Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Шифрование файлов

Студент: гр. 951007

Воривода М.А.

Руководитель:

Данилова Г. В.

Минск 2020

**СОДЕРЖАНИЕ**

[Ввеедение 5](#_Toc482833182)

[1 Анализ аналогов программного средства 7](#_Toc482833183)

[1.1 «PGPTools» **–**  надежный шифратор текстовых сообщений](#_Toc482833184) 7

[1.2 Постановка задачи](#_Toc482833186) 8

[2 Цели и задачи 9](#_Toc482833185)

[2.1 Цель курсовой работы 9](#_Toc482833186)

[2.2 Задачи для курсовой работы 9](#_Toc482833187)

[2.3 Необходимые ресурсы для курсовой работы 9](#_Toc482833188)

[3 Проектирование. Разработка курсовой работы 10](#_Toc482833189)

[3.1 Структура программного средства 10](#_Toc482833190)

[3.2 Разработка графического пользовательского интерфейса 10](#_Toc482833191)

[3.3 Потоки и сокеты](#_Toc482833193) 18

[4 Тестирование программного средства 2](#_Toc482833205)6

[5 Руководство пользователя 24](#_Toc482833206)

[5.1 Начало работы 24](#_Toc482833207)

[5.2 Запуск сервера 24](#_Toc482833208)

[5.3 Запуск клиента](#_Toc482833209) 25

[Заключение 27](#_Toc482833210)

[Список использованных источников 29](#_Toc482833211)

[Приложение А. Исходный код программы 30](#_Toc482833212)

[Приложение Б. Блок-схема метода](#_Toc482833213) 61

**ВВЕДЕНИЕ**

В 21-м веке защита информации занимает главенствующую роль в обороте данных между людьми, организациями и странами. И несмотря на то, что наука криптография существует с древних времён, с движением по исторической временной линии, значение различных методов шифрования увеличивается вплоть до текущего дня, а так как основная задача программиста – это работа с информацией, то изучение способов защиты данных есть обязательной частью его багажа профессиональных знаний.

История криптографии насчитывает около 4 тысяч лет. В качестве основного критерия периодизации криптографии возможно использовать технологические характеристики используемых методов шифрования.

Первый период – приблизительно с 3-го тысячелетия до н.э. – характеризуется господством моноалфавитных шифров (основной принцип – замена алфавита исходного текста другим алфавитом через замену букв другими буквами или символами).

Второй период – с IX века на Ближнем Востоке и с XV века в Европе – до начала XX века. Этап ознаменовался введением в обиход полиалфавитных шифров.

Третий период – с начала и до середины XX века – характеризуется внедрением электромеханических устройств в работу шифровальщиков. При этом продолжалось использование полиалфавитных шифров.

Четвёртый период – с середины до 70-х годов XX века – период перехода к математической криптографии. В работе Шеннона появляются строгие математические определения количества информации, передачи данных, энтропии, функций шифрования. Обязательным этапом создания шифра считается изучение его уязвимости для различных известных атак –криптоанализа. Однако 1975 криптография оставалась «классической» или же, более корректно, криптографией с секретным ключом.

Современный период развития криптографии – с конца 1970-х годов по настоящее время – отличается зарождением и развитием нового направления – криптография с открытым ключом. Её появление знаменуется не только новыми техническими возможностями, но и сравнительно широким распространением криптографии для использования частными лицами. Правовое регулирование использования криптографии частными лицами в разных странах сильно различается – от разрешения до полного запрета.

С широким распространением компьютеров, начиная с 1970-х годов, интерес к криптографии растёт со стороны отдельных исследователей, бизнеса и частных лиц. Этому способствовали в том числе и публикации в открытой печати – книга Дэвида Кана «Взломщики кодов», готовность научной (создание ячейки Фейстеля, работы Диффи и Хеллмана, шифров DES и RSA) и технической базы (вычислительной техники), а также наличие «заказа» со стороны бизнеса – требований к надёжной передаче информации в рамках отдельной страны и по всему миру. Одновременно с этим появилось и сопротивление со стороны государства развитию открытой криптографии (гражданской криптографии). Среди причин негативного отношения правительства указывают на недопустимость попадания надёжных систем шифрования в руки террористов, организованной преступности или вражеской разведки.

Сегодня криптографическое ПО разрабатывается в основном маленькими компаниями, созданными исключительно для одного проекта. Однако можно отметить, например, российскую компанию «КриптоПро», которая разрабатывает различные методы шифрования для различных компаний. Примечательной белорусской компанией и одновременно резидентом парка высоких технологий является «С-Терра Бел». В остальном же исследования и практическое применение в области шифрования проводятся в специальных отделах крупных компаний: «Microsoft», «Google» и так далее.

Целью данного курсового проекта является разработка компьютерного средства шифрования файлов.

# 1 АНАЛИЗ АНАЛОГОВ ПРОГРАММНОГО СРЕДСТВА

### Приложение «PGPTools» – надежный шифратор текстовых сообщений

Разработанное компанией SJ Software Development Group, приложение PGPTools для операционных систем iOS, Android, Mac OS, Windows 8-10 и Windows Phone позволит зашифровать самым надежным алгоритмом PGP (Pretty Good Privacy) любое текстовое сообщение и отправить любым удобным для пользователя способом.

Интерфейс шифратора/дешифратора понятный и доступный, без лишних значков и кнопочек. Здесь присутствуют только те функции, которые нужны для осуществления процесса:

* генерация PGP-ключей;
* импорт уже имеющихся PGP-ключей;
* шифрование/дешифрование PGP-сообщений.

Для того чтобы воспользоваться функцией дешифровки сообщений, необходимо импортировать публичный ключ получателя и только тогда он может дешифровать сообщение своим приватным ключом.



Рисунок 1.1 – Шифратор «PGPTools»

### Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства для шифрования файлов адаптированного под запуск на современных операционных системах, в частности, версиях ОС Windows 8 и старше.

В программном средстве планируется реализовать следующие функции:

* шифрование текста;
* шифрование изображений;
* шифрование различными методами;
* шифрование файлов с различным расширением.

# 2 ЦЕЛИ И ЗАДАЧИ

### 2.1 Цель курсовой работы

Целью данного курсового проекта является разработка программного средства для шифрования файлов.

### 2.2 Задачи для курсовой работы

Проанализировав аналоги, можно выделить следующие задачи:

* разработать интерфейс для шифратора файлов;
* разработать различные методы шифрования;
* разработать шифрование файлов с разными расширениями;
* обеспечить восстановление изображения из числового массива;
* обеспечить получение числового массива из текстовых/графических файлов.

### 2.3 Необходимые ресурсы для курсовой работы

Для разработки программного средства использован язык программирования Java. Среда разработки – IntelliJ IDEA 2019.2.4. Это одна из самых мощных и популярных интегрированных сред разработки.

# 3 ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА КУРСОВОЙ РАБОТЫ

### 3.1 Структура программного средства

Структура проекта разделена на несколько основных частей: раздел графического интерфейса, раздели шифратора и раздел дешифратора, а также раздел ассетов:

* раздел графического интерфейса содержит в себе файл разметки и контроллер;
* раздел шифратора хранит класс шифратора и классы текстового и графического шифраторов;
* раздел дешифратора хранит класс дешифратора и классы текстового и графического дешифраторов;
* раздел ассетов хранит в себе необходимые изображения для графического интерфейса.

Все разделы хранятся в общем пакете вместе с классом, запускающим

приложение.

### 3.2 Разработка графического пользовательского интерфейса

### Для проекта было необходимо разработать графический пользовательский интерфейс, с помощью которого пользователь сможет быстро выбрать нужный файл и провести с ним необходимые операции. Интерфейс разработан на платформе JavaFx.

Приложение насыщено различными функциями, предназначенными упростить работу с приложением, а само окно имеет компактные размеры, которые пользователь не может изменять, и практичный интерфейс. Полный код графического интерфейса можно найти в приложении А, «app.ui». GUI создан с помощью языка разметки.

При запуске пользователь может:

* выбрать необходимый файл нажав на «Browse…», на пункт в меню, на иконку папки слева от поля или с помощью сочетания клавиш «Alt+O»;
* ввести или сгенерировать ключ;
* скопировать ключ нажав на иконку ключа слева от поля;
* выбрать метод шифрования;
* воспользоваться справкой о руководстве, горячих клавишах, программе и авторе.

Интерфейс показан на рисунке 3.1.

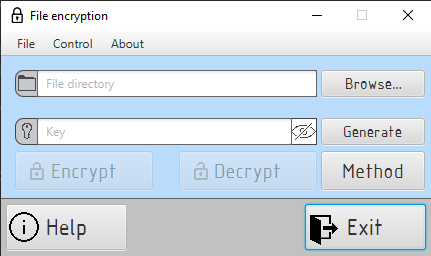


Рисунок 3.1 – Интерфейс приложения

*root* = FXMLLoader.*load*(getClass().getResource(**"ui/sample.fxml"**)); // загрузка файла разметки  
primaryStage.setTitle(**"File encryption"**); // установка заголовка приложения  
primaryStage.getIcons().add(**new** Image(**"app\\assets\\images\\mainLock.png"**)); // установка иконки заголовка  
primaryStage.setScene(**new** Scene(*root*, 430, 225)); // создание сцены и установка размеров окна  
primaryStage.setResizable(**false**); // запрет изменения размеров окна  
primaryStage.setMaximized(**false**); // отключение кнопки разворота окна на весь экран  
  
primaryStage.setOnCloseRequest(event -> {// установка обработчика события запроса на закрытие окна  
 Alert alert = **new** Alert(Alert.AlertType.***CONFIRMATION***); // создание диалогового окна  
 alert.setTitle(**"Exit"**); // установка заголовка окна  
 alert.setHeaderText(**null**); // отключение заголовочного текста  
 alert.setContentText(**"Do you want to exit?"**); // установка текста окна  
  
 ImageView image = **new** ImageView(**"app\\assets\\images\\warning.png"**); // загрузка иконки

предупреждения  
 image.setFitWidth(32); // установка ширины иконки  
 image.setFitHeight(32); // установка высоты иконки  
 alert.setGraphic(image); // загрузка иконки в окно  
  
 Optional<ButtonType> result = alert.showAndWait(); // ожидание результата  
 **if** (result.isPresent() && result.get() == ButtonType.***CANCEL***) { // проверка   
 event.consume(); // если нажата «Cancel», то отменить событие  
 } **else** {  
 Controller.*buffFile* = **null**; // если нажата «Ok», то очищаем буферный файл, чтобы его можно было

изменять вне приложения  
 }  
});   
  
primaryStage.show(); // вывод окна на экран

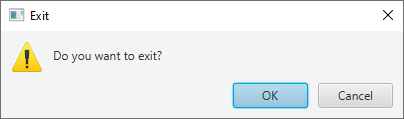
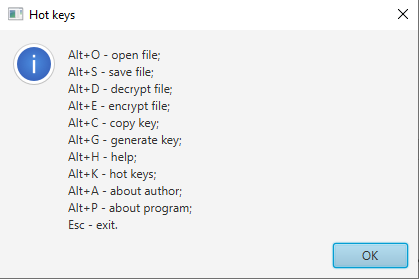


Рисунок 3.2 – Одно из диалоговых окон (запрос закрытия окна)



### Рисунок 3.3 – Список горячих клавиш

Установка горячих клавиш:

**mmChoose**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+M"**)); // выбор метода  
**mmExit**.setAccelerator(KeyCombination.*keyCombination*(**"Esc"**)); // выход  
**mmOpen**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+O"**)); // выбор файла  
**mmSave**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+S"**)); // сохранение буферного файла  
**mmCopyKey**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+C"**)); // копирование ключа  
**mmGenKey**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+G"**)); // генерирование ключа  
**mmDecrypt**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+D"**)); // расшифровка  
**mmEncrypt**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+E"**)); // зашифровка  
**mmHelp**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+H"**)); // помощь  
**mmHotKeys**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+K"**)); // список горячих клавиш  
**mmAboutAuthor**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+A"**)); // об авторе  
**mmAboutProgram**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+P"**)); // о программе





### Рисунок 3.4 – Сгенерированный ключ в скрытом и открытом виде

**eyeBtn**.setOnMouseClicked(mouseEvent -> { // скрытие и открытие ключа  
 **if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) { // проверка на нажатую ЛКМ  
 **if** (**keyPass**.isVisible()) { // если ключ скрыт  
 **eyeBtn**.setImage(**new** Image(**"app\\assets\\images\\eye\_open.png"**)); // замена иконки глаза  
 **keyField**.positionCaret(**keyPass**.getCaretPosition());//установка каретки в правильное положение  
 **keyPass**.setVisible(**false**); // открытие ключа  
 } **else** {  
 **eyeBtn**.setImage(**new** Image(**"app\\assets\\images\\eye\_close.png"**)); // замена иконки глаза  
 **keyPass**.positionCaret(**keyField**.getCaretPosition());//установка каретки в правильно положение  
 **keyPass**.setVisible(**true**); // скрытие ключа  
 }

} });

### Кнопки «Encrypt» и «Decrypt» будут неактивны, пока не будет введён ключ или же если ключ стереть (рисунок 3.5, рисунок 3.6).

