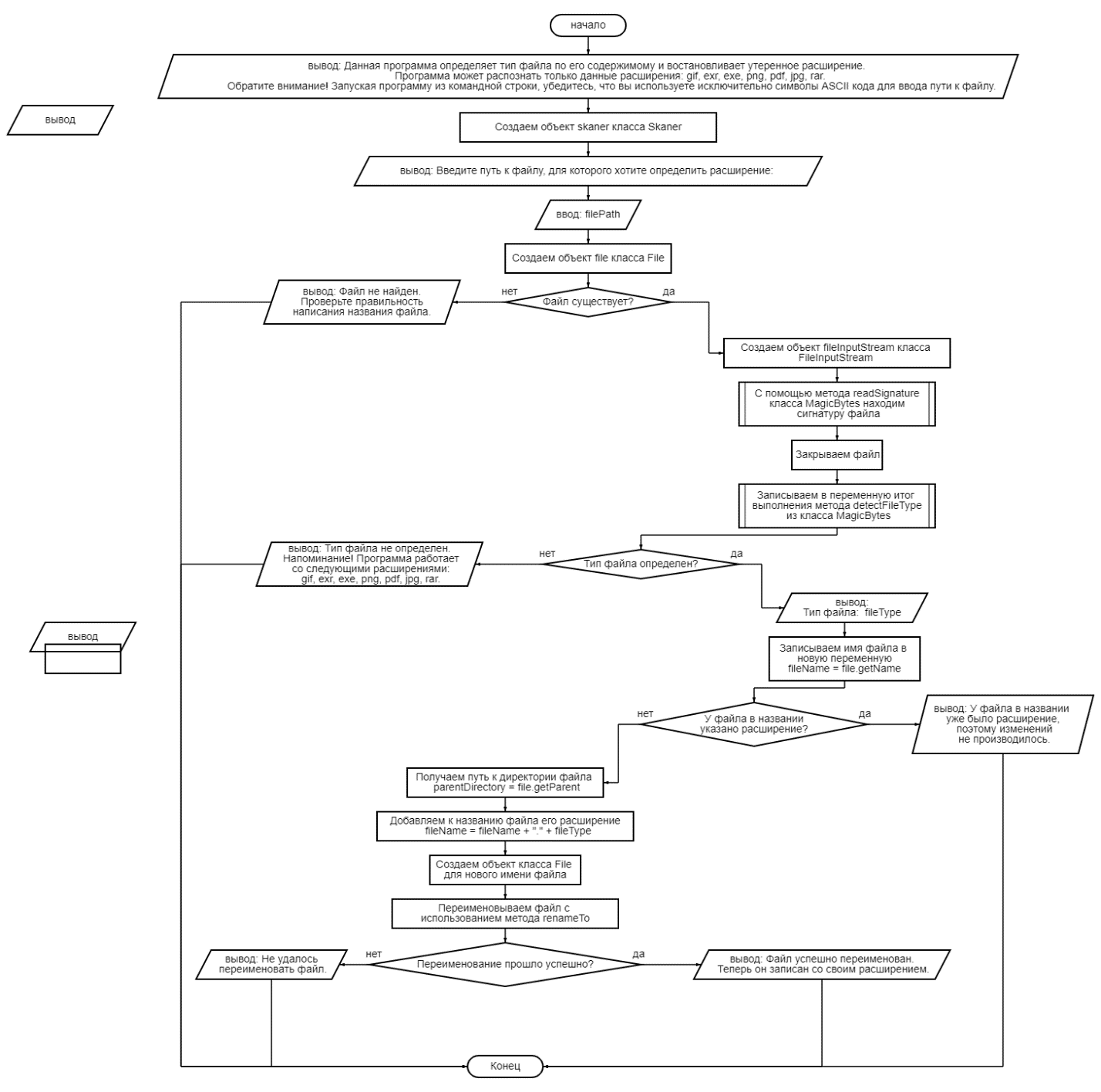


Рисунок №7. Метод detectFileType класса MagicBytes

**Класс Expansion метод expansion**



