**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 для вложения   
текстовой информации в картинки»

Выполнил

Колесник София Андреевна

Студент группы:

ИБС-22

Проверил:

Старший преподаватель

Цветков Александр Юрьевич

Оглавление

1. [Введение 2](#_Toc153742758)
2. [Основные концепции и определения 2](#_Toc153742759)
3. [Обзор разработанного приложения 7](#_Toc153742760)
4. [Работа приложения 10](#_Toc153742761)
5. [Примеры некорректной работы приложения 18](#_Toc153742762)
6. [Использование библиотек 20](#_Toc153742763)
7. [UML-диаграмма классов 23](#_Toc153742764)
8. [UML-диаграмма модулей 23](#_Toc153742765)
9. [Безопасность и надежность программы 24](#_Toc153742766)
10. [Листинг кода 26](#_Toc153742767)
11. [Источники 37](#_Toc153742768)
12. [Список литературы 38](#_Toc153742769)

Введение

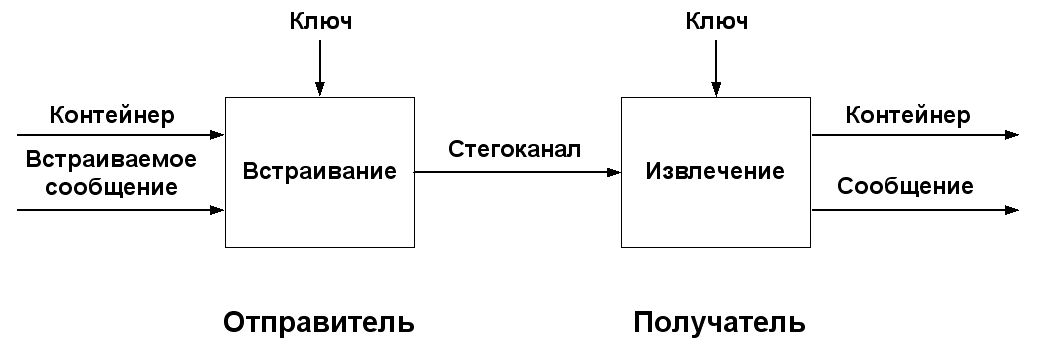
Цель курсовой работы заключается в разработке программы на языке программирования Java, которая будет осуществлять вложение текстовой информации в BMP-файл, c глубиной 24 бит на пиксель, по алгоритму НЗБ (Наименьший Значащий Бит). Программа должна обладать графическим интерфейсом, который будет предоставлять пользователю возможность выбора пути к оригинальному изображению и к изображению с вложением. Кроме того, интерфейс должен позволять отображать оригинал картинки и картинку с вложением, причем с отображением только младших битов.

Используемый метод позволит минимизировать визуальное влияние на изображение, делая вложение почти незаметным для глаза человека. В результате встроенная информация становится недоступной без использования специальных методов извлечения, что относительно обеспечивает её конфиденциальность. Более подробно алгоритм LSB описан в пункте «Основные концепции и определения».

Основные концепции и определения

**Стеганография**

Стеганография – способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи. Основным отличием стеганографии от криптографии, которая занимается шифрованием данных для обеспечения их конфиденциальности, является то, что стеганография скрывает не только содержимое, но и сам факт наличия и передачи информации. С помощью стеганографии можно вставить секретную информацию в различные файлы, например, в видео, изображение, звуковой файл или текстовый документ, без внесения заметных изменений. При этом полученный носитель называется стегоконтейнером.



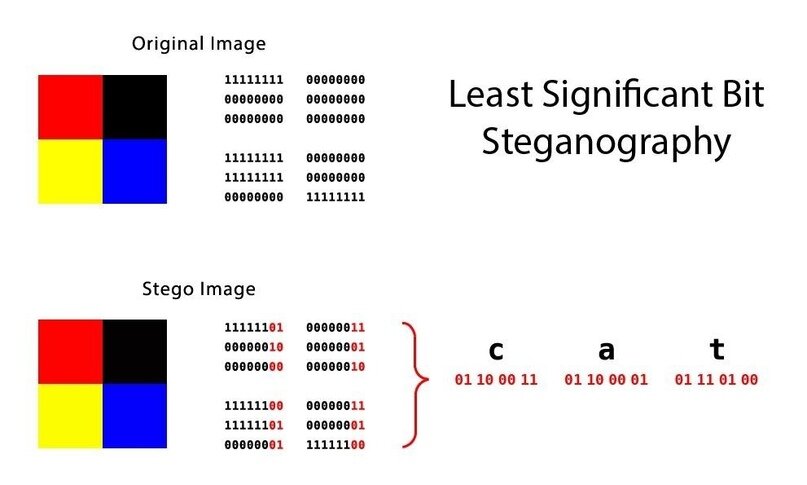
Стеганографические методы обычно используют различные техники встраивания данных в носитель. Методы могут использовать изменение спектральных характеристик звуковых или видеоданных, внедрение сообщений в whitespace (пробелы) текстовых документов, и т.д. Одним из наиболее распространённых методов является метод LSB (Last Significant Bit).

**Метод Last Significant Bit**

Метод "Last Significant Bit" (LSB) — это метод стеганографии, который позволяет скрыть информацию внутри изображения путем замены наименее значимых битов пикселей. Данный метод основывается на том, что изменение наименее значимых битов цветовых компонент пикселей обычно не заметно для человеческого глаза.

Для применения метода LSB необходимо выполнить несколько шагов. Первым шагом является выбор изображения-носителя, в котором будет скрыта информация. Обычно используются изображения в формате BMP или PNG. Затем выбирается секретное сообщение, которое будет встроено в изображение.

Для каждого пикселя изображения применяется следующий алгоритм. Сначала получают значения цветовых компонент (красного, зеленого и синего) пикселя. При наличии альфа-канала запись может происходить в том числе и в него. Затем наименее значимый бит каждой цветовой компоненты заменяется битом из секретного сообщения. Таким образом, каждый пиксель содержит новую информацию, встроенную в его наименее значимые биты. Количество заменяемых битов можно увеличить. Как правило меняют 2 бита, поскольку мониторы не способны отобразить столь незначительную разницу. При большем количестве заменяемых битов изображение будет содержать больше помех, спровоцированных заменой.



После обработки всех пикселей, полученное измененное изображение может быть сохранено и передано получателю. Для извлечения скрытой информации изображение-носитель обрабатывается аналогичным образом. Наименее значимые биты каждого пикселя извлекаются и собираются вместе, что позволяет восстановить секретное сообщение.

Одним из основных преимуществ метода LSB является его простота и незаметность для наблюдателя. Однако этот метод имеет некоторые недостатки. Во-первых, он относительно неэффективен, поскольку объем скрытой информации ограничен количеством пикселей в изображении. Во-вторых, изображения, подвергшиеся стеганографии с использованием метода LSB, могут быть подвержены различным атакам на извлечение информации, таким анализ корреляций между пикселями.

В целом, метод LSB является одним из самых простых и наиболее распространенных методов стеганографии, но его область его применения сильно ограничена за счёт малой фиксированной величины скрываемой информации и уязвимости к атакам (стегоанализу). Для повышения стойкости кодирования в качестве исходного изображения берутся изображения с высоким разбросом (шумом) наименьших битов, внедрение происходит в случайные биты, и кодируемая информация берётся малого объёма. Стоит отметить, что в данной курсовой работе пиксели изменяются последовательно с начала, что делает шифрование нестойким.

В Интернете существуют множество утилит, способных декодировать спрятанные по методу LSB сообщения. Одни из самых распространённых: stegano, openstego, LSBSteg.

**Файловый формат .bmp**

Файлы с расширением .BMP — это файлы, содержащие растровые изображения, созданные для хранения цифровых изображений. Они не зависят от графического адаптера и называются "форматом файла растрового изображения, независимого от устройства" (DIB). Это означает, что файлы .BMP могут быть открыты на разных платформах, таких как Microsoft Windows и Mac. Формат BMP позволяет хранить двумерные цифровые изображения как в монохромном, так и в цветном формате с разной глубиной цвета.

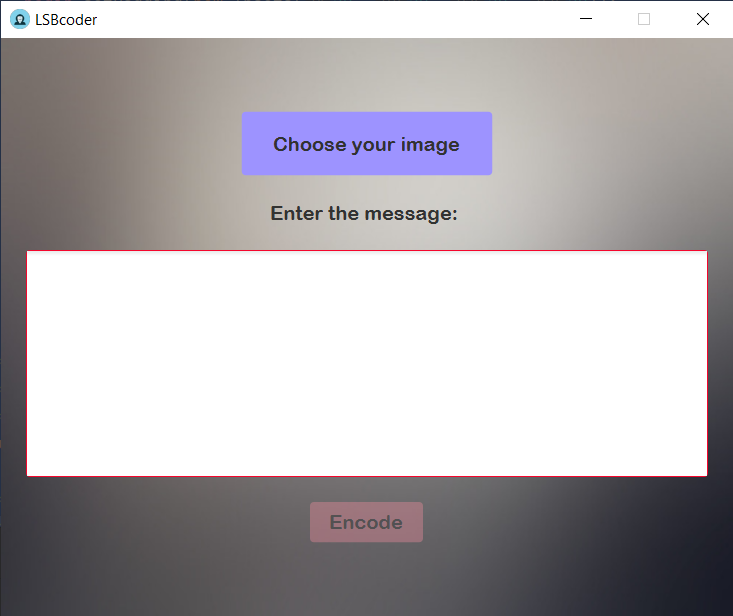
Рассмотрим структуру данного файла. Можно условно разделить его на 4 части: заголовок файла, заголовок изображения, палитру и само изображение. Для наших целей нам необходимо знать только информацию, записанную в заголовке.

Первые два байта заголовка содержат сигнатуру BM, затем размер файла в байтах записан в двойном слове, следующие 4 байта зарезервированы и должны содержать нули, и, наконец, смещение от начала файла до байтов изображения записано в еще одном двойном слове. В 24-битном bmp-файле каждый пиксель кодируется тремя байтами BGR.