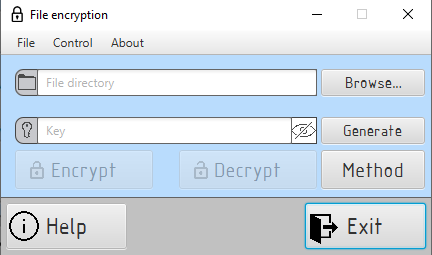


Рисунок 3.5 – Кнопки неактивны пока не введён ключ

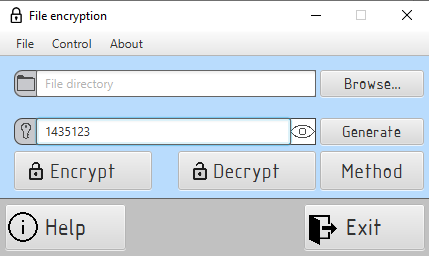
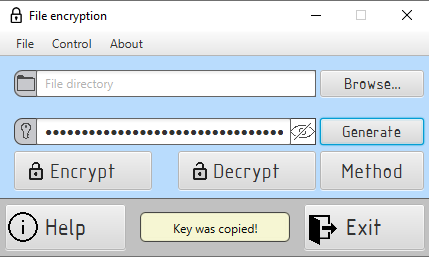


Рисунок 3.6 – Ключ введён – кнопки активны

**keyPass**.textProperty().addListener((observable, oldValue, newValue) -> { // слушатель изменения содержимого поля ключа  
 **keyField**.setText(**keyPass**.getText());  
 **keyField**.positionCaret(**keyField**.getText().length());  
 **if** (**keyPass**.getText().length() > 0) { // проверка на пустоту поля  
 enableBtns(); // включение кнопок  
 } **else** {  
 disableBtns(); // выключение кнопок  
 }  
});

### При каждом действии пользователя, успешность которого не сразу ясна, выводится анимированная подсказка (рисунок 3.7).

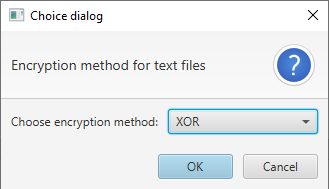


### Рисунок 3.7 – При копировании ключа возникла подсказка и, спустя секунду, началось её затухание

EventHandler<MouseEvent> copyKeyEvent = mouseEvent -> { // создание обработчика события нажатия на иконку ключа  
 **if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) { // проверка на нажатую ЛКМ  
 StringSelection stringSelection = **new** StringSelection(**keyField**.getText()); // выделение текста  
 Clipboard clipboard = Toolkit.*getDefaultToolkit*().getSystemClipboard(); // получение буфера обмена  
 clipboard.setContents(stringSelection, **null**); // занесение ключа в буфер  
 showHintWith(**"Key was copied!"**); // вывод анимированной подсказки  
 }  
};

**void** showHintWith(String content) {  
 **myTextHint**.setText(content); // установка переданного текста  
 **hintPane**.setOpacity(1); // установка прозрачности на 100%  
 (**new** Timer()).schedule(**new** TimerTask() {   
 @Override  
 **public void** run() {  
 AnimationTimer fading = **new** MyTimer();   
 fading.start(); // запуск анимации затухания  
 }  
 }, 1000); // отложить запуск на одну секунду  
}

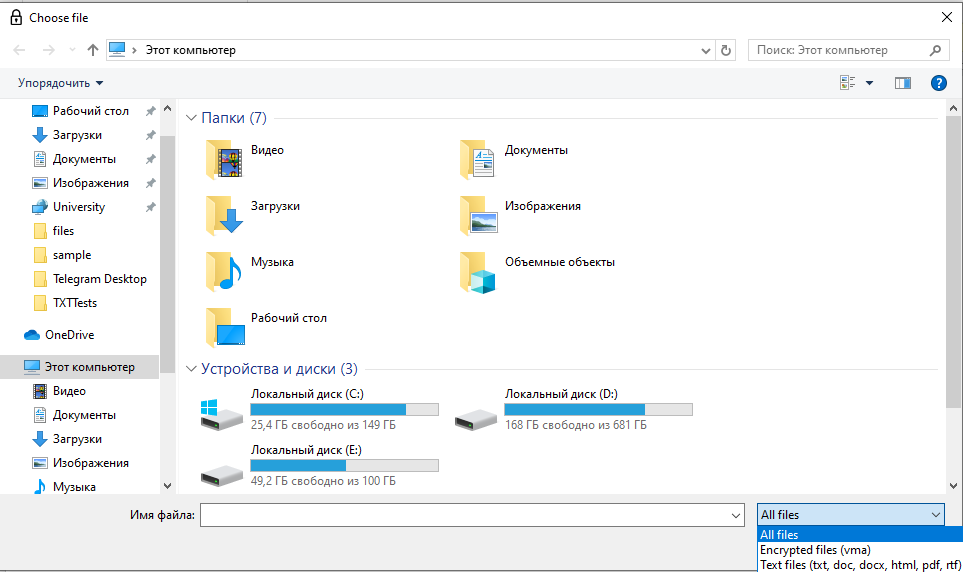
### Пользователь имеет возможность выбрать метод шифрования (рисунок 3.8).



### Рисунок 3.8 – Окно выбора метода шифрования

**public static void** methodTypeChoosing() {   
 ArrayList<String> lst = **new** ArrayList<>(); // создание списка вариантов  
 lst.add(**"XOR"**);  
 lst.add(**"Caesar"**);  
 ChoiceDialog<String> dialog = **new** ChoiceDialog<>(**"XOR"**, lst); // создание диалогового окна с созданным списком и вариантом «XOR» по умолчанию  
  
 dialog.setTitle(**"Choice dialog"**); // установка заголовка  
 dialog.setHeaderText(**"Encryption method for text files"**); // установка заголовочного текста  
 dialog.setContentText(**"Choose encryption method:"**); // установка текста окна  
  
 Optional<String> result = dialog.showAndWait(); // ожидание ответа  
 **if** (result.isPresent()) { // проверка на существование во избежание ошибок   
 **switch** (result.get()) { // получение и в зависимости от выбора присвоение необходимого метода  
 **case "XOR"**: *encryptionMethod* = 1; **break**;   
 **case "Caesar"**: *encryptionMethod* = 2; **break**;  
 }  
 }  
}

Для выбора файла нужно нажать на кнопку «Browse…», на кнопку в меню, на иконку папки или с помощью горячих клавиш. В самом окне можно выбрать фильтры расширений (рисунок 3.9).



### Рисунок 3.9 – Диалоговое окно выбора файла

**void** executeOpenFileDialog() { // создание диалогового окна выбора файла  
 FileChooser fileChooser = **new** FileChooser(); // создание экземпляра   
 fileChooser.setTitle(**"Choose file"**); // установка заголовка  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"All files"**, **"\*.txt"**, **"\*.doc"**, **"\*.docx"**, **"\*.html"**, **"\*.pdf"**, **"\*.rtf"**, **"\*.jpg"**, **"\*.jpeg"**, **"\*.bmp"**, **"\*.png"**, **"\*.mp3"**, **"\*.mp4"**, **"\*.avi"**, **"\*.mov"**, **"\*.flv"**, **"\*.vma"**)); // добавление фильтра   
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Encrypted files (vma)"**, **"\*.vma"**)); // добавление фильтра  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Text files (txt, doc, docx, html, pdf, rtf)"**, **"\*.txt"**, **"\*.doc"**, **"\*.docx"**, **"\*.html"**, **"\*.pdf"**, **"\*.rtf"**)); // добавление фильтра  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Images (jpg, jpeg, bmp, png)"**, **"\*.jpg"**, **"\*.jpeg"**, **"\*.bmp"**, **"\*.png"**)); // добавление фильтра  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Audio files (mp3)"**, **"\*.mp3"**)); // добавление фильтра  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Video files (mp4, avi, mov, flv)"**, **"\*.mp4"**, **"\*.avi"**, **"\*.mov"**, **"\*.flv"**)); // добавление фильтра  
 *opFile* = fileChooser.showOpenDialog(Main.*root*.getScene().getWindow()); // получение файла  
 **if** (*opFile* != **null**) { // если файл существует  
 **fileField**.setText(*opFile*.getAbsolutePath()); // вывести путь до него  
 }  
}

Если попытаться зашифровать файл, не выбрав его, выведется ошибка

(рисунок 3.10).

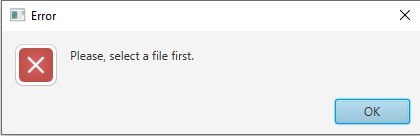


Рисунок 3.10 – Окно ошибки

**public static void** alertWith(String content, String title, Alert.AlertType type) {

// функция вывода сообщения с необходимым текстом, заголовком и типом  
 Alert alert = **new** Alert(type); // создание окна  
 alert.setTitle(title); // установка заголовка  
 alert.setHeaderText(**null**); // отключение заголовочного текста  
 alert.setContentText(content); // установка текста окна  
 alert.showAndWait(); // вывод окна  
}

Пользователь всегда имеет доступ к справке о программе (рисунок 3.11) и об авторе (рисунок 3.12), а также к помощи (рисунок 3.13).

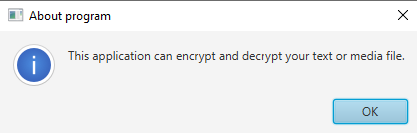


Рисунок 3.11 – Справка о программе

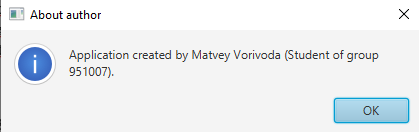


Рисунок 3.12 – Справка об авторе

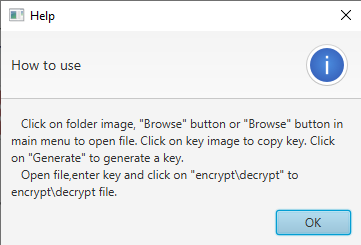


Рисунок 3.13 – Помощь

### 3.3 Шифрование

Вся информация в компьютере – это набор нулей и единиц. Каждый файл, так или иначе, можно свести к набору чисел, при это неважно текстовый ли это файл или графический. Поэтому, чтобы зашифровать какой-либо файл, необходимо получить этот набор чисел, а после, внести необходимые изменения и преобразовать получившийся в итоге набор обратно в текстовый или графический формат.

Для правильного считывания информации из файла нужно определить его тип и расширение. С этого момента оперируемый файл находится в классе «Cypher».

Производится проверка типа и отправка в соответствующий шифратор:

**static final** HashSet<String> ***txtFiles*** = **new** HashSet<>(Arrays.*asList*(**"txt"**, **"doc"**, **"docx"**, **"html"**, **"pdf"**, **"rtf"**));  
**static final** HashSet<String> ***imgFiles*** = **new** HashSet<>(Arrays.*asList*(**"jpg"**, **"jpeg"**, **"bmp"**, **"png"**));  
  
**public static** File executeEncrypt(File file, String key) **throws** IOException, BadLocationException {  
 String ext = *getExtension*(file); // запоминание расширения  
 **if** (***txtFiles***.contains(ext)) { // проверка на текстовый файл  
 **return** TextCypher.*encrypt*(file, key); // передача в текстовый шифратор  
 }  
 **if** (***imgFiles***.contains(ext)) { // проверка на графический файл  
 **return** ImageCypher.*encrypt*(file, key); // передача в графический шифратор  
 }  
 **return null**;  
}

**3.3.1** Текстовый шифратор. Класс «TextCypher»

Класс «Cypher» вызывает основную функцию текстового шифратора, передавая в него файл и ключ. В ней находится нужная последовательность действий для зашифровки текстового файла: получение данных из файла, преобразование ключа, зашифровка и создание зашифрованного файла.

