**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 файловой системы»

Выполнил:

Студент группы ИБС-22

Маркович Тимофей Максимович

Проверил:

Старший преподаватель

Цветков Александр Юрьевич

**Оглавление**

[1. Введение 2](#_Toc153205175)

[2. Основная часть 3](#_Toc153205176)

[2.1 Обзор разработанного приложения 3](#_Toc153205177)

[2.2 Анализ кода программы 5](#_Toc153205178)

[2.3 Работа приложения 10](#_Toc153205179)

[2.4 Блок-схема алгоритма приложения 15](#_Toc153205180)

[2.5 Использование библиотек 16](#_Toc153205181)

[2.6 UML Диаграмма Классов 19](#_Toc153205182)

[2.7 UML Диаграмма Прецедентов 20](#_Toc153205183)

[2.8 Основные Концепции и Определения 21](#_Toc153205184)

[2.9 Безопасность и надежность программы 24](#_Toc153205185)

[2.10 Анализ производительности 27](#_Toc153205186)

[2.11 Часто Задаваемые Вопросы (FAQ) 30](#_Toc153205187)

[3. Заключение 32](#_Toc153205188)

[4. Список источников 35](#_Toc153205189)

[5. Приложение 36](#_Toc153205190)

# Введение

**Исходная проблема и ее актуальность**

В современном мире, где технологии играют ключевую роль во многих аспектах нашей жизни, разработка программного обеспечения является одним из самых важных и актуальных вопросов. Одним из популярных языков программирования, используемых для создания различных типов приложений, является Java. В данном курсовом проекте мы рассмотрим разработку приложения на языке Java, предназначенного для анализа файловой системы.

Анализ файловой системы является неотъемлемой частью работы многих приложений и сервисов, поскольку позволяет получить подробную информацию о содержимом дисков и папок. Это может быть полезно для различных целей, таких как оптимизация использования дискового пространства, поиск и удаление дубликатов файлов, а также мониторинг состояния файловой системы и предотвращение возможных ошибок.

**Цели и задачи работы**

Целью данного курсового проекта является разработка приложения, которое будет выполнять анализ файловой системы, предоставляя пользователю подробную информацию о структуре папок, размере файлов и их расположении на диске. Для достижения этой цели мы будем использовать различные инструменты и библиотеки Java. Для достижения этой цели были поставлены следующие задачи:

1. Создание логики поиска файлов.
2. Создание графического интерфейса пользователя (GUI)
3. Разработка функционала приложения (просмотр дерева каталогов, с возможностью перехода в директорию по клику или по введенному пользователем пути, просмотр информации о создателе файла, подсчет места, занимаемого директорией или файлом на диске).
4. Тестирование приложения и внесение изменений при необходимости.

**Обоснование логической структуры работы**

# Для обеспечения полного понимания процесса разработки приложения, логическая структура отчета построена последовательно. Она начинается с описания основных проблем и целей, затем переходит к анализу кода программы и описанию работы приложения, включая примеры его функционирования, как положительные, так и негативные. Завершается отчет обзором полученных результатов. Такой подход позволяет четко проследить все этапы работы от идеи до реализации приложения.

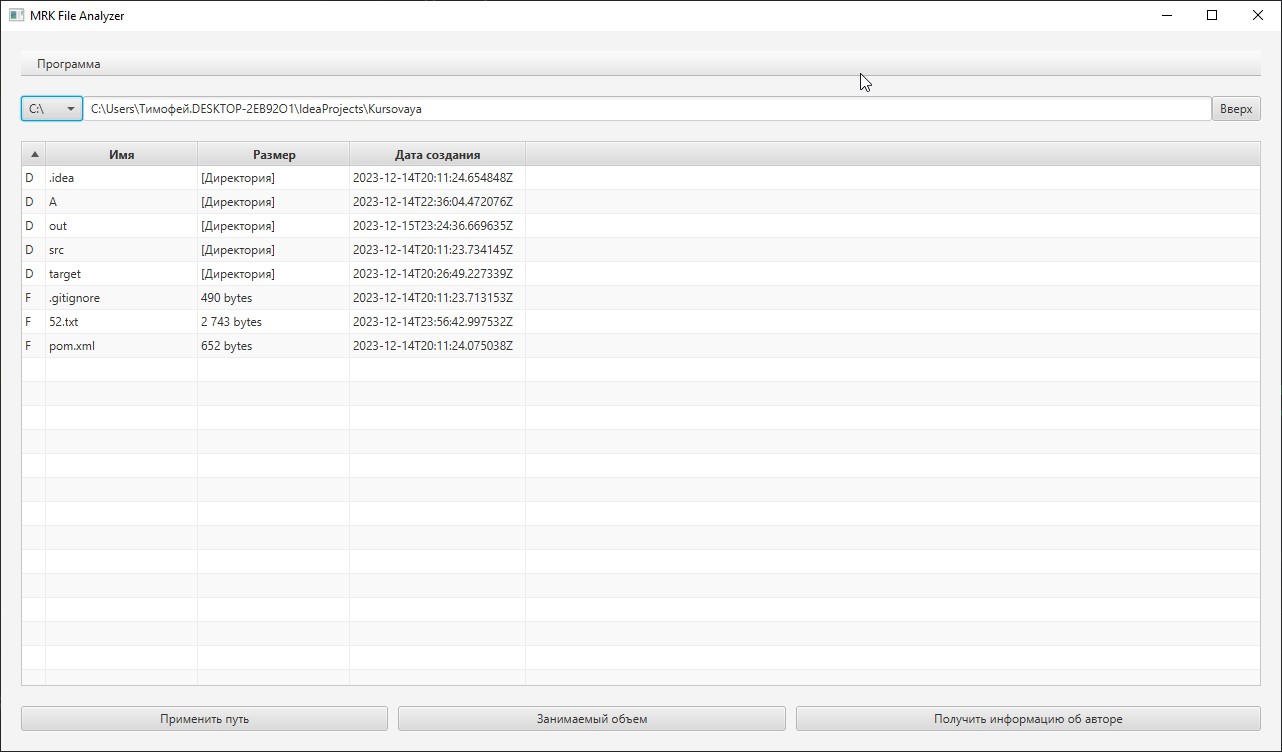
# 2. Основная часть

## 2.1 Обзор разработанного приложения

**Описание функциональности приложения**

Разработанное в процессе выполнения данного курсового проекта, приложение является утилитой для анализа файловой системы с графическим пользовательским интерфейсом (GUI). Основные функции приложения включают в себя:

1. **Просмотр содержимого директорий**: Пользователь может указать путь к каталогу либо же выбрать каталог с помощью графического интерфейса, после чего приложение проводит анализ файлов и директорий, содержащихся в указанном каталоге и выводит информацию на экран пользователю.
2. **Просмотр метаданных файла**: Пользователи могут посмотреть информацию о файле или директории, такую как размер, дата и время создания непосредственно в таблице с именами файлов.
3. **Подсчет места, занимаемого директорией**: Пользователь может при помощи клика мыши выбрать директорию или файл из каталога, после чего, нажав кнопку «Занимаемый объем» получить информацию о месте, которое занимает выбранный файл(директория).