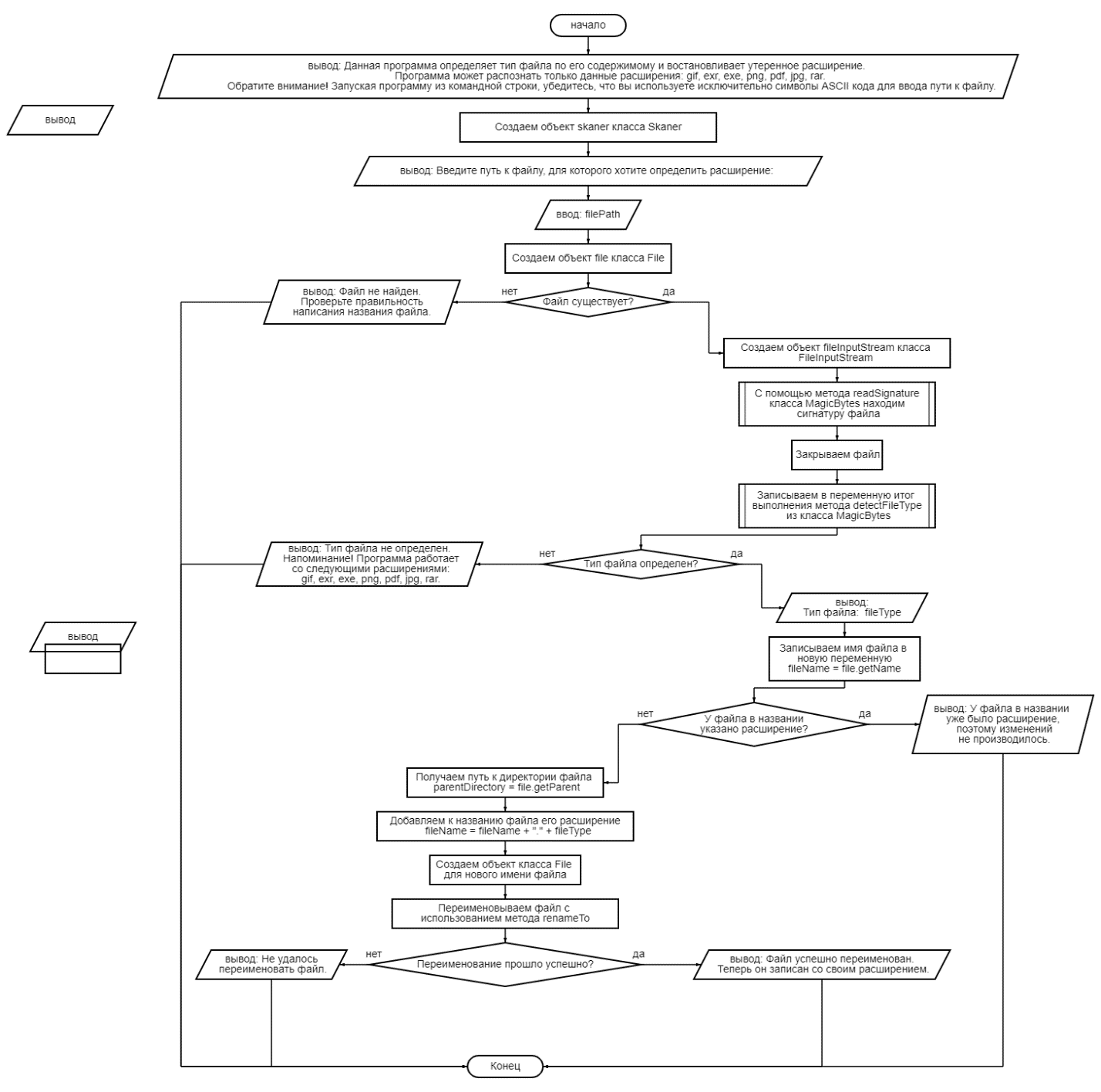
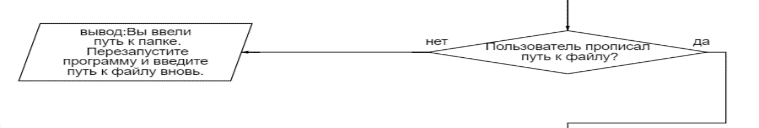
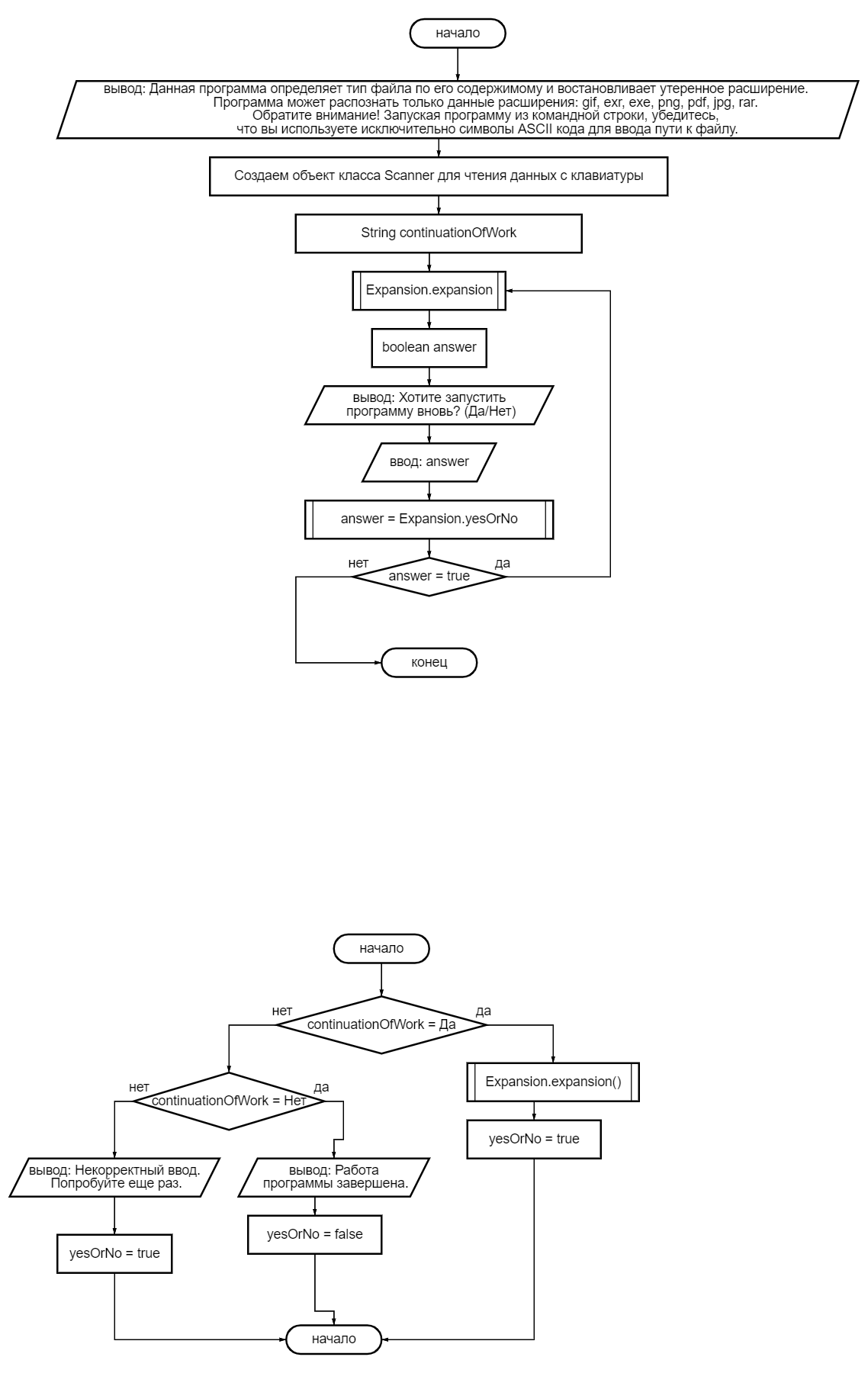