**public static** File encrypt(File file, String key) **throws** IOException, BadLocationException {  
 **char**[] fileBytes = *getTextFileContent*(file).toCharArray(); // получение содержимого   
 **char**[] keyArray = key.toCharArray(); // преобразование ключа  
 fileBytes = *chooseEncryption*(fileBytes, keyArray, Controller.*encryptionMethod*); // выборка метода и зашифровка  
 *createTextBuffFile*(fileBytes, file, Controller.*encryptionMethod*); // создание зашифрованного файла  
 **return** *buffFile*;  
}

Метод «getTextFileContent» не производит непосредственно экстракцию данных, так как на способ считывания влияет не только тип файла, но и расширение, поэтому этот метод лишь проводит проверку расширения и выбирает нужный метод для чтения файла.

**private static** String getTextFileContent(File file) **throws** IOException, BadLocationException {  
 **switch** (*getExtension*(file)) {  
 **case "txt"**:  
 **case "html"**:  
 **return** *getTxtFileContent*(file);  
 **case "doc"**:  
 **return** *getDocFileContent*(file);  
 **case "docx"**:  
 **return** *getDocxFileContent*(file);  
 **case "pdf"**:  
 **return** *getPDFFileContent*(file);  
 **case "rtf"**:  
 **return** *getRTFFileContent*(file);  
 }  
 **return ""**;  
}

Чтение «.txt» и «.html» файлов:

**private static** String getTxtFileContent(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***)); // создание буферного считывателя в кодировке UTF-8 для правильного чтения русских символо  
 String in, content = **""**;  
 **while** ((in = br.readLine()) != **null**) {  
 content = content + in + **"\n"**; // чтение  
 }  
 br.close(); // закрытие  
 **return** content.substring(0, content.length() - 1); // удаление последнего символа переноса и возврат  
}

Чтение «.doc» файлов:

**private static** String getDocFileContent(File file) **throws** IOException {  
 WordExtractor extractor = **new** WordExtractor(**new** HWPFDocument(**new** FileInputStream(file)));

// создание считывателя для файлов Word  
 String[] fileData = extractor.getParagraphText(); // чтение  
 String answer = **""**;  
 **for** (**int** i = 0; i < fileData.**length**; i++) {  
 answer = answer + fileData[i]; // преобразование в строку  
 }  
 extractor.close(); // закрытие  
 **return** answer; // возврат  
}

Чтение «.docx» файлов:

**private static** String getDocxFileContent(File file) **throws** IOException {  
 List<XWPFParagraph> paragraphs = **new** XWPFDocument(**new** FileInputStream(file)).getParagraphs();

// получение документа и получение списка абзацев   
 String answer = **""**;  
 **for** (**int** i = 0; i < paragraphs.size(); i++) {  
 answer = answer + paragraphs.get(i).getParagraphText() + **"\n"**; // преобразование в строку  
 }  
 **return** answer; // возврат  
}

Чтение «.pdf» файлов:

**private static** String getPDFFileContent(File file) **throws** IOException {  
 PDDocument reader = PDDocument.*load*(file); // загрузка файла  
 PDFTextStripper ts = **new** PDFTextStripper(); // создание считывателя  
 ts.setStartPage(1); // установка начальной страницы  
 ts.setEndPage(reader.getNumberOfPages()); // установка последней страницы  
 String answer = ts.getText(reader); // получение всего текста  
 reader.close(); // закрытие   
 **return** answer; // возврат  
}

Чтение «.rtf» файлов:

**private static** String getRTFFileContent(File file) **throws** IOException, BadLocationException {  
 RTFEditorKit editorKit = **new** RTFEditorKit(); // создание инструмента редактирования   
 Document doc = editorKit.createDefaultDocument(); // создание стандартного документа  
 editorKit.read(**new** FileInputStream(file), doc, 0); // чтение в созданный документ  
 **return** doc.getText(0, doc.getLength()); // получение текста  
}

После считывания содержимого и преобразования его в массив символов происходит вызов метода «chooseEncryption» и передача в него массива символов, ключа и метода шифрования. В нём производится выборка метода и запуск преобразования данных. В конце – возврат.

**private static char**[] chooseEncryption(**char**[] fileBytes, **char**[] keyArray, **int** method) {  
 **switch** (method) {  
 **case** 1: **return** *XOREncryption*(fileBytes, keyArray); // метод XOR  
 **case** 2: **return** *CaesarEncryption*(fileBytes, keyArray); // шифр Цезаря  
 }  
 **return null**;  
}

XOR метод:

**private static char**[] XOREncryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = (**char**) (fileBytes[i] ^ keyArray[i % keyArray.**length**]);  
 }  
 **return** fileBytes;  
}

Шифр Цезаря:

**private static char**[] CaesarEncryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = (**char**) ((fileBytes[i] + keyArray[i % keyArray.**length**]) % 65535);  
 }  
 **return** fileBytes;  
}

Последним действием является создание буферного файла с собственным расширением «.vma». В него записывается расширение, метод шифрования и зашифрованная последовательность. В последствии его можно будет сохранить в другую директорию.

**private static void** createTextBuffFile(**char**[] fileBytes, File file, **int** method) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"locked\_"** + *getMainName*(file) + **".vma"**); // создание  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*), StandardCharsets.***UTF\_8***)); // создание выходного потока с необходимой кодировкой  
 bw.write(*getExtension*(file) + **"\n"**); // запись расширения  
 bw.write(method + **"\n"**); // запись метода шифрования  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i]); // запись последовательности  
 }  
 bw.close(); // закрытие  
}

**3.3.2** Графический шифратор. Класс «ImageCypher»

Если файл, который был передан в «Cypher», оказался изображением, то управление передаётся основному методу класса «ImageCypher». Последовательность действий аналогична последовательности шифрования текста, однако нет необходимости в проверке расширения, так как считывание происходит одинаково. Также нет проверки на метод, так как графическое содержимое нужно привести к максимально нечитабельному виду, а не усложнять последовательность действий преобразования данных в ячейке информации.

**public static** File encrypt(File file, String key) **throws** IOException {  
 BufferedImage image = ImageIO.*read*(file); // загрузка файла  
  
 **int**[] fileBytes = image.getRGB(0, 0, image.getWidth(), image.getHeight(), **null**, 0, image.getWidth());

// чтение графической части файла  
 **short**[] keyArray = *getCodes*(key); // преобразование ключа  
  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = fileBytes[i] ^ keyArray[i % keyArray.**length**]; // преобразование  
 }  
  
 *createImgBuffFile*(fileBytes, file);  
 **return** *buffFile*;  
}

Существенное отличие имеется и в способе создания буферного файла. Для того чтобы картинку можно было восстановить, необходимо записать не только расширения, но и тип, ширину и высоту в пикселях, а также запись длины выводимой последовательности.

**private static void** createImgBuffFile(**int**[] fileBytes, File file) **throws** IOException {  
 BufferedImage image = ImageIO.*read*(file); // загрузка картинки  
 *createDemoImgFile*(fileBytes, file, image); // создание демонстрационного изображения  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"locked\_"** + *getMainName*(file) + **".vma"**); // создание буферного файла  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*)));

// создание выходного потока  
 bw.write(*getExtension*(file) + **"\n"**); // запись расширения  
 bw.write(image.getType() + **"\n"**); // запись типа  
 bw.write(image.getWidth() + **"\n"**); // запись ширины в пикселях  
 bw.write(image.getHeight() + **"\n"**); // запись высоты в пикселях  
 bw.write(fileBytes.**length** + **"\n"**); // запись длины выводимой последовательности (количество пикселей)  
  
 **for**(**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i] + **" "**); // запись последовательности  
 }  
  
 bw.close(); // закрытие потока  
}

**3.3.3** Текстовый дешифратор. Класс «TextDecypher»

Дешифратор расшифровывает файлы с расширением «.vma», внутри которого хранится изначальное расширение. После его проверки, если изначальный файл был текстовым, то файл передаётся в класс «TextDecypher». Основной метод класса:

**public static** File decrypt(File file, String key) **throws** IOException, BadLocationException {  
 **char**[] fileBytes = *getTextFileContent*(file).toCharArray(); // получение текстовых данных  
 **char**[] keyArray = key.toCharArray(); // преобразование ключа  
 fileBytes = *chooseDecryption*(fileBytes, keyArray, *encryptionMethod*); // преобразование данных в зависимости от выбранного метода  
 *createTextBuffFile*(fileBytes, file); // создание буферного файла  
 **return** *buffFile*;  
}

Считывание текстового файла:

**private static** String getTextFileContent(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 String in, content = **""**;  
 br.readLine(); // пропуск расширения  
 *encryptionMethod* = Integer.*parseInt*(br.readLine()); // считывание расширения  
 **while** ((in = br.readLine()) != **null**) {  
 content = content + in + **"\n"**; // чтение  
 }  
 br.close();  
 **return** content.substring(0, content.length() - 1); // возврат считанных данных кроме последнего символа \n  
}

Далее производится расшифровка и вызов метода «createTextBuffFile», который, в зависимости от расширения, вызывает соответствующий метод создания текстового файла.

**private static void** createTextBuffFile(**char**[] fileBytes, File file) **throws** IOException, BadLocationException {  
 **switch** (*getExtension*(file)) {  
 **case "doc"**:  
 **case "pdf"**:  
 **case "txt"**:  
 *createTxtBuffFile*(fileBytes, file);  
 **break**;  
 **case "html"**:  
 *createHTMLBuffFile*(fileBytes, file);  
 **break**;  
 **case "docx"**:  
 *createDocxBuffFile*(fileBytes, file);  
 **break**;  
 **case "rtf"**:  
 *createRTFBuffFile*(fileBytes, file);  
 **break**;  
 }  
}

Создание текстового файла с расширением «.txt», «.pdf» и «.doc»:

**private static void** createTxtBuffFile(**char**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".txt"**); // создание  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*), StandardCharsets.***UTF\_8***)); // создание выходного потока  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i]); // запись  
 }  
 bw.close(); // закрытие потока  
}

Создание текстового файла с расширением «.docx»:

**private static void** createDocxBuffFile(**char**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".docx"**); // создание буферного файла  
 XWPFDocument document = **new** XWPFDocument(); // создание экземпляра документа   
 String content = **""**;   
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 content = content + (fileBytes[i]); // создание выводного текста  
 }  
 XWPFParagraph bodyParagraph = document.createParagraph(); // создание экземпляра парграфа  
 XWPFRun paragraphConfig = bodyParagraph.createRun(); // подготовка к записи  
 paragraphConfig.setFontSize(14); // установка размера шрифта  
 paragraphConfig.setText(content); // запись выводного текста  
  
 FileOutputStream out = **new** FileOutputStream(*buffFile*); // создание выходного потока  
 document.write(out); // запись документ в буферный файл  
 out.close(); // закрытие потока  
}

Создание текстового файла с расширением «.rtf»:

**private static void** createRTFBuffFile(**char**[] fileBytes, File file) **throws** IOException, BadLocationException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".rtf"**); // создание файла  
 RTFEditorKit editorKit = **new** RTFEditorKit(); // создание экземпляра инструментария   
 Document doc = editorKit.createDefaultDocument(); // создание стандартного документа  
  
 String content = **""**;  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 content = content + (fileBytes[i]); // создание выводного текста  
 }  
 doc.insertString(0, content, **null**); // запись текста в документ  
 FileOutputStream fos = **new** FileOutputStream(*buffFile*); // создание выходного потока  
 editorKit.write(fos, doc, 0, doc.getLength()); // запись документа в файл  
 fos.close(); // закрытие потока  
}

**3.3.4** Графический дешифратор. Класс «ImageDecypher»

Если расшифровываемым файлом является изображение, то вызывается метод «decrypt» класса «ImageDecypher». Так как для изображений нет выбора метода шифрования и расширение изначального изображения никак не влияет на способ восстановления картинки, в классе предусмотрена только одна последовательность действий для расшифровки изображения.