Разработанная программа

**Описание использованных технологий и инструментов**

Приложение разработано с использованием следующих технологий и инструментов:

1. **Язык Программирования Java**: Java является современным языком программирования, позволяющим его пользователям создавать приложения с самым различным функционалом, обеспечивая при этом широкие возможности для добавления нового функционала при помощи библиотек, а также обеспечивает возможность работы программы на различных системах и устройствах(кросс-платформенность).
2. **JavaFX для GUI**: Данная библиотека позволяет достаточно просто и быстро создать графический интерфейс, который был бы интуитивно понятен пользователю, для приложения, написанного на языке Java.
3. **Стандартные Библиотеки Java**: Для эффективной и стабильной обработки файлов и директорий в приложении были использованы стандартные классы и методы, предоставляемые языком программирования Java. Это позволило достичь высокой производительности и надежности 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**: Данный класс является основным при запуске приложения, он инициализирует графический интерфейс и задает некоторые параметры для отображаемого окна.
* **Особенности**: В классе **Main** переопределяется метод start, который наследуется от класса Application, данный метод непосредственно создает окно, в котором пользователь производит действия.

public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("/hello-view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 1280, 720);  
 stage.setTitle("MRK File Analyzer");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

*Класс Main.java*

**Класс FileInfo**

* **Функция**: Данный класс нужен для сбора информации о файле, а также для её изменения.
* **Особенности**: Класс использует метод **FileInfo**, который принимает путь к каталогу, а затем получает информацию из файла и помещает её в соответствующие поля. В данном методе может возникнуть исключение, например, когда файл поврежден, для обработки данного исключения используется конструкция try-catch.

public class FileInfo {  
 public enum FileType{  
 *FILE*("F"), *DIRECTORY*("D");  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
 FileType(String name) {  
 this.name = name;  
 }  
 }  
 private String filename;  
 private FileType type;  
 private String dateOfCreation;  
 private long size;  
  
 public String getFilename() {  
 return filename;  
 }  
  
 public FileType getType() {  
 return type;  
 }  
  
 public String getDateOfCreation() {  
 return dateOfCreation;  
 }  
  
 public long getSize() {  
 return size;  
 }  
  
 public FileInfo(Path path) {  
 try {  
 this.filename = path.getFileName().toString();  
 this.size = Files.*size*(path);  
 this.type = Files.*isDirectory*(path) ? FileType.*DIRECTORY* : FileType.*FILE*;  
 if (this.type == FileType.*DIRECTORY*) {  
 this.size = -1L;  
 }  
 this.dateOfCreation = Files.*getAttribute*(path, "creationTime").toString();  
 } catch (IOException e) {  
 throw new RuntimeException("Невозможно получить информацию о файле!");  
 }  
 }  
  
}

*Класс FileInfo*

**Класс AppController**

* **Функция**: Отвечает за создание и управление графическим пользовательским интерфейсом приложения.
* **Особенности**: Данный класс определяет методы, которые будут вызваны после действия, произведенного пользователем, а также определяет и создает объекты графического интерфейса.

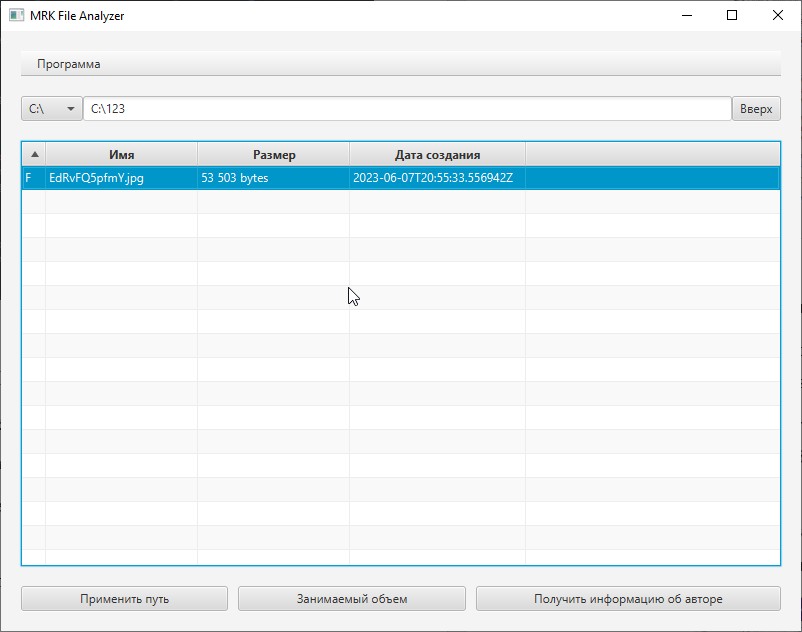
public class AppController implements Initializable {  
 @FXML  
 TableView<FileInfo> fileTable;  
 @FXML  
 ComboBox<String> diskSelect;  
 @FXML  
 TextField pathText;  
 @FXML  
 Button volumeButton;  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
 TableColumn<FileInfo, String> fileTypeColumn = new TableColumn<>();  
 fileTypeColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getType().getName()));  
 fileTypeColumn.setPrefWidth(24);  
  
 TableColumn<FileInfo, String> filenameColumn = new TableColumn<>("Имя");  
 filenameColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getFilename()));  
 filenameColumn.setPrefWidth(252);  
  
 TableColumn<FileInfo, Long> fileSizeColumn = new TableColumn<>("Размер");  
 fileSizeColumn.setCellValueFactory(param -> new SimpleObjectProperty<>(param.getValue().getSize()));  
 fileSizeColumn.setPrefWidth(152);  
 fileSizeColumn.setCellFactory(column -> {  
 return new TableCell<FileInfo, Long>() {  
 @Override  
 protected void updateItem(Long item, boolean empty) {  
 super.updateItem(item, empty);  
 if (item == null || empty){  
 setText(null);  
 setStyle("");  
 } else {  
 String text = String.*format*("%,d bytes", item);  
 if (item == -1L) {  
 text = "[Директория]";  
 }  
 setText(text);  
 }  
 }  
 };  
 });  
  
