**Задание**

**Вариант 9.**

Реализуйте модель документа, которая обеспечивает хранение всей истории

изменений значений своих атрибутов. Реализуйте программу, которая

позволяет:

1. Вносить изменения в документ

2. Фиксировать изменения значений (для сохранения текущих изменений)

3. Выполнять откат состояния документа к любым изменениям, сделанным

ранее Система должна быть расширяема по документам любой структуры.

**Проектное решение**

**Обоснование выбора паттернов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***№*** | ***Паттерн*** | ***Группа*** | ***Ключевые классы / интерфейсы*** | ***Роли элементов паттерна в проекте*** |
| ***1*** | ***Factory Method*** | ***Порождающий*** | *DocumentFactory – интерфейс; TextDocumentFactory, PdfDocumentFactory, SpreadsheetDocumentFactory – конкретные фабрики;*  *Document — продукт;* | *Каждая фабрика создаёт нужный подкласс Document, изолируя клиентский код (DocumentEditorGUI) от деталей конкретных документов* |
| ***2*** | ***Singleton*** | ***Порождающий*** | *LogCollector;* | *Статический getInstance() гарантирует единственный объект-хранилище логов для всех декораторов и GUI ogCollector.java* |
| ***3*** | ***Decorator*** | ***Структурный*** | *DocumentDecorator (базовый), EncryptionDocumentDecorator, LoggingDocumentDecorator;* | *Оборачивает любой Document, не меняя его класса. Каждый декоратор добавляет сквозную функциональность (шифрование, логирование) и может комбинироваться в произвольном порядке* |
| ***4*** | ***Observer*** | ***Поведенческий*** | *DocumentObserver – наблюдатель; ConsoleLoggerObserver – конкретный наблюдатель; Document хранит список и рассылает события;* | *Document уведомляет GUI и консоль-лог, когда содержимое меняется. Наблюдатели могут свободно подключаться/отключаться* |
| ***5*** | ***Command*** | ***Поведенческий*** | *DocumentCommand – общий интерфейс; ChangeTextCommand, UndoCommand, RedoCommand, RevertCommand – команды; DocumentEditorGUI выступает в роли Invoker;* | *Каждая операция пользователя инкапсулирована в объект-команду, что упрощает историю действий, откат/повтор и централизует логику UI-кнопок* |
| ***6*** | ***Memento*** | ***Поведенческий*** | *DocumentMemento – Memento; GenericDocumentMemento – конкретный снимок;*  *Document – Originator; DocumentHistoryLogger – Caretaker;* | *Снимки состояния (Map<String,Object>) сохраняются в стеке. Undo/Redo берут memento из DocumentHistoryLogger и восстанавливают документ* |

На рисунке 1 приведена диаграмма классов.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1. – Диаграмма классов

**Приложение 1**

**Программный код**

DocumentFactory.java

package factory**;**import documents.Document**;**public interface DocumentFactory {  
 Document createDocument()**;**}

PdfDocumentFactory.java

package factory**;**import documents.Document**;**import documents.PdfDocument**;**public class PdfDocumentFactory implements DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new PdfDocument()**;** }  
}

SpreadsheetDocumentFactory.java

package factory**;**import documents.Document**;**import documents.SpreadsheetDocument**;**public class SpreadsheetDocumentFactory implements DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new SpreadsheetDocument()**;** }  
}

TextDocumentFactory.java

package factory**;**import documents.Document**;**import documents.TextDocument**;**public class TextDocumentFactory implements DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new TextDocument()**;** }  
}

LogCollector.java

package logger**;**import java.util.ArrayList**;**import java.util.Collections**;**import java.util.List**;**public class LogCollector {  
 private static final LogCollector *INSTANCE* = new LogCollector()**;** private final List<String> logs = new ArrayList<>()**;** private LogCollector() { }  
  
 public static LogCollector getInstance() {  
 return *INSTANCE***;** }  
  
 public void add(String message) {  
 logs.add(message)**;** }  
  
 public List<String> getLogs() {  
 return Collections.*unmodifiableList*(logs)**;** }  
}