**public static** File decrypt(File file, String key) **throws** IOException {  
 **int**[] fileBytes = *getImageFileContent*(file); // получение массива чисел из файла  
 **short**[] keyArray = *getCodes*(key); // преобразование ключа  
  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = fileBytes[i] ^ keyArray[i % keyArray.**length**]; // преобразование массива  
 }  
  
 *createImageBuffFile*(fileBytes, file); // создание буферного файла  
 **return** *buffFile*;  
}

Метод получения данных из файлов с расширением «.vma» в случае, если необходимо восстановить изображение:

**private static int**[] getImageFileContent(File file) **throws** IOException {  
 Scanner scanner = **new** Scanner(file); // создание сканнера  
 scanner.nextLine(); // пропуск строки с расширением  
 *imgType* = Integer.*parseInt*(scanner.nextLine()); // чтение типа изображения  
 *imgWidth* = Integer.*parseInt*(scanner.nextLine()); // чтение ширины изображения  
 *imgHeight* = Integer.*parseInt*(scanner.nextLine()); // чтение высоты изображения  
 **int**[] fileBytes = **new int**[Integer.*parseInt*(scanner.nextLine())]; // чтение длины последовательности  
 **int** i = 0;  
 **while** (scanner.hasNextInt()) {  
 fileBytes[i] = scanner.nextInt(); // чтение последовательности  
 i++;  
 }  
 scanner.close(); // закрытие сканнера  
 **return** fileBytes; // возврат  
}

Восстановление изображения:

**private static void** createImageBuffFile(**int**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **"."** + *getExtension*(file));

// создание файла  
 BufferedImage dstImage = **new** BufferedImage(*imgWidth*, *imgHeight*, *imgType*); // создание изображения  
 dstImage.setRGB(0, 0, *imgWidth*, *imgHeight*, fileBytes, 0, *imgWidth*); // установка графической части изображения  
 ImageIO.*write*(dstImage, *getExtension*(file), *buffFile*); // запись изображения в файл  
 Controller.*buffFile* = *buffFile*; // помещение файла в буфер контроллера  
}

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Заключительным этапом разработки приложения для шифрования файлов является тестирование основных функций приложения, а также тестирование пользовательского интерфейса. Тестирование проводилось на платформе Windows 10. Проведённые тесты:

1. Тестирование графической части. Приложение имеет определённые размеры, которые нельзя изменять.

Тестируемый код:

*root* = FXMLLoader.*load*(getClass().getResource(**"ui/sample.fxml"**)); // загрузка файла разметки  
primaryStage.setTitle(**"File encryption"**); // установка заголовка  
primaryStage.getIcons().add(**new** Image(**"app\\assets\\images\\mainLock.png"**));

// установка иконки в заголовке  
primaryStage.setScene(**new** Scene(*root*, 430, 225)); // создание окна определённых размеров  
primaryStage.setResizable(**false**); // запрет изменения размеров окна  
primaryStage.setMaximized(**false**); // отключение кнопки разворота на весь экран

Результат:

После запуска приложения не удалось растянуть размеры окна, а кнопка развёртывания на весь экран была неактивна (рисунок 4.1).

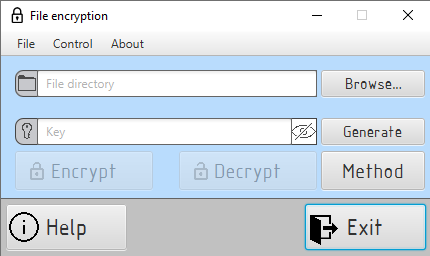


Рисунок 4.1 – Тест на возможность изменения размеров окна

2. Правильная запись скрытого и открытого ключа вне зависимости от способа ввода: ввода с клавиатуры, вставки из буфера обмена с помощью «Ctrl+V».

Тестируемый код:

**keyField**.textProperty().addListener((observable, oldValue, newValue) -> {  
 **keyPass**.setText(**keyPass**.getText()); // при любом изменении содержимого одного из полей изменения должны произойти и в другом поле  
 **keyPass**.positionCaret(**keyField**.getText().length()); // сохранение позиции каретки  
 **if** (**keyField**.getText().length() > 0) { // проверка на пустоту  
 enableBtns();  
 } **else** {  
 disableBtns();  
 }  
});  
  
**keyPass**.textProperty().addListener((observable, oldValue, newValue) -> {  
 **keyField**.setText(**keyPass**.getText());// при любом изменении содержимого одного из полей изменения должны произойти и в другом поле  
 **keyField**.positionCaret(**keyField**.getText().length()); // сохранение позиции каретки  
 **if** (**keyPass**.getText().length() > 0) {// проверка на пустоту  
 enableBtns();  
 } **else** {  
 disableBtns();  
 }  
});

Результат:

После вставки текста из буфера обмена текст в открытом виде стал аналогичным скрытому (рисунок 4.2, рисунок 4.3).



Рисунок 4.2 – Скрытый ключ



Рисунок 4.3 – Открытый ключ

3. Ключ должен генерироваться только из цифр и букв латинского алфавита.

Тестируемый код:

private String generateKey(int keyLength) {  
 String key = "";  
 Random randomizer = new Random();  
 for(int i = 0; i < keyLength; i++) {  
 key = key + nextChar((byte) Math.*abs*(randomizer.nextInt() % 128));  
 }  
 return key;  
}

Результат:

При любом количестве генераций комбинация состоит из цифр и латинских символов (рисунок 4.4).

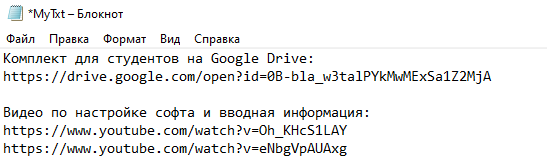


Рисунок 4.4 – Сгенерированный ключ

# 4. Зашифровка и расшифровка текстовых файлов с различными расширениями: «.txt», «.doc», «.html», «.rtf», «.docx», «.pdf». Документы формата «.txt», «.doc» и «.pdf» преобразуются в «.vma», а после в «.txt». Файлы с расширениями «.docx», «.html» и «.rtf» преобразуются в «.vma», а после в файлы с соответствующими расширениями.

Результат:

Тест успешно пройден (рисунок 4.5, рисунок 4.6, рисунок 4.7, рисунок 4.8).



# Рисунок 4.5 – Изначально содержание

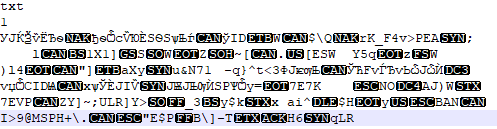


Рисунок 4.6 – Зашифрованный вид

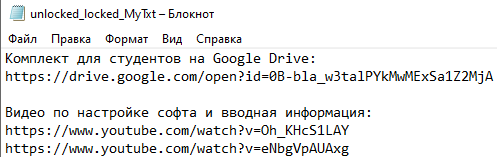


Рисунок 4.7 – Расшифрованный вид

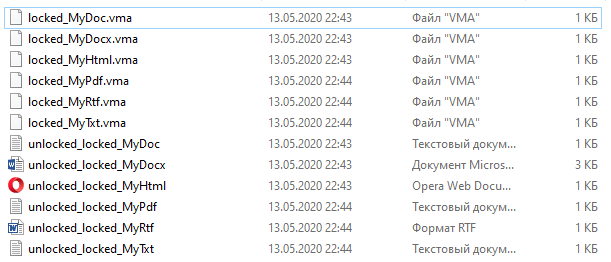


Рисунок 4.8 – Все файлы в тесте

5. Зашифровка и расшифровка изображения. Файл с расширением «.jpg» преобразуется в файл с расширением «.vma» и обратно.

Результат:

Тест пройден успешно (рисунок 4.9, рисунок 4.10, рисунок 4.11).

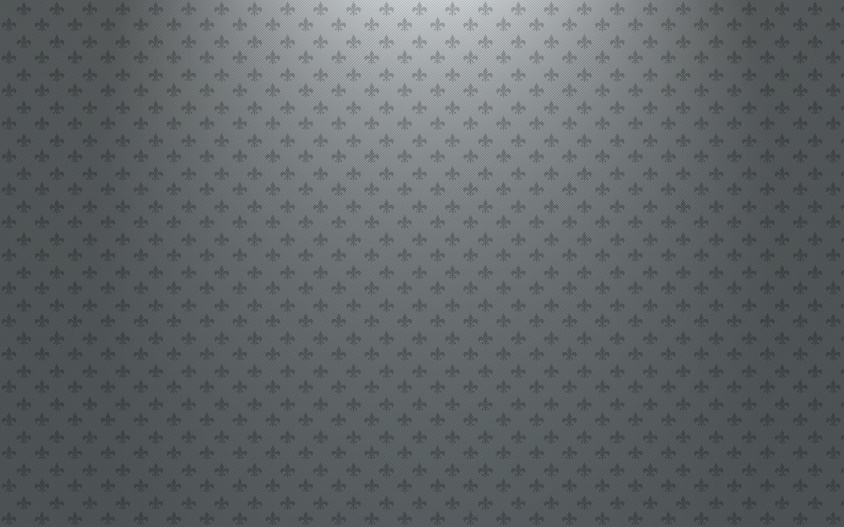


Рисунок 4.9 – Изначальное изображение

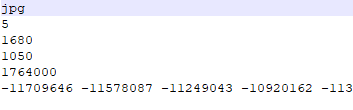


Рисунок 4.10 – Зашифрованный вид

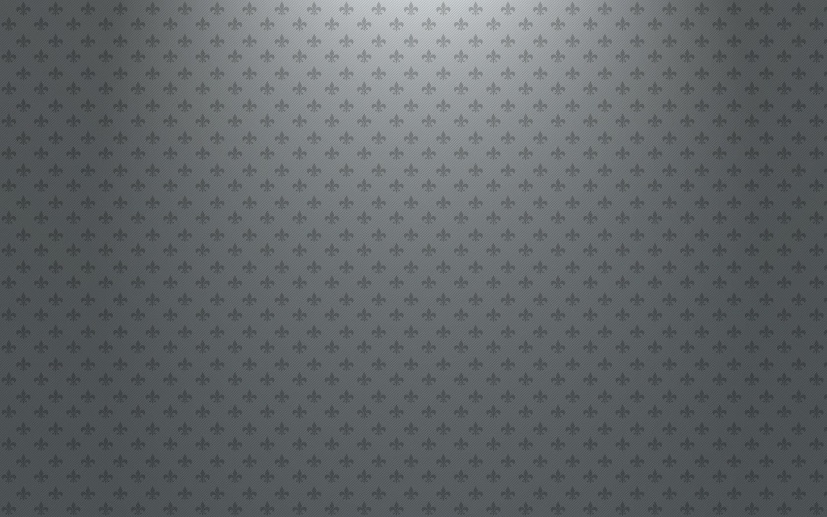


Рисунок 4.11 – Восстановленное изображение

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Запуск приложения осуществляется по средствам исполняемого файла. При открыти окна пользователю нужно выбрать файл, который необходимо зашифровать. Это можно сделать кликнув на кнопку «Browse…», на кнопку в меню, на иконку папки или с помощью сочетания клавиш «Alt+O». Шифратор поддерживает различные форматы текстовых и графических файлов, при выборе файла отображаются только разрешённые расширения.

Для зашифровки необходим ключ, который пользователь может ввести или же сгенерировать кликнув на кнопку «Generate», на кнопку в меню или с помощью сочетания клавиш «Alt+G». Ключ можно быстро скопировать нажав на иконку ключа, на кнопку в меню или с помощью сочетания клавиш «Alt+C». С помощью иконки глаза можно скрыть и открыть ключ.

Необязательным пунктом является выбор метода шифрования. Кликнув на кнопку «Method» можно вызвать диалоговое окно с выбором метода.

После того, как пользователь выбрал файл и ввёл ключ, шифратор готов к использованию (рисунок 5.1). Теперь осталось нажать на кнопку «Encrypt», на кнопку в меню или с помощью сочетания клавиш «Alt+E», чтобы зашифровать файл.