 TableColumn<FileInfo, String> fileDateColumn = new TableColumn<>("Дата создания");  
 fileDateColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getDateOfCreation().toString()));  
 filenameColumn.setPrefWidth(152);  
  
 fileTable.getColumns().addAll(fileTypeColumn, filenameColumn, fileSizeColumn, fileDateColumn);  
 fileTable.getSortOrder().add(fileTypeColumn);  
  
 diskSelect.getItems().clear();  
 for (Path p : FileSystems.*getDefault*().getRootDirectories()) {  
 diskSelect.getItems().add(p.toString());  
 }  
 diskSelect.getSelectionModel().select(0);  
 fileTable.setOnMouseClicked(new EventHandler<javafx.scene.input.MouseEvent>() {  
 @Override  
 public void handle(javafx.scene.input.MouseEvent event) {  
 if (event.getClickCount() == 2){  
 Path path = Paths.*get*(pathText.getText()).resolve(fileTable.getSelectionModel().getSelectedItem().getFilename());  
 if (Files.*isDirectory*(path)) {  
 updateList(path);  
 }  
 }  
 }  
 });  
 updateList(Paths.*get*("."));  
 }  
 public void updateList(Path path){  
 try {  
 pathText.setText(path.normalize().toAbsolutePath().toString());  
 fileTable.getItems().clear();  
 fileTable.getItems().addAll(Files.*list*(path).map(FileInfo::new).collect(Collectors.*toList*()));  
 fileTable.sort();  
 } catch (IOException e) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Не удалось получить список файлов, проверьте правильность указанного пути", ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
  
 public void btnExitAction() {  
 Platform.*exit*();  
 }  
  
 public void btnPathUpAction() {  
 Path upPath = Paths.*get*(pathText.getText()).getParent();  
 if (upPath != null) {  
 updateList(upPath);  
 }  
  
 }  
  
 public void selectDiskAction(ActionEvent actionEvent) {  
 ComboBox<String> element = (ComboBox<String>) actionEvent.getSource();  
 updateList(Paths.*get*(element.getSelectionModel().getSelectedItem()));  
 }  
  
  
 public void btnPathAction() {  
 Path newPath = Paths.*get*(pathText.getText());  
 if (newPath != null) {  
 updateList(newPath);  
 }  
 }  
  
 public void btnVolumeAction() {  
 AtomicLong size = new AtomicLong(0);  
 volumeButton.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent event) {  
 if (event.getClickCount() == 1){  
 if (getSelectedFilename() == null) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Ни один элемент не был выбран", ButtonType.*OK*);  
 alert.showAndWait();  
 return;  
 }  
 String filePath = pathText.getText() + "\\" + getSelectedFilename();  
 Path path = Paths.*get*(filePath);  
 try {  
 Files.*walkFileTree*(path, new SimpleFileVisitor<Path>() {  
 @Override  
 public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {  
 size.addAndGet(attrs.size());  
 return FileVisitResult.*CONTINUE*;  
 }  
 });  
 } catch(IOException e) {  
 throw new AssertionError();  
 }  
 String text = String.*format*("%,d bytes", size.get());  
 text = "Директория занимает: " + text;  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, text , ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
 });  
 }  
  
 public String getSelectedFilename() {  
 if(!fileTable.isFocused()) {  
 return null;  
 }  
 return fileTable.getSelectionModel().getSelectedItem().getFilename();  
 }  
 public void btnCreatorAction() {  
 if (getSelectedFilename() == null) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Ни один элемент не был выбран", ButtonType.*OK*);  
 alert.showAndWait();  
 return;  
 }  
 String filePath = pathText.getText() + "\\" + getSelectedFilename();  
 Path path = Paths.*get*(filePath);  
 try {  
 FileOwnerAttributeView file = Files.*getFileAttributeView*(path, FileOwnerAttributeView.class);  
 UserPrincipal user = file.getOwner();  
 String owner = user.getName();  
 owner = "Создатель: " + owner;  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, owner, ButtonType.*OK*);  
 alert.showAndWait();  
 } catch (IOException e) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Неудалось получить информацию об авторе", ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
  
 public void btnUpdateAction() {  
 Path path = Paths.*get*(pathText.getText());  
 updateList(path);  
 }  
}

## 2.3 Работа приложения

***Положительные примеры работы***

**Пример просмотра каталога**

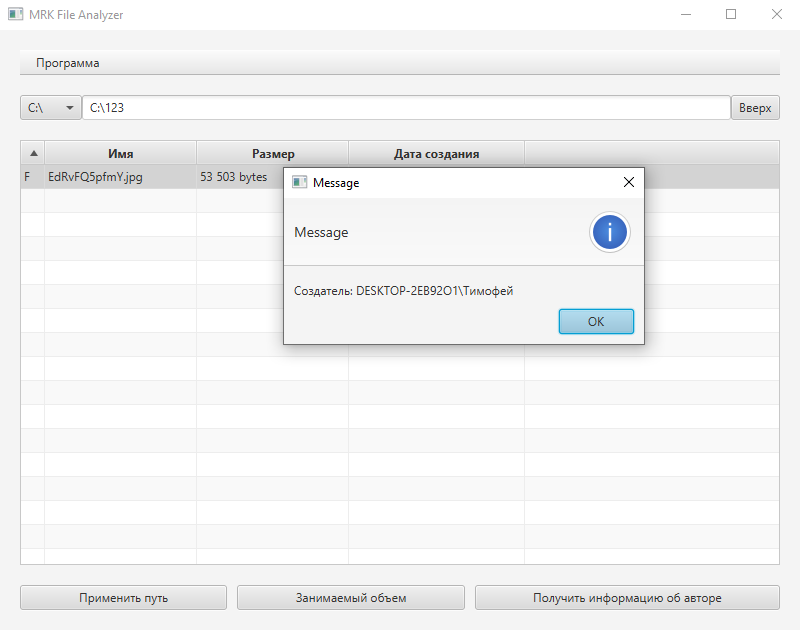
* **Сценарий**: Пользователь вводит путь и нажимает кнопку «Применить путь» либо переходит в нужный каталог, используя графический интерфейс.
* **Ожидаемый результат**: Путь был прочитан и открыт, пользователь видит на экране каталоги и файлы, хранящиеся по указанному пути, а также их размер и дату и время создания.
* **Обработка событий**: Класс **AppController** обрабатывает действие нажатия кнопки и вызывает соответствующий метод класса **FileInfo** для отображения информации о файлах и каталогах.