Document.java

package documents**;**import memento.DocumentHistoryLogger**;**import memento.DocumentMemento**;**import observer.DocumentObserver**;**import java.util.ArrayList**;**import java.util.List**;**public abstract class Document {  
 protected DocumentHistoryLogger historyLogger**;** public void setHistoryLogger(DocumentHistoryLogger logger) {  
 this.historyLogger = logger**;** }  
  
 private final List<DocumentObserver> observers = new ArrayList<>()**;** public abstract String getContent()**;** public abstract void setContent(String content)**;** public abstract DocumentMemento createMemento()**;** public abstract void restoreFromMemento(DocumentMemento memento)**;** public void addObserver(DocumentObserver observer) {  
 if (observer != null && !observers.contains(observer)) observers.add(observer)**;** }  
 public void removeObserver(DocumentObserver observer) { observers.remove(observer)**;** }  
  
 protected void notifyObservers() {  
 for (DocumentObserver obs : new ArrayList<>(observers)) obs.update(this)**;** }  
}

PdfDocument.java

package documents**;**import memento.DocumentMemento**;**import memento.GenericDocumentMemento**;**import java.util.Map**;**public class PdfDocument extends Document {  
  
 private String content = ""**;** @Override  
 public String getContent() { return content**;** }  
  
 @Override  
 public void setContent(String content) {  
 if (content == null) content = ""**;** if (!this.content.equals(content)) {  
 this.content = content**;** notifyObservers()**;** if (historyLogger != null)  
 historyLogger.addMemento(createMemento())**;** }  
 }  
  
 @Override  
 public DocumentMemento createMemento() {  
 return new GenericDocumentMemento(Map.*of*("content"**,** content))**;** }  
  
 @Override  
 public void restoreFromMemento(DocumentMemento memento) {  
 String restored = (String) memento.getState().getOrDefault("content"**,** "")**;** if (!content.equals(restored)) {  
 content = restored**;** notifyObservers()**;** }  
 }  
}

SpreadsheetDocument.java

package documents**;**import memento.DocumentMemento**;**import memento.GenericDocumentMemento**;**import java.util.Map**;**public class SpreadsheetDocument extends Document {  
  
 private String content = ""**;** @Override  
 public String getContent() { return content**;** }  
  
 @Override  
 public void setContent(String content) {  
 if (content == null) content = ""**;** if (!this.content.equals(content)) {  
 this.content = content**;** notifyObservers()**;** if (historyLogger != null)  
 historyLogger.addMemento(createMemento())**;** }  
 }  
  
 @Override  
 public DocumentMemento createMemento() {  
 return new GenericDocumentMemento(Map.*of*("content"**,** content))**;** }  
  
 @Override  
 public void restoreFromMemento(DocumentMemento memento) {  
 String restored = (String) memento.getState().getOrDefault("content"**,** "")**;** if (!content.equals(restored)) {  
 content = restored**;** notifyObservers()**;** }  
 }  
}

TextDocument.java

package documents**;**import memento.DocumentMemento**;**import memento.GenericDocumentMemento**;**import java.util.Map**;**public class TextDocument extends Document {  
  
 private String content = ""**;** @Override  
 public String getContent() { return content**;** }  
  
 @Override  
 public void setContent(String content) {  
 if (content == null) content = ""**;** if (!this.content.equals(content)) {  
 this.content = content**;** notifyObservers()**;** if (historyLogger != null)  
 historyLogger.addMemento(createMemento())**;** }  
 }  
  
 @Override  
 public DocumentMemento createMemento() {  
 return new GenericDocumentMemento(Map.*of*("content"**,** content))**;** }  
  
 @Override  
 public void restoreFromMemento(DocumentMemento memento) {  
 String restored = (String) memento.getState().getOrDefault("content"**,** "")**;** if (!content.equals(restored)) {  
 content = restored**;** notifyObservers()**;** }  
 }  
}

DocumentDecorator.java

package decorators**;**import documents.Document**;**import memento.DocumentMemento**;**import observer.DocumentObserver**;**public abstract class DocumentDecorator extends Document {  
 protected Document document**;** public DocumentDecorator(Document document) {  
 if (document == null) {  
 throw new IllegalArgumentException("Оборачиваемый документ не может быть null")**;** }  
 this.document = document**;** }  
  
 @Override  
 public String getContent() {  
 return document.getContent()**;** }  
  
 @Override  
 public void setContent(String content) {  
 document.setContent(content)**;** }  
  