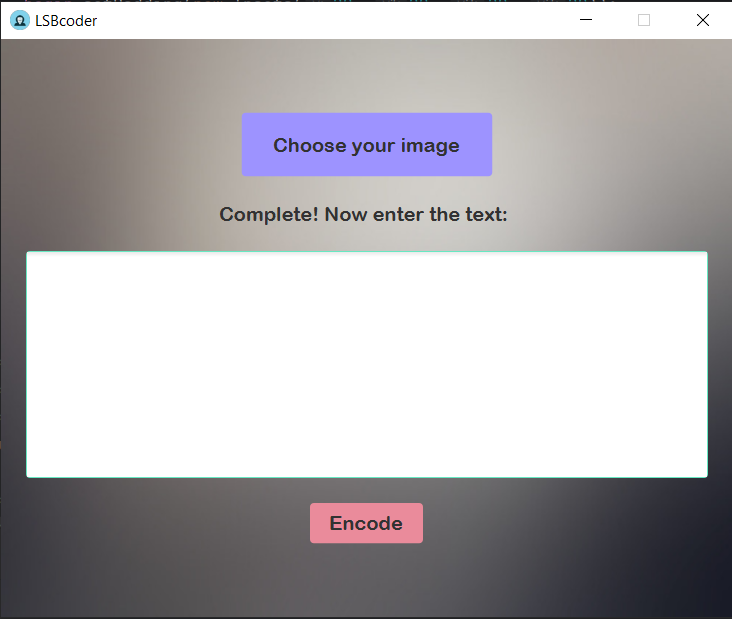
|  |  |  |  |
| --- | --- | --- | --- |
| **Структура** | **Необязательно** | **Размер** | **Назначение** |
| Заголовок файла | Нет | 14 | Для хранения общей информации о файле растрового изображения |
| Заголовок DIB | Нет | Фиксированный размер | Для хранения подробной информации о растровом изображении и определения формата пикселей |
| Дополнительные битовые маски | Да | 12 или 16 байт | Для определения формата пикселей |
| Цветовая палитра | Да | Переменный размер | Для определения цветов, используемых данными растрового изображения |
| Gap1 | Да | Размер переменной | Выравнивание структуры |
| Массив пикселей | Нет | Переменный размер | Формат пикселей определяется заголовком DIB или дополнительными битовыми масками. |
| Gap2 | Да | Переменный размер | Выравнивание структуры |
| Цветовой профиль ICC | Да | Размер переменной | Для определения цветового профиля для управления цветом |

Обзор разработанного приложения

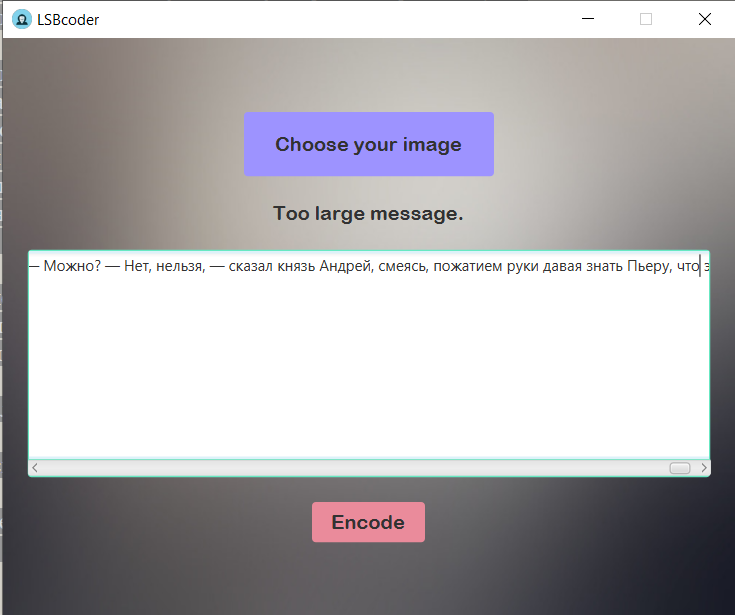
При запуске приложения пользователю открывается главное меню. Окно фиксированного размера. Сразу после открытия пользователю доступна только кнопка Choose File во избежания возникновения сторонних ошибок остальные элементы интерфейса заблокированы.

  
*Главное меню*: с*остояние 1.*

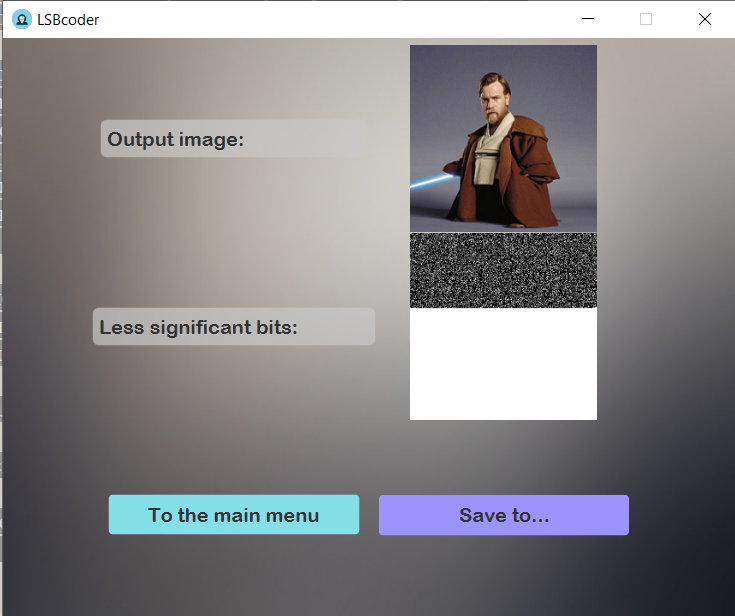
До того момента пока выбранный пользователем файл не будет выбран и успешно найден программой элементы интерфейса будут заблокированы. После успешного ввода исходного изображения окно текстового ввод станет доступно для редактирования и окрасится по краям в зелёный. Кнопка кодирования так же будет заблокирована до того момента, пока сообщение не будет введено.

  
*Главное меню: состояние 2.*

После ввода сообщения происходит проверка на вместимость введенного текста. В случае, если проверка прошла успешно, начнётся процесс кодирования. В противном случае пользователю высветится alert с ошибкой и запустить процесс кодирование будет невозможно до того момента, пока сообщение либо не будет заменено или сокращено до длины возможного размера, либо пока не будет заменено изображение на файл большего разрешения.

  
*Главное меню: состояние 3.*

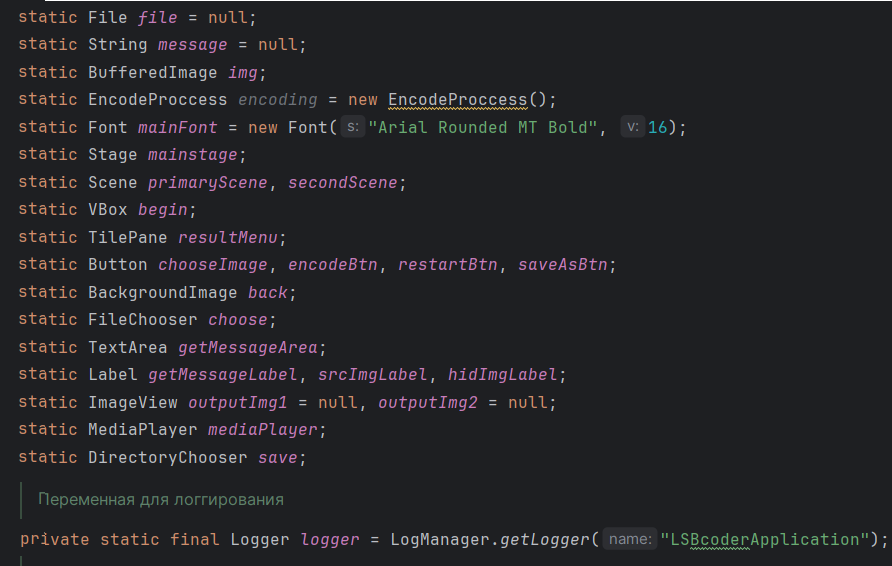
После успешного кодирования отобразится меню результата. Оно вмещает в себя два изображения: изображение со встроенным lsb и сами младшие биты. Чем темнее оттенок бита, тем в большем количестве байтов цвета произведена замена, то есть тем сильнее полученный байт отличается от исходного. Соответственно, белые пиксели – это пиксели, либо не содержащие фрагмента вложенной информации, либо полностью совпадающие с ним.

  
*Меню результата*

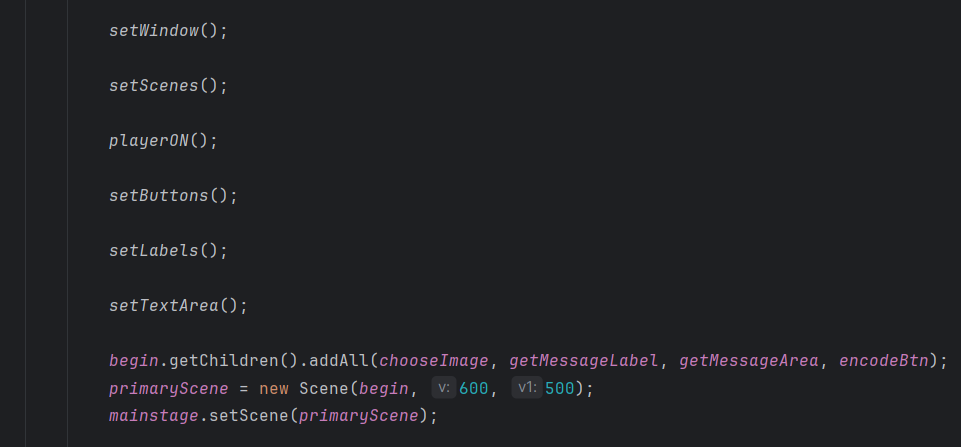
Далее пользователь может сохранить результат, нажав на кнопку   
Save to…, после чего откроется проводник, или перезапустить программу нажатием кнопки To the main menu. Выход стандартно осуществляется через крестик в правом верхнем углу.

