**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**

**Описание Spring**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***№*** | ***Компонент*** | ***Группа*** | ***Бин*** | ***Ключевые аннотации*** | ***Роль в проекте*** |
| ***1*** | ***AppConfig*** | ***singleton*** | ***✓*** | ***@Configuration, @ComponentScan, @EnableAspectJAutoProxy*** | ***Настройка Spring-контейнера и AOP*** |
| ***2*** | ***LoggingAspect*** | ***singleton*** | ***✓*** | ***@Aspect, @Component*** | ***AOP-аспект логирования (@Loggable)*** |
| ***3*** | ***LogCollector*** | ***singleton*** | ***✓*** | ***@Component*** | ***Сбор и хранение логов*** |
| ***4*** | ***TextDocumentFactory*** | ***singleton*** | ***✓*** | ***@Component*** | ***Фабрика создания TextDocument*** |
| ***5*** | ***PdfDocumentFactory*** | ***singleton*** | ***✓*** | ***@Component*** | ***Фабрика создания PdfDocument*** |
| ***6*** | ***SpreadsheetDocumentFactory*** | ***singleton*** | ***✓*** | ***@Component*** | ***Фабрика создания SpreadsheetDocument*** |
| ***7*** | ***EncryptionDocumentDecorator*** | ***prototype*** | ***✓*** | ***@Component, @Scope("prototype")*** | ***Декоратор шифрования/дешифрования*** |
| ***8*** | ***DocumentHistoryLogger*** | ***prototype*** | ***✓*** | ***@Component, @Scope("prototype")*** | ***Хранение истории (Memento), undo/redo*** |
| ***9*** | ***DocumentEditorGUI*** | ***singleton*** | ***✓*** | ***@Component, @PostConstruct*** | ***Swing-GUI: построение интерфейса и запуск команд*** |
| ***10*** | ***DocumentCommand (interface)*** | ***—*** | ***✗*** | ***—*** | ***Контракт команд: execute(), undo()*** |
| ***11*** | ***ChangeTextCommand*** | ***—*** | ***✗*** | ***—*** | ***Изменение текста документа*** |
| ***12*** | ***UndoCommand*** | ***—*** | ***✗*** | ***—*** | ***Откат к предыдущему состоянию*** |
| ***13*** | ***RedoCommand*** | ***—*** | ***✗*** | ***—*** | ***Возврат после отката*** |
| ***14*** | ***RevertCommand*** | ***—*** | ***✗*** | ***—*** | ***Переход к выбранному снимку истории*** |
| ***15*** | ***Document (abstract)*** | ***—*** | ***✗*** | ***—*** | ***Базовый класс документов: контент, Memento, Observer*** |
| ***16*** | ***TextDocument*** | ***—*** | ***✗*** | ***—*** | ***Конкретный текстовый документ*** |
| ***17*** | ***PdfDocument*** | ***—*** | ***✗*** | ***—*** | ***Конкретный PDF-документ*** |
| ***18*** | ***SpreadsheetDocument*** | ***—*** | ***✗*** | ***—*** | ***Конкретный документ-таблица*** |
| ***19*** | ***DocumentDecorator (abstract)*** | ***—*** | ***✗*** | ***—*** | ***Базовый класс для декораторов*** |
| ***20*** | ***DocumentMemento (interface)*** | ***—*** | ***✗*** | ***—*** | ***Интерфейс снимка состояния (Memento)*** |
| ***21*** | ***GenericDocumentMemento*** | ***—*** | ***✗*** | ***—*** | ***Реализация DocumentMemento*** |
| ***22*** | ***DocumentObserver (interface)*** | ***—*** | ***✗*** | ***—*** | ***Интерфейс наблюдателя документа*** |
| ***23*** | ***ConsoleLoggerObserver*** | ***—*** | ***✗*** | ***—*** | ***Вывод изменений документа в консоль*** |
| ***24*** | ***Loggable (annotation)*** | ***—*** | ***✗*** | ***@Target, @Retention*** | ***Маркер для AOP-логирования*** |
| ***25*** | ***DocumentFactory (interface)*** | ***—*** | ***✗*** | ***—*** | ***Контракт фабрик документов*** |
| ***26*** | ***Main*** | ***—*** | ***✗*** | ***—*** | ***Точка входа: старт Spring-контекста и GUI*** |

На рисунке 1 приведена диаграмма классов.

Изображение выглядит как диаграмма, текст, План, зарисовка

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1. – Диаграмма классов

**Приложение 1**

**Программный код**

DocumentFactory.java

package ru.doc.factory**;**import ru.doc.document.Document**;**public interface DocumentFactory {  
 Document createDocument()**;** String getName()**;**}

PdfDocumentFactory.java

package ru.doc.factory**;**import org.springframework.beans.factory.ObjectProvider**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.document.PdfDocument**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
public class PdfDocumentFactory implements DocumentFactory {  
 private final ObjectProvider<PdfDocument> pdfProvider**;** private final ObjectProvider<DocumentHistoryLogger> historyProvider**;** @Autowired  
 public PdfDocumentFactory(ObjectProvider<PdfDocument> pdfProvider**,** ObjectProvider<DocumentHistoryLogger> historyProvider) {  
 this.pdfProvider = pdfProvider**;** this.historyProvider = historyProvider**;** }  
  
 @Loggable(category = LogCollector.Category.*FACTORY***,** withResult = true**,** description = "Document creation")  
 @Override  
 public Document createDocument() {  
 PdfDocument doc = pdfProvider.getObject()**;** doc.setHistoryLogger(historyProvider.getObject())**;** return doc**;** }  
  
 @Override  
 public String getName() {  
 return "PDF"**;** }  
}

SpreadsheetDocumentFactory.java

package ru.doc.factory**;**import org.springframework.beans.factory.ObjectProvider**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.document.SpreadsheetDocument**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
public class SpreadsheetDocumentFactory implements DocumentFactory {  
 private final ObjectProvider<SpreadsheetDocument> sheetProvider**;** private final ObjectProvider<DocumentHistoryLogger> historyProvider**;** @Autowired  
 public SpreadsheetDocumentFactory(ObjectProvider<SpreadsheetDocument> sheetProvider**,** ObjectProvider<DocumentHistoryLogger> historyProvider) {  
 this.sheetProvider = sheetProvider**;** this.historyProvider = historyProvider**;** }  
  