 @Override  
 public DocumentMemento createMemento() {  
 return document.createMemento()**;** }  
  
 @Override  
 public void restoreFromMemento(DocumentMemento memento) {  
 document.restoreFromMemento(memento)**;** }  
  
 @Override  
 public void addObserver(DocumentObserver observer) {  
 document.addObserver(observer)**;** }  
  
 @Override  
 public void removeObserver(DocumentObserver observer) {  
 document.removeObserver(observer)**;** }  
}

EncryptionDocumentDecorator.java

package decorators**;**import documents.Document**;**import memento.DocumentMemento**;**public class EncryptionDocumentDecorator extends DocumentDecorator {  
  
 public EncryptionDocumentDecorator(Document document) {  
 super(document)**;** }  
  
 @Override  
 public void setContent(String content) {  
 String encrypted = encrypt(content)**;** super.setContent(encrypted)**;** }  
  
 @Override  
 public String getContent() {  
 String encrypted = super.getContent()**;** return decrypt(encrypted)**;** }  
  
 private String encrypt(String input) {  
 if (input == null) return null**;** System.*out*.println("EncryptionDecorator: Шифрование данных...")**;** return new StringBuilder(input).reverse().toString()**;** }  
  
 private String decrypt(String input) {  
 if (input == null) return null**;** System.*out*.println("EncryptionDecorator: Дешифрование данных...")**;** return new StringBuilder(input).reverse().toString()**;** }  
}

LoggingDocumentDecorator.java

package decorators**;**import documents.Document**;**import logger.LogCollector**;**import memento.DocumentMemento**;**import observer.DocumentObserver**;**public class LoggingDocumentDecorator extends DocumentDecorator {  
 public LoggingDocumentDecorator(Document document) {  
 super(document)**;** }  
  
 @Override  
 public String getContent() {  
 LogCollector.*getInstance*().add("GET content")**;** return super.getContent()**;** }  
  
 @Override  
 public void setContent(String content) {  
 LogCollector.*getInstance*().add("SET content")**;** super.setContent(content)**;** }  
  
 @Override  
 public DocumentMemento createMemento() {  
 LogCollector.*getInstance*().add("CREATE memento")**;** return super.createMemento()**;** }  
  
 @Override  
 public void restoreFromMemento(DocumentMemento memento) {  
 LogCollector.*getInstance*().add("RESTORE from memento")**;** super.restoreFromMemento(memento)**;** }  
  
 @Override  
 public void addObserver(DocumentObserver observer) {  
 LogCollector.*getInstance*().add("ADD observer")**;** super.addObserver(observer)**;** }  
  
 @Override  
 public void removeObserver(DocumentObserver observer) {  
 LogCollector.*getInstance*().add("REMOVE observer")**;** super.removeObserver(observer)**;** }  
}

ConsoleLoggerObserver.java

package observer**;**import documents.Document**;**public class ConsoleLoggerObserver implements DocumentObserver {  
 @Override  
 public void update(Document document) {  
 System.*out*.println("ConsoleLoggerObserver: Документ был обновлен. Новое содержимое:")**;** System.*out*.println(">>> " + document.getContent())**;** }  
}

DocumentObserver.java

package observer**;**import documents.Document**;**public interface DocumentObserver {  
 void update(Document document)**;**}

DocumentMemento.java

package memento**;**import java.io.Serializable**;**import java.util.Map**;**public interface DocumentMemento extends Serializable {  
 Map<String**,** Object> getState()**;**}

GenericDocumentMemento.java

package memento**;**import java.io.Serial**;**import java.io.Serializable**;**import java.util.Map**;**public class GenericDocumentMemento implements DocumentMemento**,** Serializable {  
  
 @Serial  
 private static final long *serialVersionUID* = 1L**;** private final Map<String**,** Object> state**;** public GenericDocumentMemento(Map<String**,** Object> state) {  
 this.state = Map.*copyOf*(state)**;** }  
  
 @Override  
 public Map<String**,** Object> getState() {  
 return state**;** }  
}