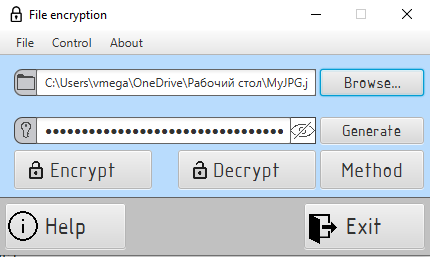


Рисунок 5.1 – Файл готов к зашифровке

Для расшифровки необходимо выбрать файл с расширением «.vma», ввести нужный ключ и нажать на кнопку «Decrypt», на кнопку в меню или с помощью сочетания клавиш «Alt+D» (рисунок 5.2).

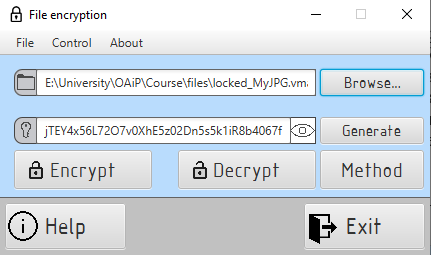


Рисунок 5.2 – Файл готов к расшифровке

# ЗАКЛЮЧЕНИЕ

Основная задача программиста – это работа с информацией. Изучение способов защиты данных есть обязательной частью его багажа профессиональных знаний. Цель данного курсового проекта – разработка шифратора, который поможет людям обезопасить свои данные.

Разработка данного приложения требовала познаний в области различных способов шифрования, понимания разницы при шифровании файлов различного расширения.

В ходе разработки были изучены возможности визуальной среды разработки IntelliJ IDEA, получены навыки создания графического пользовательского интерфейса с использованием JavaFx.

В результате выполнения данной курсовой работы разработан шифратор, который выполняет следующие функции:

* шифрование текста;
* шифрование изображений;
* шифрование различными методами;
* шифрование файлов с различным расширением.

Шифратор отлично подойдет для передачи личных сообщений или деловых документов.

Приложение имеет возможность совершенствования функционала и расширения возможностей. В приложение будут добавлены возможности шифрования видео и аудио, а также различных файлов с большим количеством расширений, дополнительные методы шифрования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Java. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.

[2] Java2. Практическое руководство.: практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2005. – 814с.

[3] Программирование на Java / Патрик Нимейер, Дэниэл Леук ; [пер. с англ. М. А. Райтмана]. – Москва : Эксмо, 2014. – 1216 с.

[4] Скотт К. Java для студента. – СПб.: БХВ-Петербург, 2007. – 448 с.

[5] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2018. – 464 с.: ил.

[6] Роберт Сенджвик, Кевин Уэйн. Алгоритмы на Java. – Москва.: Вильямс, 2012. – 848.

[7] Глухова, Л. А. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. / Л. А. Глухова, Е. Е. Фадеева, Е. П. Фадеева. – Минск: БГУИР, 2007. – Ч. 3. – 51 с.

[8] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.

[9] Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003.

# ПРИЛОЖЕНИЕ А

# Исходный код программы

**Main:**

**package** app;  
  
**import** app.ui.Controller;  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.Alert;  
**import** javafx.scene.control.ButtonType;  
**import** javafx.scene.image.Image;  
**import** javafx.scene.image.ImageView;  
**import** javafx.stage.Stage;  
  
**import** java.io.File;  
**import** java.util.Optional;  
  
**public class** Main **extends** Application {  
  
 **public static** Parent *root*;  
 **public static** String *rootDir* = **new** File(**""**).getAbsolutePath();  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 *root* = FXMLLoader.*load*(getClass().getResource(**"ui/sample.fxml"**));  
 primaryStage.setTitle(**"File encryption"**);  
 primaryStage.getIcons().add(**new** Image(**"app\\assets\\images\\mainLock.png"**));  
 primaryStage.setScene(**new** Scene(*root*, 430, 225));  
 primaryStage.setResizable(**false**);  
 primaryStage.setMaximized(**false**);  
  
 primaryStage.setOnCloseRequest(event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***CONFIRMATION***);  
 alert.setTitle(**"Exit"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Do you want to exit?"**);  
  
 ImageView image = **new** ImageView(**"app\\assets\\images\\warning.png"**);  
 image.setFitWidth(32);  
 image.setFitHeight(32);  
 alert.setGraphic(image);  
  
 Optional<ButtonType> result = alert.showAndWait();  
 **if** (result.isPresent() && result.get() == ButtonType.***CANCEL***) {  
 event.consume();  
 } **else** {  
 Controller.*buffFile* = **null**;  
 }  
 });  
  
 primaryStage.show();  
 }  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

**User Interface:**

**sample.fxml:**

*<?***xml version="1.0" encoding="UTF-8"***?>  
  
<?***import javafx.geometry.Insets***?>  
<?***import javafx.scene.Cursor***?>  
<?***import javafx.scene.control.Button***?>  
<?***import javafx.scene.control.Label***?>  
<?***import javafx.scene.control.Menu***?>  
<?***import javafx.scene.control.MenuBar***?>  
<?***import javafx.scene.control.MenuItem***?>  
<?***import javafx.scene.control.PasswordField***?>  
<?***import javafx.scene.control.SeparatorMenuItem***?>  
<?***import javafx.scene.control.TextField***?>  
<?***import javafx.scene.image.Image***?>  
<?***import javafx.scene.image.ImageView***?>  
<?***import javafx.scene.layout.Pane***?>  
<?***import javafx.scene.text.Font***?>*<**Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="225.0" prefWidth="430.0" style="-fx-background-color: #B9DCFD;" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="app.ui.Controller"**>  
 <**children**>  
 <**MenuBar prefHeight="25.0" prefWidth="430.0"**>  
 <**menus**>  
 <**Menu mnemonicParsing="false" text="File"**>  
 <**items**>  
 <**MenuItem fx:id="mmOpen" mnemonicParsing="false" text="Open..."** />  
 <**MenuItem fx:id="mmSave" mnemonicParsing="false" text="Save..."** />  
 <**SeparatorMenuItem mnemonicParsing="false"** />  
 <**MenuItem fx:id="mmExit" mnemonicParsing="false" text="Exit"** />  
 </**items**>  
 </**Menu**>  
 <**Menu mnemonicParsing="false" text="Control"**>  
 <**items**>  
 <**MenuItem fx:id="mmEncrypt" disable="true" mnemonicParsing="false" text="Encrypt"** />  
 <**MenuItem fx:id="mmDecrypt" disable="true" mnemonicParsing="false" text="Decrypt"** />  
 <**SeparatorMenuItem mnemonicParsing="false"** />  
 <**MenuItem fx:id="mmGenKey" mnemonicParsing="false" text="Generate key"** />  
 <**MenuItem fx:id="mmCopyKey" mnemonicParsing="false" text="Copy key"** />  
 <**MenuItem fx:id="mmChoose" mnemonicParsing="false" text="Choose method"** />  
 </**items**>  
 </**Menu**>  
 <**Menu mnemonicParsing="false" text="About"**>  
 <**items**>  
 <**MenuItem fx:id="mmHelp" mnemonicParsing="false" text="Help"** />  
 <**MenuItem fx:id="mmHotKeys" mnemonicParsing="false" text="Hot keys"** />  
 <**SeparatorMenuItem mnemonicParsing="false"** />  
 <**MenuItem fx:id="mmAboutProgram" mnemonicParsing="false" text="About program"** />  
 <**MenuItem fx:id="mmAboutAuthor" mnemonicParsing="false" text="About author"** />  
 </**items**>  
 </**Menu**>  
 </**menus**>  
 </**MenuBar**>  
 <**Pane layoutY="167.0" prefHeight="58.0" prefWidth="430.0" style="-fx-background-color: #C1C1C1; -fx-border-width: 1 0 0 0; -fx-border-color: #626262;"**>  
 <**children**>  
 <**Button fx:id="exitBtn" layoutX="304.0" layoutY="6.0" mnemonicParsing="false" prefHeight="46.0" prefWidth="121.0" text="Exit"**>  
 <**font**>  
 <**Font name="GOST Common" size="22.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**Button fx:id="helpBtn" layoutX="5.0" layoutY="6.0" mnemonicParsing="false" prefHeight="46.0" prefWidth="121.0" text="Help"**>  
 <**font**>  
 <**Font name="GOST Common" size="22.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**ImageView fitHeight="30.0" fitWidth="30.0" layoutX="8.0" layoutY="14.0" mouseTransparent="true" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/help.png"** />  
 </**image**></**ImageView**>  
 <**ImageView fitHeight="30.0" fitWidth="30.0" layoutX="308.0" layoutY="17.0" mouseTransparent="true" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/exit.png"** />  
 </**image**></**ImageView**>  
 <**Pane fx:id="hintPane" layoutX="140.0" layoutY="14.0" opacity="0.0" prefHeight="30.0" prefWidth="150.0" style="-fx-background-color: #F6F6D3 #F6F6D3; -fx-border-color: #3B3B3B; -fx-border-radius: 5; -fx-background-radius: 5;"**>  
 <**children**>  
 <**Label fx:id="myTextHint" alignment="CENTER" layoutY="8.0" prefHeight="17.0" prefWidth="150.0" text="TextHint" textFill="#000000fa"** />  
 </**children**>  
 </**Pane**>  
 </**children**>  
 <**opaqueInsets**>  
 <**Insets** />  
 </**opaqueInsets**>  
 </**Pane**>  
 <**Button fx:id="browseBtn" layoutX="320.0" layoutY="39.0" mnemonicParsing="false" prefHeight="25.0" prefWidth="104.0" text="Browse..."**>  
 <**font**>  
 <**Font name="GOST Common" size="15.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**Button fx:id="genBtn" layoutX="320.0" layoutY="87.0" mnemonicParsing="false" prefHeight="25.0" prefWidth="104.0" text="Generate"**>  
 <**font**>  
 <**Font name="GOST Common" size="15.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**Button fx:id="encryptBtn" disable="true" layoutX="14.0" layoutY="120.0" mnemonicParsing="false" prefHeight="25.0" prefWidth="138.0" text="Encrypt"**>  
 <**font**>  
 <**Font name="GOST Common" size="20.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**Button fx:id="decryptBtn" disable="true" layoutX="178.0" layoutY="120.0" mnemonicParsing="false" prefHeight="39.0" prefWidth="138.0" text="Decrypt"**>  
 <**font**>  
 <**Font name="GOST Common" size="20.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 <**ImageView fx:id="closeLockImg" fitHeight="18.0" fitWidth="15.0" layoutX="29.0" layoutY="131.0" mouseTransparent="true" opacity="0.2" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/lockClose.png"** />  
 </**image**>  
 </**ImageView**>  
 <**ImageView fx:id="openLockImg" fitHeight="18.0" fitWidth="15.0" layoutX="193.0" layoutY="131.0" mouseTransparent="true" opacity="0.2" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/lockOpen.png"** />  
 </**image**>  
 </**ImageView**>  
 <**Pane layoutX="14.0" layoutY="39.0" prefHeight="27.0" prefWidth="302.0" style="-fx-background-color: #C7C8CC; -fx-background-radius: 8; -fx-border-width: 1; -fx-border-color: #444444; -fx-border-radius: 8;"**>  
 <**children**>  
 <**TextField fx:id="fileField" layoutX="22.0" prefHeight="18.0" prefWidth="280.0" promptText="File directory" style="-fx-border-color: #656565;"** />  
 <**ImageView fx:id="folderBtn" accessibleRole="BUTTON" fitHeight="18.0" fitWidth="30.0" layoutX="3.0" layoutY="4.0" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/folder.png"** />  
 </**image**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**ImageView**>  
 </**children**>  
 </**Pane**>  
 <**Pane layoutX="14.0" layoutY="87.0" prefHeight="27.0" prefWidth="302.0" style="-fx-background-color: #C7C8CC; -fx-background-radius: 8; -fx-border-width: 1; -fx-border-color: #444444; -fx-border-radius: 8 0 0 8;"**>  
 <**children**>  
 <**ImageView fx:id="keyBtn" accessibleRole="BUTTON" fitHeight="18.0" fitWidth="30.0" layoutX="3.0" layoutY="4.0" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/3643767-key-keys-main-password-privilege\_113429.png"** />  
 </**image**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**ImageView**>  
 <**ImageView fx:id="eyeBtn" accessibleRole="BUTTON" fitHeight="25.0" fitWidth="25.0" layoutX="276.0" layoutY="1.0" pickOnBounds="true" preserveRatio="true"**>  
 <**image**>  
 <**Image url="@../assets/images/eye\_close.png"** />  
 </**image**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**ImageView**>  
 <**TextField fx:id="keyField" layoutX="22.0" prefHeight="18.0" prefWidth="255.0" promptText="Key" style="-fx-border-color: #656565;"** />  
 <**PasswordField fx:id="keyPass" layoutX="22.0" prefHeight="18.0" prefWidth="255.0" promptText="Key" style="-fx-border-color: #656565;"** />  
 </**children**>  
 </**Pane**>  
 <**Button fx:id="chooseBtn" layoutX="320.0" layoutY="120.0" mnemonicParsing="false" prefHeight="39.0" prefWidth="104.0" text="Method"**>  
 <**font**>  
 <**Font name="GOST Common" size="20.0"** />  
 </**font**>  
 <**cursor**>  
 <**Cursor fx:constant="HAND"** />  
 </**cursor**>  
 </**Button**>  
 </**children**>  
</**Pane**>