Работа приложения

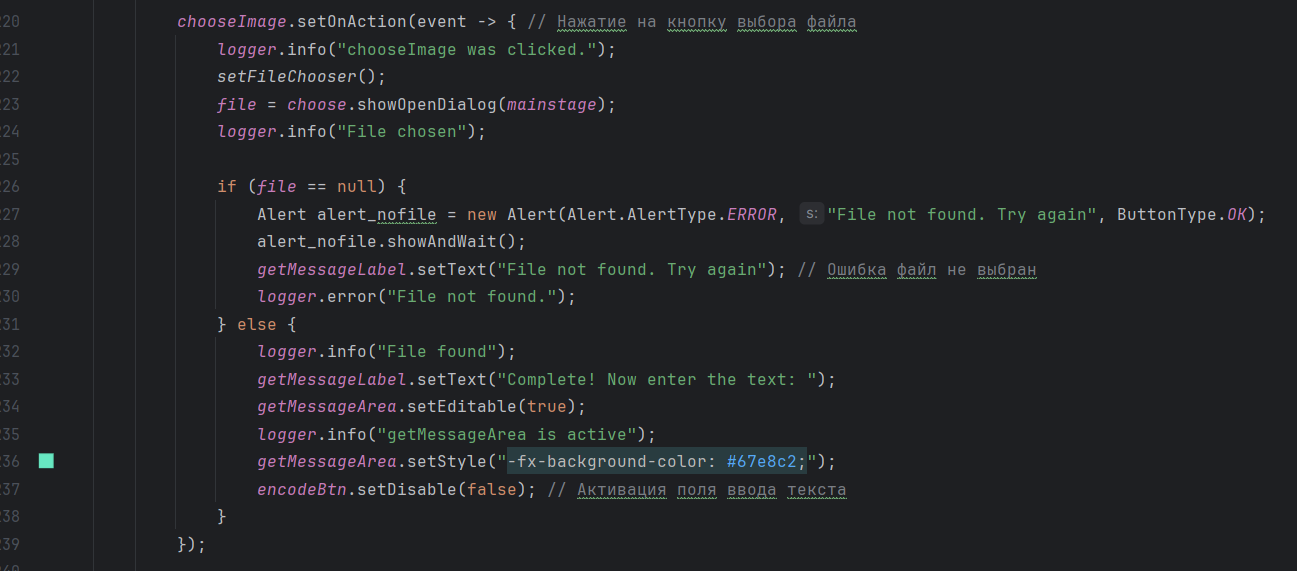
1. После запуска приложения сразу создаются все статические переменные и объекты, необходимые для работы приложения в классе LSBcoderApplication (более подробно в javadoc). Они являются статическими, поскольку большинство используется для элементов интерфейса, а file, message и img являются основными, поэтому они задействованы в каждой функции. Во избежание вызова функций от десятка и более переменных было принято решение сделать их статическими.

 *Переменные и объекты, объявленные в LSBcoderAppalication.*

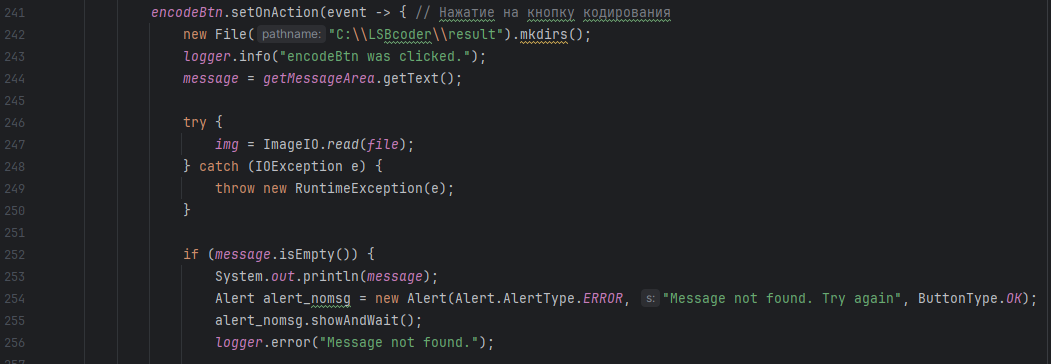
1. Далее поочередно вызываются функции, которые задают параметры элементам интерфейса, а именно: устанавливают окно, сцены, плеер, кнопки, надписи, текстовое поле и ставит главное меню с определенным набором элементов (кнопка выбора изображения, надпись над текстовым полем, текстовое поле и кнопка кодирования). Задаётся размер первой сцены.

  
*Вызов функций, задающих параметры*

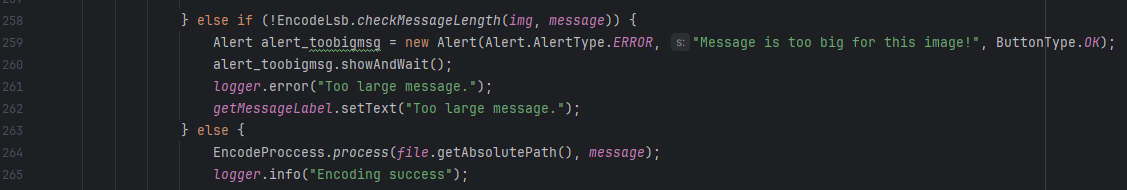
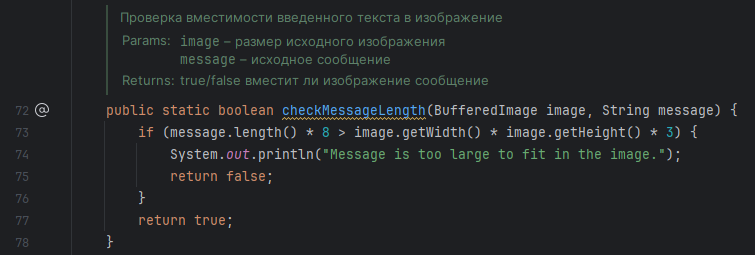
1. При нажатии на кнопку chooseImage вызывается функция, устанавливающая параметры fileChooser и открытие проводника посредством функции showOpenDialog. Пользователь может выбрать только файлы расширения .bmp, поскольку в функции setFileChooser устанавливается фильтр. При неуспешном чтении файла программа выдает alert о том, что файл не найден, логгируется ошибка. При успешном чтении файла и сохранении в переменную file, логгируется успех, высвечивается надпись об успехе и активируется поле ввода и кнопка encodeBtn.

  
*Событие при нажатии на кнопке chooseImage.*

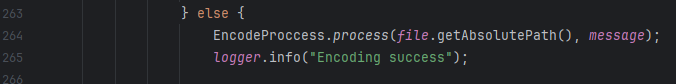
1. После нажатии кнопки encodeBtn происходит считывание сообщения, введенного в getMessageArea. В случае пустого сообщения выводится alert о пустом тексте, ошибка логгируется.

  
*Событие при нажатии на кнопке encodeBtn 1.*

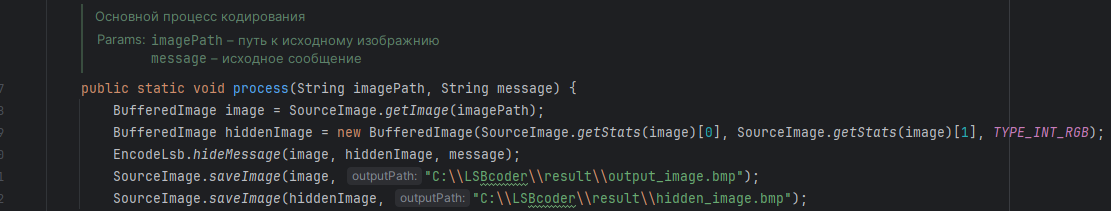
1. Если сообщение не пустое, далее производится проверка на вместимость, посредством функции checkMessageLength из класса EncodeLsb (умножение на 3, поскольку каждый пиксель кодируется 3 байтами BGR). В случае если сообщение первышает размер и выводится alert, логгируется ошибка.

  
*Событие при нажатии на кнопке encodeBtn 2.*  
 *Функция проверки checkMessageLength.*

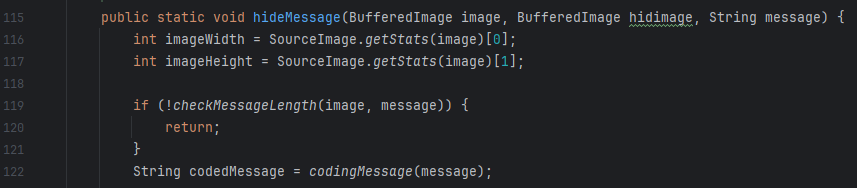
1. После успешного прохождения двух этапов проверки начинается процесс кодирования. Вызывается функция process из класса EncodingProcess.

  
*Процесс кодирования 1.*

1. Создаётся копия исходного bmp-изображения в переменную image типа BufferedImage. Сразу создаётся изображение для отображения заменённых наименьших битов. Вызывается функция класса EncodeLsb hidemessage.

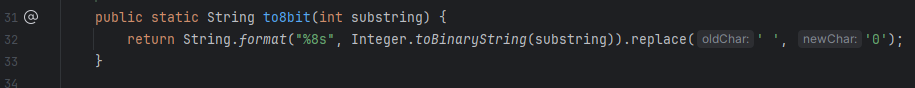
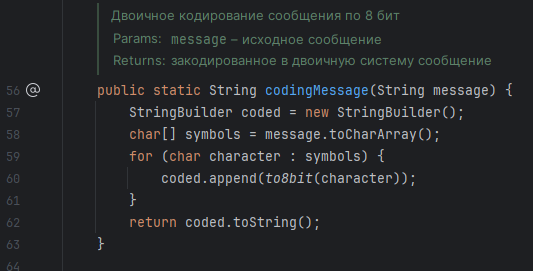
  
*Процесс кодирования 2.*

1. В переменные сохраняется размер изображения, проводится дополнительная проверка на вместимость.

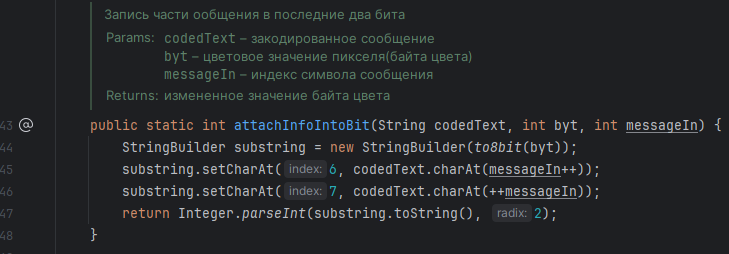
  
*Процесс кодирования 3.*

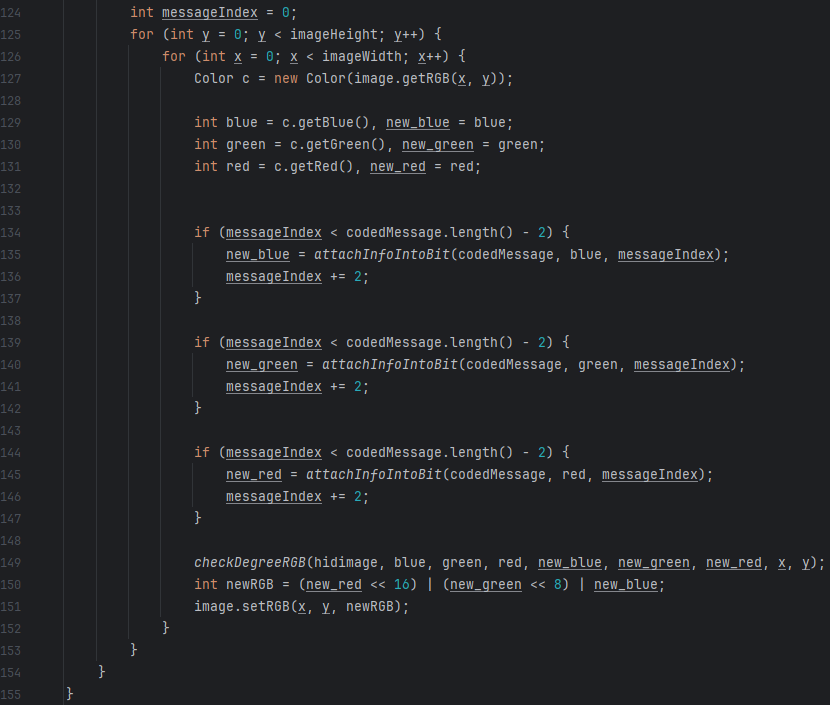
1. Сообщение переводится в двоичную систему счисления, при этом каждый символ кодируется 8-ю битами.