DocumentHistoryLogger.java

package memento**;**import java.io.\***;**import java.util.ArrayList**;**import java.util.List**;**import java.util.Stack**;**public class DocumentHistoryLogger {  
  
 private final Stack<DocumentMemento> undoStack = new Stack<>()**;** private final Stack<DocumentMemento> redoStack = new Stack<>()**;** public void addMemento(DocumentMemento memento) {  
 if (memento == null) return**;** undoStack.push(memento)**;** redoStack.clear()**;** }  
  
 public boolean canUndo() { return !undoStack.isEmpty()**;** }  
 public boolean canRedo() { return !redoStack.isEmpty()**;** }  
  
 public DocumentMemento undo(DocumentMemento current) {  
 if (!canUndo()) return null**;** if (current != null) redoStack.push(current)**;** return undoStack.pop()**;** }  
  
 public DocumentMemento redo(DocumentMemento current) {  
 if (!canRedo()) return null**;** if (current != null) undoStack.push(current)**;** return redoStack.pop()**;** }  
  
 public List<DocumentMemento> getHistory() {  
 return new ArrayList<>(undoStack)**;** }  
 public DocumentMemento getMementoAt(int index) {  
 if (index < 0 || index >= undoStack.size()) return null**;** return undoStack.get(index)**;** }  
 public void clearRedo() { redoStack.clear()**;** }  
  
 public void saveHistoryToFile(File file) throws IOException {  
 try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file))) {  
 oos.writeObject(new ArrayList<>(undoStack))**;** }  
 }  
 @SuppressWarnings("unchecked")  
 public void loadHistoryFromFile(File file) throws IOException**,** ClassNotFoundException {  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {  
 List<DocumentMemento> list = (List<DocumentMemento>) ois.readObject()**;** undoStack.clear()**;** redoStack.clear()**;** undoStack.addAll(list)**;** }  
 }  
}

DocumentCommand.java

package commands**;**public interface DocumentCommand {  
 void execute()**;** void undo()**;**}

ChangeTextCommand.java

package commands**;**import documents.Document**;**public class ChangeTextCommand implements DocumentCommand {  
 private Document document**;** private String newText**;** private String previousText**;** public ChangeTextCommand(Document document**,** String newText) {  
 this.document = document**;** this.newText = newText**;** }  
  
 @Override  
 public void execute() {  
 previousText = document.getContent()**;** document.setContent(newText)**;** }  
  
 @Override  
 public void undo() {  
 document.setContent(previousText)**;** }  
}

UndoCommand.java

package commands**;**import documents.Document**;**import memento.DocumentHistoryLogger**;**import memento.DocumentMemento**;**public class UndoCommand implements DocumentCommand {  
  
 private final Document document**;** private final DocumentHistoryLogger logger**;** public UndoCommand(Document document**,** DocumentHistoryLogger logger) {  
 this.document = document**;** this.logger = logger**;** }  
  
 @Override  
 public void execute() {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.undo(current)**;** if (target != null) document.restoreFromMemento(target)**;** }  
  
 @Override  
 public void undo() {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.redo(current)**;** if (target != null) document.restoreFromMemento(target)**;** }  
}

RedoCommand.java

package commands**;**import documents.Document**;**import memento.DocumentHistoryLogger**;**import memento.DocumentMemento**;**public class RedoCommand implements DocumentCommand {  
  
 private final Document document**;** private final DocumentHistoryLogger logger**;** public RedoCommand(Document document**,** DocumentHistoryLogger logger) {  
 this.document = document**;** this.logger = logger**;** }  
  
 @Override  
 public void execute() {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.redo(current)**;** if (target != null) document.restoreFromMemento(target)**;** }  
  
 @Override  
 public void undo() {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.undo(current)**;** if (target != null) document.restoreFromMemento(target)**;** }  
}

RevertCommand.java

package commands**;**import documents.Document**;**import memento.DocumentHistoryLogger**;**import memento.DocumentMemento**;**public class RevertCommand implements DocumentCommand {  
  
 private final Document document**;** private final DocumentHistoryLogger historyLogger**;** private DocumentMemento savedCurrent**;** public RevertCommand(Document document**,** DocumentHistoryLogger historyLogger) {  
 this.document = document**;** this.historyLogger = historyLogger**;** }  
  
 @Override  
 public void execute() {  
 savedCurrent = document.createMemento()**;** DocumentMemento target = historyLogger.undo(savedCurrent)**;** if (target != null) document.restoreFromMemento(target)**;** }  
  
