МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Информационной безопасности

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Модели безопасности компьютерных систем»
Тема: Информационный поток по памяти

Горбунова Д. А.
Студенты гр. 1361 _____ Кравцов И. Ю.
Преподаватель _____ Шкляр Е. В.

Санкт-Петербург
2024

Теоретическая справка.

Сущность в произвольный момент времени может быть однозначно представлена словом некоторого языка (набором данных), которое может рассматриваться как состояние сущности.

Субъект – сущность, инициирующая выполнение операций над сущностями.

Объект (object), или контейнер (container) – сущность, содержащая или получающая информацию, и над которой субъекты выполняют операции.

Контейнеры могут состоять из объектов или других контейнеров.

Субъекты могут получать доступ к объектам целиком, но не к их части.

Субъекты могут получать доступ к контейнеру и сущностям, из которых он состоит.

Для выполнения операций над сущностями субъекты осуществляют доступы к ним.

Основные виды доступов:

read – на чтение из сущности;

write – на запись в сущность;

append – на запись в конец слова, описывающего состояние сущности;

execute – на активацию субъекта из сущности.

Информационный поток по памяти – информационный поток, при реализации которого фактор времени не является существенным.

Информационный поток по времени – информационный поток, при реализации которого фактор времени является существенным (например, передача данных осуществляется путем изменения продолжительности интервалов времени между событиями в КС или путем изменения последовательности событий).

Дискреционная политика управления доступом – политика, соответствующая следующим требованиям:

- Все сущности идентифицированы (т.е. каждой сущности присвоен уникальный идентификатор);

- Задана матрица доступов, каждая строка которой соответствует субъекту, а столбец – сущности КС, ячейка содержит список прав доступа субъекта к сущности;
- Субъект обладает правом доступа к сущности КС тогда и только тогда, когда в соответствующей ячейке матрицы доступов содержится данное право доступа.

Мандатная политика управления доступом – политика, соответствующая следующим требованиям:

- Все сущности идентифицированы;
- Задана решетка уровней конфиденциальности информации;
- Каждой сущности присвоен уровень конфиденциальности, задающий установленные ограничения на доступ к данной сущности;
- Каждому субъекту присвоен уровень доступа, задающий уровень полномочий данного субъекта в КС;
- Субъект обладает правом доступа к сущности КС тогда, когда уровень доступа субъекта позволяет предоставить ему данный доступ к сущности с заданным уровнем конфиденциальности, и реализация доступа не приведет к возникновению информационных потоков от сущностей с высоким уровнем конфиденциальности к сущностям с низким уровнем.

Политика ролевого управления доступом - политика, соответствующая следующим требованиям:

- Все сущности идентифицированы;
- Задано множество ролей, каждой из которых ставится в соответствие некоторое множество прав доступа к сущностям;
- Каждый субъект обладает некоторым множеством ролей;
- Субъект обладает правом доступа к сущности КС тогда, когда он обладает ролью, которой соответствует множество прав доступа, содержащее право доступа к данной сущности.

Является развитием дискреционного управления доступом, при этом позволяет определить более четкие и понятные для пользователей правила. Позволяет реализовывать гибкие правила управления доступом.

Политика безопасности информационных потоков основана на разделении всех возможных информационных потоков (ИП) между сущностями КС на два непересекающихся множества: множество разрешенных ИП и множество запрещенных ИП.

Цель – обеспечить невозможность возникновения в КС запрещенных ИП.

Как правило, реализуется в сочетании с политикой другого вида (дискреционной, мандатной или ролевой). Реализация крайне сложна, т.к. требует защиту от возникновения запрещенных потоков по времени.

Политика изолированной программной среды реализуется путем изоляции субъектов КС друг от друга и путем контроля порождения новых субъектов так, чтобы в системе могли активи-зироваться только субъекты из определенного списка.

Цель – задание порядка безопасного взаимодей-ствия субъектов КС, обеспечивающего невоз-можность воздействия на систему защиты КС и модификации ее параметров или конфигурации, результатом которого могло бы стать изменение политики управления доступом.

Задача.

Реализовать программу, реализующую создание файла, хранящего строку текста с заданным названием в приватную папку, а также копирование этого файла по запросу пользователя в общедоступную папку.

Реализовать программу нарушителя, которая позволяет просматривать содержимое публичной папки и копировать новые файлы с информацией в папку нарушителя.