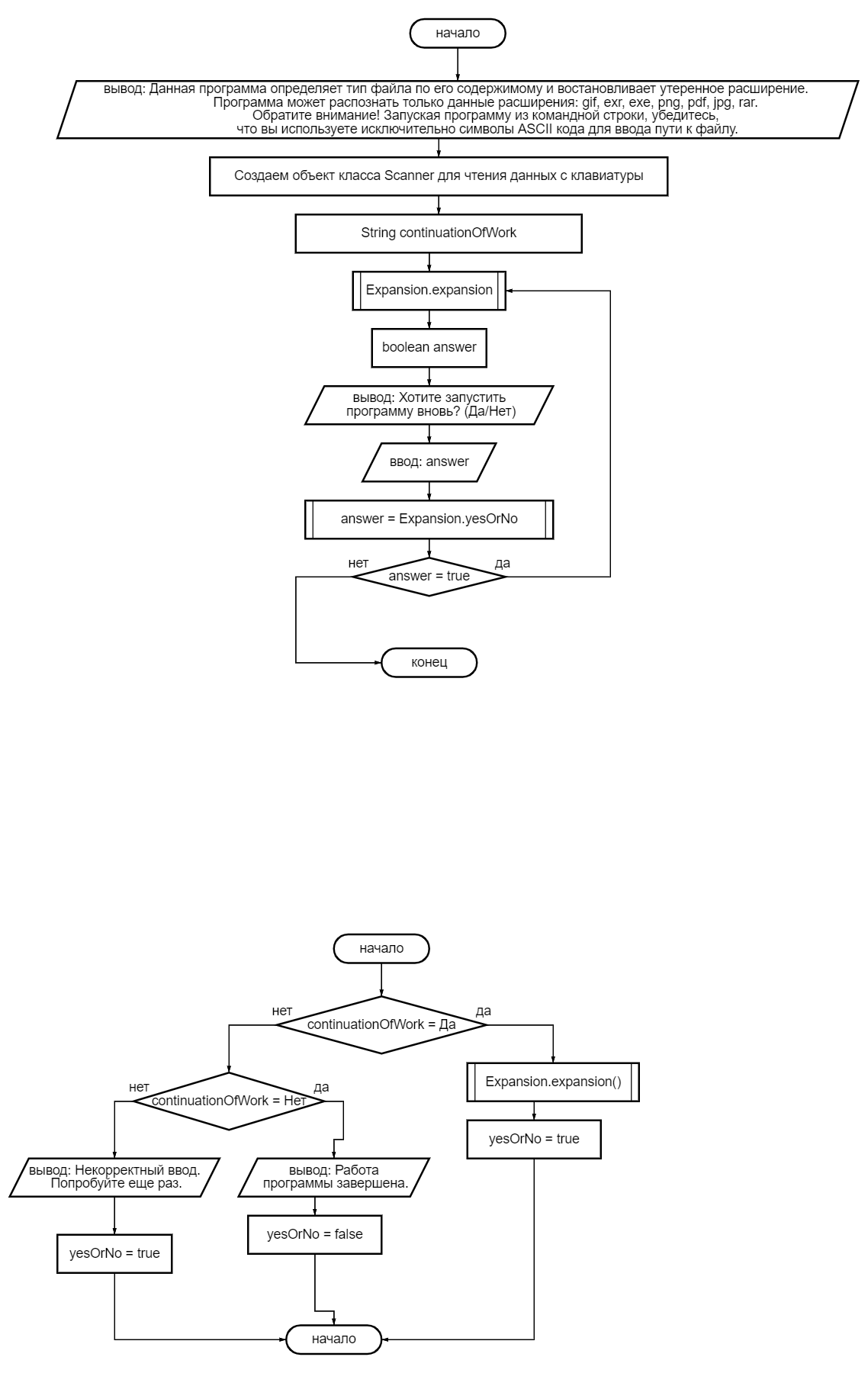
Рисунок №4. Класс Expansion

**Класс Expansion метод yesOrNo**

****

**Рисунок №5. Класс Expansion метод yesOrNo**

**Класс Main**

****

**Рисунок №6. Класс Main**

* 1. **Работа программы**

### ***Положительная работа приложения***

Пример положительной работы программы можно представить следующим образом:

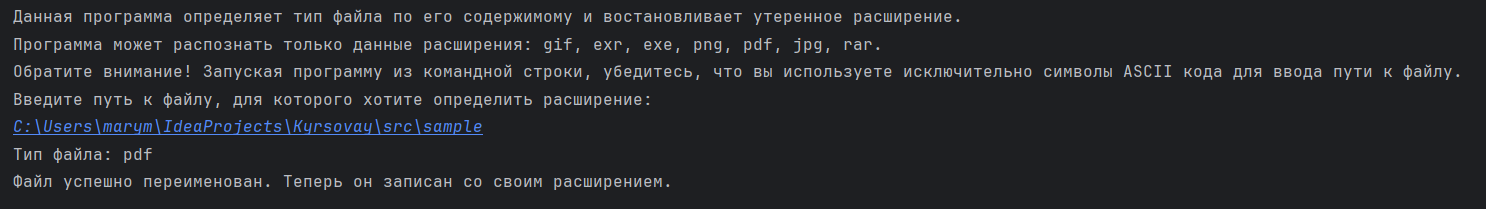
**

Рисунок №7. Работа программы. Запись нового расширения.

Комментарий:

- Пользователь вводит путь к файлу C:\Users\marym\IdeaProjects\Kyrsovay\src\sample

- Программа определяет, что тип файла - jpg.

- Имя файла " sample " не содержит расширения.

- Программа добавляет к названию файла расширение ".pdf " и переименовывает файл.

- Файл успешно переименован и теперь его имя - " sample.pdf ".

Еще один пример положительной работы приложения:

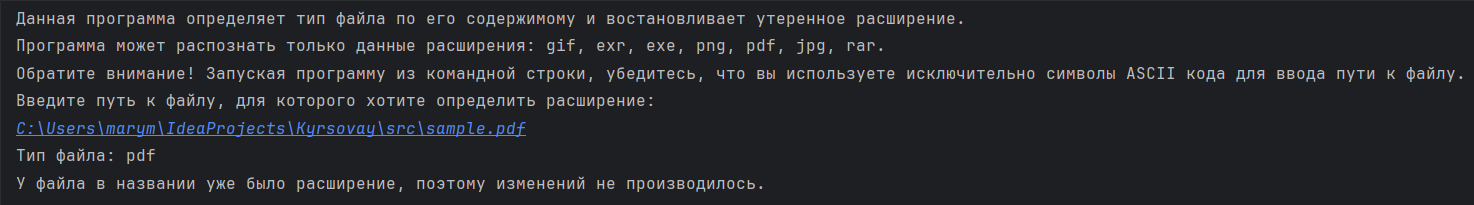


Рисунок №8. Работа программы. Определение типа. Изменения не производятся.

Комментарий:

- Пользователь вводит путь к файлу C:\Users\marym\IdeaProjects\Kyrsovay\src\sample.pdf

- Программа определяет, что тип файла - pdf.

- Имя файла " sample.pdf " содержит расширение.

- Программа выводит результат: У файла в названии уже было расширение, поэтому изменений не производилось.

### ***Отрицательная работа приложения***

Для объективной оценки работы приложения нам стоит привести пример отрицательной работы приложения.

1. **Некорректный путь к файлу**

В данном случае, так как вводится некорректный путь к файлу, программа выведет сообщение "Файл не найден. Проверьте правильность написания названия файла." и предложит ввести путь снова. Ни одно действие программы, связанное с определением типа файла, восстановлением расширения и переименованием файла, не будет выполнено.