 @Loggable(category = LogCollector.Category.*FACTORY***,** withResult = true**,** description = "Document creation")  
 @Override  
 public Document createDocument() {  
 SpreadsheetDocument doc = sheetProvider.getObject()**;** doc.setHistoryLogger(historyProvider.getObject())**;** return doc**;** }  
  
 @Override  
 public String getName() {  
 return "Spreadsheet"**;** }  
}

TextDocumentFactory.java

package ru.doc.factory**;**import org.springframework.beans.factory.ObjectProvider**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.document.TextDocument**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
public class TextDocumentFactory implements DocumentFactory {  
 private final ObjectProvider<TextDocument> textProvider**;** private final ObjectProvider<DocumentHistoryLogger> historyProvider**;** @Autowired  
 public TextDocumentFactory(ObjectProvider<TextDocument> textProvider**,** ObjectProvider<DocumentHistoryLogger> historyProvider) {  
 this.textProvider = textProvider**;** this.historyProvider = historyProvider**;** }  
  
 @Loggable(category = LogCollector.Category.*FACTORY***,** withResult = true**,** description = "Document creation")  
 @Override  
 public Document createDocument() {  
 TextDocument doc = textProvider.getObject()**;** doc.setHistoryLogger(historyProvider.getObject())**;** return doc**;** }  
  
 @Override  
 public String getName() {  
 return "Text"**;** }  
}

LogCollector.java

package ru.doc.logging**;**import org.springframework.stereotype.Component**;**import java.time.LocalDateTime**;**import java.time.format.DateTimeFormatter**;**import java.util.Collections**;**import java.util.List**;**import java.util.concurrent.CopyOnWriteArrayList**;**import java.util.stream.Collectors**;**@Component  
public class LogCollector {  
 public enum Level { *INFO***,** *DEBUG***,** *ERROR* }  
 public enum Category {  
 *DOCUMENT***,** *HISTORY***,** *FACTORY***,** *DECORATOR***,** *UI* }  
 public enum OperationType {  
 *CALL*("→")**,** *RETURN*("←")**,** *ERROR*("⚠")**;** private final String symbol**;** OperationType(String symbol) {  
 this.symbol = symbol**;** }  
 public String getSymbol() {  
 return symbol**;** }  
 }  
 private final List<String> records = new CopyOnWriteArrayList<>()**;** private static final DateTimeFormatter *timeFormatter* = DateTimeFormatter.*ofPattern*("HH:mm:ss.SSS")**;** public void add(Level level**,** Category category**,** OperationType type**,** String msg) {  
 String record = String.*format*("[%s] [%s] [%s] [%s] %s"**,** LocalDateTime.*now*().format(*timeFormatter*)**,** level**,** category**,** type.getSymbol()**,** msg)**;** records.add(record)**;** }  
  
 public void add(Level level**,** String msg) {  
 add(level**,** Category.*DOCUMENT***,** OperationType.*CALL***,** msg)**;** }  
  
 public void add(String msg) {  
 add(Level.*INFO***,** msg)**;** }  
  
 public List<String> getRecordsByCategory(Category category) {  
 return records.stream()  
 .filter(r -> r.contains("[" + category + "]"))  
 .collect(Collectors.*toList*())**;** }  
  
 public List<String> getRecordsByCategoryAndLevel(Category category**,** Level level) {  
 return records.stream()  
 .filter(r -> r.contains("[" + category + "]") && r.contains("[" + level + "]"))  
 .collect(Collectors.*toList*())**;** }  
  
 public List<String> getRecords() {  
 return Collections.*unmodifiableList*(records)**;** }  
  
 public List<String> getRecordsByLevel(Level level) {  
 return records.stream()  
 .filter(r -> r.contains("[" + level + "]"))  
 .collect(Collectors.*toList*())**;** }  
  
 public void clear() {  
 records.clear()**;** }  
}

Loggable.java

package ru.doc.logging**;**import java.lang.annotation.ElementType**;**import java.lang.annotation.Retention**;**import java.lang.annotation.RetentionPolicy**;**import java.lang.annotation.Target**;**@Target({ElementType.*METHOD***,** ElementType.*TYPE*})  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface Loggable {  
 LogCollector.Category category() default LogCollector.Category.*DOCUMENT***;** LogCollector.Level level() default LogCollector.Level.*INFO***;** boolean withArgs() default false**;** boolean withResult() default false**;** boolean withTime() default true**;** String description() default ""**;**}

LoggingAspect.java

package ru.doc.aop**;**import org.aspectj.lang.ProceedingJoinPoint**;**import org.aspectj.lang.annotation.Around**;**import org.aspectj.lang.annotation.Aspect**;**import org.aspectj.lang.annotation.Pointcut**;**import org.aspectj.lang.reflect.MethodSignature**;**import org.springframework.stereotype.Component**;**import ru.doc.logging.LogCollector**;**import ru.doc.logging.Loggable**;**import java.lang.reflect.Method**;**@Aspect  
@Component  
public class LoggingAspect {  
  
 private final LogCollector logCollector**;** public LoggingAspect(LogCollector logCollector) {  
 this.logCollector = logCollector**;** }  
  
 @Pointcut("execution(\* ru.doc.document..\*(..)) || execution(\* ru.doc.factory..\*(..)) || execution(\* ru.doc.commands..\*(..))")  
 public void applicationPointcut() {}  
  
 @Pointcut("@annotation(ru.doc.logging.Loggable)")  
 public void loggableMethod() {}  
  