Ход работы.

1) В первую очередь, была реализована программа пользователя, которая может создавать файлы с заданным именем и содержанием в приватной папке, перемещать файлы из приватной папки в общедоступную. Интерфейс программы представлен на рисунке 1.

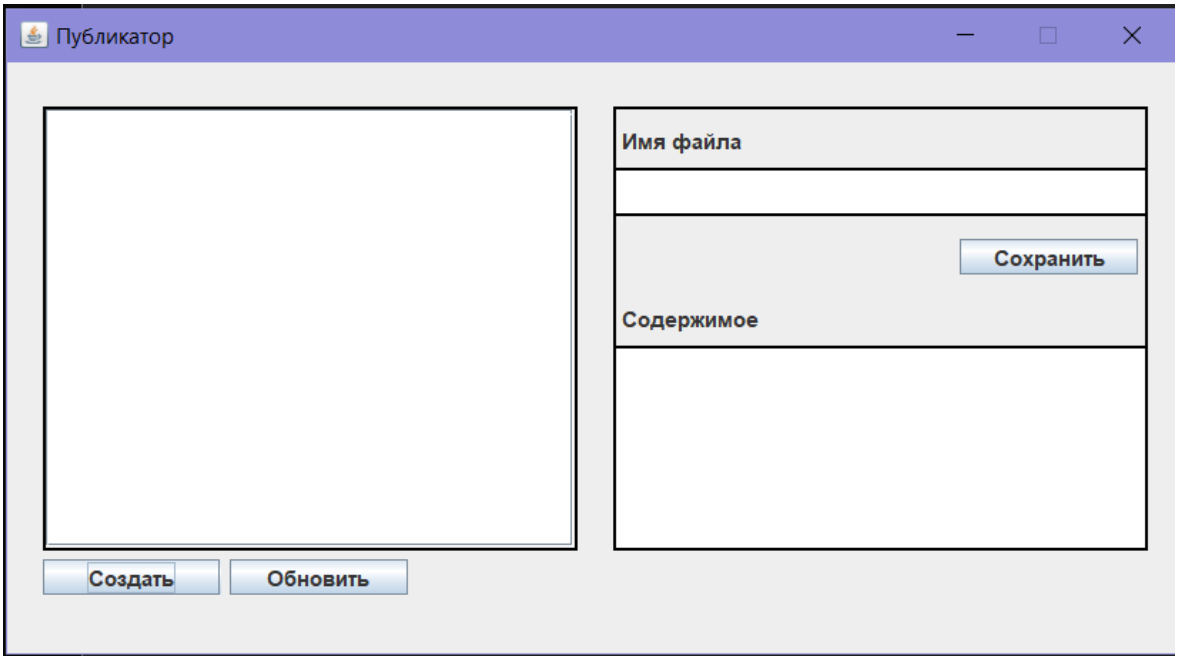

Рисунок 1 – Интерфейс программы пользователя

Создадим файл, записав в первое поле ввода “Hello World!” а во второе поле – название файла test.txt. Нажмем кнопку “Сохранить”. Результат представлен на рисунке 2.