*Процесс кодирования 4. Перевод сообщения в двоичную   
систему счисления.*

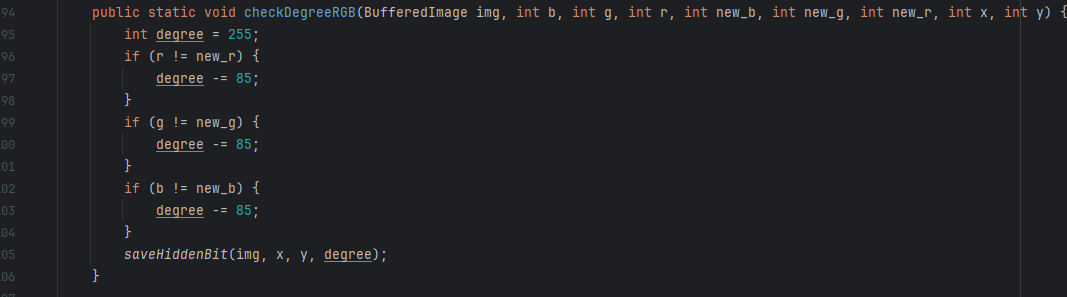
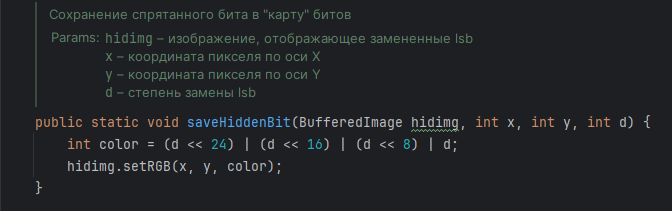
1. Создаётся переменная для отслеживания индекса кодируемого сообщения. В двойном цикле по координатам X и Y создаются парные переменные bgr и new\_bgr. Первые три хранят исходное значение байта соответствующего цвета (blue, green, red), вторые три хранят преобразованное и переведенное в десятичную систему счисления. Перед записью информации в бит проводится проверка на окончание сообщения. В данном алгоритме заменяются 2 последних бита.

*  
Процесс кодирования 5. Функция замены двух последних бит в байте цвета.*

  
*Процесс кодирования 6. Двойной цикл по всем пикселям.*

1. В каждой итерации вызывается проверка степени замены байта через функцию checkDegree и в изображение с наименьшими битами (карту) сохраняется соответствующей темноты пиксель на месте того же пикселя в исходном изображении.



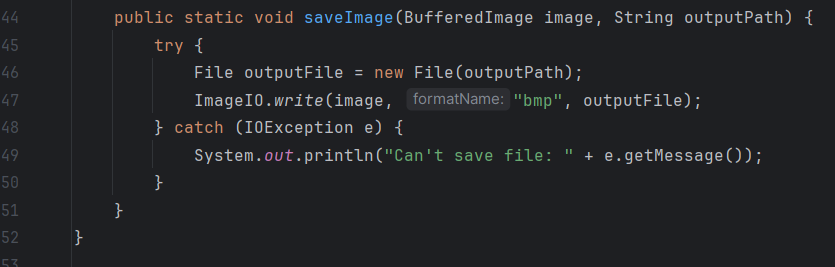
  
  
*Процесс кодирования 7. Сохранение бита в карту измененных битов.*

1. Новое BGR-значение сохраняется в копию исходника.

  
*Процесс кодирование 8. Замена байтов на изменённые.*

1. Файлы сохраняются функцией класса SourceImage saveImage в папку result, которая по определению создаётся в директории «C:\LSBcoder\».

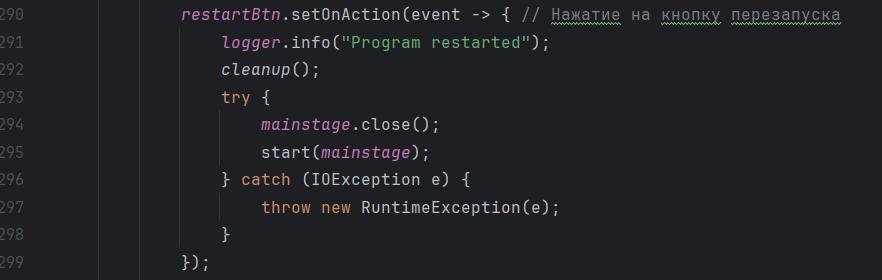


  
*Сохранение карты измененных битов и закодированного изображения.*

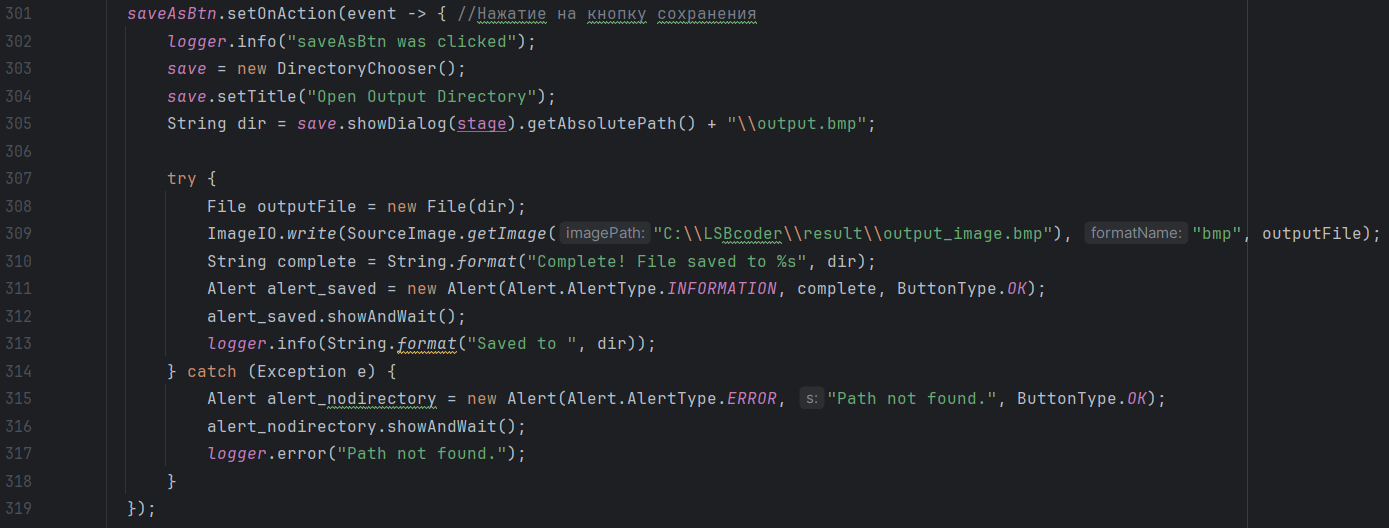
1. Далее создаются два ImageView на основании изображений, полученных в процессе кодирования. Меняется сцена с добавлением двух надписей, двух изображений и двух кнопок restartBtn и saveAsBtn.

  
*Активация меню результата.*

1. При нажатии на restartBtn происходит отчистка переменных с помощью функции cleanup и перезапуск приложения.

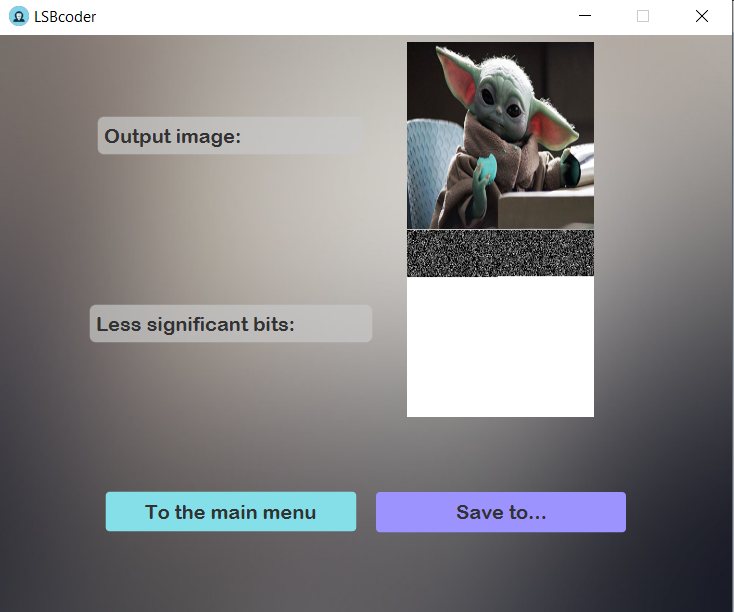
  
*Событие при нажатии на кнопку перезапуска.*

1. При нажатии на кнопку saveAsBtn инициализируется DirectoryChooser для выбора директории сохранения, открывается проводник, где пользователь выбирает необходимую директорию. В случае верного указания директории файл сохраняется под названием output формата bmp и того же разрешения, что и исходник, отображается alert об успешном сохранении. В противном случае отобразится alert с ошибкой «Путь не найден».