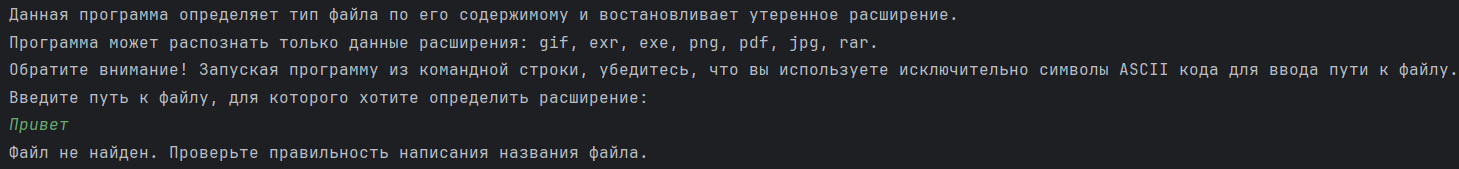


Рисунок №9. Некорректный путь к файлу.

Файл так же может быть не найден при работе через консоль. Некоторые файлы в названии имеют символы, не соответствующие ASCII коду. Пользователю в начале работы программы сообщается о том, что консоль воспринимает только данные символы. Приведем пример со знаком умножения (крестиком) в разрешении картинки в названии файла, имеющего расширение jpg: *C:\Users\marym\IdeaProjects\Kyrsovay\src\sample\_1280×853*

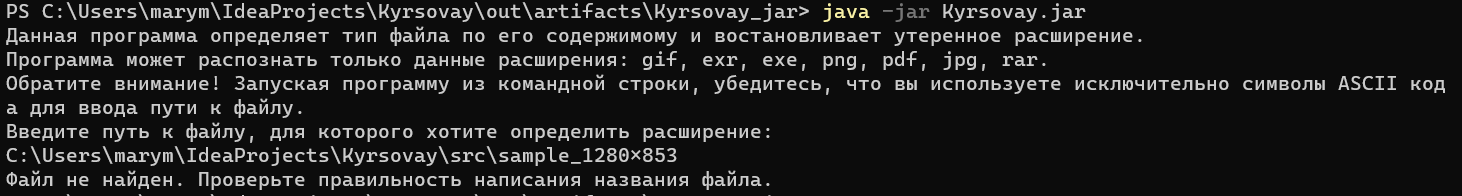


Рисунок №10. Запуск программы через консоль с символами в пути к файлу, не входящими в ASCII код.

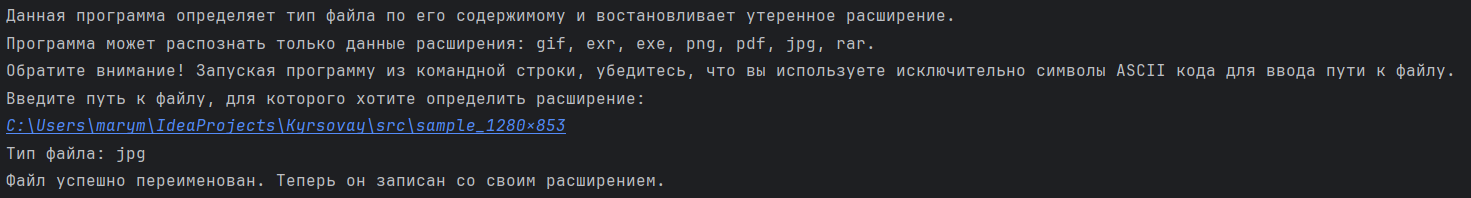


Рисунок №11. Запуск программы через IJ с тем же путем.

1. **Пользователь ввел путь к папке**

Путь, который был введен: C:\Users\marym\IdeaProjects\Kyrsovay\src\

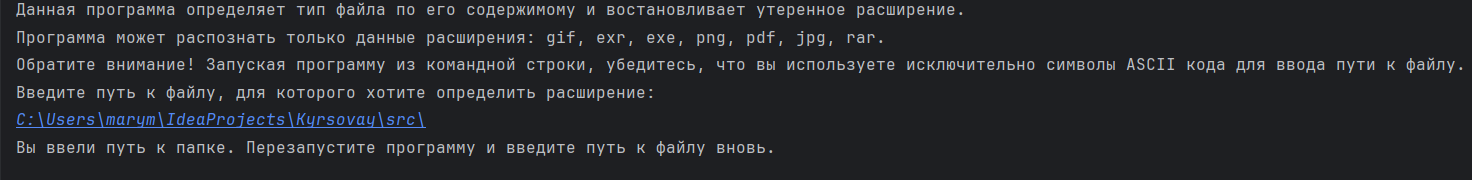


Рисунок №12. Введен путь к папке.

При вводе пути к директории, а не к файлу программа среагирует на это и выведет следующую информацию: «Вы ввели путь к папке. Перезапустите программу и введите путь к файлу вновь.» Пользователь должен ответить «Да» на предложение о перезапуске программы и ввести уже путь к файлу, для которого требуется определить расширение.

1. **Пользователь ввел файл, который имеет расширение, не прописанное в словаре программы**

Программа рассчитана для определения следующих расширений: gif, exr, exe, png, pdf, jpg, rar. Если пользователь вставит файл, для которого программа, прочитав сигнатуру, не сможет определить тип, выведется следующая информация на экран монитора:

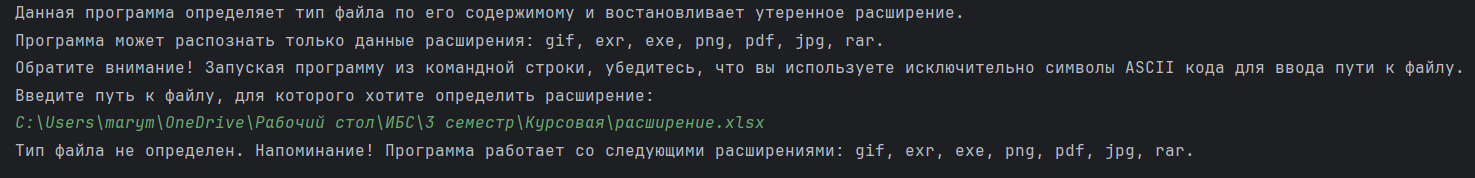


Рисунок №13. Отсутствие расширения в словаре.

В дальнейшем возможно расширение словаря. Пополнение сигнатур в данный момент в этой программе возможно только программисту, имеющему доступ к коду.

1. **Неудача при переименовании файла**

Программа не сможет переименовать файл в случае, когда файл с данным названием и расширением уже существует в указанной директории. В этом случае пользователь должен либо самостоятельно удалить файл, который был в папке до начала работы программы (уже имеющий расширение), либо убедиться в правильности написания названия файла.