 @Override  
 public void undo() {  
 if (savedCurrent == null) return**;** DocumentMemento back = document.createMemento()**;** historyLogger.addMemento(back)**;** document.restoreFromMemento(savedCurrent)**;** savedCurrent = null**;** }  
}

DocumentEditorGUI.java

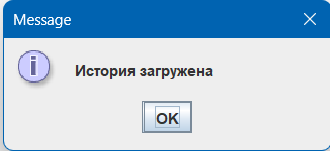
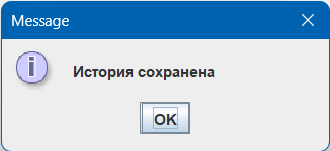
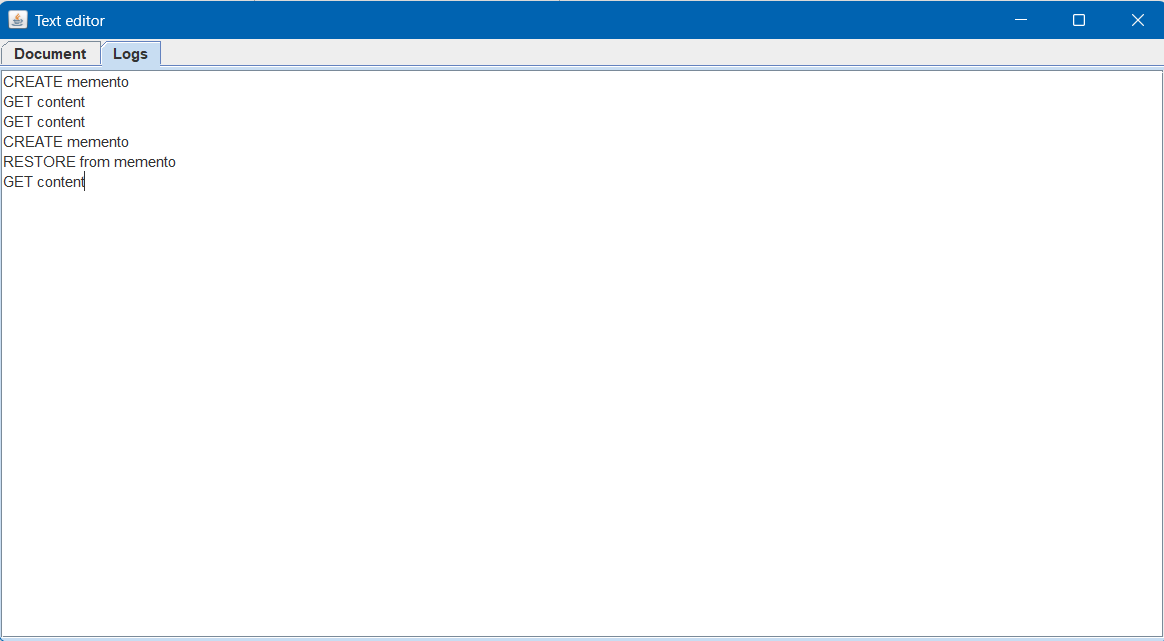
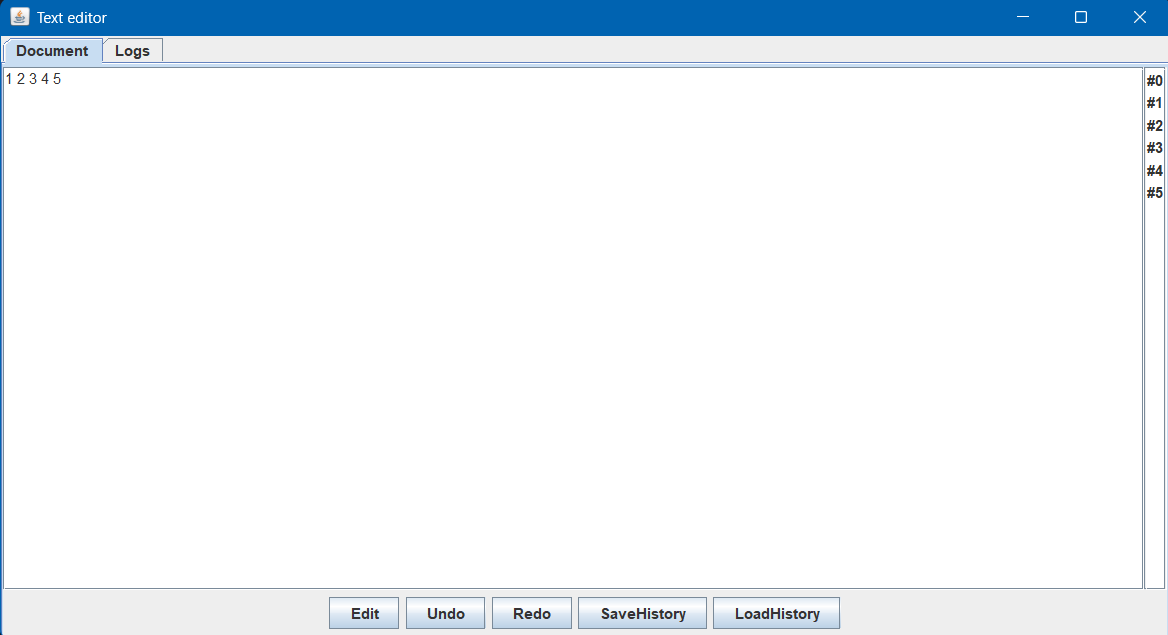
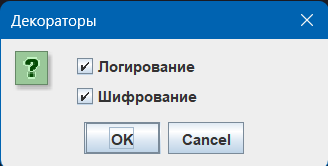
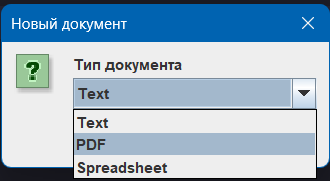
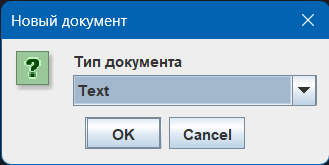
package main**;**import commands.\***;**import decorators.EncryptionDocumentDecorator**;**import decorators.LoggingDocumentDecorator**;**import documents.Document**;**import factory.\***;**import logger.LogCollector**;**import memento.DocumentHistoryLogger**;**import memento.DocumentMemento**;**import observer.ConsoleLoggerObserver**;**import observer.DocumentObserver**;**import javax.swing.\***;**import java.awt.\***;**import java.io.File**;**import java.util.stream.IntStream**;**public class DocumentEditorGUI extends JFrame implements DocumentObserver {  
  