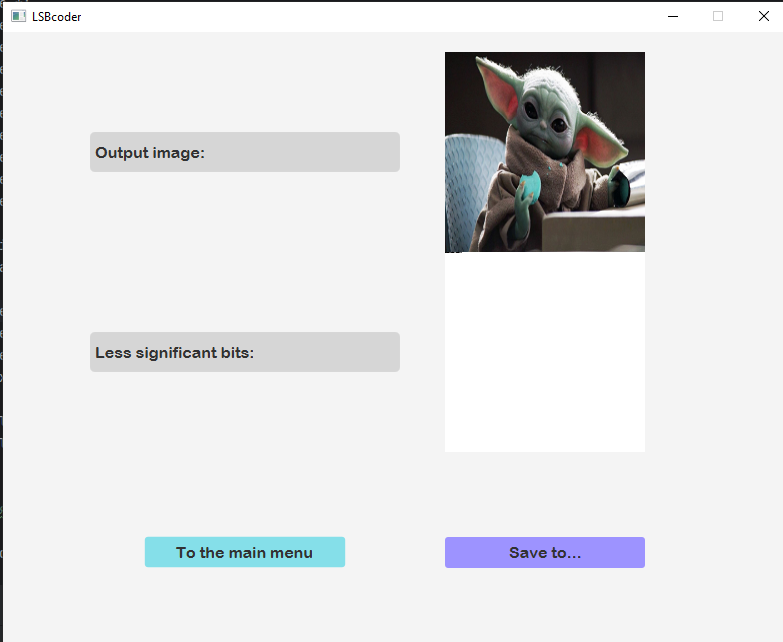
  
*Событие при нажатии на кнопку сохранения.*

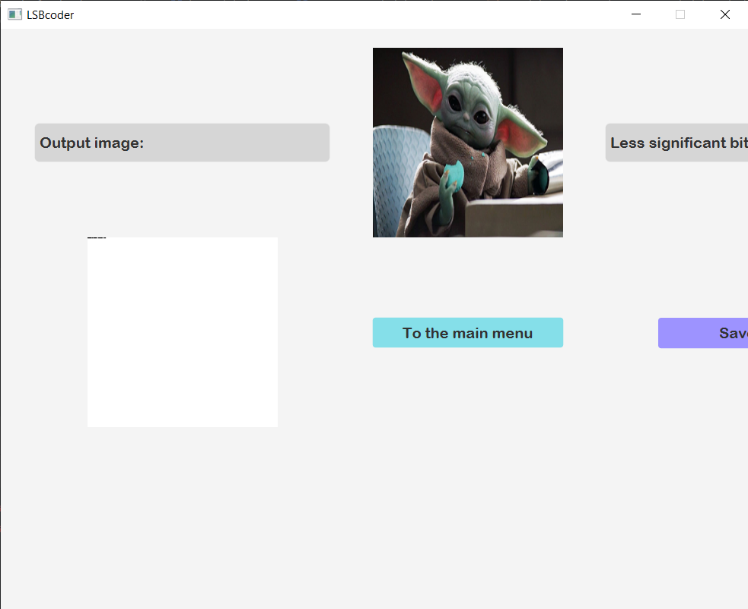
Примеры некорректной работы приложения

Из-за того, что оба ImageView для отображения результата имеют фиксированную квадратную величину изображения с соотношением сторон не 1:1 будет отображаться некорректно, то есть сжато по бокам.

*****Пример некорректной работы приложения 1.*

Величина окна фиксированная, поэтому приложение не адаптируется под разные настройки масштаба экрана пользователя. При разрешении экрана 1920 на 1080 и масштабе 100% отображение окна и его содержимого будет корректно, тем не менее при отображении 125% и больше некоторые элементы интерфейса будут вынесены за пределы окна, что сделает невозможным работу с приложением.

  
*Пример некорректной работы приложения 2.1. Масштаб экрана 100%.*

  
*Пример некорректной работы приложения 2.2. Масштаб экрана 150%.*

*Примечание:* последние скриншоты сделаны из 2-ой версии приложения, адаптированной для сборки jar-файла (подробнее в README.txt). Первая версия приложения подобных проблем не имеет и отображается корректно независимо от масштаба экрана.

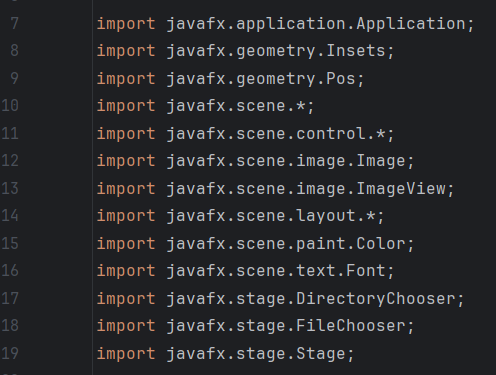
Использование библиотек

**JavaFX**

Разработка минимального графического интерфейса входит в основную задачу, поэтому для его создания в программе используется библиотека JavaFX.

JavaFX — это открытая платформа следующего поколения для клиентских приложений настольных компьютеров, мобильных устройств и встроенных систем, разработанная на языке Java.

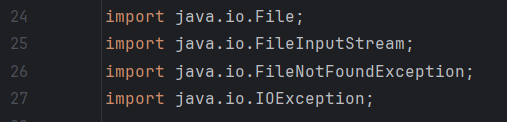
Первоначально программа задействовала библиотеку Swing, после переписана под JavaFX. Смена инструментария обусловлена более расширенным функционалом новой библиотеки и удобством при написании кода. Также JavaFX имеет ПО для удобной и наглядной работы по созданию графического интерфейса – JavaFX Scene Builder. Однако инструментарий имеет и свои недостатки. В частности, сложность сборки jar-файла из проекта JavaFX.

  
*Используемые компоненты JavaFX*

**IO**

Для вывода и ввода данных использовалась библиотека IO. Её использование обусловлено удобным и освоенным инструментарием. В проекте используется для создания, чтения и записи изображений, ввода сообщения для кодирования и обработки исключений.

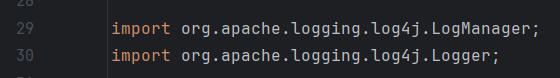
Java IO (Input/Output) — это пакет стандартной библиотеки Java, предоставляющий классы и интерфейсы для ввода и вывода данных в приложениях. Он предоставляет возможность обрабатывать различные типы данных из различных источников, таких как файлы, сеть, память и устройства. В Java IO существует две основные группы классов: классы для байтового ввода/вывода и классы для символьного ввода/вывода.

*  
Используемые компоненты IO*

**Apache Logging Log4j2**

Для логгирования ключевых событий и ошибок используется библиотека Apache Logging Log4j2. Логгирование происходит в папку logs, которая по определению создаётся либо в той же директории, что и запускаемый файл (при запуске jar-файла), либо в файлах проекта в главной директории. Логгируется запуск и завершение программы, ошибки, начало и завершение процесса кодирования, нажатие на кнопки, принятие сообщение, сохранение результата и директория вывода, а также перезапуск.

Apache Log4j 2 — это библиотека логирования для Java, которая предоставляет разработчикам гибкий и мощный инструментарий для управления логами в их приложениях. Она поддерживает различные форматы конфигурации, аппендеры для определения места назначения логов, уровни логирования, фильтры и возможность асинхронного логирования. Библиотека также расширяема и легко интегрируется с другими библиотеками и фреймворками Java.

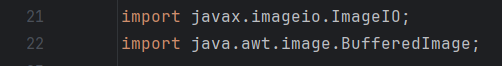
  
*Используемые компоненты Apache Logging Log4j2*

**Awt и imageio**

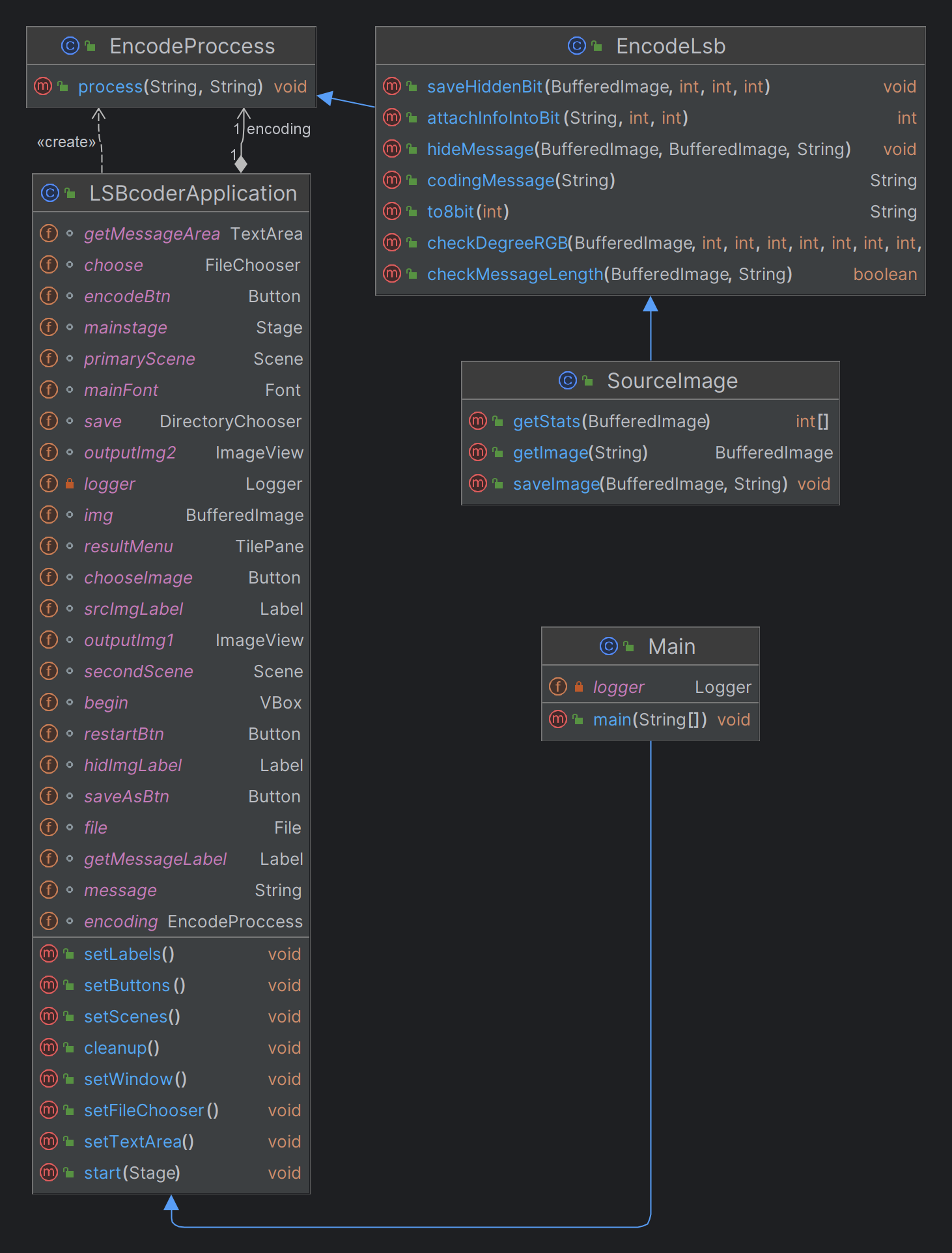
Для работы с изображениями используются библиотеки awt и imageio. В основном используется класс BufferedImage для удобной обработки изображений. Процесс вложение информации в изображение работает только с изображениями типа BufferedImage.