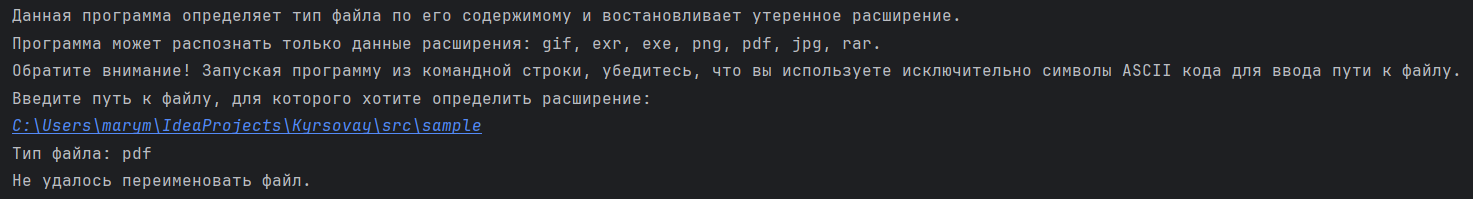


Рисунок №14. Неудача при переименовании файла.

1. **Ввод некорректного ответа на вопрос о перезапуске программы**

В случае, если пользователь некорректно отвечает на вопрос о перезапуске программы (ответ, отличный от «Yes» или «No»), программа сообщит об этом и предложит перезапустить программу вновь.

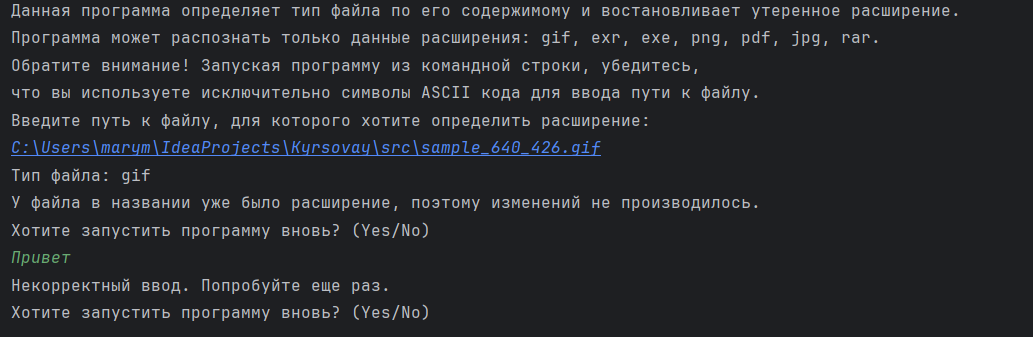


Рисунок №15. Ответ на вопрос Да \ Нет

* 1. **Использование библиотек**

**1. java.io.FileInputStream:**

- Назначение: Класс FileInputStream используется для чтения байтов из файлов на диске. Он предоставляет возможность чтения различных типов данных, таких как байты, целочисленные значения, строки и т. д.

- Эффективность: FileInputStream эффективен при чтении больших файлов, так как он работает напрямую с байтами и не выполняет излишних преобразований данных.

**2. java.io.IOException:**

- Назначение: IOException - это исключение, которое генерируется при возникновении ошибок ввода-вывода. Оно является общим для всех классов, связанных с вводом-выводом данных.

- Эффективность: IOException эффективно обрабатывает ошибки ввода-вывода и предоставляет детали о проблеме, что позволяет разработчикам легко определить и устранить ошибки.

**3. java.util.HashMap:**

- Назначение: Класс HashMap предоставляет реализацию хэш-таблицы, которая позволяет хранить данные в виде пар ключ-значение. Он обеспечивает высокую производительность при операциях добавления, удаления и доступа к данным.

- Эффективность: Время выполнения основных операций, таких как добавление, удаление и поиск элемента, в HashMap является практически постоянным, независимо от размера набора данных. Это делает HashMap очень эффективным для работы с большими объемами данных.

**4. java.util.Map:**

- Назначение: Интерфейс Map представляет собой коллекцию, которая хранит данные в виде пар ключ-значение. Он определяет методы для добавления, удаления и доступа к данным, а также для выполнения различных операций, таких как поиск, перебор и модификация элементов коллекции.

- Эффективность: Эффективность использования Map зависит от его конкретной реализации. Некоторые реализации, такие как HashMap, обеспечивают высокую производительность, особенно при работе с большими объемами данных.

**5. java.io.File:**

- Назначение: Класс File используется для работы с файлами и директориями. Он предоставляет методы для создания, удаления, переименования и получения информации о файлах и директориях.

- Эффективность: File предоставляет удобные методы для манипулирования файлами и директориями, но его эффективность зависит от операционной системы и файловой системы. В целом, он обеспечивает эффективную работу с файлами и директориями.

**6. java.util.Scanner:**

- Назначение: Класс Scanner используется для чтения данных из различных источников, таких как файлы, строки и потоки ввода. Он предоставляет методы для чтения различных типов данных, таких как строки, числа, булевые значения и т. д.

- Эффективность: Scanner является довольно эффективным при чтении и обработке ввода данных. Он обеспечивает гибкость и удобство в использовании, что делает его популярным инструментом для чтения и анализа данных из различных источников.

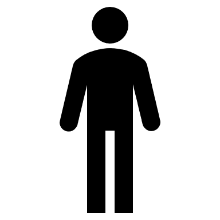
**7. java.util.Objects**

Библиотека java.util.Objects в языке Java предоставляет утилитарные методы для работы с объектами. Она содержит статические методы, которые могут быть использованы для множества операций над объектами, включая проверку наличия объекта, сравнение, хэширование и т.д.

- Назначение: Назначение библиотеки java.util.Objects заключается в предоставлении удобных инструментов для работы с объектами в различных сценариях программирования.

- Эффективность: Библиотека java.util.Objects эффективна в плане простоты использования и предоставления универсальных методов для работы с объектами. Она помогает упростить код и делает его более читаемым. Кроме того, этой библиотеки достаточно для базовых операций с объектами, что устраняет необходимость в написании дополнительного кода для повседневных задач. Это позволяет программистам сосредоточиться на более важных аспектах разработки.

* 1. **UML Диаграмма Прецедентов**



**Пользователь**

Рисунок № 16. UML Диаграмма Прецедентов

* 1. **UML Диаграмма Классов**

UML (Unified Modeling Language) - это стандартный язык для визуализации, спецификации, конструирования и документирования системных архитектур и дизайна программных систем. Одним из наиболее распространенных и полезных видов диаграмм UML является диаграмма классов.