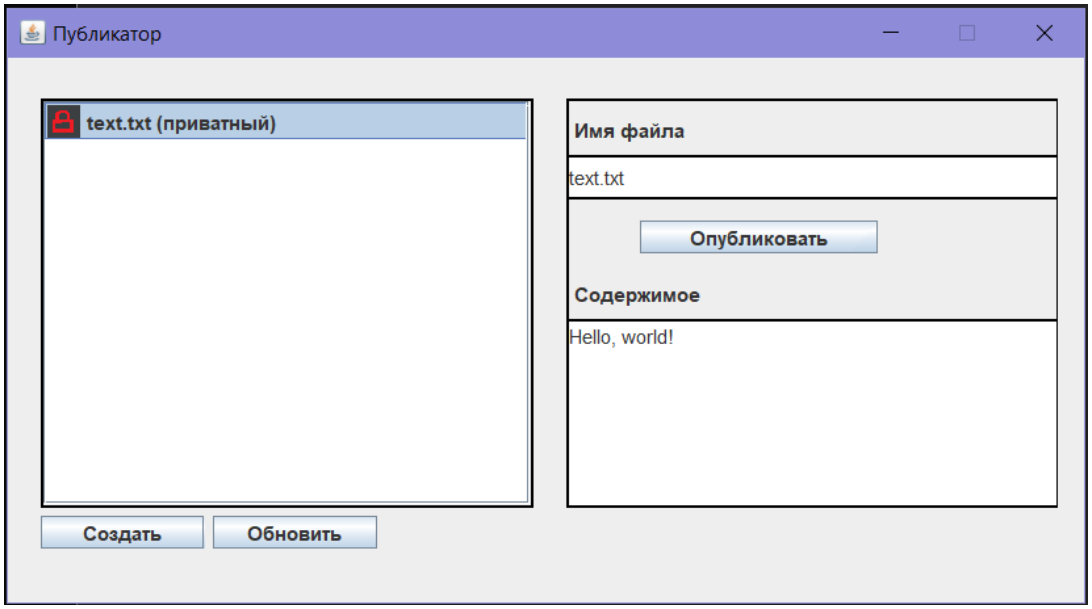

Рисунок 2 – Создание файла

Как видно из рисунка 2, после создания файла, список приватной папки автоматически обновился и в нем появился новый файл. Откроем этот файл через проводник и проверим его содержимое. Результат на рисунке 3.

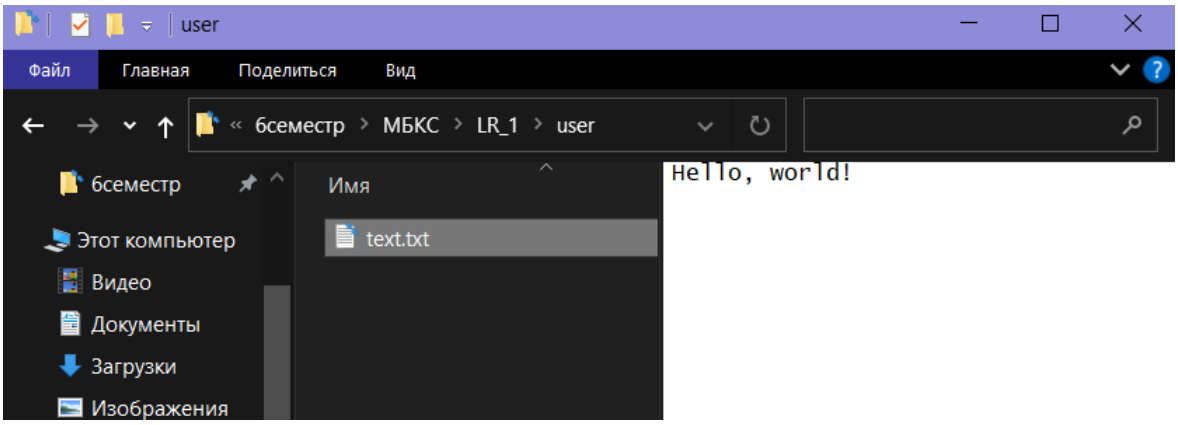

Рисунок 3 – Созданный файл

Как видно из рисунка 3 – файл создался корректно. Теперь выберем его и нажмем кнопку “Опубликовать”. Результат представлен на рисунке 4.