Библиотека Java AWT (Abstract Window Toolkit) предоставляет инструменты для создания графических пользовательских интерфейсов (GUI) в приложениях на Java. Она включает в себя набор классов и интерфейсов, которые позволяют создавать окна, обработчики событий, изображения, элементы управления и другие компоненты интерфейса.

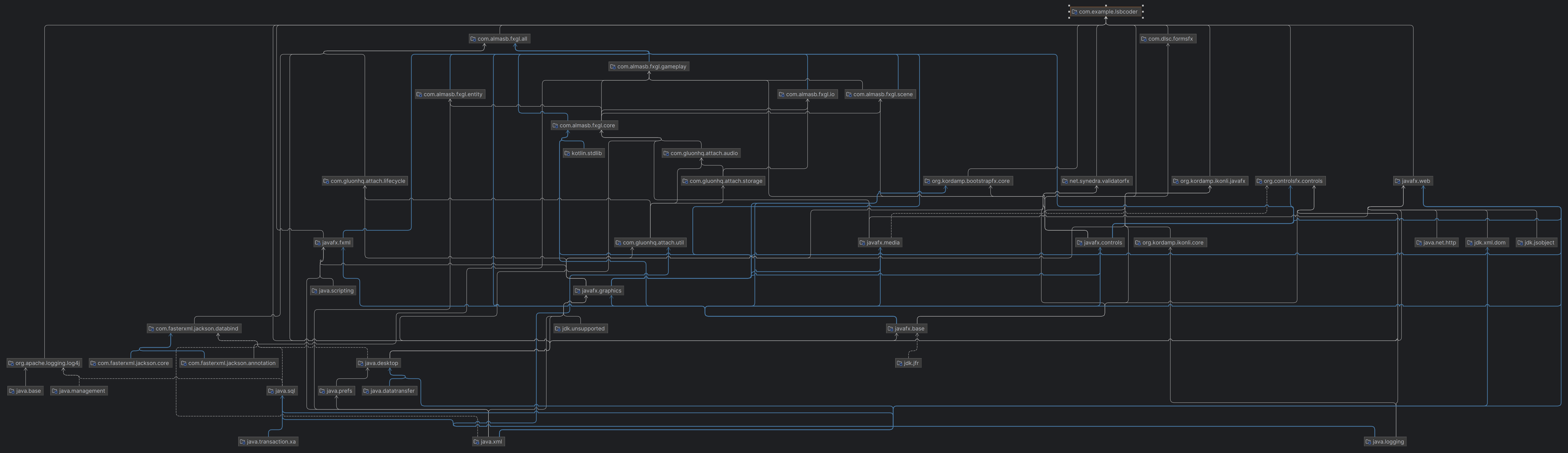
AWT была введена в первых версиях языка Java и предоставляла основные элементы для создания GUI-приложений, такие как кнопки, поля для ввода, метки, меню и прочие. Однако с развитием фреймворка Swing и впоследствии JavaFX, AWT стала использоваться в меньшей степени на пользу более продвинутых и гибких способов создания интерфейсов.

  
*Используемые компоненты imageIO и awt*

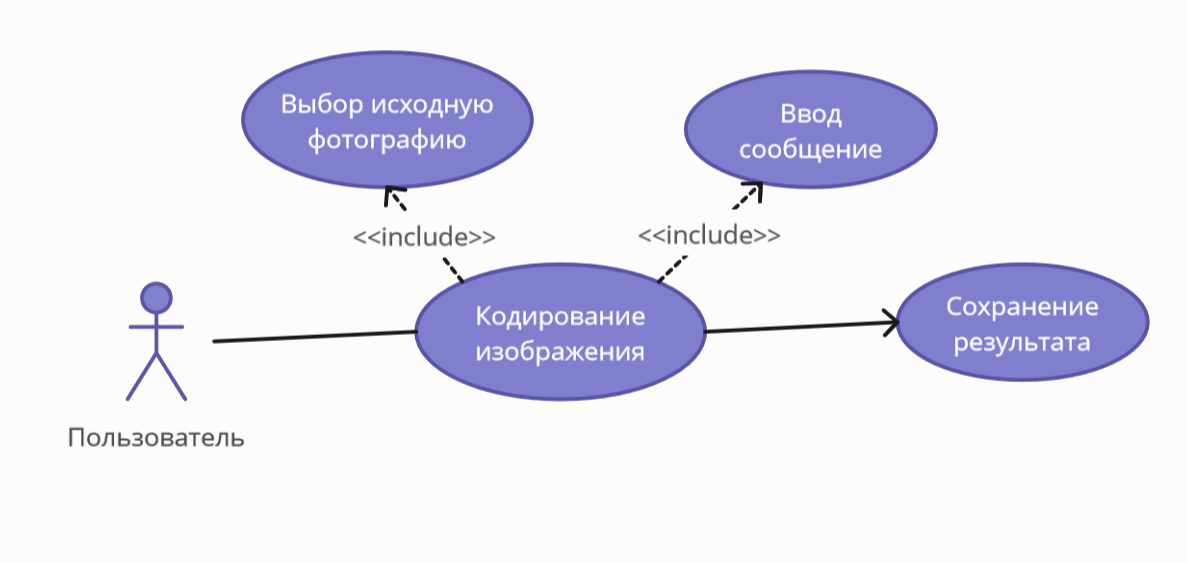
UML-диаграмма классов



UML-диаграмма модулей

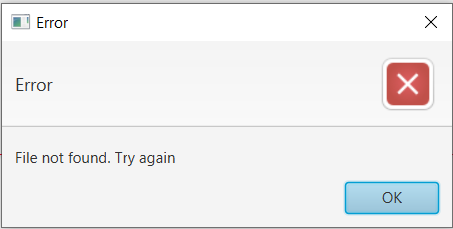


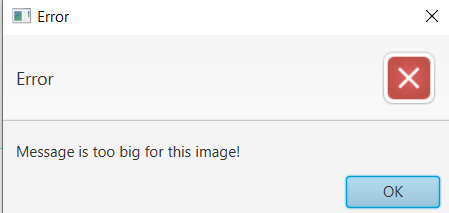
*Примечание:* Диаграммы для более удобного просмотра находятся в папке diagrams.

**UML-диаграмма прецедентов**

Безопасность и надежность программы

Во избежание появления ошибок при работе приложения интерфейс ограничивает действия пользователя. К примеру, до выбора изображение текстовое поле для ввода и кнопка кодирования заблокированы. О всех действиях, приводящих к гипотетической ошибке, пользователя уведомляют предупреждения.



*  
Примеры предупреждений.*

Также во время теста было выяснено приложение способно обрабатывать любые символы: знаки препинания, латиницу, кириллицу, пробелы, китайские и арабские иероглифы и т.д., что делает его устойчивым к ошибкам при вводе сообщения.

Из уязвимостей можно выявить отсутствие проверки заголовка файла, то есть если файл test.mp4 переименовать в test.bmp приложение выдаст ошибку и сломается.

Листинг кода

**Main.java**

package com.example.lsbcoder;  
  
import javafx.application.Application;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
*/\*\*  
 \* Переписанный лаунчер для сборки  
 \*/*public class Main extends LSBcoderApplication {  
 */\*\*  
 \* Переменная для логгирования  
 \*/* private static final Logger *logger* = LogManager.*getLogger*("Main");  
  
 */\*\*  
 \* Запуск программы  
 \* @param args пустые аргументы, поскольку ввод двнных через интерфейс  
 \*/* public static void main(String[] args) {  
 *logger*.info("Program successfully launched.");  
 Application.*launch*(LSBcoderApplication.class, args);  
 *logger*.info("Program was finished.");  
 }  
}

**LSBcoderApplication.java**

package com.example.lsbcoder;  
  
import back.EncodeLsb;  
import back.EncodeProccess;  
import back.SourceImage;  
  
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.\*;  
import javafx.scene.control.\*;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.\*;  
import javafx.scene.paint.Color;  
import javafx.scene.text.Font;  
import javafx.stage.DirectoryChooser;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
import javax.imageio.ImageIO;  
import java.awt.image.BufferedImage;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
*/\*\*  
 \* Класс приложения с реализацией графического интерфейса  
 \*/*public class LSBcoderApplication extends Application {  
 */\*\*  
 \* Файл с исходным изображением  
 \*/* static File *file* = null;  
 */\*\*  
 \* Сообщение для кодирования  
 \*/* static String *message* = null;  
 */\*\*  
 \* изменяемое изображение, генерируемое из file  
 \*/* static BufferedImage *img*;  
 static EncodeProccess *encoding* = new EncodeProccess();  
 */\*\*  
 \* Основной шрифт  
 \*/* static Font *mainFont* = new Font("Arial Rounded MT Bold", 16);  
 */\*\*  
 \* Основное окно  
 \*/* static Stage *mainstage*;  
 */\*\*  
 \* Сцены (начальное меню и меню результата)  
 \*/* static Scene *primaryScene*, *secondScene*;  
 */\*\*  
 \* Макет содержимого начального меню  
 \*/* static VBox *begin*;  
 */\*\*  
 \* Макет содержимого меню результата  
 \*/* static TilePane *resultMenu*;  
 */\*\*  
 \* Кнопки (выбора исходника, запуска кодирования, перезапуска, сохранения)  
 \*/* static Button *chooseImage*, *encodeBtn*, *restartBtn*, *saveAsBtn*;  
 */\*\*  
 \* FileChooser для файла исходника через проводник  
 \*/* static FileChooser *choose*;  
 */\*\*  
 \* TextArea для получения сообщения для кодирования  
 \*/* static TextArea *getMessageArea*;  
 */\*\*  
 \* Подписи (указаний действий в начальном меню, выходного изображения, изображения с lsb)  
 \*/* static Label *getMessageLabel*, *srcImgLabel*, *hidImgLabel*;  
 */\*\*  
 \* изображения (выходное, отображение lsb)  
 \*/* static ImageView *outputImg1* = null, *outputImg2* = null;  
 */\*\*  
 \* Переменная для логгирования  
 \*/* private static final Logger *logger* = LogManager.*getLogger*("LSBcoderApplication");  
 */\*\*  
 \* DirectoryChooser для сохранения выходного файла через проводник  
 \*/* static DirectoryChooser *save*;  
  