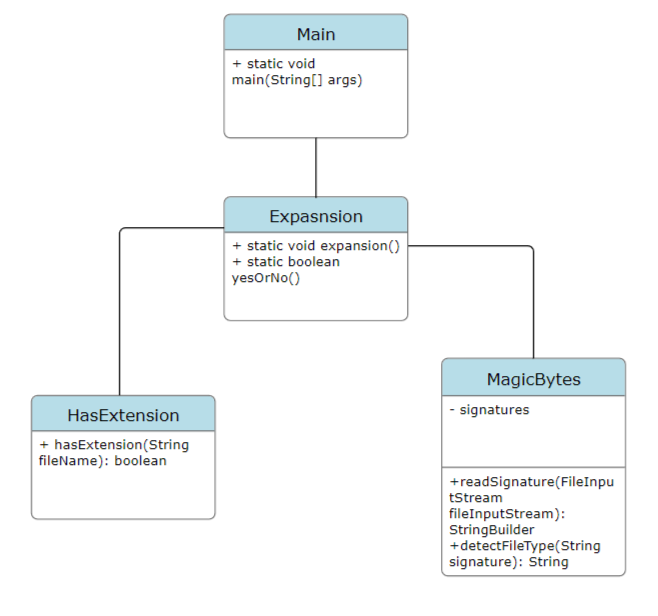
****Диаграмма классов UML отображает структуру классов в системе, их атрибуты, методы и отношения между ними. Она представляет абстрактные классы, интерфейсы и конкретные классы, а также связи наследования, ассоциации, агрегации и композиции. Диаграмма классов помогает в понимании общей структуры системы, иерархии классов и взаимодействия между ними.

Рисунок №17. UML Диаграмма Классов

* 1. **Основные термины**

**Байт-код** представляет собой промежуточный код, который создается компилятором Java при компиляции исходного кода Java. Исходный код на языке Java сначала компилируется в байт-код, который затем может быть выполнен виртуальной машиной Java (JVM).

Байт-код представляет собой последовательность байтовых инструкций, которые выполняют базовые операции и команды виртуальной машины. Каждая инструкция в байт-коде обычно занимает один или несколько байтов в файле .class. Инструкции байт-кода могут выполнять арифметические операции, загрузку и сохранение значений в переменные, вызов методов, переходы и другие операции.

Виртуальная машина Java (JVM) является интерпретатором байт-кода и выполняет байт-код, переводя его в реальные инструкции, которые выполняются процессором компьютера.

Преимуществом использования байт-кода в Java является его платформенная независимость. Байт-код генерируется компилятором для JVM, который может работать на разных операционных системах и аппаратных платформах. Это позволяет разработчикам писать код один раз и запускать его на любой платформе, где установлена JVM.

**Сигнатура** файла представляет собой уникальное значение, которое идентифицирует тип файла. Для определения типа файла можно использовать сигнатуру, которая обычно является последовательностью байтов в начале файла. Сигнатура часто называется "магическим числом" или "сигнатурой файла".

В Java для чтения и определения сигнатуры файла обычно используется класс FileInputStream. Файл открывается для чтения, и из него считывается определенное количество байтов, которые затем анализируются для определения типа файла.

Для примера, рассмотрим сигнатуру двух распространенных типов файлов - JPEG и PDF.

Сигнатура файла JPEG состоит из двух байтов: 0xFF, 0xD8. То есть, в начале файла должны находиться эти два байта для того, чтобы файл был определен как изображение в формате JPEG.

Сигнатура файла PDF состоит из четырех байтов: 0x25, 0x50, 0x44, 0x46. Если в начале файла находятся эти четыре байта, то файл определяется как документ в формате PDF.

Для определения сигнатуры файла в программе можно использовать методы класса FileInputStream или другие сторонние библиотеки, которые предоставляют функционал для анализа байтовых данных файлов.

* 1. **Точность и эффективность работы программы**

***Эффективность работы программы:***

- Эффективность работы программы может быть оценена по нескольким аспектам:

- Точность определения типа файла: программа использует сигнатуры, или "магические числа", чтобы определить тип файла по его содержимому. Если сигнатура не соответствует ожидаемой, программа может некорректно определить тип файла. Однако, для расширений, указанных в программе (gif, exr, exe, png, pdf, jpg, rar), эта вероятность ошибки будет невелика.

- Точность восстановления утерянного расширения: программа проверяет, содержит ли имя файла расширение, и если нет, добавляет соответствующее расширение. Это может быть эффективным, если пользователь случайно удалил расширение файла или оно было утеряно по другим причинам. Однако, если у файла уже есть расширение, программа не будет производить изменений.

- В общем, эффективность работы программы будет зависеть от корректности вводимой информации пользователем и соответствия файла ожидаемым сигнатурам.

***Точность работы программы:***

- Программа осуществляет запрос у пользователя о вводе пути до файла и затем пытается определить его тип по сигнатуре. Если путь указан некорректно, программа выведет соответствующее сообщение об ошибке.

- Если тип файла может быть определен, программа выводит информацию о нем. Если тип не найден в списке предопределенных расширений, программа предупреждает пользователя об этом.

- Программа также проверяет, содержит ли имя файла расширение, и если нет, добавляет соответствующее расширение. Однако, если у файла уже есть расширение, программа не изменяет его.

- После завершения работы с одним файлом, программа спрашивает пользователя, хочет ли он продолжить работу со следующим файлом, и продолжает выполнение в соответствии с выбранным ответом.

- В целом, точность работы программы будет зависеть от правильности вводимой информации пользователем и соответствия файла ожидаемым сигнатурам.

* 1. **Безопасность программы. Производительность**

Приватность в этой программе не является основным аспектом, так как она не обрабатывает или передает конфиденциальные данные. Однако, для обеспечения безопасности важно учитывать следующие аспекты:

1. *Безопасное использование файлового потока:* В данной программе используется класс FileInputStream для чтения сигнатуры файла. Однако, необходимо убедиться, что файловый поток используется безопасным образом, чтобы избежать утечки информации или взлома. Это может включать проверку разрешений на чтение файла и обработку возможных исключений.

2*. Защита словаря сигнатур*: Словарь сигнатур хранит информацию о типах файлов и их соответствующих сигнатурах. Важно обеспечить безопасное хранение и доступ к этому словарю, чтобы предотвратить его изменение или несанкционированное использование. Один из способов обеспечить безопасность словаря - сделать его приватным.

3*. Оценка безопасности операций чтения и записи*: Программа читает сигнатуры из файлов и сравнивает их с сигнатурами в словаре. Необходимо убедиться, что эти операции выполняются безопасным образом, чтобы предотвратить потенциальные атаки на файловую систему или некорректную обработку данных.

В целом, безопасность является важным аспектом в разработке современных приложений. Наличие правильных мер безопасности помогает предотвратить утечку конфиденциальной информации, защитить пользователей от вредоносного ПО и предотвратить несанкционированный доступ к системе.

В целом, производительность данной программы зависит от нескольких факторов:

1. Объем и тип обрабатываемых файлов: Если программа обрабатывает большие файлы или файлы с неоптимальным форматом, это может сказаться на производительности. Но так как программа считывает лишь первые восемь байт в сигнатуре файла, процесс не замедляется.