 @Pointcut("@within(ru.doc.logging.Loggable)")  
 public void loggableClass() {}  
  
 @Around("loggableMethod() || loggableClass()")  
 public Object aroundLoggableMethod(ProceedingJoinPoint pjp) throws Throwable {  
 MethodSignature signature = (MethodSignature) pjp.getSignature()**;** Method method = signature.getMethod()**;** Loggable loggable = method.getAnnotation(Loggable.class)**;** if (loggable == null) {  
 loggable = method.getDeclaringClass().getAnnotation(Loggable.class)**;** }  
 if (loggable == null) {  
 return pjp.proceed()**;** }  
  
 LogCollector.Category category = loggable.category()**;** LogCollector.Level level = loggable.level()**;** boolean withArgs = loggable.withArgs()**;** boolean withResult = loggable.withResult()**;** boolean withTime = loggable.withTime()**;** String description = loggable.description()**;** String methodName = signature.toShortString()**;** String message = description.isEmpty() ? methodName : description + " (" + methodName + ")"**;** if (withArgs && pjp.getArgs().length > 0) {  
 StringBuilder args = new StringBuilder()**;** boolean first = true**;** for (Object arg : pjp.getArgs()) {  
 if (!first) {  
 args.append(", ")**;** }  
 first = false**;** try {  
 args.append(arg != null ? arg.toString() : "null")**;** } catch (Exception e) {  
 args.append("[Error getting argument]")**;** }  
 }  
 message += " Args: [" + args + "]"**;** }  
  
 logCollector.add(level**,** category**,** LogCollector.OperationType.*CALL***,** message)**;** long startTime = System.*currentTimeMillis*()**;** try {  
 Object result = pjp.proceed()**;** long elapsedTime = System.*currentTimeMillis*() - startTime**;** String returnMessage = message**;** if (withTime) {  
 returnMessage += " Time: " + elapsedTime + "ms"**;** }  
  
 if (withResult && result != null) {  
 try {  
 returnMessage += " Result: [" + result + "]"**;** } catch (Exception e) {  
 returnMessage += " Result: [Error getting result]"**;** }  
 }  
  
 logCollector.add(level**,** category**,** LogCollector.OperationType.*RETURN***,** returnMessage)**;** return result**;** } catch (Throwable ex) {  
 long elapsedTime = System.*currentTimeMillis*() - startTime**;** String errorMessage = message + " Exception: " + ex.getClass().getSimpleName()**;** if (withTime) {  
 errorMessage += " Time: " + elapsedTime + "ms"**;** }  
  
 logCollector.add(LogCollector.Level.*ERROR***,** category**,** LogCollector.OperationType.*ERROR***,** errorMessage)**;** throw ex**;** }  
 }  
  
 @Around("applicationPointcut() && !(loggableMethod() || loggableClass())")  
 public Object trace(ProceedingJoinPoint pjp) throws Throwable {  
 String sig = pjp.getSignature().toShortString()**;** logCollector.add(LogCollector.Level.*INFO***,** "CALL " + sig)**;** try {  
 Object res = pjp.proceed()**;** logCollector.add(LogCollector.Level.*INFO***,** "RET " + sig)**;** return res**;** } catch (Throwable ex) {  
 logCollector.add(LogCollector.Level.*ERROR***,** "THRW " + sig + " " + ex.getClass().getSimpleName())**;** throw ex**;** }  
 }  
}

Document.java

package ru.doc.document**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.memento.GenericDocumentMemento**;**import ru.doc.observer.DocumentObserver**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**import java.util.ArrayList**;**import java.util.HashMap**;**import java.util.List**;**import java.util.Map**;**public abstract class Document {  
 private String content = ""**;** private DocumentHistoryLogger historyLogger**;** private final List<DocumentObserver> observers = new ArrayList<>()**;** @Loggable(category = LogCollector.Category.*DOCUMENT***,** withResult = true)  
 public String getContent() {  
 return content**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withArgs = true)  
 public void setContent(String content) {  
 this.content = content**;** notifyObservers()**;** }  
  
 public void setHistoryLogger(DocumentHistoryLogger historyLogger) {  
 this.historyLogger = historyLogger**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** level = LogCollector.Level.*DEBUG*)  
 public DocumentMemento createMemento() {  
 Map<String**,** Object> state = new HashMap<>()**;** state.put("content"**,** content)**;** return new GenericDocumentMemento(state)**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** level = LogCollector.Level.*DEBUG*)  
 public void restoreFromMemento(DocumentMemento memento) {  
 Map<String**,** Object> state = memento.getState()**;** this.content = (String) state.get("content")**;** notifyObservers()**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** level = LogCollector.Level.*DEBUG*)  
 public void addObserver(DocumentObserver observer) {  
 if (!observers.contains(observer)) {  
 observers.add(observer)**;** }  
 }  
  
 public void removeObserver(DocumentObserver observer) {  
 observers.remove(observer)**;** }  
  
 private void notifyObservers() {  
 for (DocumentObserver observer : observers) {  
 observer.update(this)**;** }  
 }  
}

PdfDocument.java

package ru.doc.document**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
public class PdfDocument extends Document {  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withArgs = true)  
 @Override  
 public void setContent(String content) {  
 super.setContent(content)**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withResult = true)  
 @Override  
 public String getContent() {  
 return super.getContent()**;** }  
}

SpreadsheetDocument.java

package ru.doc.document**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
public class SpreadsheetDocument extends Document {  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withArgs = true)  
 @Override  
 public void setContent(String content) {  
 super.setContent(content)**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withResult = true)  
 @Override  
 public String getContent() {  
 return super.getContent()**;** }  
}

TextDocument.java

package ru.doc.document**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
public class TextDocument extends Document {  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withArgs = true)  
 @Override  
 public void setContent(String content) {  
 super.setContent(content)**;** }  
  
 @Loggable(category = LogCollector.Category.*DOCUMENT***,** withResult = true)  
 @Override  
 public String getContent() {  
 return super.getContent()**;** }  
}