 private JTextArea textArea**;** private JTextArea logArea**;** private DefaultListModel<String> historyModel**;** private JButton undoBtn**;** private JButton redoBtn**;** private Document document**;** private DocumentHistoryLogger logger**;** public DocumentEditorGUI() {  
 init()**;** setVisible(true)**;** }  
  
 private void init() {  
 String[] kinds = {"Text"**,** "PDF"**,** "Spreadsheet"}**;** String kind = (String) JOptionPane.*showInputDialog*(  
 null**,** "Тип документа"**,** "Новый документ"**,** JOptionPane.*QUESTION\_MESSAGE***,** null**,** kinds**,** kinds[0])**;** if (kind == null) System.*exit*(0)**;** DocumentFactory factory = switch (kind) {  
 case "PDF" -> new PdfDocumentFactory()**;** case "Spreadsheet" -> new SpreadsheetDocumentFactory()**;** default -> new TextDocumentFactory()**;** }**;** logger = new DocumentHistoryLogger()**;** document = factory.createDocument()**;** document.setHistoryLogger(logger)**;** document.addObserver(this)**;** document.addObserver(new ConsoleLoggerObserver())**;** JCheckBox logCB = new JCheckBox("Логирование"**,** true)**;** JCheckBox encCB = new JCheckBox("Шифрование"**,** true)**;** JPanel opts = new JPanel(new GridLayout(0**,**1))**;** opts.add(logCB)**;** opts.add(encCB)**;** if (JOptionPane.*showConfirmDialog*(  
 null**,** opts**,** "Декораторы"**,** JOptionPane.*OK\_CANCEL\_OPTION*) != JOptionPane.*OK\_OPTION*) System.*exit*(0)**;** if (encCB.isSelected()) document = new EncryptionDocumentDecorator(document)**;** if (logCB.isSelected()) document = new LoggingDocumentDecorator(document)**;** logger.addMemento(document.createMemento())**;** buildUI(kind)**;** }  
  