2. Эффективность алгоритмов и структур данных: В данной программе используется словарь сигнатур, представленный в виде HashMap. Это позволяет быстро и эффективно определить тип файла по его сигнатуре. Однако, эффективность может зависеть от количества и размера сигнатур в словаре, если в дальнейшем он будет пополняться.

3. Ввод/вывод данных: Программа выполняет чтение с клавиатуры, чтение из файла и запись в консоль. Эти операции могут занимать время, особенно при работе с большими объемами данных.

4. Наличие циклов и условных операторов: Программа содержит циклы и условные операторы для взаимодействия с пользователем и обработки данных. Оптимизация этих конструкций может повлиять на производительность программы.

Производительность данной программы зависит от различных факторов, таких как объем и тип обрабатываемых файлов, эффективность алгоритмов и структур данных, ввод/вывод данных и оптимизация циклов и условных операторов. В дальнейшем, чтобы оптимизировать производительность, можно рассмотреть возможности асинхронного выполнения операций, использование более эффективных алгоритмов и структур данных, а также оптимизацию кода с целью уменьшения накладных расходов времени выполнения.

1. **Выводы**

В ходе выполнения данной работы были успешно достигнуты все поставленные задачи и цели, и проделанная работа принесла много пользы.

В результате исследования методов и подходов к определению типа файла по его содержимому были получены необходимые знания о сигнатурах файлов и их форматах. Были изучены принципы работы сигнатур и различные типы файлов, что позволило разработать алгоритм для анализа содержимого файла и определения его типа.

Затем была создана база данных типов файлов, в которой были собраны информация о различных типах файлов, их расширениях и соответствующих магических числах. База данных стала основным инструментом для определения типа файла в приложении.

Приложение для анализа и восстановления расширения файла успешно разработано. Оно принимает на вход путь к файлу, считывает его содержимое и сравнивает магическое число с данными из базы данных. На основе этой информации приложение определяет тип файла и восстанавливает утерянное расширение. Программа была протестирована на различных типах файлов и показала высокую точность определения типа и восстановления расширения.

Работа принесла много пользы, так как она предоставляет удобный инструмент для определения типа файла и восстановления расширения в случае его утери. Это может быть полезно в различных областях, где требуется определение типа файлов и работа с ними, например, в области информационной безопасности или при разработке программного обеспечения для работы с медиафайлами.

1. **Источники**

1. Java оценивает тип загруженного файла по магическому номеру - русские блоги [Электронный ресурс] // Russian Blogs. — URL: https://russianblogs.com/article/77251198216/ (дата посещения: 2.12.2023).

2. File (Java Platform SE 8) [Электронный ресурс] // Oracle. — URL: https://docs.oracle.com/javase/8/docs/api/java/io/file.html (дата посещения: 2.12.2023).

3. HashMap (Java SE 17 & JDK 17) [Электронный ресурс] // Oracle. — URL: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/hashmap.html (дата посещения: 2.12.2023).

4. Список сигнатур файлов — Википедия [Электронный ресурс] // Wikipedia. — URL: https://ru.wikipedia.org/wiki/список\_сигнатур\_файлов (дата посещения: 2.12.2023).

5. FileInputStream (Java SE 17 & JDK 17) [Электронный ресурс] // Oracle. — URL: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/fileinputstream.html (дата посещения: 3.12.2023).

6. Расширение файла: типы файлов, таблица форматов [Электронный ресурс] // Neumeka.ru. — URL: https://neumeka.ru/tipy\_faylov.html (дата посещения: 3.12.2023).

7. List of File Formats – Wikipedia [Электронный ресурс] // Wikipedia. — URL: https://en.wikipedia.org/wiki/list\_of\_file\_formats (дата посещения: 3.12.2023).

8. Переименование файла в Java [Электронный ресурс] // Techiedelight. — URL: https://www.techiedelight.com/ru/rename-a-file-in-java/ (дата посещения: 4.12.2023).

9. Байтовый код: структура, магический номер и номер версии файла класса - русские блоги [Электронный ресурс] // Russian Blogs. — URL: https://russianblogs.com/article/83232760668/ (дата посещения: 4.12.2023).

10. ASCII — Википедия [Электронный ресурс] // Wikipedia. — URL: https://ru.wikipedia.org/wiki/ascii (дата посещения: 4.12.2023).

11. Байт-код — Википедия [Электронный ресурс] // Wikipedia. — URL: https://ru.wikipedia.org/wiki/байт-код#java (дата посещения: 5.12.2023).

12. Magic Number (Programming) – Wikipedia [Электронный ресурс] // Wikipedia. — URL: https://en.wikipedia.org/wiki/magic\_number\_(programming) (дата посещения: 5.12.2023).

13. Path (Java SE 17 & JDK 17) [Электронный ресурс] // Oracle. — URL: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/path.html (дата посещения: 5.12.2023).

14. Блок-схемы алгоритмов. ГОСТ. Примеры — Блог программиста [Электронный ресурс] // Pro-Prof. — URL: https://pro-prof.com/archives/1462 (дата посещения: 5.12.2023).

15. Диаграмма классов — Википедия [Электронный ресурс] // Wikipedia. — URL: https://ru.wikipedia.org/wiki/диаграмма\_классов (дата посещения: 6.12.2023).

16. IntelliJ IDEA: стиль и форматирование кода [Электронный ресурс] // JavaRush. — URL: https://javarush.com/groups/posts/2009-intellij-idea-stiljh-i-formatirovanie-koda (дата посещения: 6.12.2023).

17. Уроки по Java [Электронный ресурс] // YouTube.com : [сайт]. — URL: https://www.youtube.com/playlist?list=pl786bpilqejrdxpakybzpdtaoyswyjtcx (дата посещения: 11.12.2023).

18. Валерий Яценков. Java за неделю [Книга] / Валерий Яценков. — ISBN 978-5-4490-4684-0. — Издательство: Ридеро, 2018. — 367 с.