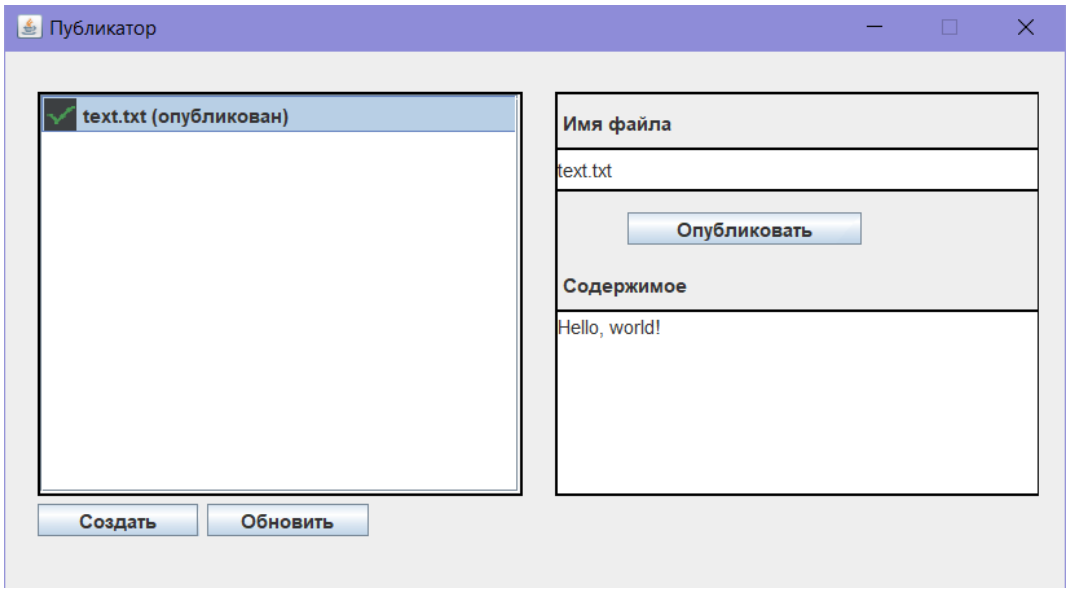

Рисунок 4 – Перенос файла

Копия файла появилась в публичной папке.

Создадим много новых файлов в проводнике. Результат на рисунке 5.

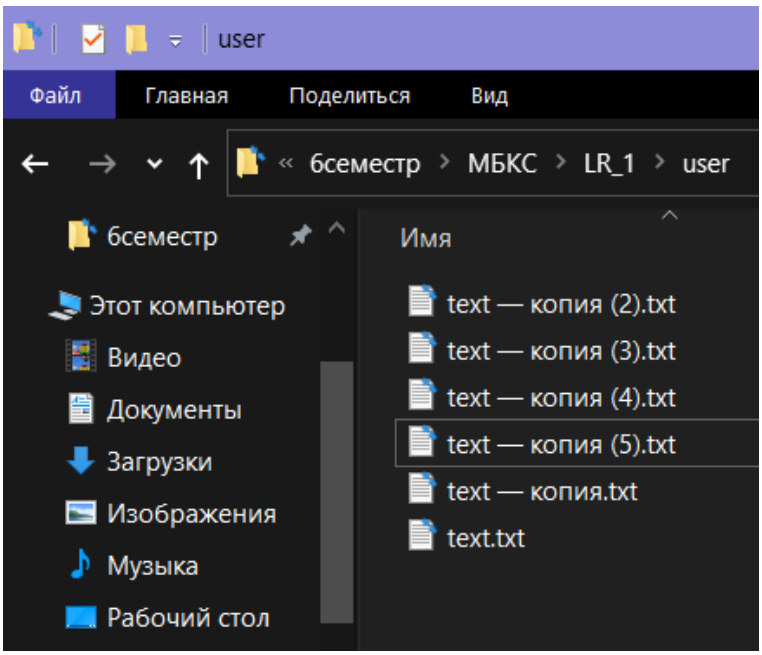

Рисунок 5 – Много новых файлов

Нажмем на кнопку “Обновить”. Результат после нажатия на рисунке 6.