Пример просмотра каталога

**Пример просмотра информации об авторе файла**

* **Сценарий**: Пользователь вводит путь и нажимает кнопку «Применить путь» либо переходит в нужный каталог, используя графический интерфейс. После чего выбирает файл нажатием мыши, а затем нажимает кнопку «Получить информацию об авторе».
* **Ожидаемый результат**: Приложение отображает имя компьютера и имя пользователя, на котором был создан файл.
* **Обработка событий**: Класс **AppController** обрабатывает действия пользователя, после чего вызывает метод для получения информации о создателе файла и выводит ее на экран.

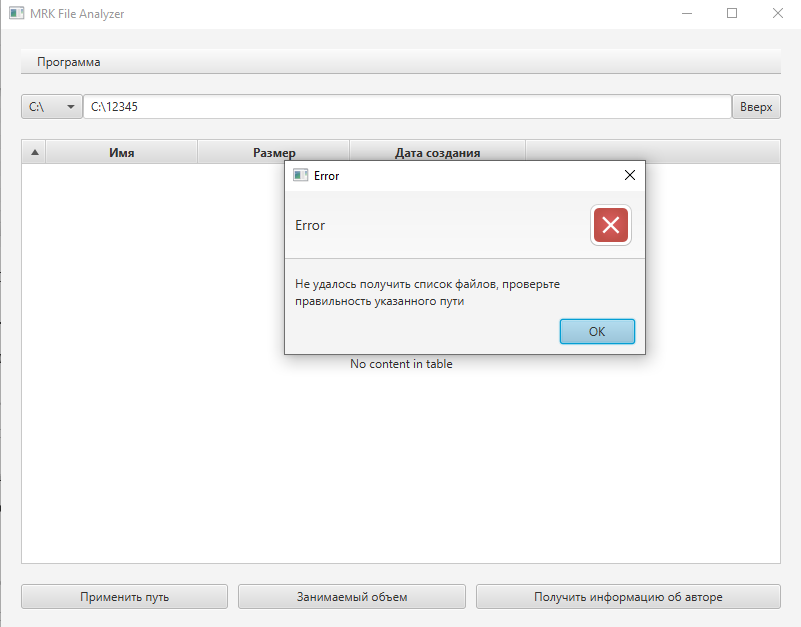


Получение информации об авторе

***Негативные примеры работы***

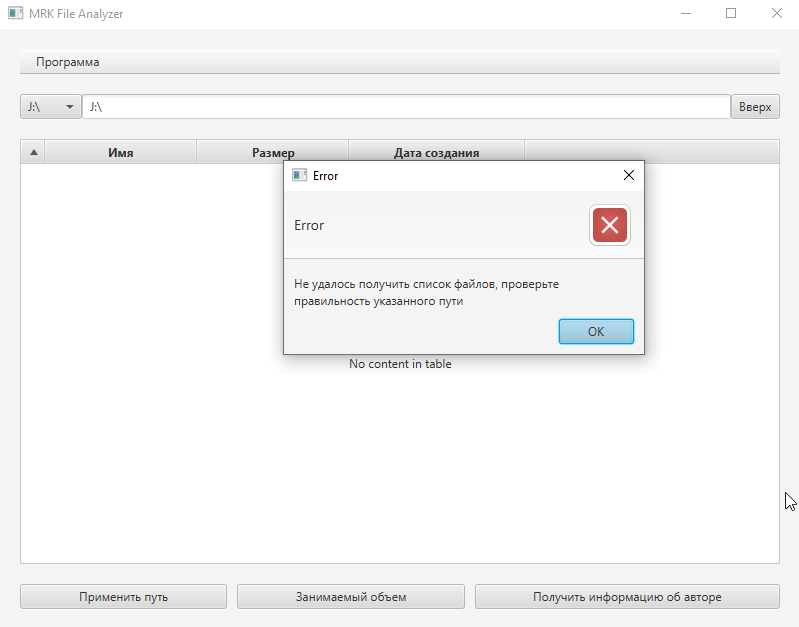
**Примеры обработки ошибок**

* **Сценарий 1**: Пользователь вводит некорректный путь к каталогу и нажимает кнопку "Применить путь".
* **Ожидаемый результат**: Приложение выводит сообщение об ошибке, указывая на невозможность получения файловой информации в указанном пути.
* **Обработка событий**: Обработка исключения в классе **FileInfo** и отображение ошибки через класс **AppController**.



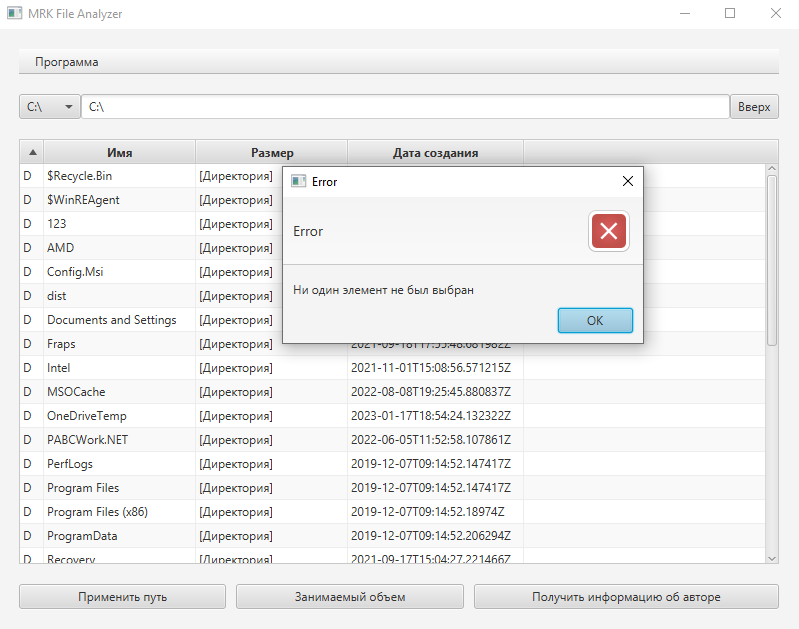
Сценарий 1

* **Сценарий 2**: Пользователь пытается получить список файлов диска, который не подключен.
* **Ожидаемый результат**: Приложение выводит сообщение об ошибке, указывая на невозможность получения списка файлов в указанном пути.
* **Обработка событий**: Обработка исключения в классе **FileInfo** и отображение ошибки через класс **AppController**.