DocumentDecorator.java

package ru.doc.decorators**;**import ru.doc.document.Document**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.observer.DocumentObserver**;**public abstract class DocumentDecorator extends Document {  
 protected Document document**;** public DocumentDecorator(Document document) {  
 if (document == null) {  
 throw new IllegalArgumentException("Wrapped document cannot be 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 ru.doc.decorators**;**import org.springframework.aop.framework.AopContext**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**import ru.doc.document.Document**;**@Component  
@Scope("prototype")  
public class EncryptionDocumentDecorator extends DocumentDecorator {  
 public EncryptionDocumentDecorator(Document document) {  
 super(document)**;** }  
  
 @Override  
 public void setContent(String content) {  
 EncryptionDocumentDecorator proxy = (EncryptionDocumentDecorator) AopContext.*currentProxy*()**;** String encrypted = proxy.encrypt(content)**;** super.setContent(encrypted)**;** }  
  
 @Override  
 public String getContent() {  
 String encrypted = super.getContent()**;** EncryptionDocumentDecorator proxy = (EncryptionDocumentDecorator) AopContext.*currentProxy*()**;** return proxy.decrypt(encrypted)**;** }  
  
 @Loggable(category = LogCollector.Category.*DECORATOR***,** level = LogCollector.Level.*DEBUG***,** withTime = true**,** description = "Data encryption")  
 public String encrypt(String input) {  
 if (input == null) return null**;** return new StringBuilder(input).reverse().toString()**;** }  
  
 @Loggable(category = LogCollector.Category.*DECORATOR***,** level = LogCollector.Level.*DEBUG***,** withTime = true**,** description = "Data decryption")  
 public String decrypt(String input) {  
 if (input == null) return null**;** return new StringBuilder(input).reverse().toString()**;** }  
}

ConsoleLoggerObserver.java

package ru.doc.observer**;**import ru.doc.document.Document**;**public class ConsoleLoggerObserver implements DocumentObserver {  
 @Override  
 public void update(Document document) {  
 System.*out*.println("ConsoleLoggerObserver: Document has been updated. New content:")**;** System.*out*.println(">>> " + document.getContent())**;** }  
}

DocumentObserver.java

package ru.doc.observer**;**import ru.doc.document.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 ru.doc.history**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.memento.DocumentMemento**;**import java.io.\***;**import java.util.ArrayList**;**import java.util.List**;**import java.util.Stack**;**@Component  
@Scope("prototype")  
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.size() > 1**;** }  
  
 public boolean canRedo() {  
 return !redoStack.isEmpty()**;** }  
  
 public DocumentMemento undo(DocumentMemento current) {  
 if (undoStack.size() <= 1) return null**;** redoStack.push(undoStack.pop())**;** return undoStack.peek()**;** }  
  
 public DocumentMemento redo(DocumentMemento current) {  
 if (redoStack.isEmpty()) return null**;** DocumentMemento memento = redoStack.pop()**;** undoStack.push(memento)**;** return memento**;** }  
  
 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 ru.doc.commands**;**public interface DocumentCommand {  
 void execute()**;** void undo()**;**}

ChangeTextCommand.java

package ru.doc.commands**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
public class ChangeTextCommand implements DocumentCommand {  
 private final Document document**;** private final String newText**;** private String previousText**;** public ChangeTextCommand(Document document**,** String newText) {  
 this.document = document**;** this.newText = newText**;** }  
  
 @Loggable(category = LogCollector.Category.*HISTORY*)  
 @Override  
 public void execute() {  
 previousText = document.getContent()**;** document.setContent(newText)**;** }  
  
 @Override  
 public void undo() {  
 document.setContent(previousText)**;** }  
}

UndoCommand.java

package ru.doc.commands**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
@Loggable(category = LogCollector.Category.*HISTORY*)  
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 ru.doc.commands**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
@Loggable(category = LogCollector.Category.*HISTORY*)  
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.redo(current)**;** if (target != null) {  
 document.restoreFromMemento(target)**;** }  
 }  
}

RevertCommand.java

package ru.doc.commands**;**import org.springframework.context.annotation.Scope**;**import org.springframework.stereotype.Component**;**import ru.doc.document.Document**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.logging.Loggable**;**import ru.doc.logging.LogCollector**;**@Component  
@Scope("prototype")  
@Loggable(category = LogCollector.Category.*HISTORY*)  
public class RevertCommand implements DocumentCommand {  
 private final Document document**;** private final DocumentHistoryLogger historyLogger**;** private final int targetIndex**;** private DocumentMemento savedCurrent**;** public RevertCommand(Document document**,** DocumentHistoryLogger historyLogger**,** int targetIndex) {  
 this.document = document**;** this.historyLogger = historyLogger**;** this.targetIndex = targetIndex**;** }  
  
 @Override  
 public void execute() {  
 savedCurrent = document.createMemento()**;** DocumentMemento target = historyLogger.getMementoAt(targetIndex)**;** 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**;** }  
}

AppConfig.java

package ru.doc.config**;**import org.springframework.context.annotation.ComponentScan**;**import org.springframework.context.annotation.Configuration**;**import org.springframework.context.annotation.EnableAspectJAutoProxy**;**@Configuration  
@ComponentScan(basePackages = "ru.doc")  
@EnableAspectJAutoProxy(proxyTargetClass = true**,** exposeProxy = true)  
public class AppConfig {  
}