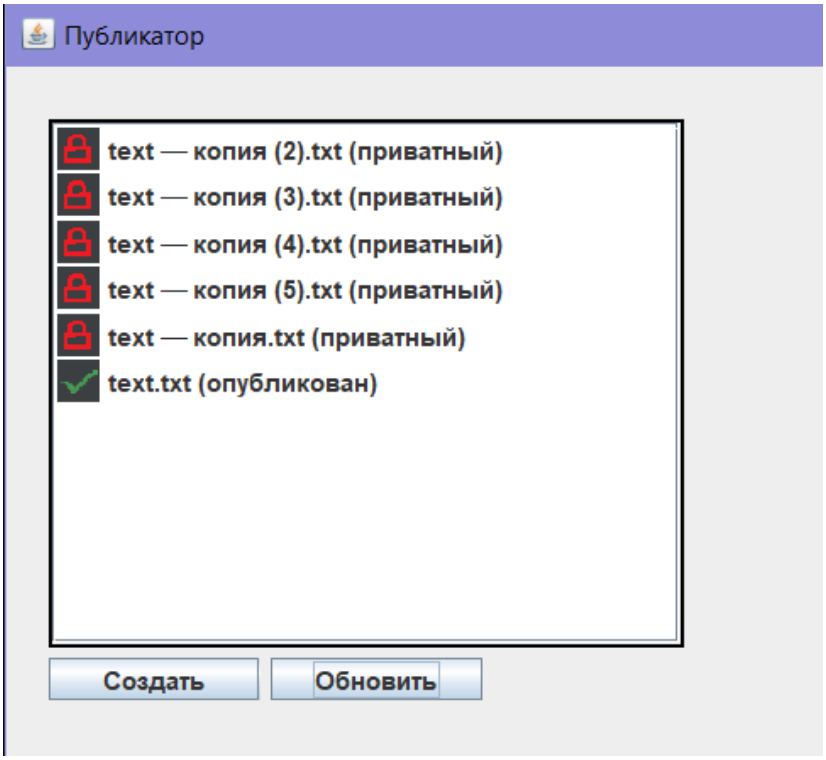

Рисунок 6 – Обновление списка через проводник

Все файлы появились в приложении.

2) В первую очередь, была реализована программа нарушитель.

- Был выбран консольный интерфейс программы, так как хакер в первую очередь, должен уметь работать с консольной строкой. Конечный интерфейс программы представлен на рисунке 7.

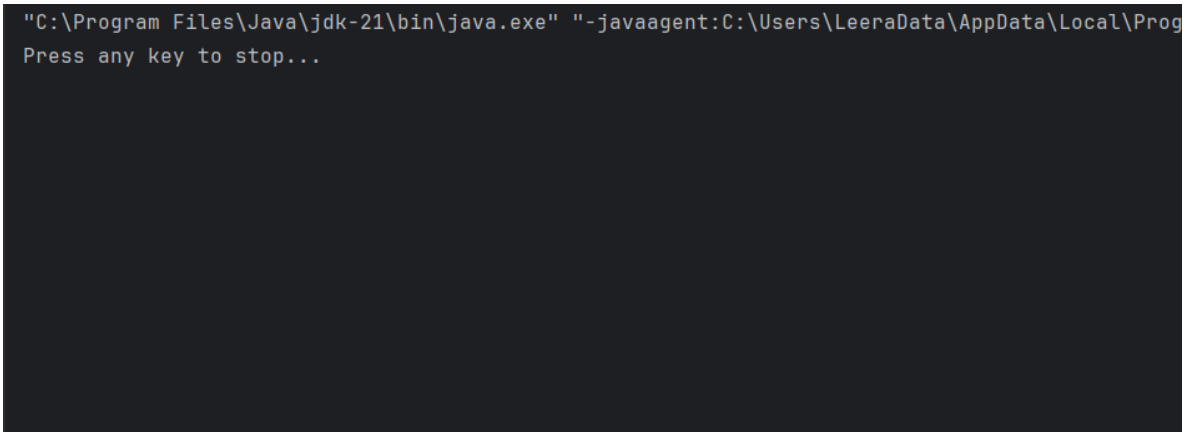

Рисунок 7 – Интерфейс программы

- Как только мы запустили программу она начала сканирование публичной папки. Он следит за папкой, фиксируя любы события, которые с этой папкой происходят (добавление нового файла, перезапись существуещего или удаление файла).

- Если мы опубликуем новый или перезаписанный файл из приватной папки, то сканер выведет сообшение «Шалость удалась – полный_путь_к_файлу». К примеру опубликуем файл «text.txt». Результат работы сканнера изображен на рисунке 8.

Рисунок 8 – Результат работы сканнера.

Выводы.

В результате выполнения данной лабораторной работы было реализовано два различных программных средства:

1. Программное средство создающее уязвимость путем создания и копирования содержимого файла в директорию, позволяющую субъекту, не обладающему доступом на чтение прочитать его содержимое и использовать в угодных себе целях.

2. Программное средство эксплуатирующее уязвимость по копированию содержимого приватного файла, содержащего в себе секретную информацию.

Обе программы показывают важность распределения доступа к файлам для различных групп субъектов и ограничения доступа информационного потока по памяти.

Код приложения.