**Controller:**

**package** app.ui;  
  
**import** java.awt.\*;  
**import** java.awt.datatransfer.Clipboard;  
**import** java.awt.datatransfer.StringSelection;  
**import** java.io.\*;  
**import** java.net.URL;  
**import** java.nio.file.Files;  
**import** java.util.\*;  
  
**import** app.Main;  
**import** app.cypher.Cypher;  
**import** app.decypher.Decypher;  
**import** javafx.animation.AnimationTimer;  
**import** javafx.application.Platform;  
**import** javafx.event.ActionEvent;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.MenuItem;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.image.Image;  
**import** javafx.scene.image.ImageView;  
**import** javafx.scene.input.KeyCombination;  
**import** javafx.scene.input.MouseButton;  
**import** javafx.scene.input.MouseEvent;  
**import** javafx.scene.layout.Pane;  
**import** javafx.stage.FileChooser;  
  
**import** javax.swing.text.BadLocationException;  
  
**public class** Controller {  
  
 **private class** MyTimer **extends** AnimationTimer {  
 @Override  
 **public void** handle(**long** now) {  
 doHandle();  
 }  
  
 **private void** doHandle() {  
 **hintPane**.setOpacity(**hintPane**.getOpacity() - 0.02);  
 **if** (**hintPane**.getOpacity() <= 0) {  
 stop();  
 }  
 }  
 }  
  
 **public static** File *opFile*, *buffFile*;  
 **public static int** *encryptionMethod* = 1;  
  
 @FXML  
 **private** ResourceBundle **resources**;  
  
 @FXML  
 **private** URL **location**;  
  
 @FXML  
 **private** Button **exitBtn**;  
  
 @FXML  
 **private** Button **helpBtn**;  
  
 @FXML  
 **private** Button **browseBtn**;  
  
 @FXML  
 **private** Button **genBtn**;  
  
 @FXML  
 **private** Button **encryptBtn**;  
  
 @FXML  
 **private** Button **decryptBtn**;  
  
 @FXML  
 **private** Button **chooseBtn**;  
  
 @FXML  
 **private** ImageView **folderBtn**;  
  
 @FXML  
 **private** ImageView **keyBtn**;  
  
 @FXML  
 **private** ImageView **eyeBtn**;  
  
 @FXML  
 **private** MenuItem **mmOpen**;  
  
 @FXML  
 **private** MenuItem **mmSave**;  
  
 @FXML  
 **private** MenuItem **mmChoose**;  
  
 @FXML  
 **private** MenuItem **mmExit**;  
  
 @FXML  
 **private** MenuItem **mmEncrypt**;  
  
 @FXML  
 **private** MenuItem **mmDecrypt**;  
  
 @FXML  
 **private** MenuItem **mmGenKey**;  
  
 @FXML  
 **private** MenuItem **mmCopyKey**;  
  
 @FXML  
 **private** MenuItem **mmHelp**;  
  
 @FXML  
 **private** MenuItem **mmHotKeys**;  
  
 @FXML  
 **private** MenuItem **mmAboutProgram**;  
  
 @FXML  
 **private** MenuItem **mmAboutAuthor**;  
  
 @FXML  
 **private** TextField **keyField**;  
  
 @FXML  
 **private** TextField **fileField**;  
  
 @FXML  
 **private** PasswordField **keyPass**;  
  
 @FXML  
 **private** Label **myTextHint**;  
  
 @FXML  
 **private** Pane **hintPane**;  
  
 @FXML  
 **private** ImageView **closeLockImg**;  
  
 @FXML  
 **private** ImageView **openLockImg**;  
  
 @FXML  
 **void** initialize() {  
 EventHandler<ActionEvent> exitEvent = event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***CONFIRMATION***);  
 alert.setTitle(**"Exit"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Do you want to exit?"**);  
  
 ImageView image = **new** ImageView(**"app\\assets\\images\\warning.png"**);  
 image.setFitWidth(32);  
 image.setFitHeight(32);  
 alert.setGraphic(image);  
  
 Optional<ButtonType> result = alert.showAndWait();  
 **if** (result.isPresent() && result.get() == ButtonType.***OK***) {  
 *buffFile* = **null**;  
 Platform.*exit*();  
 }  
 };  
  
 EventHandler<ActionEvent> genEvent = event -> {  
 **keyField**.setText(generateKey(((**new** Random()).nextInt() % 5) + 35));  
 **keyPass**.setText(**keyField**.getText());  
 showHintWith(**"Key generated!"**);  
 };  
  
 EventHandler<ActionEvent> helpEvent = event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
 alert.setTitle(**"Help"**);  
 alert.setHeaderText(**"How to use"**);  
 alert.setContentText(**" Click on folder image, \"Browse\" button or \"Browse\" button in main menu to "** +  
 **"open file. Click on key image to copy key. Click on \"Generate\" to generate a key.\n Open file,"** +  
 **"enter key and click on \"encrypt\\decrypt\" to encrypt\\decrypt file."**);  
 alert.showAndWait();  
 };  
  
 EventHandler<MouseEvent> copyKeyEvent = mouseEvent -> {  
 **if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) {  
 StringSelection stringSelection = **new** StringSelection(**keyField**.getText());  
 Clipboard clipboard = Toolkit.*getDefaultToolkit*().getSystemClipboard();  
 clipboard.setContents(stringSelection, **null**);  
 showHintWith(**"Key was copied!"**);  
 }  
 };  
  
 EventHandler<ActionEvent> mmCopyKeyEvent = event -> {  
 StringSelection stringSelection = **new** StringSelection(**keyField**.getText());  
 Clipboard clipboard = Toolkit.*getDefaultToolkit*().getSystemClipboard();  
 clipboard.setContents(stringSelection, **null**);  
 showHintWith(**"Key was copied!"**);  
 };  
  
 EventHandler<ActionEvent> openFileEvent = event -> {  
 executeOpenFileDialog();  
 };  
  
 EventHandler<ActionEvent> encryptEvent = event -> {  
 **if** (**fileField**.getText().length() == 0) {  
 *alertWith*(**"Please, select a file first."**, **"Error"**, Alert.AlertType.***ERROR***);  
 } **else** {  
 **try** {  
 **if** (*opFile* == **null**) {  
 *buffFile* = Cypher.*executeEncrypt*(**new** File(**fileField**.getText()), **keyField**.getText());  
 } **else** {  
 *buffFile* = Cypher.*executeEncrypt*(*opFile*, **keyField**.getText());  
 }  
 showHintWith(**"File encrypted"**);  
 } **catch** (IOException | BadLocationException e) {  
 e.printStackTrace();  
 }  
 }  
 };  
  
 EventHandler<ActionEvent> decryptEvent = event -> {  
 **if** (**fileField**.getText().length() == 0) {  
 *alertWith*(**"Please, select a file first."**, **"Error"**, Alert.AlertType.***ERROR***);  
 } **else** {  
 **try** {  
 **if** (*opFile* == **null**) {  
 *buffFile* = Decypher.*executeDecrypt*(**new** File(**fileField**.getText()), **keyField**.getText());  
 } **else** {  
 *buffFile* = Decypher.*executeDecrypt*(*opFile*, **keyField**.getText());  
 }  
 showHintWith(**"File decrypted"**);  
 } **catch**(IOException | BadLocationException e){  
 e.printStackTrace();  
 }  
 }  
 };  
  
 EventHandler<ActionEvent> chooseEvent = event -> {  
 *methodTypeChoosing*();  
 };  
  
 **eyeBtn**.setOnMouseClicked(mouseEvent -> {  
 **if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) {  
 **if** (**keyPass**.isVisible()) {  
 **eyeBtn**.setImage(**new** Image(**"app\\assets\\images\\eye\_open.png"**));  
 **keyField**.positionCaret(**keyPass**.getCaretPosition());  
 **keyPass**.setVisible(**false**);  
 } **else** {  
 **eyeBtn**.setImage(**new** Image(**"app\\assets\\images\\eye\_close.png"**));  
 **keyPass**.positionCaret(**keyField**.getCaretPosition());  
 **keyPass**.setVisible(**true**);  
 }  
 }  
 });  
  
 **folderBtn**.setOnMouseClicked(mouseEvent -> {  
 **if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) {  
 executeOpenFileDialog();  
 }  
 });  
  
 **keyField**.textProperty().addListener((observable, oldValue, newValue) -> {  
 **keyPass**.setText(**keyPass**.getText());  
 **keyPass**.positionCaret(**keyField**.getText().length());  
 **if** (**keyField**.getText().length() > 0) {  
 enableBtns();  
 } **else** {  
 disableBtns();  
 }  
 });  
  
 **keyPass**.textProperty().addListener((observable, oldValue, newValue) -> {  
 **keyField**.setText(**keyPass**.getText());  
 **keyField**.positionCaret(**keyField**.getText().length());  
 **if** (**keyPass**.getText().length() > 0) {  
 enableBtns();  
 } **else** {  
 disableBtns();  
 }  
 });  
  
 **mmSave**.setOnAction(event -> {  
 **try** {  
 executeSaveFileDialog();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 });  
  
 **mmHotKeys**.setOnAction(event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
 alert.setTitle(**"Hot keys"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Alt+O - open file;\nAlt+S - save file;\nAlt+D - decrypt file;\n"** +  
 **"Alt+E - encrypt file;\nAlt+C - copy key;\nAlt+G - generate key;\nAlt+H - help;\n"** + **"Alt+K - hot keys;\nAlt+A - about author;\nAlt+P - about program;\nEsc - exit."**);  
 alert.showAndWait();  
 });  
  
 **mmAboutProgram**.setOnAction(event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
 alert.setTitle(**"About program"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"This application can encrypt and decrypt your text or media file."**);  
 alert.showAndWait();  
 });  
  
 **mmAboutAuthor**.setOnAction(event -> {  
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
 alert.setTitle(**"About author"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Application created by Matvey Vorivoda (Student of group 951007)."**);  
 alert.showAndWait();  
 });  
  