Сценарий 2

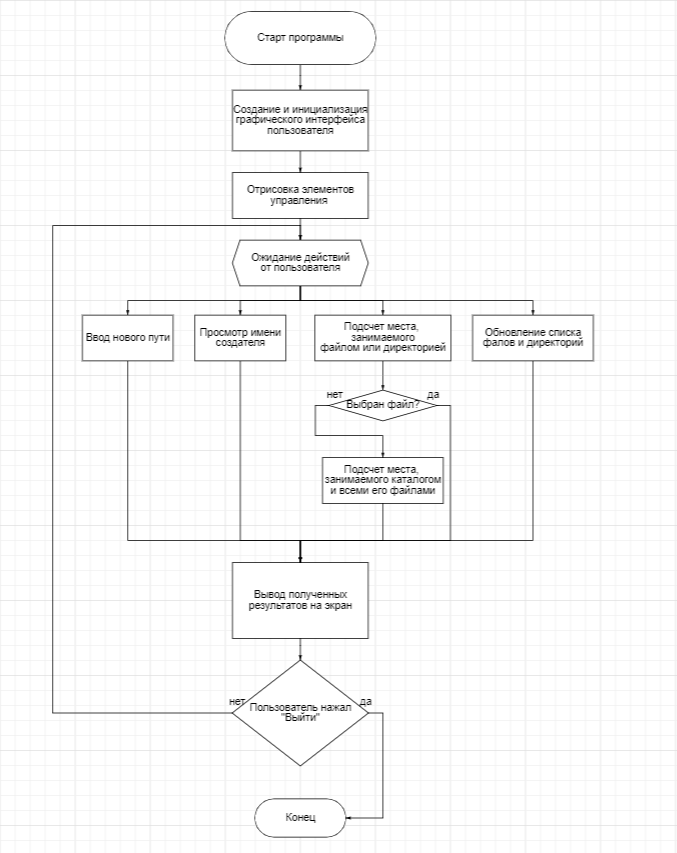
* **Сценарий 3**: Пользователь не выбрал ни один файл и нажимает кнопку "Занимаемый объем".
* **Ожидаемый результат**: Приложение выводит сообщение об ошибке, указывая на то, что ни один файл не был выбран.
* **Обработка событий**: Получение некорректного значения для имени файла и последующий вывод сообщения в классе AppController.



Сценарий 3

## 2.4 Блок-схема алгоритма приложения

Для лучшего понимания функционирования приложения полезно создать блок-схему его алгоритма. Эта схема наглядно представляет последовательность выполнения программы и взаимосвязь между ее составляющими. Блок-схема способствует лучшему восприятию логики работы приложения и служит основой для последующих усовершенствований и поиска ошибок.



Блок схема программы

## 2.5 Использование библиотек

**Обзор использованных библиотек**

В процессе разработки приложения для анализа файловой системы были использованы следующие основные библиотеки и технологии:

1. **Java FX**:
   * **Назначение**: Создание графического пользовательского интерфейса (GUI).
   * **Эффективность**: JavaFX предоставляет инструменты для создания и работы с графическим интерфейсом, данная библиотека является более простой и удобной в использовании, чем Swing, при этом не лишена преимуществ данной библиотеки, таких как: простота интеграции, надежность работы и кросс-платформенность.
2. **Java Standard Library**:
   * **Назначение**: Обработка файлов, работа с коллекциями, управление потоками и другие базовые функции.

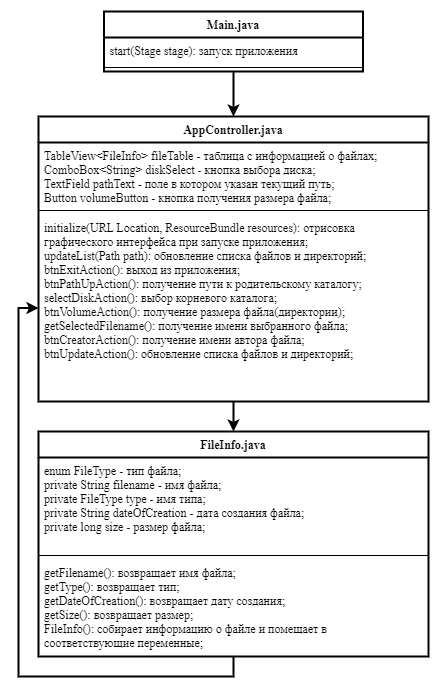
* **Эффективность**: Стандартная библиотека Java включает в себя широкий спектр инструментов для выполнения базовых операций, обеспечивающих стабильность, производительность и безопасность приложения.

1. **Java I/O и NIO**:
   * **Назначение**: Чтение и запись файлов, обработка файловых путей и атрибутов.
   * **Эффективность**: Java NIO и Java I/O предоставляют удобные инструменты для работы с файловой системой, которые необходимы для корректного функционирования поиска файлов в приложении.

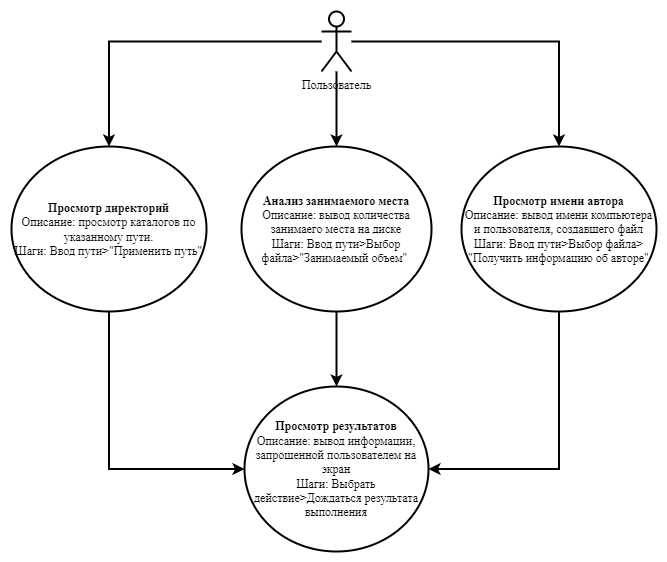
**Почему были выбраны эти библиотеки**

* **Легкая интеграция с Java**: использование этих библиотек не вызовет затруднений благодаря их совместимости с основным стеком технологий Java, что снижает вероятность возникновения проблем и конфликтов.
* **Высокая надежность и стабильность**: применение этих библиотек оправдано благодаря их всесторонней проверке и использованию в больших количествах проектов по всему миру. Это служит гарантией их надежности и стабильности.
* **Активное сообщество поддержки**: благодаря широкому распространению, эти библиотеки обеспечены обширной документацией и поддержкой со стороны сообщества разработчиков, что облегчает процесс изучения и эксплуатации.