1. Программа пользователя.

```
package gorbuno.mkbs.lab1;
import java.io.*;
import gorbuno.mkbs.lab1.gui.FilePublisher;
public class User {
    public static void main(String[] args) {
        File privateFolder = new File(args[0]);
        File publicFolder = new File(args[1]);
        if (!privateFolder.exists() || !publicFolder.exists()) {
            System.out.println("Enter    correct    folders    on
startup!");
            return;}
        new       FilePublisher("Публикатор",       privateFolder,
publicFolder);}}
```

2. Программа нарушителя.

```
package gorbuno.mkbs.lab1;
import java.io.*;

import gorbuno.mkbs.lab1.services.FsWatcher;

public class Hacker {
    public static void main(String[] args) {
        File hackerFolder = new File(args[0]);
        File publicFolder = new File(args[1]);

        if (!hackerFolder.exists() || !publicFolder.exists()
            ||            !hackerFolder.isDirectory()            ||
!publicFolder.isDirectory()) {
            System.out.println("Enter    correct    folders    on
startup!");
            return;
        }

        Thread      watcherThread      =      new      Thread(new
FsWatcher(publicFolder, hackerFolder));
        watcherThread.start();

        System.out.println("Press any key to stop...");
        try {
            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }

        watcherThread.interrupt();
    }
```

```
}
```