 */\*\*  
 \* Задаёт параметры окон  
 \*  
 \* @throws FileNotFoundException исключение в случае ненайденной иконки приложения  
 \*/* public static void setWindow() throws FileNotFoundException {  
 *mainstage* = new Stage();  
 *mainstage*.setTitle("LSBcoder");  
 *mainstage*.setResizable(false);  
 *mainstage*.setWidth(800);  
 *mainstage*.setHeight(650);  
 }  
  
 */\*\*  
 \* Задаёт параметры кнопкам  
 \*/* public static void setButtons() {  
 *chooseImage* = new Button("Choose your image");  
 *chooseImage*.setFont(*mainFont*);  
 *chooseImage*.setStyle("-fx-background-color: #9d93ff;");  
 *chooseImage*.setPrefSize(200, 50);  
  
 *encodeBtn* = new Button("Encode");  
 *encodeBtn*.setStyle("-fx-background-color: #ea8b9b;");  
 *encodeBtn*.setFont(*mainFont*);  
 *encodeBtn*.setPrefSize(90, 30);  
 *encodeBtn*.setDisable(true);  
  
 *restartBtn* = new Button("To the main menu");  
 *restartBtn*.setStyle("-fx-background-color: #85dfe9;");  
 *restartBtn*.setFont(*mainFont*);  
 *restartBtn*.setPrefSize(200, 20);  
  
  
 *saveAsBtn* = new Button("Save to...");  
 *saveAsBtn*.setFont(*mainFont*);  
 *saveAsBtn*.setStyle("-fx-background-color: #9d93ff;");  
 *saveAsBtn*.setPrefSize(200, 20);  
  
 }  
  
 */\*\*  
 \* Задаёт параметры сцен  
 \*/* public static void setScenes() {  
 *begin* = new VBox();  
 *begin*.setSpacing(20);  
 *begin*.setPadding(new Insets(20, 20, 20, 20));  
 *begin*.setAlignment(Pos.*CENTER*);  
  
 *resultMenu* = new TilePane();  
 *resultMenu*.setPadding(new Insets(20, 20, 20, 20));  
 *resultMenu*.setAlignment(Pos.*TOP\_CENTER*);  
 }  
  
 */\*\*  
 \* Задаёт параметры надписей  
 \*/* public static void setLabels() {  
 *getMessageLabel* = new Label("Enter the message: ");  
 *getMessageLabel*.setFont(*mainFont*);  
 *getMessageLabel*.setAlignment(Pos.*CENTER\_LEFT*);  
  
 *srcImgLabel* = new Label("Output image:");  
 *srcImgLabel*.setBackground(new Background(new BackgroundFill(Color.*rgb*(200, 200, 200, 0.7), new CornerRadii(5.0), new Insets(-5.0))));  
 *srcImgLabel*.setFont(*mainFont*);  
 *srcImgLabel*.setMinWidth(300);  
 *srcImgLabel*.setMinHeight(30);  
  
 *hidImgLabel* = new Label("Less significant bits:");  
 *hidImgLabel*.setBackground(new Background(new BackgroundFill(Color.*rgb*(200, 200, 200, 0.7), new CornerRadii(5.0), new Insets(-5.0))));  
 *hidImgLabel*.setFont(*mainFont*);  
 *hidImgLabel*.setMinWidth(300);  
 *hidImgLabel*.setMinHeight(30);  
 }  
  
 */\*\*  
 \* Задаёт параметры выборщику файлов  
 \*/* public static void setFileChooser() {  
 *choose* = new FileChooser();  
 *choose*.getExtensionFilters().addAll(  
 new FileChooser.ExtensionFilter("BMP", "\*.bmp"));  
 *choose*.setTitle("Open Resource File");  
 }  
  
 */\*\*  
 \* Задаёт параметры поля для ввода текста  
 \*/* public static void setTextArea() {  
 *getMessageArea* = new TextArea();  
 *getMessageArea*.setStyle("-fx-background-color: #f5062c;");  
 *getMessageArea*.setEditable(false);  
 }  
  
 */\*\*  
 \* Очищает переменные перед перезапуском  
 \*/* public static void cleanup() {  
 *file* = null;  
 *message* = null;  
 *img* = null;  
 }  
  
 */\*\*  
 \* Основная функция активации приложения со сценариями событий при нажатии на кнопки  
 \*  
 \* @param stage дефолтное окно  
 \* @throws IOException в случае отсуствия окна  
 \*/* @Override  
 public void start(Stage stage) throws IOException {  
 *setWindow*();  
 *setScenes*();  
 *setButtons*();  
 *setLabels*();  
 *setTextArea*();  
  
 *begin*.getChildren().addAll(*chooseImage*, *getMessageLabel*, *getMessageArea*, *encodeBtn*);  
 *primaryScene* = new Scene(*begin*, 800, 650);  
 *mainstage*.setScene(*primaryScene*);  
  
 *chooseImage*.setOnAction(event -> { // Нажатие на кнопку выбора файла  
 *logger*.info("chooseImage was clicked.");  
 *setFileChooser*();  
 *file* = *choose*.showOpenDialog(*mainstage*);  
 *logger*.info("File chosen");  
  
 if (*file* == null) {  
 Alert alert\_nofile = new Alert(Alert.AlertType.*ERROR*, "File not found. Try again", ButtonType.*OK*);  
 alert\_nofile.showAndWait();  
 *getMessageLabel*.setText("File not found. Try again"); // Ошибка файл не выбран  
 *logger*.error("File not found.");  
 } else {  
 *logger*.info("File found");  
 *getMessageLabel*.setText("Complete! Now enter the text: ");  
 *getMessageArea*.setEditable(true);  
 *logger*.info("getMessageArea is active");  
 *getMessageArea*.setStyle("-fx-background-color: #67e8c2;");  
 *encodeBtn*.setDisable(false); // Активация поля ввода текста  
 }  
 });  
  
 *encodeBtn*.setOnAction(event -> { // Нажатие на кнопку кодирования  
 new File("C:\\LSBcoder\\result").mkdirs();  
 *logger*.info("encodeBtn was clicked.");  
 *message* = *getMessageArea*.getText();  
  
 try {  
 *img* = ImageIO.*read*(*file*);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
  
 if (*message*.isEmpty()) {  
 System.*out*.println(*message*);  
 Alert alert\_nomsg = new Alert(Alert.AlertType.*ERROR*, "Message not found. Try again", ButtonType.*OK*);  
 alert\_nomsg.showAndWait();  
 *logger*.error("Message not found.");  
  
 } else if (!EncodeLsb.*checkMessageLength*(*img*, *message*)) {  
 Alert alert\_toobigmsg = new Alert(Alert.AlertType.*ERROR*, "Message is too big for this image!", ButtonType.*OK*);  
 alert\_toobigmsg.showAndWait();  
 *logger*.error("Too large message.");  
 *getMessageLabel*.setText("Too large message.");  
 } else {  
 EncodeProccess.*process*(*file*.getAbsolutePath(), *message*);  
 *logger*.info("Encoding success");  
  
 try {  
 *outputImg1* = new ImageView(new Image(new FileInputStream("C:\\LSBcoder\\result\\output\_image.bmp")));  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 *outputImg1*.setFitHeight(200);  
 *outputImg1*.setFitWidth(200);  
  
 try {  
 *outputImg2* = new ImageView(new Image(new FileInputStream("C:\\LSBcoder\\result\\hidden\_image.bmp")));  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 *outputImg2*.setFitHeight(200);  
 *outputImg2*.setFitWidth(200);  
  
 *resultMenu*.getChildren().addAll(*srcImgLabel*, *outputImg1*, *hidImgLabel*, *outputImg2*, *restartBtn*, *saveAsBtn*);  
  
 *secondScene* = new Scene(*resultMenu*, 800, 650);  
 *mainstage*.setScene(*secondScene*);  
 }  
 });  
  
 *restartBtn*.setOnAction(event -> { // Нажатие на кнопку перезапуска  
 *logger*.info("Program restarted");  
 *cleanup*();  
 try {  
 *mainstage*.close();  
 start(*mainstage*);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 });  
  
 *saveAsBtn*.setOnAction(event -> { //Нажатие на кнопку сохранения  
 *logger*.info("saveAsBtn was clicked");  
 *save* = new DirectoryChooser();  
 *save*.setTitle("Open Output Directory");  
 String dir = *save*.showDialog(stage).getAbsolutePath() + "\\output.bmp";  
  
 try {  
 File outputFile = new File(dir);  
 ImageIO.*write*(SourceImage.*getImage*("C:\\LSBcoder\\result\\output\_image.bmp"), "bmp", outputFile);  
 String complete = String.*format*("Complete! File saved to %s", dir);  
 Alert alert\_saved = new Alert(Alert.AlertType.*INFORMATION*, complete, ButtonType.*OK*);  
 alert\_saved.showAndWait();  
 *logger*.info(String.*format*("Saved to ", dir));  
 } catch (Exception e) {  
 Alert alert\_nodirectory = new Alert(Alert.AlertType.*ERROR*, "Path not found.", ButtonType.*OK*);  
 alert\_nodirectory.showAndWait();  
 *logger*.error("Path not found.");  
 }  
 });  
  
 *mainstage*.show();  
 }  
}

**EncodeProcess.java**

package back;  
  
import java.awt.image.BufferedImage;  
import static java.awt.image.BufferedImage.*TYPE\_INT\_RGB*;  
  
*/\*\*  
 \* Класс, реализующий шифрование  
 \*/*public class EncodeProccess {  
  
 */\*\*  
 \* Основной процесс кодирования  
 \*  
 \* @param imagePath путь к исходному изображнию  
 \* @param message исходное сообщение  
 \*/* public static void process(String imagePath, String message) {  
 BufferedImage image = SourceImage.*getImage*(imagePath);  
 BufferedImage hiddenImage = new BufferedImage(SourceImage.*getStats*(image)[0], SourceImage.*getStats*(image)[1], *TYPE\_INT\_RGB*);  
 EncodeLsb.*hideMessage*(image, hiddenImage, message);  
 SourceImage.*saveImage*(image, "C:\\LSBcoder\\result\\output\_image.bmp");  
 SourceImage.*saveImage*(hiddenImage, "C:\\LSBcoder\\result\\hidden\_image.bmp");  
 }  
}

**EncodeLsb.java**

package back;  
  
import com.example.lsbcoder.LSBcoderApplication;  
  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
  