 **keyBtn**.setOnMouseClicked(copyKeyEvent);  
 **mmCopyKey**.setOnAction(mmCopyKeyEvent);  
  
 **exitBtn**.setOnAction(exitEvent);  
 **mmExit**.setOnAction(exitEvent);  
  
 **helpBtn**.setOnAction(helpEvent);  
 **mmHelp**.setOnAction(helpEvent);  
  
 **browseBtn**.setOnAction(openFileEvent);  
 **mmOpen**.setOnAction(openFileEvent);  
  
 **encryptBtn**.setOnAction(encryptEvent);  
 **mmEncrypt**.setOnAction(encryptEvent);  
  
 **decryptBtn**.setOnAction(decryptEvent);  
 **mmDecrypt**.setOnAction(decryptEvent);  
  
 **genBtn**.setOnAction(genEvent);  
 **mmGenKey**.setOnAction(genEvent);  
  
 **chooseBtn**.setOnAction(chooseEvent);  
 **mmChoose**.setOnAction(chooseEvent);  
  
 **mmChoose**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+M"**));  
 **mmExit**.setAccelerator(KeyCombination.*keyCombination*(**"Esc"**));  
 **mmOpen**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+O"**));  
 **mmSave**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+S"**));  
 **mmCopyKey**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+C"**));  
 **mmGenKey**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+G"**));  
 **mmDecrypt**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+D"**));  
 **mmEncrypt**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+E"**));  
 **mmHelp**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+H"**));  
 **mmHotKeys**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+K"**));  
 **mmAboutAuthor**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+A"**));  
 **mmAboutProgram**.setAccelerator(KeyCombination.*keyCombination*(**"Alt+P"**));  
 }  
  
private String generateKey(int keyLength) {  
 String key = "";  
 Random randomizer = new Random();  
 for(int i = 0; i < keyLength; i++) {  
 key = key + nextChar((byte) Math.*abs*(randomizer.nextInt() % 128));  
 }  
 return key;  
}  
  
private char nextChar(byte chr) {  
 if (chr > 57) {  
 if (chr > 90) {  
 chr = (byte) (97 + chr % 26);  
 } else {  
 chr = (byte) (65 + chr % 26);  
 }  
 } else {  
 chr = (byte) (48 + chr % 10);  
 }  
 return (char) (chr);  
}

**public static void** alertWith(String content, String title, Alert.AlertType type) {  
 Alert alert = **new** Alert(type);  
 alert.setTitle(title);  
 alert.setHeaderText(**null**);  
 alert.setContentText(content);  
 alert.showAndWait();  
 }  
  
 **public static void** methodTypeChoosing() {  
 ArrayList<String> lst = **new** ArrayList<>();  
 lst.add(**"XOR"**);  
 lst.add(**"Caesar"**);  
 ChoiceDialog<String> dialog = **new** ChoiceDialog<>(**"XOR"**, lst);  
  
 dialog.setTitle(**"Choice dialog"**);  
 dialog.setHeaderText(**"Encryption method for text files"**);  
 dialog.setContentText(**"Choose encryption method:"**);  
  
 Optional<String> result = dialog.showAndWait();  
 **if** (result.isPresent()) {  
 **switch** (result.get()) {  
 **case "XOR"**: *encryptionMethod* = 1; **break**;  
 **case "Caesar"**: *encryptionMethod* = 2; **break**;  
 }  
 }  
 }  
  
 **void** enableBtns() {  
 **encryptBtn**.setDisable(**false**);  
 **decryptBtn**.setDisable(**false**);  
 **mmEncrypt**.setDisable(**false**);  
 **mmDecrypt**.setDisable(**false**);  
 **openLockImg**.setOpacity(1);  
 **closeLockImg**.setOpacity(1);  
 }  
  
 **void** disableBtns() {  
 **encryptBtn**.setDisable(**true**);  
 **decryptBtn**.setDisable(**true**);  
 **mmEncrypt**.setDisable(**true**);  
 **mmDecrypt**.setDisable(**true**);  
 **openLockImg**.setOpacity(0.2);  
 **closeLockImg**.setOpacity(0.2);  
 }  
  
 **void** showHintWith(String content) {  
 **myTextHint**.setText(content);  
 **hintPane**.setOpacity(1);  
 (**new** Timer()).schedule(**new** TimerTask() {  
 @Override  
 **public void** run() {  
 AnimationTimer fading = **new** MyTimer();  
 fading.start();  
 }  
 }, 1000);  
 }  
  
 **void** executeSaveFileDialog() **throws** IOException {  
 FileChooser fileChooser = **new** FileChooser();  
 fileChooser.setTitle(**"Save file"**);  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Buffered file"**, **"\*."** + Cypher.*getExtension*(*buffFile*)));  
 *opFile* = fileChooser.showSaveDialog(Main.*root*.getScene().getWindow());  
 Files.*copy*(*buffFile*.toPath(), *opFile*.toPath());  
 }  
  
 **void** executeOpenFileDialog() {  
 FileChooser fileChooser = **new** FileChooser();  
 fileChooser.setTitle(**"Choose file"**);  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"All files"**, **"\*.txt"**, **"\*.doc"**, **"\*.docx"**, **"\*.html"**, **"\*.pdf"**, **"\*.rtf"**, **"\*.jpg"**, **"\*.jpeg"**, **"\*.bmp"**, **"\*.png"**, **"\*.mp3"**, **"\*.mp4"**, **"\*.avi"**, **"\*.mov"**, **"\*.flv"**, **"\*.vma"**));  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Encrypted files (vma)"**, **"\*.vma"**));  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Text files (txt, doc, docx, html, pdf, rtf)"**, **"\*.txt"**, **"\*.doc"**, **"\*.docx"**, **"\*.html"**, **"\*.pdf"**, **"\*.rtf"**));  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Images (jpg, jpeg, bmp, png)"**, **"\*.jpg"**, **"\*.jpeg"**, **"\*.bmp"**, **"\*.png"**));  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Audio files (mp3)"**, **"\*.mp3"**));  
 fileChooser.getExtensionFilters().add(**new** FileChooser.ExtensionFilter(**"Video files (mp4, avi, mov, flv)"**, **"\*.mp4"**, **"\*.avi"**, **"\*.mov"**, **"\*.flv"**));  
 *opFile* = fileChooser.showOpenDialog(Main.*root*.getScene().getWindow());  
 **if** (*opFile* != **null**) {  
 **fileField**.setText(*opFile*.getAbsolutePath());  
 }  
 }  
}

**Cypher:**

**package** app.cypher;  
  
**import** javax.swing.text.BadLocationException;  
**import** java.io.\*;  
**import** java.util.Arrays;  
**import** java.util.HashSet;  
  
**public class** Cypher {  
  
 **static final** HashSet<String> ***txtFiles*** = **new** HashSet<>(Arrays.*asList*(**"txt"**, **"doc"**, **"docx"**, **"html"**, **"pdf"**, **"rtf"**));  
 **static final** HashSet<String> ***imgFiles*** = **new** HashSet<>(Arrays.*asList*(**"jpg"**, **"jpeg"**, **"bmp"**, **"png"**));  
 **static final** HashSet<String> ***audioFiles*** = **new** HashSet<>(Arrays.*asList*(**"mp3"**, **"wav"**));  
 **static final** HashSet<String> ***videoFiles*** = **new** HashSet<>(Arrays.*asList*(**"mp4"**, **"avi"**, **"mov"**, **"flv"**));  
  
 **public static** File executeEncrypt(File file, String key) **throws** IOException, BadLocationException {  
 String ext = *getExtension*(file);  
 **if** (***txtFiles***.contains(ext)) {  
 **return** TextCypher.*encrypt*(file, key);  
 }  
 **if** (***imgFiles***.contains(ext)) {  
 **return** ImageCypher.*encrypt*(file, key);  
 }  
 **if** (***audioFiles***.contains(ext)) {  
 *//* }  
 **if** (***videoFiles***.contains(ext)) {  
 *//* }  
 **return null**;  
 }  
  
 **public static** String getExtension(File file) {  
 **return** file.getName().substring(file.getName().lastIndexOf(**'.'**) + 1);  
 }  
}

**TextCypher:**

**package** app.cypher;  
  
**import** app.ui.Controller;  
**import** app.Main;  
**import** org.apache.pdfbox.pdmodel.PDDocument;  
**import** org.apache.pdfbox.text.PDFTextStripper;  
**import** org.apache.poi.hwpf.HWPFDocument;  
**import** org.apache.poi.hwpf.extractor.WordExtractor;  
**import** org.apache.poi.xwpf.usermodel.XWPFDocument;  
**import** org.apache.poi.xwpf.usermodel.XWPFParagraph;  
  
**import** javax.swing.text.BadLocationException;  
**import** javax.swing.text.Document;  
**import** javax.swing.text.rtf.RTFEditorKit;  
**import** java.io.\*;  
**import** java.nio.charset.StandardCharsets;  
**import** java.util.List;  
  
**public class** TextCypher {  
  
 **static** File *buffFile*;  
  
 **public static** File encrypt(File file, String key) **throws** IOException, BadLocationException {  
 **char**[] fileBytes = *getTextFileContent*(file).toCharArray();  
 **char**[] keyArray = key.toCharArray();  
 fileBytes = *chooseEncryption*(fileBytes, keyArray, Controller.*encryptionMethod*);  
 *createTextBuffFile*(fileBytes, file, Controller.*encryptionMethod*);  
 **return** *buffFile*;  
 }  
  
 **private static char**[] chooseEncryption(**char**[] fileBytes, **char**[] keyArray, **int** method) {  
 **switch** (method) {  
 **case** 1: **return** *XOREncryption*(fileBytes, keyArray);  
 **case** 2: **return** *CaesarEncryption*(fileBytes, keyArray);  
 }  
 **return null**;  
 }  
  
 **private static char**[] CaesarEncryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = (**char**) ((fileBytes[i] + keyArray[i % keyArray.**length**]) % 65535);  
 }  
 **return** fileBytes;  
 }  
  
 **private static char**[] XOREncryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = (**char**) (fileBytes[i] ^ keyArray[i % keyArray.**length**]);  
 }  
 **return** fileBytes;  
 }  
  
 **private static** String getTextFileContent(File file) **throws** IOException, BadLocationException {  
 **switch** (*getExtension*(file)) {  
 **case "txt"**:  
 **case "html"**:  
 **return** *getTxtFileContent*(file);  
 **case "doc"**:  
 **return** *getDocFileContent*(file);  
 **case "docx"**:  
 **return** *getDocxFileContent*(file);  
 **case "pdf"**:  
 **return** *getPDFFileContent*(file);  
 **case "rtf"**:  
 **return** *getRTFFileContent*(file);  
 }  
 **return ""**;  
 }  
  
 **private static void** createTextBuffFile(**char**[] fileBytes, File file, **int** method) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"locked\_"** + *getMainName*(file) + **".vma"**);  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*), StandardCharsets.***UTF\_8***));  
 bw.write(*getExtension*(file) + **"\n"**);  
 bw.write(method + **"\n"**);  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i]);  
 }  
 bw.close();  
 }  
  
 **private static** String getRTFFileContent(File file) **throws** IOException, BadLocationException {  
 RTFEditorKit editorKit = **new** RTFEditorKit();  
 Document doc = editorKit.createDefaultDocument();  
 editorKit.read(**new** FileInputStream(file), doc, 0);  
 **return** doc.getText(0, doc.getLength());  
 }  
  
 **private static** String getTxtFileContent(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 String in, content = **""**;  
 **while** ((in = br.readLine()) != **null**) {  
 content = content + in + **"\n"**;  
 }  
 br.close();  
 **return** content.substring(0, content.length() - 1);  
 }  
  
 **private static** String getPDFFileContent(File file) **throws** IOException {  
 PDDocument reader = PDDocument.*load*(file);  
 PDFTextStripper ts = **new** PDFTextStripper();  
 ts.setStartPage(1);  
 ts.setEndPage(reader.getNumberOfPages());  
 String answer = ts.getText(reader);  
 reader.close();  
 **return** answer;  
 }  
  
 **private static** String getDocFileContent(File file) **throws** IOException {  
 WordExtractor extractor = **new** WordExtractor(**new** HWPFDocument(**new** FileInputStream(file)));  
 String[] fileData = extractor.getParagraphText();  
 String answer = **""**;  
 **for** (**int** i = 0; i < fileData.**length**; i++) {  
 answer = answer + fileData[i];  
 }  
 extractor.close();  
 **return** answer;  
 }  
  
 **private static** String getDocxFileContent(File file) **throws** IOException {  
 List<XWPFParagraph> paragraphs = **new** XWPFDocument(**new** FileInputStream(file)).getParagraphs();  
 String answer = **""**;  
 **for** (**int** i = 0; i < paragraphs.size(); i++) {  
 answer = answer + paragraphs.get(i).getParagraphText() + **"\n"**;  
 }  
 **return** answer;  
 }  
  
 **public static** String getExtension(File file) {  
 **return** file.getName().substring(file.getName().lastIndexOf(**'.'**) + 1);  
 }  
  