3. Вспомогательные классы

```
package gorbuno.mkbs.lab1.gui;
import static gorbuno.mkbs.lab1.utils.FsUtils.readFile;
import static gorbuno.mkbs.lab1.utils.FsUtils.updateFile;
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.io.*;
import java.util.List;
import java.util.*;
import javax.swing.*;
import javax.swing.border.LineBorder;
import org.apache.commons.io.FileUtils;
public class FilePublisher extends JFrame {
    private File privateFolder;
    private File publicFolder;
    public JPanel leftPanel = new JPanel(null);
    public JPanel fileListPanel = new JPanel(new BorderLayout());
    public     DefaultListModel<File>     listModel     =     new
DefaultListModel<>();
    private JList<File> fileList = new JList<>(listModel);
    public JButton newFileButton = new JButton("Создать");
    public JButton refreshButton = new JButton("Обновить");
    public JPanel rightPanel = new JPanel(null);
    public JLabel fileNameLabel = new JLabel("Имя файла");
    public JTextField filenameField = new JTextField();
    public JLabel contentLabel = new JLabel("Содержимое");
    public JTextArea contentArea = new JTextArea();
    public JButton saveFileButton = new JButton("Сохранить");
    public JButton publishButton = new JButton("Опубликовать");
    public         List<JComponent>         borderedItems         =
Arrays.asList(fileListPanel,        rightPanel,        contentArea,
filenameField);

    public  FilePublisher(final  String  title,  File  privateFolder,
File publicFolder) {
        super(title);
        this.privateFolder = privateFolder;
        this.publicFolder = publicFolder;
        setupUiComponents();
        composeUiComponents();
        setupActions();
        setupWindow();
        refresh();
    }

    public void setupUiComponents() {
        leftPanel.setBounds(20, 25, 300, 300);
        fileListPanel.setBounds(0, 0, 300, 250);
        newFileButton.setBounds(0, 255, 100, 20);
```

```
        refreshButton.setBounds(105, 255, 100, 20);
        rightPanel.setBounds(340, 25, 300, 250);
        fileNameLabel.setBounds(5, 7, 150, 25);
        filenameField.setBounds(0, 35, 300, 27);
        saveFileButton.setBounds(195, 75, 100, 20);
        publishButton.setBounds(45, 75, 145, 20);
        contentLabel.setBounds(5, 107, 200, 25);
        contentArea.setBounds(0, 135, 300, 115);

        for (JComponent item : borderedItems) {
            item.setBorder(new LineBorder(Color.BLACK, 2));}

        fileList.setCellRenderer(new
FileListCellRenderer(publicFolder));}

    public void composeUiComponents() {
        this.add(leftPanel);
        this.add(rightPanel);
        leftPanel.add(fileListPanel);
        leftPanel.add(newFileButton);
        leftPanel.add(refreshButton);
        rightPanel.add(fileNameLabel);
        rightPanel.add(filenameField);
        rightPanel.add(saveFileButton);
        rightPanel.add(publishButton);
        rightPanel.add(contentLabel);
        rightPanel.add(contentArea);
        fileListPanel.add(new                JScrollPane(fileList),
BorderLayout.CENTER);}

    public void setupWindow() {(null = по x y)
        setLayout(null);
        setSize(670, 370);
        setResizable(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addWindowListener(new WindowListener() {
            @Override
            public void windowOpened(WindowEvent arg0) {}
            @Override
            public void windowIconified(WindowEvent arg0) {}
            @Override
            public void windowDeiconified(WindowEvent arg0) {}
            @Override
            public void windowActivated(WindowEvent arg0) {}
            @Override
            public void windowDeactivated(WindowEvent arg0) {}
            @Override
            public void windowClosing(WindowEvent arg0) {}
            @Override
            public void windowClosed(WindowEvent arg0) {
                System.exit(0);}});
        this.setVisible(true);}
```

```
    public void setupActions() {
        refreshButton.addActionListener(e -> refresh());
        fileList.addListSelectionListener(e -> {
            int selectedIndex = fileList.getSelectedIndex();
            if (selectedIndex < 0) return;}
            File                  selectedValue                    =
listModel.getElementAt(selectedIndex);
            filenameField.setText(selectedValue.getName());
            try {
                contentArea.setText(readFile(selectedValue));
            } catch (IOException ex) {
                System.out.println("Ошибка    чтения    выбранного
файла");
                refresh();}
            rightPanel.show();
            publishButton.show();
            saveFileButton.hide();});
        newFileButton.addActionListener(ev -> {
            fileList.clearSelection();
            rightPanel.setVisible(true);
            filenameField.setText("");
            contentArea.setText("");
            publishButton.hide();
            saveFileButton.show();});
        saveFileButton.addActionListener(ev -> {
            int selectedIndex = fileList.getSelectedIndex();
            String                targetFileName                   =
privateFolder.getAbsolutePath()      +      File.separator      +
filenameField.getText();
            File  rewrittenFile  =  selectedIndex  <  0  ?  null  :
listModel.getElementAt(selectedIndex);
            try {
                updateFile(rewrittenFile,          targetFileName,
contentArea.getText());
            } catch (IOException e) {
                System.out.println("Ошибка  Сохранения  файла  "  +
targetFileName);}
            refresh();});
        publishButton.addActionListener(ev -> {
            File                    source                           =
listModel.getElementAt(fileList.getSelectedIndex());
            File  target  =  new  File(publicFolder.getAbsolutePath()
+ File.separator + source.getName());
            try {
                FileUtils.copyFile(source, target);
            } catch (IOException e) {
                System.out.println("Ошибка     копирования     при
публикации файла");}
            refresh();});
        ((javax.swing.text.AbstractDocument)
contentArea.getDocument()).setDocumentFilter(new
InputCallbackFilter(false, this));
```

```
        ((javax.swing.text.AbstractDocument)
filenameField.getDocument()).setDocumentFilter(new
InputCallbackFilter(true, this));}
    private void refresh() {
        rightPanel.setVisible(false);
        listModel.clear();
        File[] files = privateFolder.listFiles();
        if (files != null) {
            for (File file : files) {
                listModel.addElement(file);}}
        this.revalidate();
        this.repaint();}}
```

```
package gorbuno.mkbs.lab1.gui;
import                                                    static
gorbuno.mkbs.lab1.utils.FsUtils.areFileNamesAndContentEqual;
import static gorbuno.mkbs.lab1.utils.FsUtils.areFileNamesEqual;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.util.stream.Stream;
import javax.swing.*;

public class FileListCellRenderer extends DefaultListCellRenderer
{
    ImageIcon             publicIcon             =             new
ImageIcon("src/main/resources/icons/public.png");
    ImageIcon              oldIcon               =             new
ImageIcon("src/main/resources/icons/old.png");
    ImageIcon            privateIcon             =             new
ImageIcon("src/main/resources/icons/private.png");
    File folder;
    public FileListCellRenderer(final File folder) {
        super();
        this.folder = folder;}
    @Override
    public Component getListCellRendererComponent(JList<?> list,
Object value, int index, boolean isSelected, boolean cellHasFocus)
{JLabel label = (JLabel) super.getListCellRendererComponent(list,
value, index, isSelected, cellHasFocus);
        String addition;
        File[] publicFiles = folder.listFiles();
        if        (!(value        instanceof        File)        ||
Objects.isNull(publicFiles)) {
            return label;}
        File file = (File) value;
        if          (Stream.of(publicFiles).anyMatch(f          ->
areFileNamesEqual(f, file))) {
            if (Stream.of(publicFiles).anyMatch(f -> {
```

```
                try {
                    return areFileNamesAndContentEqual(f, file);
                } catch (IOException ex) {
                    throw new RuntimeException(ex);}})) {
                label.setIcon(publicIcon);
                addition = " (опубликован)";
            } else {
                label.setIcon(oldIcon);
                addition = " (изменен после публикации)";}
        } else {
            label.setIcon(privateIcon);
            addition = " (приватный)";}
        label.setText(file.getName() + addition);
        return label;}}
```

```
package gorbuno.mkbs.lab1.utils;
import java.io.*;
import java.nio.file.Paths;
import java.security.MessageDigest;
import java.util.*;
import org.apache.commons.io.FileUtils;
public class FsUtils {
    public static boolean areFileNamesEqual(File f1, File f2) {
        return f1.getName().equals(f2.getName());}
    public static String readFile(File file) throws IOException {
        Scanner scanner = new
Scanner(Paths.get(file.getAbsolutePath()));
        StringJoiner content = new StringJoiner("\n");
        while(scanner.hasNextLine()) {
            content.add(scanner.nextLine()); }
        scanner.close(); // убиваем сканер
        return content.toString();}
    public static boolean areFileNamesAndContentEqual(File f1,
File f2) throws IOException {
        return f1.getName().equals(f2.getName()) &&
FileUtils.contentEquals(f1, f2);}
    public static void updateFile(File file, String fileName,
String content) throws IOException {
            File newFile = new File(fileName);
            FileWriter writer = new FileWriter(newFile);
            writer.write(content);
            writer.close();
            if (Objects.nonNull(file) && file.exists() &&
!file.getName().equals(newFile.getName())) {
                file.delete();
            }}
    public static String computeFileSha256HashCode(File file) {
        try {
```

```
            MessageDigest digest = MessageDigest.getInstance("SHA-
256");
            FileInputStream fis = new FileInputStream(file);
            byte[] buffer = new byte[8192];
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                digest.update(buffer, 0, bytesRead);
            }
            fis.close();
            byte[] hash = digest.digest();
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) {
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