*/\*\*  
 \* Класс, реализующий функции для шифрования  
 \*/*public class EncodeLsb extends EncodeProccess {  
 */\*\*  
 \* Сохранение спрятанного бита в "карту" битов  
 \*  
 \* @param hidimg изображение, отображающее замененные lsb  
 \* @param x координата пикселя по оси X  
 \* @param y координата пикселя по оси Y  
 \* @param d степень замены lsb  
 \*/* public static void saveHiddenBit(BufferedImage hidimg, int x, int y, int d) {  
 int color = (d << 24) | (d << 16) | (d << 8) | d;  
 hidimg.setRGB(x, y, color);  
 }  
  
 */\*\*  
 \* Перевод десятичного числа в 8-битную двоичную запись  
 \*  
 \* @param substring число, которое необходимо представить в двоичной  
 \* @return 8-битную двоичную запись числа  
 \*/* public static String to8bit(int substring) {  
 return String.format("%8s", Integer.toBinaryString(substring)).replace(' ', '0');  
 }  
  
 */\*\*  
 \* Запись части ообщения в последние два бита  
 \*  
 \* @param codedText закодированное сообщение  
 \* @param byt цветовое значение пикселя(байта цвета)  
 \* @param messageIn индекс символа сообщения  
 \* @return измененное значение байта цвета  
 \*/* public static int attachInfoIntoBit(String codedText, int byt, int messageIn) {  
 StringBuilder substring = new StringBuilder(to8bit(byt));  
 substring.setCharAt(6, codedText.charAt(messageIn++));  
 substring.setCharAt(7, codedText.charAt(++messageIn));  
 return Integer.parseInt(substring.toString(), 2);  
 }  
  
 */\*\*  
 \* Двоичное кодирование сообщения по 8 бит  
 \*  
 \* @param message исходное сообщение  
 \* @return закодированное в двоичную систему сообщение  
 \*/* public static String codingMessage(String message) {  
 StringBuilder coded = new StringBuilder();  
 char[] symbols = message.toCharArray();  
 for (char character : symbols) {  
 coded.append(to8bit(character));  
 }  
 return coded.toString();  
 }  
  
 */\*\*  
 \* Проверка вместимости введенного текста в изображение  
 \*  
 \* @param image размер исходного изображения  
 \* @param message исходное сообщение  
 \* @return true/false вместит ли изображение сообщение  
 \*/* public static boolean checkMessageLength(BufferedImage image, String message) {  
 if (message.length() \* 8 > image.getWidth() \* image.getHeight() \* 3) {  
 System.out.println("Message is too large to fit in the image.");  
 return false;  
 }  
 return true;  
 }  
  
 */\*\*  
 \* Проверка на степень изменения байта цвета  
 \*  
 \* @param img исходное изображение  
 \* @param b исходное значение байта синего  
 \* @param g исходное значение байта зеленого  
 \* @param r исходное значение байта красного  
 \* @param new\_b значение изменённого байта синего  
 \* @param new\_g значение изменённого байта зеленого  
 \* @param new\_r значение изменённого байта красного  
 \* @param x координата пикселя по оси X  
 \* @param y координата пикселя по оси Y  
 \*/* public static void checkDegreeRGB(BufferedImage img, int b, int g, int r, int new\_b, int new\_g, int new\_r, int x, int y) {  
 int degree = 255;  
 if (r != new\_r) {  
 degree -= 85;  
 }  
 if (g != new\_g) {  
 degree -= 85;  
 }  
 if (b != new\_b) {  
 degree -= 85;  
 }  
 *saveHiddenBit*(img, x, y, degree);  
 }  
  
 */\*\*  
 \* Основной прогон по каждому пикселю и вложение в него информации  
 \*  
 \* @param image исходное изображение  
 \* @param hidimage изображение, отображающее изменённые lsb  
 \* @param message исходное сообщение  
 \*/* public static void hideMessage(BufferedImage image, BufferedImage hidimage, String message) {  
 int imageWidth = SourceImage.*getStats*(image)[0];  
 int imageHeight = SourceImage.*getStats*(image)[1];  
  
 if (!checkMessageLength(image, message)) {  
 return;  
 }  
 String codedMessage = codingMessage(message);  
  
 int messageIndex = 0;  
 for (int y = 0; y < imageHeight; y++) {  
 for (int x = 0; x < imageWidth; x++) {  
 Color c = new Color(image.getRGB(x, y));  
  
 int blue = c.getBlue(), new\_blue = blue;  
 int green = c.getGreen(), new\_green = green;  
 int red = c.getRed(), new\_red = red;  
  
  
 if (messageIndex < codedMessage.length() - 2) {  
 new\_blue = attachInfoIntoBit(codedMessage, blue, messageIndex);  
 messageIndex += 2;  
 }  
  
 if (messageIndex < codedMessage.length() - 2) {  
 new\_green = attachInfoIntoBit(codedMessage, green, messageIndex);  
 messageIndex += 2;  
 }  
  
 if (messageIndex < codedMessage.length() - 2) {  
 new\_red = attachInfoIntoBit(codedMessage, red, messageIndex);  
 messageIndex += 2;  
 }  
  
 checkDegreeRGB(hidimage, blue, green, red, new\_blue, new\_green, new\_red, x, y);  
 int newRGB = (new\_red << 16) | (new\_green << 8) | new\_blue;  
 image.setRGB(x, y, newRGB);  
 }  
 }  
 }  
}

**SourceImage.java**

package back;  
  
import javax.imageio.ImageIO;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.IOException;  
  
*/\*\*  
 \* Класс для обработки изображений  
 \*/*public class SourceImage extends EncodeLsb {  
 */\*\*  
 \* Считывание изображения из файла  
 \*  
 \* @param imagePath путь к исходному изображению  
 \* @return изображение-копия исходного, доступное для обработки  
 \*/* public static BufferedImage getImage(String imagePath) {  
 BufferedImage image = null;  
 try {  
 image = ImageIO.*read*(new File(imagePath));  
 } catch (IOException e) {  
 System.*out*.println("Can't open file: " + e.getMessage());  
 }  
 return image;  
 }  
  
 */\*\*  
 \* Получение данных изображения  
 \*  
 \* @param image исходное изображение  
 \* @return ширина и высота исходного изображения  
 \*/* public static int[] getStats(BufferedImage image) {  
 return new int[]{image.getWidth(), image.getHeight()};  
 }  
  
 */\*\*  
 \* Сохранение изображения  
 \*  
 \* @param image исходное изображение  
 \* @param outputPath путь в директорию для сохранения изображения  
 \*/* public static void saveImage(BufferedImage image, String outputPath) {  
 try {  
 File outputFile = new File(outputPath);  
 ImageIO.write(image, "bmp", outputFile);  
 } catch (IOException e) {  
 System.out.println("Can't save file: " + e.getMessage());  
 }  
 }  
}

*Примечание:* В листинге отображены .java файлы второй версии программы.

Источники

Места заимствований, переработки кода, получения информации (статьи, используемые при подготовке, адреса сайтов с кодом, туториалы, ссылки на видео):

1. <https://docs.fileformat.com/ru/image/bmp/>
2. <https://www.jetbrains.com/help/idea/class-diagram.html>
3. [Приемы и советы. Как избежать NullPointerException в Java приложениях](https://javarush.com/groups/posts/680-priemih-i-sovetih-kak-izbezhatjh-nullpointerexception-v-java-prilozhenijakh)
4. [java - Can&apos;t run jar file of JavaFx application - Stack Overflow](https://stackoverflow.com/questions/74346018/cant-run-jar-file-of-javafx-application)
5. [java - Error: JavaFX runtime components are missing, and are required to run this application with JDK 11 - Stack Overflow](https://stackoverflow.com/questions/51478675/error-javafx-runtime-components-are-missing-and-are-required-to-run-this-appli)
6. [HexEd.it - ​​онлайн и офлайн редактирование шестнадцатеричного кода через браузер](https://hexed.it/)
7. [java - How to force IntelliJ to download javadocs with Maven? - Stack Overflow](https://stackoverflow.com/questions/31072667/how-to-force-intellij-to-download-javadocs-with-maven)
8. [JavaFX | Кнопки и метки](https://metanit.com/java/javafx/4.1.php)
9. [JavaFX - текст - CoderLessons.com](https://coderlessons.com/tutorials/java-tekhnologii/vyuchi-javafx/javafx-tekst)
10. [JavaFX How to - Save an Image to a file](http://www.java2s.com/Tutorials/Java/JavaFX_How_to/Image/Save_an_Image_to_a_file.htm)
11. [JavaFX | Класс Scene](https://metanit.com/java/javafx/1.5.php)
12. [Save As Button In Preview Sample Controller Skeleton Dialog · Issue #381 · gluonhq/scenebuilder](https://github.com/gluonhq/scenebuilder/issues/381)
13. [Java из двоичной системы в десятичную и обратно | user12vv](https://user12vv.wordpress.com/2014/01/16/java-%D0%B8%D0%B7-%D0%B4%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B9-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D0%B2-%D0%B4%D0%B5%D1%81%D1%8F%D1%82%D0%B8%D1%87%D0%BD%D1%83%D1%8E-%D0%B8-%D0%BE%D0%B1/)
14. [LSB/Steganography.java at master · Bharat23/LSB](https://github.com/Bharat23/LSB/blob/master/Steganography.java)
15. [byte - LSB/MSB handling in Java - Stack Overflow](https://stackoverflow.com/questions/5167079/lsb-msb-handling-in-java)
16. [BMP — формат файла изображения](https://docs.fileformat.com/ru/image/bmp/)
17. [jar JavaFX runtime components are missing, and are required to run this application - YouTube](https://www.youtube.com/results?search_query=jar+JavaFX+runtime+components+are+missing%2C+and+are+required+to+run+this+application)
18. [java: module eu.hansolo.tilesfx - YouTube](https://www.youtube.com/results?search_query=java%3A+module+eu.hansolo.tilesfx)

Список литературы

1. «LSB стеганография», *Горьков А., 2011 г.   
   Источник:* <https://habr.com/ru/articles/112976/>
2. «Стегоанализ метода сокрытия информации в изображении замены наименее значащего бита (LSB)», *2018 г., Назаренко Ю.Л.  
   Источник:* <https://cyberleninka.ru/article/n/stegoanaliz-metoda-sokrytiya-informatsii-v-izobrazhenii-zameny-naimenshego-znachaschego-bita-lsb>
3. «Скрывать не скрывая. Еще раз о LSB-стеганографии, хи-квадрате и… сингулярности?», *2018 г., Pandora*

*Источник:* [*https://habr.com/ru/articles/422593/*](https://habr.com/ru/articles/422593/)