1. **Приложение**

Листинг программы

import java.io.IOException;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) throws IOException {  
 // Выводим сообщение с информацией о работе программы  
 System.*out*.println("""  
 Данная программа определяет тип файла по его содержимому и востановливает утеренное расширение.  
 Программа может распознать только данные расширения: gif, exr, exe, png, pdf, jpg, rar.  
 Обратите внимание! Запуская программу из командной строки, убедитесь,  
 что вы используете исключительно символы ASCII кода для ввода пути к файлу.""");  
  
 // Создаем объект класса Scanner для чтения данных с клавиатуры  
 Scanner scanner = new Scanner(System.*in*);  
 // Создаем переменную для записи ответа пользователя  
 String continuationOfWork;  
  
 // Работа основной программы  
 Expansion.*expansion*();  
  
 boolean answer;  
 do {  
 // Задаем вопрос пользователю, требуется ли запустить программу снова  
 System.*out*.println("Хотите запустить программу вновь? (Да/Нет)");  
  
 // Считываем введенную строку с помощью метода nextLine()  
 continuationOfWork = scanner.nextLine();  
  
 // Определяем дальнейшую работу программы после ввода пользователем своего ответа с помощью метода класса  
 answer = Expansion.*yesOrNo*(continuationOfWork);  
 } while (answer);  
  
 }  
  
}

import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Objects;  
import java.util.Scanner;  
  
public class Expansion {  
 public static void expansion() throws IOException {  
 do {  
 // Создаем объект класса Scanner для чтения данных с клавиатуры  
 Scanner scanner = new Scanner(System.*in*);  
  
 // Выводим сообщение с просьбой ввести путь к файлу  
 System.*out*.println("Введите путь к файлу, для которого хотите определить расширение: ");  
  
 // Считываем введенную строку с помощью метода nextLine()  
 String filePath = scanner.nextLine();  
  
 // Создаем объект класса File с указанным путем  
 File file = new File(filePath);  
  
 // Проверяем, не является ли введенный пользователем путь папкой  
 if (file.isDirectory()) {  
 System.*out*.println("Вы ввели путь к папке. Пожалуйста, введите путь к файлу вновь.");  
 return;  
 }  
  
 // Убеждаемся, что файл существует  
 if (!file.exists()) {  
 System.*out*.println("Файл не найден. Проверьте правильность написания названия файла.");  
 return;  
 }  
  
 // Открываем файл для чтения  
 FileInputStream fileInputStream = new FileInputStream(filePath);  
  
 // Создаем строковую переменную для хранения магического числа (сигнатуры) и записываем ее туда с помощью readSignature  
 String signature = String.*valueOf*(MagicBytes.*readSignature*(fileInputStream));  
  
 // Закрываем файл  
 fileInputStream.close();  
  
 //Записываем в переменную итог выполнения метода detectFileType из класса MagicBytes  
 String fileType = MagicBytes.*detectFileType*(signature);  
  
 // Определяем, удалось ли определить тип файла  
 if (fileType == null) {  
 System.*out*.println("Тип файла не определен. Напоминание! Программа работает со следующими расширениями: " +  
 "gif, exr, exe, png, pdf, jpg, rar.");  
 return;  
 }  
  
 // Выводим тип файла на экран  
 System.*out*.println("Тип файла: " + fileType);  
  
 // Записываем имя файла в новую переменную  
 String fileName = file.getName();  
  
 // Проверяем, есть ли у файла уже расширение в имени  
 if (HasExtension.*hasExtension*(fileName)) {  
 System.*out*.println("У файла в названии уже было расширение, поэтому изменений не производилось.");  
 return;  
 }  
  
 // Получаем путь к директории файла  
 String parentDirectory = file.getParent();  
  
 // Добавляем к названию файла его расширение  
 fileName = fileName + "." + fileType;  
  
 // Создаем объект класса File для нового имени файла с тем же путем к директории  
 File newFile = new File(parentDirectory + File.*separator* + fileName);  
  
 // Переименовываем файл с использованием метода renameTo()  
 boolean renamed = file.renameTo(newFile);  
  
 // Проверяем, успешно ли прошло переименование  
 if (renamed) {  
 System.*out*.println("Файл успешно переименован. Теперь он записан со своим расширением. ");  
 } else {  
 System.*out*.println("Не удалось переименовать файл.");  
 break;  
 }  
 } while (true);  
 }  
  
 // Определяем работу программы после ввода пользователем своего ответа  
 public static boolean yesOrNo(String continuationOfWork) throws IOException {  
 if (Objects.*equals*(continuationOfWork, "Да")) {  
 Expansion.*expansion*();  
 return true;  
 } else if (Objects.*equals*(continuationOfWork, "Нет")) {  
 System.*out*.println("Работа программы завершена.");  
 return false;  
 } else {  
 System.*out*.println("Некорректный ввод. Попробуйте еще раз.");  
 return true;  
 }  
 }  
}

import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.HashMap;  
import java.util.Map;  
  
public class MagicBytes {  
 // Создаем HashMap для хранения сигнатур файлов  
 private static final HashMap<String, String> *signatures* = new HashMap<>();  
  
 // Добавляем сигнатуры для каждого типа файла в словарь  
 static {  
 *signatures*.put("gif", "47 49 46 38");  
 *signatures*.put("exr", "76 2f 31 1");  
 *signatures*.put("exe", "4d 5a");  
 *signatures*.put("png", "89 50 4e 47");  
 *signatures*.put("pdf", "25 50 44 46 2d");  
 *signatures*.put("jpg", "ff d8 ff e0");  
 *signatures*.put("rar", "52 61 72 21");  
 }  
  
 // Создаем метод, с помощью которого читаем сигнатуру из содердержимого байт-кода файла  
 public static StringBuilder readSignature(FileInputStream fileInputStream) throws IOException {  
  
 // Создаем переменную для записи сигнатуры  
 StringBuilder signature = new StringBuilder(" ");  
  
 // Считываем первые 8 байт из файла  
 for (int i = 0; i < 8; i++) {  
 int bit = fileInputStream.read();  
  
 // Преобразуем каждый байт в шестнадцатеричную строку и добавляем к магическому числу(сигнатуры)  
 assert false;  
 signature.append(Integer.*toHexString*(bit));  
 if (i < 7) {  
  
 // добавляем пробел между каждым байтом в сигнатуре  
 signature.append(" ");  
 }  
 }  
 return signature; // возвращаем сигнатуру в виде объекта StringBuilder  
 }  
  
 // Для определения типа файла по его сигнатуре, передаем сигнатуру в метод  
 // Возвращается тип файла или null, если сигнатура не найдена в словаре  
 public static String detectFileType(String signature) {  
 for (Map.Entry<String, String> entry : *signatures*.entrySet()) {  
 // Проверяем, есть ли в сигнатуре нашего файла сигнатура расширений, прописанных в словаре  
 if (signature.contains(entry.getValue())) {  
  
 return entry.getKey(); // Возвращаем тип файла  
 }  
 }  
 return null; // Сигнатура не найдена  
 }  
  
}

public class HasExtension {  
 // Метод, который возвращает true, если у файла в названии прописано расширение  
 public static boolean hasExtension(String fileName) {  
 // Ищем последнюю точку в имени файла  
 int dotIndex = fileName.lastIndexOf(".");  
  
 // Проверяем, что точка есть и не является последним символом в имени файла  
 return dotIndex > 0 && dotIndex < fileName.length() - 1;  
 }  
  
}

**Ссылка на код (GitHub):**

https://github.com/MashaMoroshkina/Kyrsovay\_rabota.git