 private void buildUI(String kind) {  
 setTitle(kind + " editor")**;** setDefaultCloseOperation(*EXIT\_ON\_CLOSE*)**;** setLayout(new BorderLayout())**;** setPreferredSize(new Dimension(950**,** 520))**;** JTabbedPane tabs = new JTabbedPane()**;** JPanel docPanel = new JPanel(new BorderLayout())**;** textArea = new JTextArea()**;** textArea.setLineWrap(true)**;** docPanel.add(new JScrollPane(textArea)**,** BorderLayout.*CENTER*)**;** historyModel = new DefaultListModel<>()**;** JList<String> historyList = new JList<>(historyModel)**;** historyList.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*)**;** historyList.addListSelectionListener(e -> {  
 if (!e.getValueIsAdjusting()) {  
 int idx = historyList.getSelectedIndex()**;** if (idx >= 0) jumpToHistory(idx)**;** }  
 })**;** docPanel.add(new JScrollPane(historyList)**,** BorderLayout.*EAST*)**;** JPanel buttons = new JPanel(new FlowLayout())**;** JButton editBtn = new JButton("Edit")**;** undoBtn = new JButton("Undo")**;** redoBtn = new JButton("Redo")**;** JButton saveHistBtn = new JButton("SaveHistory")**;** JButton loadHistBtn = new JButton("LoadHistory")**;** editBtn.addActionListener(e -> editContent())**;** undoBtn.addActionListener(e -> { new UndoCommand(document**,** logger).execute()**;** refresh()**;** })**;** redoBtn.addActionListener(e -> { new RedoCommand(document**,** logger).execute()**;** refresh()**;** })**;** saveHistBtn.addActionListener(e -> saveHistoryToFile())**;** loadHistBtn.addActionListener(e -> loadHistoryFromFile())**;** buttons.add(editBtn)**;** buttons.add(undoBtn)**;** buttons.add(redoBtn)**;** buttons.add(saveHistBtn)**;** buttons.add(loadHistBtn)**;** docPanel.add(buttons**,** BorderLayout.*SOUTH*)**;** tabs.add("Document"**,** docPanel)**;** logArea = new JTextArea()**;** logArea.setEditable(false)**;** tabs.add("Logs"**,** new JScrollPane(logArea))**;** add(tabs**,** BorderLayout.*CENTER*)**;** pack()**;** setLocationRelativeTo(null)**;** refresh()**;** }  
  
 private void editContent() {  
 String current = document.getContent()**;** String newText = JOptionPane.*showInputDialog*(this**,** "Текст:"**,** current)**;** if (newText != null && !newText.equals(current)) {  
 new ChangeTextCommand(document**,** newText).execute()**;** refresh()**;** }  
 }  
  
 private void jumpToHistory(int index) {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.getMementoAt(index)**;** if (target != null) {  
 document.restoreFromMemento(target)**;** logger.addMemento(current)**;** logger.clearRedo()**;** refresh()**;** }  
 }  
  
 private void saveHistoryToFile() {  
 JFileChooser fc = new JFileChooser()**;** if (fc.showSaveDialog(this) == JFileChooser.*APPROVE\_OPTION*) {  
 try {  
 logger.addMemento(document.createMemento())**;** logger.saveHistoryToFile(fc.getSelectedFile())**;** JOptionPane.*showMessageDialog*(this**,** "История сохранена")**;** } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(this**,** "Ошибка сохранения: " + ex.getMessage())**;** }  
 }  
 }  
  
 private void loadHistoryFromFile() {  
 JFileChooser fc = new JFileChooser()**;** if (fc.showOpenDialog(this) == JFileChooser.*APPROVE\_OPTION*) {  
 try {  
 logger.loadHistoryFromFile(fc.getSelectedFile())**;** refresh()**;** JOptionPane.*showMessageDialog*(this**,** "История загружена")**;** } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(this**,** "Ошибка загрузки: " + ex.getMessage())**;** }  
 }  
 }  
  
 private void refresh() {  
 historyModel.clear()**;** IntStream.*range*(0**,** logger.getHistory().size())  
 .forEach(i -> historyModel.addElement("#" + i))**;** undoBtn.setEnabled(logger.canUndo())**;** redoBtn.setEnabled(logger.canRedo())**;** textArea.setText(document.getContent())**;** logArea.setText(String.*join*("\n"**,** LogCollector.*getInstance*().getLogs()))**;** }  
  