```
package gorbuno.mkbs.lab1.services;
import                                                    static
gorbuno.mkbs.lab1.utils.FsUtils.computeFileSha256HashCode;
import java.io.*;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import java.util.*;
import java.util.stream.Stream;
import org.apache.commons.io.FileUtils;
public class FsWatcher implements Runnable {
    private File publicFolder;
    private File hackerFolder;
    public FsWatcher(File publicFolder, File hackerFolder) {
        this.publicFolder = publicFolder;
        this.hackerFolder = hackerFolder;
    }

    @Override
    public void run() {
        try {
            Path path = Paths.get(publicFolder.getAbsolutePath());
```

```
            WatchService                watchService                =
FileSystems.getDefault().newWatchService();
            path.register(watchService,
StandardWatchEventKinds.ENTRY_CREATE,
StandardWatchEventKinds.ENTRY_MODIFY);
            while (true) {
                WatchKey key = watchService.take();
                for  (WatchEvent<?>  event  :  key.pollEvents())  {
WatchEvent<Path> ev = (WatchEvent<Path>) event;
                    Path fileName = ev.context();
                    File    changedFile    =    new    File(path    +
File.separator + fileName);
                    if (changedFile.isFile()) {
                        tryCommitFile(changedFile);
                    }
                }
                key.reset();
            }
        } catch (Exception e) {
            System.out.println("Наблюдение за папкой прервано");
        }}


    private void tryCommitFile(File file) {
        File             targetFolder             =             new
File(hackerFolder.getAbsolutePath()     +     File.separator     +
file.getName());
        if  (!targetFolder.exists()  &&  !targetFolder.mkdir()  ||
targetFolder.exists() && targetFolder.isFile()) {
            System.out.println("При  попытке  доступа  к  целевой
папке возникла ошибка");
            return;
        }
        String hashName = computeFileSha256HashCode(file);
        if (hashName.isEmpty()) {
            System.out.println("При попытке анализа файла возникла
ошибка");
            return;}
        if    (Stream.of(targetFolder.listFiles()).noneMatch(f    ->
f.getName().equals(hashName))) {
            String  newFilePath  =  targetFolder.getAbsolutePath()  +
File.separator + hashName + ".txt";
            try {
                FileUtils.copyFile(file, new File(newFilePath));
                System.out.println("Шалость  удалась  -  Скопирован
файл: " + file.getAbsolutePath());
            } catch (IOException e) {
                System.out.println("При копировании файла возникла
ошибка");}}}}
```