 **public static** String getMainName(File file) {  
 **return** file.getName().substring(0, file.getName().lastIndexOf(**'.'**));  
 }  
}

**ImageCypher:**

**package** app.cypher;  
  
**import** app.ui.Controller;  
**import** app.Main;  
  
**import** javax.imageio.ImageIO;  
**import** java.awt.image.BufferedImage;  
**import** java.io.\*;  
  
**public class** ImageCypher {  
  
 **static** File *buffFile*;  
  
 **public static** File encrypt(File file, String key) **throws** IOException {  
 BufferedImage image = ImageIO.*read*(file);  
  
 **int**[] fileBytes = image.getRGB(0, 0, image.getWidth(), image.getHeight(), **null**, 0, image.getWidth());  
 **short**[] keyArray = *getCodes*(key);  
  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = fileBytes[i] ^ keyArray[i % keyArray.**length**];  
 }  
  
 *createImgBuffFile*(fileBytes, file);  
 **return** *buffFile*;  
 }  
  
 **private static void** createImgBuffFile(**int**[] fileBytes, File file) **throws** IOException {  
 BufferedImage image = ImageIO.*read*(file);  
 *createDemoImgFile*(fileBytes, file, image);  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"locked\_"** + *getMainName*(file) + **".vma"**);  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*)));  
 bw.write(*getExtension*(file) + **"\n"**);  
 bw.write(image.getType() + **"\n"**);  
 bw.write(image.getWidth() + **"\n"**);  
 bw.write(image.getHeight() + **"\n"**);  
 bw.write(fileBytes.**length** + **"\n"**);  
  
 **for**(**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i] + **" "**);  
 }  
  
 bw.close();  
 }  
  
 *// DEMONSTRATION FUNCTION* **private static void** createDemoImgFile(**int**[] fileBytes, File file, BufferedImage image) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **"."** + *getExtension*(file));  
 BufferedImage dstImage = **new** BufferedImage(image.getWidth(), image.getHeight(), image.getType());  
 dstImage.setRGB(0, 0, image.getWidth(), image.getHeight(), fileBytes, 0, image.getWidth());  
 ImageIO.*write*(dstImage, *getExtension*(file), *buffFile*);  
 Controller.*buffFile* = *buffFile*;  
 }  
 *// DEMONSTRATION FUNCTION* **public static** String getMainName(File file) {  
 **return** file.getName().substring(0, file.getName().lastIndexOf(**'.'**));  
 }  
  
 **public static** String getExtension(File file) {  
 **return** file.getName().substring(file.getName().lastIndexOf(**'.'**) + 1);  
 }  
  
 **private static short**[] getCodes(String content) {  
 **short**[] answer = **new short**[content.length()];  
 **for** (**int** i = 0; i < answer.**length**; i++) {  
 answer[i] = (**short**) content.charAt(i);  
 }  
 **return** answer;  
 }  
}

**Decypher:**

**package** app.decypher;  
  
**import** javax.swing.text.BadLocationException;  
**import** java.io.\*;  
**import** java.nio.charset.StandardCharsets;  
**import** java.util.Arrays;  
**import** java.util.HashSet;  
  
**public class** Decypher {  
  
 **static final** HashSet<String> ***txtFiles*** = **new** HashSet<>(Arrays.*asList*(**"txt"**, **"doc"**, **"docx"**, **"html"**, **"pdf"**, **"rtf"**));  
 **static final** HashSet<String> ***imgFiles*** = **new** HashSet<>(Arrays.*asList*(**"jpg"**, **"jpeg"**, **"bmp"**, **"png"**, **"ico"**, **"svg"**));  
 **static final** HashSet<String> ***audioFiles*** = **new** HashSet<>(Arrays.*asList*(**"mp3"**, **"wav"**));  
 **static final** HashSet<String> ***videoFiles*** = **new** HashSet<>(Arrays.*asList*(**"mp4"**, **"avi"**, **"mov"**, **"flv"**));  
  
 **public static** File executeDecrypt(File file, String key) **throws** IOException, BadLocationException {  
 String ext = *getExtension*(file);  
 **if** (***txtFiles***.contains(ext)) {  
 **return** TextDecypher.*decrypt*(file, key);  
 }  
 **if** (***imgFiles***.contains(ext)) {  
 **return** ImageDecypher.*decrypt*(file, key);  
 }  
 **if** (***audioFiles***.contains(ext)) {  
 *//* }  
 **if** (***videoFiles***.contains(ext)) {  
 *//* }  
 **return null**;  
 }  
  
 **public static** String getExtension(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 **return** br.readLine().trim();  
 }  
}

**TextDecypher:**

**package** app.decypher;  
  
**import** app.Main;  
**import** org.apache.poi.xwpf.usermodel.XWPFDocument;  
**import** org.apache.poi.xwpf.usermodel.XWPFParagraph;  
**import** org.apache.poi.xwpf.usermodel.XWPFRun;  
  
**import** javax.swing.text.BadLocationException;  
**import** javax.swing.text.Document;  
**import** javax.swing.text.rtf.RTFEditorKit;  
**import** java.io.\*;  
**import** java.nio.charset.StandardCharsets;  
  
**public class** TextDecypher {  
  
 **static** File *buffFile*;  
 **static int** *encryptionMethod*;  
  
 **public static** File decrypt(File file, String key) **throws** IOException, BadLocationException {  
 **char**[] fileBytes = *getTextFileContent*(file).toCharArray();  
 **char**[] keyArray = key.toCharArray();  
 fileBytes = *chooseDecryption*(fileBytes, keyArray, *encryptionMethod*);  
 *createTextBuffFile*(fileBytes, file);  
 **return** *buffFile*;  
 }  
  
 **private static char**[] chooseDecryption(**char**[] fileBytes, **char**[] keyArray, **int** method) {  
 **switch** (method) {  
 **case** 1: **return** *XORDecryption*(fileBytes, keyArray);  
 **case** 2: **return** *CaesarDecryption*(fileBytes, keyArray);  
 }  
 **return null**;  
 }  
  
 **private static char**[] CaesarDecryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 **int** temp;  
 **if** ((temp = fileBytes[i] - keyArray[i % keyArray.**length**]) < 0) {  
 fileBytes[i] = (**char**) (65535 + temp);  
 } **else** {  
 fileBytes[i] = (**char**) temp;  
 }  
 }  
 **return** fileBytes;  
 }  
  
 **private static char**[] XORDecryption(**char**[] fileBytes, **char**[] keyArray) {  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = (**char**) (fileBytes[i] ^ keyArray[i % keyArray.**length**]);  
 }  
 **return** fileBytes;  
 }  
  
 **private static** String getTextFileContent(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 String in, content = **""**;  
 br.readLine();  
 *encryptionMethod* = Integer.*parseInt*(br.readLine());  
 **while** ((in = br.readLine()) != **null**) {  
 content = content + in + **"\n"**;  
 }  
 br.close();  
 **return** content.substring(0, content.length() - 1);  
 }  
  
 **private static void** createTextBuffFile(**char**[] fileBytes, File file) **throws** IOException, BadLocationException {  
 **switch** (*getExtension*(file)) {  
 **case "doc"**:  
 **case "pdf"**:  
 **case "txt"**:  
 *createTxtBuffFile*(fileBytes, file);  
 **break**;  
 **case "html"**:  
 *createHTMLBuffFile*(fileBytes, file);  
 **break**;  
 **case "docx"**:  
 *createDocxBuffFile*(fileBytes, file);  
 **break**;  
 **case "rtf"**:  
 *createRTFBuffFile*(fileBytes, file);  
 **break**;  
 }  
 }  
  
 **private static void** createRTFBuffFile(**char**[] fileBytes, File file) **throws** IOException, BadLocationException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".rtf"**);  
 RTFEditorKit editorKit = **new** RTFEditorKit();  
 Document doc = editorKit.createDefaultDocument();  
  
 String content = **""**;  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 content = content + (fileBytes[i]);  
 }  
 doc.insertString(0, content, **null**);  
 FileOutputStream fos = **new** FileOutputStream(*buffFile*);  
 editorKit.write(fos, doc, 0, doc.getLength());  
 fos.close();  
 }  
  
 **private static void** createDocxBuffFile(**char**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".docx"**);  
 XWPFDocument document = **new** XWPFDocument();  
 String content = **""**;  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 content = content + (fileBytes[i]);  
 }  
 XWPFParagraph bodyParagraph = document.createParagraph();  
 XWPFRun paragraphConfig = bodyParagraph.createRun();  
 paragraphConfig.setFontSize(14);  
 paragraphConfig.setText(content);  
  
 FileOutputStream out = **new** FileOutputStream(*buffFile*);  
 document.write(out);  
 out.close();  
 }  
  
 **private static void** createHTMLBuffFile(**char**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".html"**);  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*), StandardCharsets.***UTF\_8***));  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i]);  
 }  
 bw.close();  
 }  
  
 **private static void** createTxtBuffFile(**char**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **".txt"**);  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(*buffFile*), StandardCharsets.***UTF\_8***));  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 bw.write(fileBytes[i]);  
 }  
 bw.close();  
 }  
  
 **public static** String getMainName(File file) {  
 **return** file.getName().substring(0, file.getName().lastIndexOf(**'.'**));  
 }  
  
 **public static** String getExtension(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 **return** br.readLine().trim();  
 }  
}

**ImageDecypher:**

**package** app.decypher;  
  
**import** app.ui.Controller;  
**import** app.Main;  
  
**import** javax.imageio.ImageIO;  
**import** java.awt.image.BufferedImage;  
**import** java.io.\*;  
**import** java.nio.charset.StandardCharsets;  
**import** java.util.Scanner;  
  
**public class** ImageDecypher {  
  
 **static** File *buffFile*;  
 **static int** *imgWidth*, *imgHeight*, *imgType*;  
  
 **public static** File decrypt(File file, String key) **throws** IOException {  
 **int**[] fileBytes = *getImageFileContent*(file);  
 **short**[] keyArray = *getCodes*(key);  
  
 **for** (**int** i = 0; i < fileBytes.**length**; i++) {  
 fileBytes[i] = fileBytes[i] ^ keyArray[i % keyArray.**length**];  
 }  
  
 *createImageBuffFile*(fileBytes, file);  
 **return** *buffFile*;  
 }  
  
 **private static void** createImageBuffFile(**int**[] fileBytes, File file) **throws** IOException {  
 *buffFile* = **new** File(Main.*rootDir* + **"/files"**, **"unlocked\_"** + *getMainName*(file) + **"."** + *getExtension*(file));  
 BufferedImage dstImage = **new** BufferedImage(*imgWidth*, *imgHeight*, *imgType*);  
 dstImage.setRGB(0, 0, *imgWidth*, *imgHeight*, fileBytes, 0, *imgWidth*);  
 ImageIO.*write*(dstImage, *getExtension*(file), *buffFile*);  
 Controller.*buffFile* = *buffFile*;  
 }  
  
 **private static int**[] getImageFileContent(File file) **throws** IOException {  
 Scanner scanner = **new** Scanner(file);  
 scanner.nextLine();  
 *imgType* = Integer.*parseInt*(scanner.nextLine());  
 *imgWidth* = Integer.*parseInt*(scanner.nextLine());  
 *imgHeight* = Integer.*parseInt*(scanner.nextLine());  
 **int**[] fileBytes = **new int**[Integer.*parseInt*(scanner.nextLine())];  
 **int** i = 0;  
 **while** (scanner.hasNextInt()) {  
 fileBytes[i] = scanner.nextInt();  
 i++;  
 }  
 scanner.close();  
 **return** fileBytes;  
 }  
  
 **private static short**[] getCodes(String content) {  
 **short**[] answer = **new short**[content.length()];  
 **for** (**int** i = 0; i < answer.**length**; i++) {  
 answer[i] = (**short**) content.charAt(i);  
 }  
 **return** answer;  
 }  
  
 **public static** String getExtension(File file) **throws** IOException {  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file), StandardCharsets.***UTF\_8***));  
 **return** br.readLine().trim();  
 }  
  
 **public static** String getMainName(File file) {  
 **return** file.getName().substring(0, file.getName().lastIndexOf(**'.'**));  
 }  
}

# 

# ПРИЛОЖЕНИЕ Б

# Блок-схема методов encrypt, generateKey