## 2.6 UML Диаграмма Классов



## 2.7 UML Диаграмма Прецедентов



## 2.8 Основные Концепции и Определения

В рамках разработки приложения для анализа файловой системы, важно понять ряд ключевых теоретических понятий, которые лежат в основе функциональности программы.

**Анализ файловой системы**

* **Файловая система**: Это структура, которая организует и хранит данные на компьютере. Она определяет, как файлы и папки хранятся, как к ним осуществляется доступ и как они управляются.
* **Дерево каталогов**: Это иерархическая структура, используемая для организации файлов и папок на компьютере.

**Обработка Файлов в Java**

* **Java I/O и NIO**: В Java для работы с файлами используются пакеты ввода-вывода (I/O) и новые ввод-выводы (NIO). Эти пакеты предоставляют классы и интерфейсы для чтения, записи и управления файлами и файловыми системами.
* **Особенности и Применение**: Эти пакеты обеспечивают эффективное выполнение операций с файлами, таких как чтение данных из файла, запись в файл или получение информации о файлах и каталогах.

**Теоретический Обзор Алгоритмов Поиска**

Поиск файлов и папок требует использования специализированных алгоритмов. Алгоритм, используемый в нашем приложении, можно описать следующими шагами:

1. **Чтение Директории**: Программа начинает с чтения указанной пользователем директории для получения списка файлов.
2. **Получение данных**: Из полученного списка файлов, мы получаем файлы, у которых запрашиваем такие данные как дата создания, размер и т.п.
3. **Возврат Результатов**: Файлы и директории, которые находятся по указанному пути, а также некоторые их метаданные возвращаются как результат поиска.

Этот процесс демонстрирует важность эффективной работы с файлами и алгоритмов поиска в задачах обработки данных.

**Важность Эффективности и Точности**

* **Эффективность**: Учитывая, что операции поиска могут выполняться в крупных файловых системах, эффективность алгоритма критически важна. Она обеспечивает минимальное время отклика и оптимальное использование ресурсов.
* **Точность**: Точность алгоритма поиска обеспечивает, что результаты будут надежными и соответствующими заданным пользователем критериям. Это особенно важно в приложениях, где требуется высокая степень точности, например, в системах безопасности или при обработке конфиденциальных данных.

**Методы Обработки Файлов**

Обработка файлов - ключевой компонент в приложении. Рассмотрим подробнее методы и практики, связанные с этой областью.

**Работа с Файловой Системой в Java**

* **Использование Java I/O и NIO**: В Java для чтения и записи файлов часто используются классы из пакетов **java.io** и **java.nio**. Эти пакеты предоставляют широкие возможности для работы с файлами, включая чтение и запись данных, обработку файловых путей и управление атрибутами файлов.
* **Буферизированное Чтение и Запись**: Буферизированные потоки (**BufferedReader**, **BufferedWriter**) используются для повышения эффективности чтения и записи, минимизируя количество операций ввода-вывода.

**Аспекты Безопасности в Программировании**

При работе с файлами и данными важно учитывать аспекты безопасности, чтобы обеспечить защиту информации и предотвратить уязвимости.

**Защита Данных**

* **Контроль Доступа**: Важно обеспечить, чтобы доступ к файлам и операциям с ними был строго регламентирован и контролируем.
* **Шифрование**: В некоторых случаях может потребоваться шифрование данных для обеспечения их конфиденциальности.

**Предотвращение Уязвимостей**

* **Обработка Исключений**: Корректная обработка исключений предотвращает "падение" программы при возникновении ошибок.
* **Проверка Ввода**: Валидация пользовательского ввода и данных, получаемых из внешних источников, необходима для предотвращения атак, например, инъекций кода.

## 2.9 Безопасность и надежность программы

**Важность Безопасности в Современных Приложениях**

В современных приложениях безопасность является одним из наиболее важных аспектов. Она обеспечивает защиту данных пользователей и стабильную работу системы. Основные угрозы включают неавторизованный доступ к файлам, исполнение вредоносного кода и повреждение данных. Для обеспечения безопасности применяются меры, такие как обработка исключений, валидация ввода, использование надежных библиотек и шифрование данных. Надежность приложения достигается через резервное копирование и восстановление, тестирование и отладку, а также регулярные обновления и патчи. Робастность приложения обеспечивается его устойчивостью к ошибкам и способностью адаптироваться к различным условиям. Процедуры восстановления, такие как механизмы восстановления и резервное копирование, также играют важную роль в обеспечении надежности приложения.

**Основные Угрозы**

* **Неавторизованный 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**: Некорректная 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 путей файлов и ограничение доступа к критически важным системным файлам и директориям.

**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 также включает в себя меры по предотвращению несанкционированного доступа к данным, а также способность приложения адаптироваться к различным системным конфигурациям и техническим характеристикам оборудования. Важно также учитывать человеческий фактор и разрабатывать программное обеспечение, которое будет понятным и удобным для пользователей, независимо от их уровня технической подготовки.

**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**

* **Защита Чувствительной Информации**: В случаях, когда приложение обрабатывает чувствительные данные, важно применять методы шифрования для их защиты.
* **Применение Современных Алгоритмов**: Использование проверенных и современных алгоритмов шифрования, таких как AES (Advanced Encryption Standard), обеспечивает высокий уровень безопасности.

**Обновления и Патчи**

* **Регулярные Обновления**: Предоставление регулярных обновлений программного обеспечения помогает устранять обнаруженные уязвимости и улучшать функциональность.
* **Быстрое Реагирование на Уязвимости**: Способность оперативно выпускать патчи и обновления в ответ на обнаружение новых угроз повышает надежность приложения.

**Обеспечение Надежности**

* **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

Анализ производительности программного обеспечения помогает определить его эффективность и выявить возможности для улучшения. В приложении для анализа файловой системы производительность играет критическую роль, особенно при работе с большими объемами данных.

**Ключевые параметры производительности**

Скорость обработки:

– Время отклика: Измерение времени, затраченного на выполнение поиска файлов, важно для оценки скорости работы приложения. Время, затраченное на сканирование директорий также учитывается.

Использование ресурсов:

– Использование памяти: Анализ использования памяти приложением помогает определить эффективность его работы. Управление памятью особенно важно при работе с большим количеством файлов или крупными файлами.

– Загрузка процессора: Мониторинг загрузки процессора во время работы приложения позволяет выявить возможные проблемы с производительностью.

**Анализ производительности приложения**

Практические тесты:

– Тестирование на разных объемах данных: Производительность приложения проверяется при работе с разными размерами директорий – от малых до очень больших, чтобы оценить влияние изменения объема данных на скорость и эффективность.