DocumentEditorGUI.java

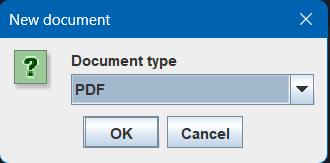
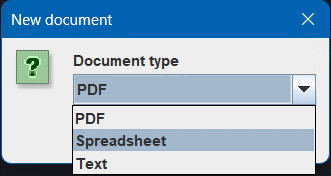
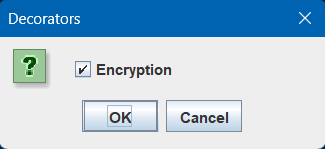
package ru.doc.gui**;**import ru.doc.commands.ChangeTextCommand**;**import ru.doc.commands.RedoCommand**;**import ru.doc.commands.UndoCommand**;**import ru.doc.commands.RevertCommand**;**import ru.doc.decorators.EncryptionDocumentDecorator**;**import ru.doc.document.Document**;**import ru.doc.factory.DocumentFactory**;**import ru.doc.logging.LogCollector**;**import ru.doc.history.DocumentHistoryLogger**;**import ru.doc.memento.DocumentMemento**;**import ru.doc.observer.ConsoleLoggerObserver**;**import ru.doc.observer.DocumentObserver**;**import org.springframework.beans.factory.ObjectProvider**;**import org.springframework.context.ApplicationContext**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.stereotype.Component**;**import javax.swing.\***;**import java.awt.\***;**import java.awt.event.MouseAdapter**;**import java.awt.event.MouseEvent**;**import java.io.BufferedWriter**;**import java.io.FileWriter**;**import java.util.List**;**import java.util.stream.IntStream**;**import jakarta.annotation.PostConstruct**;**@Component  
public class DocumentEditorGUI extends JFrame implements DocumentObserver {  
 private final List<DocumentFactory> factories**;** private final LogCollector logCollector**;** private final ObjectProvider<DocumentHistoryLogger> historyLoggerProvider**;** private final ApplicationContext context**;** private JTextArea textArea**;** private JTextArea logArea**;** private DefaultListModel<String> historyModel**;** private JButton undoBtn**;** private JButton redoBtn**;** private Document document**;** private DocumentHistoryLogger logger**;** private JRadioButton allBtn**;** private JRadioButton infoBtn**;** private JRadioButton debugBtn**;** private JRadioButton errorBtn**;** private JRadioButton allCatBtn**;** private JRadioButton docCatBtn**;** private JRadioButton histCatBtn**;** private JRadioButton factCatBtn**;** private JRadioButton decCatBtn**;** @Autowired  
 public DocumentEditorGUI(List<DocumentFactory> factories**,** LogCollector logCollector**,** ObjectProvider<DocumentHistoryLogger> historyLoggerProvider**,** ApplicationContext context) {  
 this.factories = factories**;** this.logCollector = logCollector**;** this.historyLoggerProvider = historyLoggerProvider**;** this.context = context**;** }  
  
 @PostConstruct  
 public void initialize() {  
 SwingUtilities.*invokeLater*(() -> {  
 try {  
 init()**;** setVisible(true)**;** } catch (Exception e) {  
 logCollector.add(LogCollector.Level.*ERROR***,** LogCollector.Category.*UI***,** LogCollector.OperationType.*ERROR***,** "Initialization error: " + e.getMessage())**;** JOptionPane.*showMessageDialog*(null**,** "Error: " + e.getMessage()**,** "Initialization Error"**,** JOptionPane.*ERROR\_MESSAGE*)**;** }  
 })**;** }  
  
 private void init() {  
 String[] names = factories.stream().map(DocumentFactory::getName).toArray(String[]::new)**;** String kind = (String) JOptionPane.*showInputDialog*(null**,** "Document type"**,** "New document"**,** JOptionPane.*QUESTION\_MESSAGE***,** null**,** names**,** names[0])**;** if (kind == null) System.*exit*(0)**;** DocumentFactory factory = factories.stream().filter(f -> f.getName().equals(kind)).findFirst().orElseThrow()**;** logger = historyLoggerProvider.getObject()**;** document = factory.createDocument()**;** document.setHistoryLogger(logger)**;** document.addObserver(this)**;** document.addObserver(new ConsoleLoggerObserver())**;** JCheckBox encCB = new JCheckBox("Encryption"**,** true)**;** JPanel opts = new JPanel(new GridLayout(0**,**1))**;** opts.add(encCB)**;** if (JOptionPane.*showConfirmDialog*(null**,** opts**,** "Decorators"**,** JOptionPane.*OK\_CANCEL\_OPTION*) != JOptionPane.*OK\_OPTION*)  
 System.*exit*(0)**;** if (encCB.isSelected()) {  
 document = context.getBean(EncryptionDocumentDecorator.class**,** 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)**;** }  
 })**;** JPopupMenu historyMenu = new JPopupMenu()**;** JMenuItem revertItem = new JMenuItem("Revert")**;** historyMenu.add(revertItem)**;** historyList.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mousePressed(MouseEvent e) {  
 if (e.isPopupTrigger() || SwingUtilities.*isRightMouseButton*(e)) {  
 int idx = historyList.locationToIndex(e.getPoint())**;** if (idx >= 0) {  
 historyList.setSelectedIndex(idx)**;** historyMenu.show(historyList**,** e.getX()**,** e.getY())**;** }  
 }  
 }  
  