 @Override  
 public void update(Document d) {  
 if (d == document) textArea.setText(d.getContent())**;** }  
}

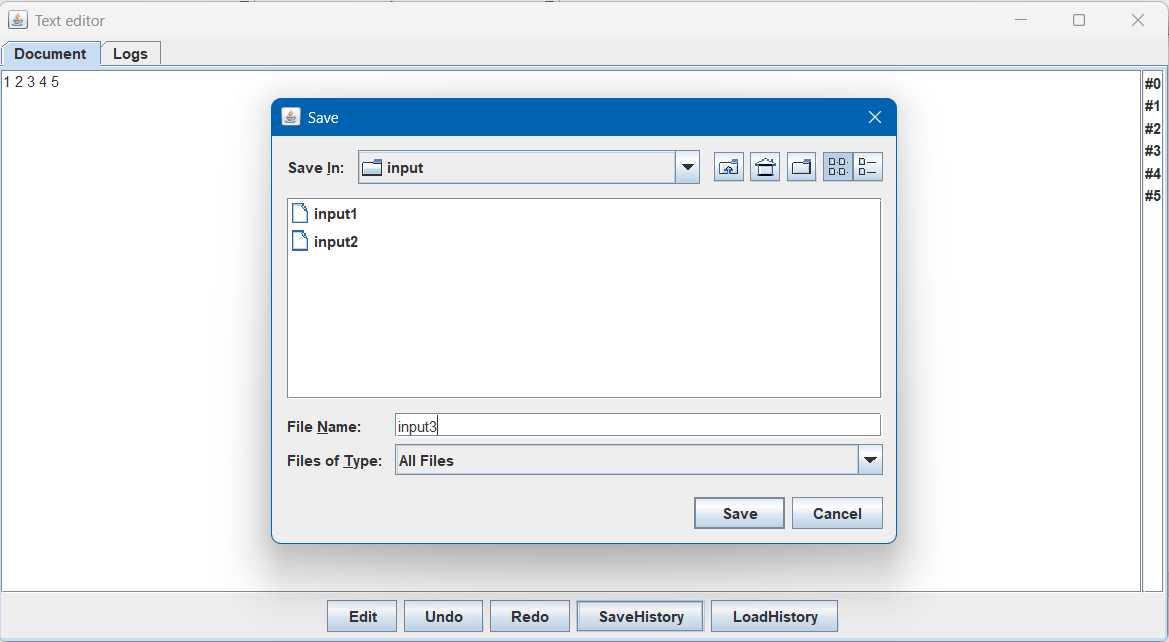
Main.java

package main**;**import javax.swing.SwingUtilities**;**public class Main {  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 DocumentEditorGUI editorGUI = new DocumentEditorGUI()**;** editorGUI.setVisible(true)**;** })**;** }  
}

**Приложение 2**

**Результаты тестирования**





**Сценарии работы**

1. **Редактирование**

*Нажать «Edit»* → модальное окно вводит новый текст → ChangeTextCommand.execute() меняет содержимое и автоматически фиксирует снимок в DocumentHistoryLogger.

1. **Отмена / повтор**

Кнопки **Undo** и **Redo** создают текущий снимок, просят у DocumentHistoryLogger предыдущий или следующий, восстанавливают документ и обновляют список.

1. **Прыжок к любому состоянию**

Щёлкнув по элементу #i в списке, GUI берёт i-й memento, восстанавливает документ, а текущий снимок помещает в стек «вперёд». Таким образом можно «скакать» по истории без потери данных.

1. **Сохранение истории**

*Нажать «SaveHistory»* → выбирается файл (\*.bin). DocumentHistoryLogger сериализует стек undoStack через ObjectOutputStream. Перед сохранением добавляется актуальный memento, чтобы изменения не потерялись.

1. **Загрузка истории**

*Нажать «LoadHistory»* → стек считывается из выбранного файла; список мгновенно перерисовывается, а документ показывает состояние самого последнего memento.