**2.11 Часто Задаваемые Вопросы (FAQ)**

Раздел часто задаваемых вопросов (FAQ) помогает пользователям находить ответы на распространенные вопросы, упрощая использование приложения и помогая решать возникающие проблемы.

**Вопросы и Ответы**

1. **Вопрос**: Что делать, если приложение не находит файлы?
   * **Ответ**: Убедитесь, что путь к директории введен корректно и выбранный диск подключен. Также стоит проверить состояние жесткого диска, возможно файлы повреждены.
2. **Вопрос**: Можно ли использовать приложение для анализа на внешнем диске?
   * **Ответ**: Да, приложение может искать файлы на любом подключенном устройстве хранения данных, если у вас есть доступ к его файловой системе.
3. **Вопрос**: Может ли приложение работать на разных операционных системах?
   * **Ответ**: Да, приложение разработано на Java, что обеспечивает его совместимость с различными операционными системами.

# Заключение

**Итоги Работы**

В процессе выполнения данной курсовой работы было создано приложение, предназначенное для анализа файловой системы. Проект успешно реализует заявленные функциональные требования, позволяя пользователям осуществлять чтение каталогов из дерева и просматривать информацию о файлах. Разработка приложения проводилась на языке программирования Java, что обеспечило его кроссплатформенность и высокую доступность.

**Основные Достижения**

1. **Реализация Графического Пользовательского Интерфейса (GUI)**: Использование библиотеки JavaFX позволило создать интуитивно понятный и функциональный интерфейс.
2. **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 показывает хорошую эффективность и надежность при выполнении своих задач.
* **Расширение функционала:** В будущем приложение может быть дополнено новыми функциями, такими как просмотр свободной памяти на диске, обнаружение дубликатов файлов, возможность копирования и перемещения файлов непосредственно в приложении.
* **Оптимизация производительности**: Есть потенциал для дальнейшего повышения производительности, особенно при работе с крупными объемами данных.
* **Улучшение пользовательского интерфейса**: Интерфейс можно улучшить для обеспечения более удобного и интуитивного использования.

# Список источников

1. Руководство по Java FX [Электронный ресурс] // Metanit.com : [сайт]. — URL: https://metanit.com/java/javafx/
2. Работа с файловой системой в Java: чтение и запись файлов и директорий / [Электронный ресурс] // Nastroika-pc.ru : [сайт]. — URL: https://nastroika-pc.ru/rabota-s-failovoi-sistemoi-v-java-ctenie-i-zapis-failov-i-direktorii/ (дата обращения: 21.11.2023).
3. Исключения в Java: перехват и обработка / [Электронный ресурс] // javarush.com : [сайт]. — URL: https://javarush.com/groups/posts/1943-iskljuchenija-perekhvat-i-obrabotka
4. Изучаем Java [Книга] / Сьерра K. Бэйтс Б. — ISBN 9785699545742. — Москва: ЭКСМО, 2012 — 720 c.
5. GeekBrains: "Разработка простого файлового менеджера на Java" - 31.01.20// [Электронный ресурс] // Youtube.com : [сайт]. — URL: https://www.youtube.com/watch?v=LILeZhSHf1k

# Приложение

Листинг программы:

1. **Main.java**

package org.example;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("/hello-view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 1280, 720);  
 stage.setTitle("MRK File Analyzer");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

1. **AppController.java**

package org.example;  
  
import javafx.application.Platform;  
import javafx.beans.property.SimpleObjectProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.fxml.FXML;  
import javafx.fxml.Initializable;  
import javafx.scene.control.\*;  
import javafx.scene.input.MouseEvent;  
  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.\*;  
import java.nio.file.attribute.BasicFileAttributes;  
import java.nio.file.attribute.FileOwnerAttributeView;  
import java.nio.file.attribute.UserPrincipal;  
import java.util.ResourceBundle;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.Collectors;  
  
public class AppController implements Initializable {  
 @FXML  
 TableView<FileInfo> fileTable;  
 @FXML  
 ComboBox<String> diskSelect;  
 @FXML  
 TextField pathText;  
 @FXML  
 Button volumeButton;  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
 TableColumn<FileInfo, String> fileTypeColumn = new TableColumn<>();  
 fileTypeColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getType().getName()));  
 fileTypeColumn.setPrefWidth(24);  
  
 TableColumn<FileInfo, String> filenameColumn = new TableColumn<>("Имя");  
 filenameColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getFilename()));  
 filenameColumn.setPrefWidth(252);  
  
 TableColumn<FileInfo, Long> fileSizeColumn = new TableColumn<>("Размер");  
 fileSizeColumn.setCellValueFactory(param -> new SimpleObjectProperty<>(param.getValue().getSize()));  
 fileSizeColumn.setPrefWidth(152);  
 fileSizeColumn.setCellFactory(column -> {  
 return new TableCell<FileInfo, Long>() {  
 @Override  
 protected void updateItem(Long item, boolean empty) {  
 super.updateItem(item, empty);  
 if (item == null || empty){  
 setText(null);  
 setStyle("");  
 } else {  
 String text = String.*format*("%,d bytes", item);  
 if (item == -1L) {  
 text = "[Директория]";  
 }  
 setText(text);  
 }  
 }  
 };  
 });  
  
 TableColumn<FileInfo, String> fileDateColumn = new TableColumn<>("Дата создания");  
 fileDateColumn.setCellValueFactory(param -> new SimpleStringProperty(param.getValue().getDateOfCreation().toString()));  
 filenameColumn.setPrefWidth(152);  
  
 fileTable.getColumns().addAll(fileTypeColumn, filenameColumn, fileSizeColumn, fileDateColumn);  
 fileTable.getSortOrder().add(fileTypeColumn);  
  
 diskSelect.getItems().clear();  
 for (Path p : FileSystems.*getDefault*().getRootDirectories()) {  
 diskSelect.getItems().add(p.toString());  
 }  
 diskSelect.getSelectionModel().select(0);  
 fileTable.setOnMouseClicked(new EventHandler<javafx.scene.input.MouseEvent>() {  
 @Override  
 public void handle(javafx.scene.input.MouseEvent event) {  
 if (event.getClickCount() == 2){  
 Path path = Paths.*get*(pathText.getText()).resolve(fileTable.getSelectionModel().getSelectedItem().getFilename());  
 if (Files.*isDirectory*(path)) {  
 updateList(path);  
 }  
 }  
 }  
 });  
 updateList(Paths.*get*("."));  
 }  
 public void updateList(Path path){  
 try {  
 pathText.setText(path.normalize().toAbsolutePath().toString());  
 fileTable.getItems().clear();  
 fileTable.getItems().addAll(Files.*list*(path).map(FileInfo::new).collect(Collectors.*toList*()));  
 fileTable.sort();  
 } catch (IOException e) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Не удалось получить список файлов, проверьте правильность указанного пути", ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
  
 public void btnExitAction() {  
 Platform.*exit*();  
 }  
  
 public void btnPathUpAction() {  
 Path upPath = Paths.*get*(pathText.getText()).getParent();  
 if (upPath != null) {  
 updateList(upPath);  
 }  
  