 @Override  
 public void mouseReleased(MouseEvent e) {  
 if (e.isPopupTrigger() || SwingUtilities.*isRightMouseButton*(e)) {  
 int idx = historyList.locationToIndex(e.getPoint())**;** if (idx >= 0) {  
 historyList.setSelectedIndex(idx)**;** historyMenu.show(historyList**,** e.getX()**,** e.getY())**;** }  
 }  
 }  
  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (e.getClickCount() == 2 && SwingUtilities.*isLeftMouseButton*(e)) {  
 int idx = historyList.locationToIndex(e.getPoint())**;** if (idx >= 0) {  
 context.getBean(RevertCommand.class**,** document**,** logger**,** idx).execute()**;** logger.addMemento(document.createMemento())**;** logger.clearRedo()**;** refresh()**;** }  
 }  
 }  
 })**;** revertItem.addActionListener(e -> {  
 int idx = historyList.getSelectedIndex()**;** if (idx >= 0) {  
 context.getBean(RevertCommand.class**,** document**,** logger**,** idx).execute()**;** logger.addMemento(document.createMemento())**;** logger.clearRedo()**;** refresh()**;** }  
 })**;** 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("Save history")**;** JButton loadHistBtn = new JButton("Load history")**;** editBtn.addActionListener(e -> editContent())**;** undoBtn.addActionListener(e -> {  
 context.getBean(UndoCommand.class**,** document**,** logger).execute()**;** refresh()**;** })**;** redoBtn.addActionListener(e -> {  
 context.getBean(RedoCommand.class**,** 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)**;** JPanel logPanel = new JPanel(new BorderLayout())**;** JPanel filterPanel = new JPanel(new FlowLayout(FlowLayout.*LEFT*))**;** ButtonGroup group = new ButtonGroup()**;** allBtn = new JRadioButton("ALL"**,** true)**;** infoBtn = new JRadioButton("INFO")**;** debugBtn = new JRadioButton("DEBUG")**;** errorBtn = new JRadioButton("ERROR")**;** group.add(allBtn)**;** group.add(infoBtn)**;** group.add(debugBtn)**;** group.add(errorBtn)**;** filterPanel.add(new JLabel("Filter level:"))**;** filterPanel.add(allBtn)**;** filterPanel.add(infoBtn)**;** filterPanel.add(debugBtn)**;** filterPanel.add(errorBtn)**;** allBtn.addActionListener(e -> refreshLogs())**;** infoBtn.addActionListener(e -> refreshLogs())**;** debugBtn.addActionListener(e -> refreshLogs())**;** errorBtn.addActionListener(e -> refreshLogs())**;** JPanel categoryPanel = new JPanel(new FlowLayout(FlowLayout.*LEFT*))**;** ButtonGroup categoryGroup = new ButtonGroup()**;** allCatBtn = new JRadioButton("All Categories"**,** true)**;** docCatBtn = new JRadioButton("DOCUMENT")**;** histCatBtn = new JRadioButton("HISTORY")**;** factCatBtn = new JRadioButton("FACTORY")**;** decCatBtn = new JRadioButton("DECORATOR")**;** categoryGroup.add(allCatBtn)**;** categoryGroup.add(docCatBtn)**;** categoryGroup.add(histCatBtn)**;** categoryGroup.add(factCatBtn)**;** categoryGroup.add(decCatBtn)**;** categoryPanel.add(new JLabel("Filter category:"))**;** categoryPanel.add(allCatBtn)**;** categoryPanel.add(docCatBtn)**;** categoryPanel.add(histCatBtn)**;** categoryPanel.add(factCatBtn)**;** categoryPanel.add(decCatBtn)**;** allCatBtn.addActionListener(e -> refreshLogs())**;** docCatBtn.addActionListener(e -> refreshLogs())**;** histCatBtn.addActionListener(e -> refreshLogs())**;** factCatBtn.addActionListener(e -> refreshLogs())**;** decCatBtn.addActionListener(e -> refreshLogs())**;** JPanel filtersContainer = new JPanel(new BorderLayout())**;** filtersContainer.add(filterPanel**,** BorderLayout.*NORTH*)**;** filtersContainer.add(categoryPanel**,** BorderLayout.*CENTER*)**;** logPanel.add(filtersContainer**,** BorderLayout.*NORTH*)**;** logPanel.add(new JScrollPane(logArea)**,** BorderLayout.*CENTER*)**;** JPanel exportPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*))**;** JButton exportBtn = new JButton("Export logs")**;** exportBtn.setPreferredSize(new Dimension(160**,**30))**;** exportBtn.addActionListener(e -> exportLogsToFile())**;** exportPanel.add(exportBtn)**;** logPanel.add(exportPanel**,** BorderLayout.*SOUTH*)**;** tabs.add("Logs"**,** logPanel)**;** add(tabs**,** BorderLayout.*CENTER*)**;** pack()**;** setLocationRelativeTo(null)**;** refresh()**;** }  
  
 private void editContent() {  
 String current = document.getContent()**;** String newText = JOptionPane.*showInputDialog*(this**,** "Text:"**,** current)**;** if (newText != null && !newText.equals(current)) {  
 context.getBean(ChangeTextCommand.class**,** document**,** newText).execute()**;** logger.addMemento(document.createMemento())**;** logger.clearRedo()**;** refresh()**;** }  
 }  
  
 private void jumpToHistory(int index) {  
 DocumentMemento current = document.createMemento()**;** DocumentMemento target = logger.getMementoAt(index)**;** if (target != null) {  
 context.getBean(RevertCommand.class**,** document**,** logger**,** index).execute()**;** logger.addMemento(current)**;** logger.clearRedo()**;** refresh()**;** }  
 }  
  
 private void saveHistoryToFile() {  
 JFileChooser fc = new JFileChooser()**;** if (fc.showSaveDialog(this) == JFileChooser.*APPROVE\_OPTION*) {  
 try {  
 logger.saveHistoryToFile(fc.getSelectedFile())**;** JOptionPane.*showMessageDialog*(this**,** "History saved")**;** } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(this**,** "Save error: " + ex.getMessage())**;** }  
 }  
 }  
  
 private void loadHistoryFromFile() {  
 JFileChooser fc = new JFileChooser()**;** if (fc.showOpenDialog(this) == JFileChooser.*APPROVE\_OPTION*) {  
 try {  
 logger.loadHistoryFromFile(fc.getSelectedFile())**;** if (!logger.getHistory().isEmpty()) {  
 DocumentMemento last = logger.getHistory().get(logger.getHistory().size() - 1)**;** document.restoreFromMemento(last)**;** }  
 refresh()**;** JOptionPane.*showMessageDialog*(this**,** "History loaded")**;** } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(this**,** "Load error: " + 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())**;** refreshLogs()**;** }  
  
 private void refreshLogs() {  
 List<String> logs**;** if (allBtn.isSelected()) {  
 if (allCatBtn.isSelected()) {  
 logs = logCollector.getRecords()**;** } else if (docCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*DOCUMENT*)**;** } else if (histCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*HISTORY*)**;** } else if (factCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*FACTORY*)**;** } else {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*DECORATOR*)**;** }  
 } else {  
 LogCollector.Level level = infoBtn.isSelected() ? LogCollector.Level.*INFO* :  
 debugBtn.isSelected() ? LogCollector.Level.*DEBUG* :  
 LogCollector.Level.*ERROR***;** if (allCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByLevel(level)**;** } else {  
 LogCollector.Category category = docCatBtn.isSelected() ? LogCollector.Category.*DOCUMENT* :  
 histCatBtn.isSelected() ? LogCollector.Category.*HISTORY* :  
 factCatBtn.isSelected() ? LogCollector.Category.*FACTORY* :  
 LogCollector.Category.*DECORATOR***;** logs = logCollector.getRecordsByCategoryAndLevel(category**,** level)**;** }  
 }  
 logArea.setText(String.*join*("\n"**,** logs))**;** }  
  
 private void exportLogsToFile() {  
 JFileChooser fc = new JFileChooser()**;** if (fc.showSaveDialog(this) == JFileChooser.*APPROVE\_OPTION*) {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(fc.getSelectedFile()))) {  
 List<String> logs**;** if (allBtn.isSelected()) {  
 if (allCatBtn.isSelected()) {  
 logs = logCollector.getRecords()**;** } else if (docCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*DOCUMENT*)**;** } else if (histCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*HISTORY*)**;** } else if (factCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*FACTORY*)**;** } else {  
 logs = logCollector.getRecordsByCategory(LogCollector.Category.*DECORATOR*)**;** }  
 } else {  
 LogCollector.Level level = infoBtn.isSelected() ? LogCollector.Level.*INFO* :  
 debugBtn.isSelected() ? LogCollector.Level.*DEBUG* :  
 LogCollector.Level.*ERROR***;** if (allCatBtn.isSelected()) {  
 logs = logCollector.getRecordsByLevel(level)**;** } else {  
 LogCollector.Category category = docCatBtn.isSelected() ? LogCollector.Category.*DOCUMENT* :  
 histCatBtn.isSelected() ? LogCollector.Category.*HISTORY* :  
 factCatBtn.isSelected() ? LogCollector.Category.*FACTORY* :  
 LogCollector.Category.*DECORATOR***;** logs = logCollector.getRecordsByCategoryAndLevel(category**,** level)**;** }  
 }  
 for (String record : logs) {  
 writer.write(record)**;** writer.newLine()**;** }  
 JOptionPane.*showMessageDialog*(this**,** "Logs exported")**;** } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(this**,** "Export error: " + ex.getMessage())**;** }  
 }  
 }  
  
 @Override  
 public void update(Document d) {  
 if (d == document) textArea.setText(d.getContent())**;** }  
}