 }  
  
 public void selectDiskAction(ActionEvent actionEvent) {  
 ComboBox<String> element = (ComboBox<String>) actionEvent.getSource();  
 updateList(Paths.*get*(element.getSelectionModel().getSelectedItem()));  
 }  
  
  
 public void btnPathAction() {  
 Path newPath = Paths.*get*(pathText.getText());  
 if (newPath != null) {  
 updateList(newPath);  
 }  
 }  
  
 public void btnVolumeAction() {  
 AtomicLong size = new AtomicLong(0);  
 volumeButton.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent event) {  
 if (event.getClickCount() == 1){  
 if (getSelectedFilename() == null) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Ни один элемент не был выбран", ButtonType.*OK*);  
 alert.showAndWait();  
 return;  
 }  
 String filePath = pathText.getText() + "\\" + getSelectedFilename();  
 Path path = Paths.*get*(filePath);  
 try {  
 Files.*walkFileTree*(path, new SimpleFileVisitor<Path>() {  
 @Override  
 public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {  
 size.addAndGet(attrs.size());  
 return FileVisitResult.*CONTINUE*;  
 }  
 });  
 } catch(IOException e) {  
 throw new AssertionError();  
 }  
 String text = String.*format*("%,d bytes", size.get());  
 text = "Директория занимает: " + text;  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, text , ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
 });  
 }  
  
 public String getSelectedFilename() {  
 if(!fileTable.isFocused()) {  
 return null;  
 }  
 return fileTable.getSelectionModel().getSelectedItem().getFilename();  
 }  
 public void btnCreatorAction() {  
 if (getSelectedFilename() == null) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Ни один элемент не был выбран", ButtonType.*OK*);  
 alert.showAndWait();  
 return;  
 }  
 String filePath = pathText.getText() + "\\" + getSelectedFilename();  
 Path path = Paths.*get*(filePath);  
 try {  
 FileOwnerAttributeView file = Files.*getFileAttributeView*(path, FileOwnerAttributeView.class);  
 UserPrincipal user = file.getOwner();  
 String owner = user.getName();  
 owner = "Создатель: " + owner;  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, owner, ButtonType.*OK*);  
 alert.showAndWait();  
 } catch (IOException e) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Неудалось получить информацию об авторе", ButtonType.*OK*);  
 alert.showAndWait();  
 }  
 }  
  
 public void btnUpdateAction() {  
 Path path = Paths.*get*(pathText.getText());  
 updateList(path);  
 }  
}

1. **FileInfo.java**
2. package org.example;  
     
   import java.io.IOException;  
   import java.nio.file.Files;  
   import java.nio.file.Path;  
     
   public class FileInfo {  
    public enum FileType{  
    *FILE*("F"), *DIRECTORY*("D");  
    private String name;  
     
    public String getName() {  
    return name;  
    }  
    FileType(String name) {  
    this.name = name;  
    }  
    }  
    private String filename;  
    private FileType type;  
    private String dateOfCreation;  
    private long size;  
     
    public String getFilename() {  
    return filename;  
    }  
     
    public FileType getType() {  
    return type;  
    }  
     
    public String getDateOfCreation() {  
    return dateOfCreation;  
    }  
     
    public long getSize() {  
    return size;  
    }  
     
    public FileInfo(Path path) {  
    try {  
    this.filename = path.getFileName().toString();  
    this.size = Files.*size*(path);  
    this.type = Files.*isDirectory*(path) ? FileType.*DIRECTORY* : FileType.*FILE*;  
    if (this.type == FileType.*DIRECTORY*) {  
    this.size = -1L;  
    }  
    this.dateOfCreation = Files.*getAttribute*(path, "creationTime").toString();  
    } catch (IOException e) {  
    throw new RuntimeException("Невозможно получить информацию о файле!");  
    }  
    }  
     
   }
3. **hello-view.fxml**
4. <?xml version="1.0" encoding="UTF-8"?>  
     
   <?import javafx.geometry.Insets?>  
   <?import javafx.scene.control.Label?>  
   <?import javafx.scene.layout.VBox?>  
     
   <?import javafx.scene.control.Button?>  
   <?import javafx.scene.control.MenuBar?>  
   <?import javafx.scene.control.Menu?>  
   <?import javafx.scene.control.MenuItem?>  
   <?import javafx.scene.control.TableView?>  
   <?import javafx.scene.layout.HBox?>  
   <?import javafx.scene.control.ComboBox?>  
   <?import javafx.scene.control.TextField?>  
   <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"  
    fx:controller="org.example.AppController">  
    <padding>  
    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>  
    </padding>  
    <MenuBar>  
    <Menu text = "Программа">  
    <MenuItem text="Обновить" onAction="#btnUpdateAction"></MenuItem>  
    <MenuItem text ="Выйти" onAction="#btnExitAction"></MenuItem>  
     
    </Menu>  
    </MenuBar>  
    <HBox>  
    <ComboBox fx:id="diskSelect" onAction="#selectDiskAction">  
    </ComboBox>  
    <TextField fx:id="pathText" HBox.hgrow="ALWAYS" editable="true" ></TextField>  
    <Button text="Вверх" onAction="#btnPathUpAction"></Button>  
    </HBox>  
    <TableView fx:id="fileTable" VBox.vgrow="ALWAYS">  
    </TableView>  
    <HBox spacing="10">  
    <Button text="Применить путь" maxWidth="Infinity" HBox.hgrow="ALWAYS" onAction="#btnPathAction" />  
    <Button fx:id="volumeButton" text="Занимаемый объем" maxWidth="Infinity" HBox.hgrow="ALWAYS" onAction="#btnVolumeAction" focusTraversable="false" />  
    <Button text="Получить информацию об авторе" maxWidth="Infinity" HBox.hgrow="ALWAYS" onAction="#btnCreatorAction" focusTraversable="false"/>  
    </HBox>  
   </VBox>

**Ссылка на код (GitHub):**

https://github.com/Ne0n52/Kursach/tree/main/Kursovaya