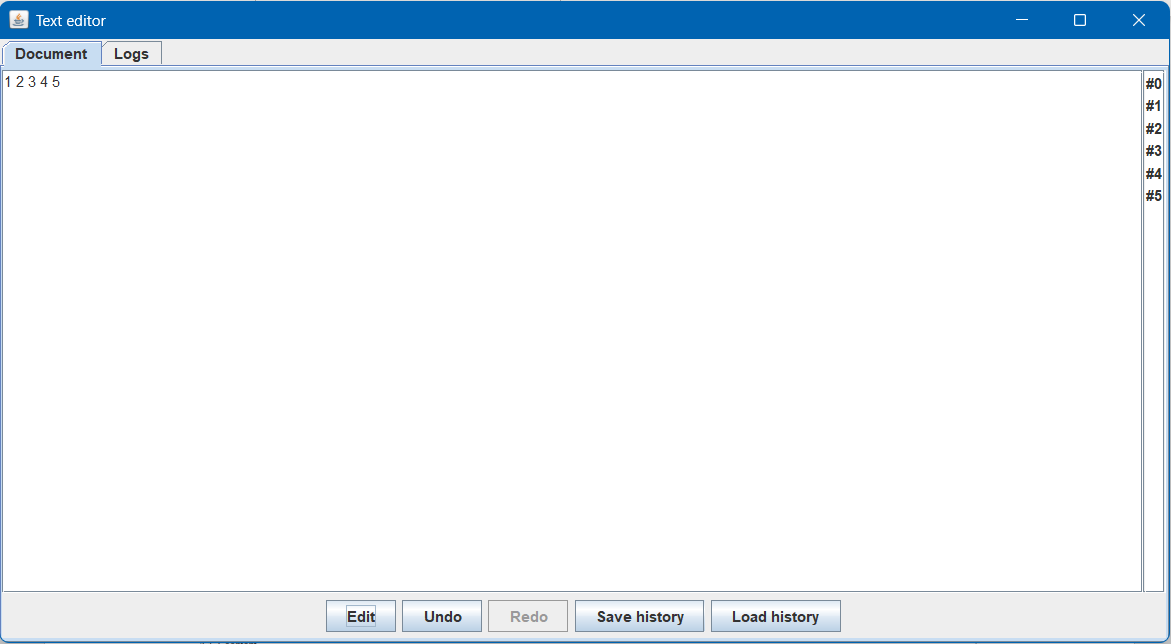
Main.java

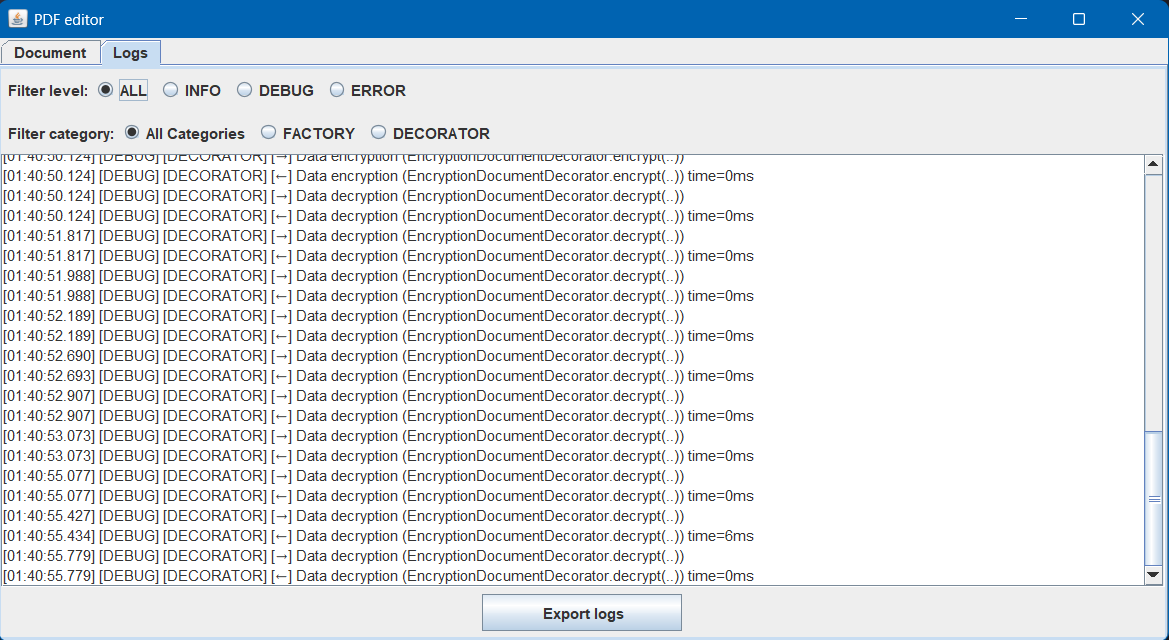
package ru.doc**;**import org.springframework.context.annotation.AnnotationConfigApplicationContext**;**import ru.doc.config.AppConfig**;**import ru.doc.gui.DocumentEditorGUI**;**import javax.swing.SwingUtilities**;**public class Main {  
 public static void main(String[] args) {  
 AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class)**;** Runtime.*getRuntime*().addShutdownHook(new Thread(ctx::close))**;** SwingUtilities.*invokeLater*(() -> ctx.getBean(DocumentEditorGUI.class))**;** }  
}

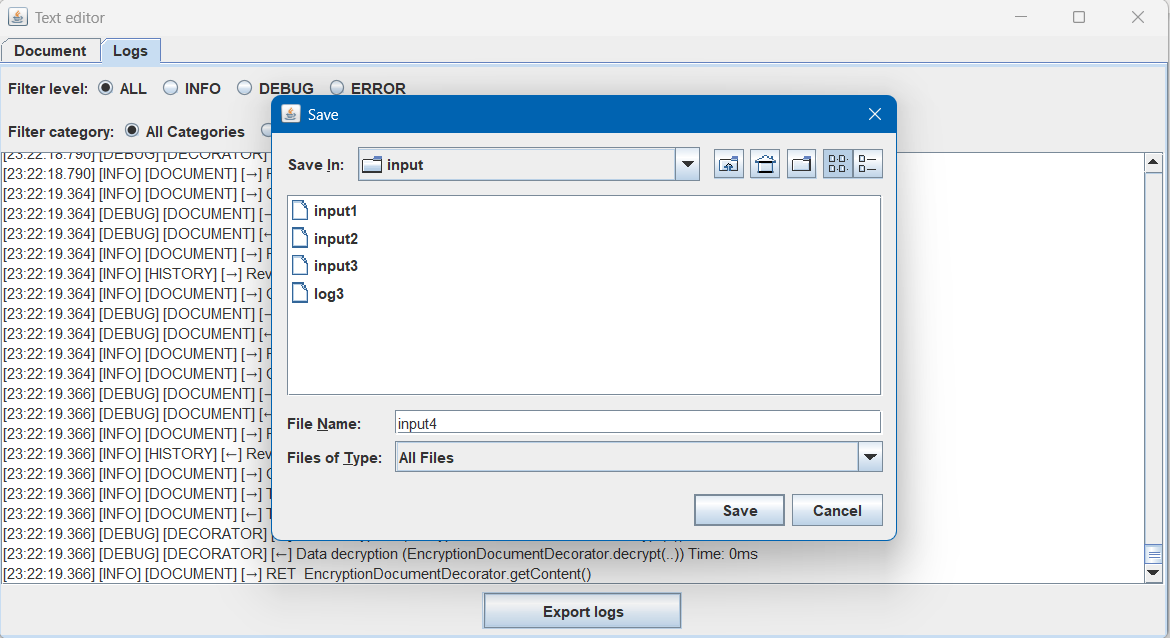
**Приложение 2**

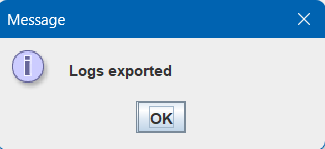
**Результаты тестирования**









**Сценарии работы**

1. **Редактирование документа**

Нажатие кнопки "Edit" открывает модальное окно для ввода текста, после внесения изменений и нажатия OK создаётся команда ChangeTextCommand.execute(), документ обновляется, автоматически фиксируется снимок в DocumentHistoryLogger, очищается история Redo

1. **Отмена / повтор / прыжок**

Кнопка Undo создаёт текущий снимок DocumentHistoryLogger предоставляет предыдущее состояние, документ восстанавливается, список истории обновляется

Кнопка Redo создаёт текущий снимок DocumentHistoryLogger предоставляет следующее состояние, документ восстанавливается, список истории обновляется

При щелчке по элементу #i в списке истории GUI получает i-й memento, восстанавливает документ из этого состояния, текущий снимок помещается в стек "вперёд", таким образом возможно перемещение 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.

1. **Фильтрация по уровню**

На вкладке Logs предлагаются переключатели: ALL, INFO, DEBUG, ERROR, при выборе логи немедленно фильтруются по выбранному уровню

1. **Фильтрация по категории**

Доступны категории: All Categories, DOCUMENT, HISTORY, FACTORY, DECORATOR, возможно комбинирование с фильтром по уровню для точной выборки

1. **Экспорт логов**

Кнопка Export logs открывает диалог выбора файла, экспортируются только записи, видимые после применения текущих фильтров, логи сохраняются в текстовом формате, каждая запись на отдельной строке, после успешного сохранения появляется сообщение Logs exported