**1.**

Сегодня мы напишем **HTML** редактор с графическим интерфейсом.  
В качестве библиотеки для создания графического интерфейса воспользуемся **Swing**.  
А в качестве архитектурного каркаса нашего приложения будем использовать **MVC** модель.  
1.1. Объяви класс **Controller** и класс **View**.  
Класс **View** должен быть унаследован от **JFrame** и реализовывать интерфейс **ActionListener**.  
1.2. Добавь в класс **Controller** поля, отвечающие за представление **View** view и модель **HTMLDocument** document. Здесь и далее, классы, которые мы не объявляли, но используем в коде скорее всего все находятся в библиотеке **swing**.  
Например, класс **HTMLDocument** реализован в пакете ***javax.swing.text.html.***  
1.3. Добавь в класс **Controller** поле, которое будет отвечать за файл, который сейчас открыт в нашем редакторе (*текущий файл*) **File** currentFile.  
1.4. Добавь конструктор класса **Controller**. Он должен принимать в качестве параметра представление и инициализировать соответствующее поле класса.  
1.5. Добавь в **Controller** пустой метод main.  
1.6. Добавь в класс **View** поле **Controller** controller.  
1.7. Добавь в класс **View** **сеттер** и **геттер** для поля controller.  
1.8. Добавь пустую реализацию метода, который объявлен в интерфейсе **ActionListener**.

**Требования:**

* •

Класс Controller должен быть создан в отдельном файле.

* •

Класс View должен быть создан в отдельном файле.

* •

Класс View должен быть унаследован от JFrame и реализовывать интерфейс ActionListener.

* •

В классе Controller должны быть приватные поля: View view, HTMLDocument document, File currentFile.

* •

В классе Controller должен быть создан метод public static void main (String[] args).

* •

Класс Controller должен содержать конструктор с одним параметром, инициализирующий поле view.

* •

В классе View должно быть приватное поле Controller controller, а также сеттер и геттер к нему.

* •

В классе View должен быть метод public void actionPerformed(ActionEvent e).

**2.**

2.1. Добавь в контроллер и представление по методу init(), пока без реализаций. Они будут отвечать за инициализацию контроллера и представления.  
2.2. Теперь напишем в классе **Controller** метод main().  
Он должен:  
2.2.1. Создавать объект представления.  
2.2.2. Создавать контроллер, используя представление.  
2.2.3. Устанавливать у представления контроллер.  
2.2.4. Инициализировать представление.  
2.2.5. Инициализировать контроллер. Контроллер должен инициализироваться после представления.  
2.3. Добавь в контроллер метод exit(), он должен вызывать статический метод exit у класса **System**.  
2.3.1. Метод exit в классе **Controller** не должен быть статическим.  
2.4. Добавь в представление метод exit(), он должен вызывать exit() у контроллера.

**Требования:**

* •

Класс Controller должен содержать метод public void init().

* •

Класс View должен содержать метод public void init().

* •

Класс Controller должен реализовывать метод public void exit().

* •

Класс View должен реализовывать метод public void exit().

* •

Реализуй, согласно условию, метод public static void main(String[] args) у класса Controller.

**3.**

Графический интерфейс будет представлять собой окно, в котором будет меню и панель с двумя вкладками.  
На первой вкладке будет располагаться текстовая панель, которая будет отрисовывать **html** страницу. На ней можно будет форматировать и редактировать текст страницы.  
На второй вкладке будет редактор, который будет отображать код **html** страницы, в нем будут видны все используемые **html** теги. В нем также можно будет менять текст страницы, добавлять и удалять различные теги.  
3.1. Добавь и проинициализируй поля в классе представления:  
3.1.1. **JTabbedPane tabbedPane** - это будет панель с двумя вкладками.  
3.1.2. **JTextPane htmlTextPane** - это будет компонент для визуального редактирования **html**.  
Он будет размещен на первой вкладке.  
3.1.3. **JEditorPane plainTextPane** - это будет компонент для редактирования **html** в виде текста, он будет отображать код **html** (*теги и их содержимое*).  
3.2. Добавь класс **FrameListener** в пакет listeners. Он должен:  
3.2.1. Быть унаследован от **WindowAdapter**.  
3.2.2. Иметь поле **View** view.  
3.2.3. В конструкторе принимать **View** и инициализировать внутреннее поле.  
3.2.4. Иметь переопределенный метод windowClosing(WindowEvent **windowEvent**), который должен вызывать exit() у представления.

**Требования:**

* •

Класс View должен содержать приватное проинициализированое поле JTabbedPane tabbedPane.

* •

Класс View должен содержать приватное проинициализированое поле JTextPane htmlTextPane.

* •

Класс View должен содержать приватное проинициализированое поле JEditorPane plainTextPane.

* •

Класс FrameListener должен быть создан в отдельном файле.

* •

Класс FrameListener должен содержать приватное поле View view.

* •

Класс FrameListener должен содержать конструктор с одним параметром, инициализирующий поле view.

* •

В классе FrameListenerМетод должен быть метод windowClosing(WindowEvent windowEvent).

**4.**

4.1. Объяви методы initMenuBar() и initEditor() в классе **View**. Они будут отвечать за инициализацию меню и панелей редактора.  
4.2. Объяви в представлении метод initGui(). Он будет инициализировать графический интерфейс.  
Вызови из него инициализацию меню initMenuBar(), инициализацию редактора initEditor() и метод pack(), реализацию которого мы унаследовали от класса **JFrame**.  
Разберись что делает метод pack().  
4.3. Реализуй метод init() представления. **Он должен:**  
4.3.1. Вызывать инициализацию графического интерфейса initGui().  
4.3.2. Добавлять слушателя событий нашего окна. В качестве подписчика создай и используй объект класса **FrameListener**.  
В качестве метода для добавления подписчика используй подходящий метод из класса **Window** от которого наследуется и наш класс через классы **JFrame** и **Frame**.  
4.3.3. Показывать наше окно. Используй метод setVisible с правильным параметром.  
На этом этапе приложение при запуске должно показывать окно, которое можно растягивать, разворачивать, закрыть и т.д.

**Требования:**

* •

Класс View должен содержать метод public void initMenuBar().

* •

Класс View должен содержать метод public void initEditor().

* •

Класс View должен реализовывать метод public void initGui().

* •

Реализуй метод init() класса View, согласно задания.

**5.**

5.1. Объяви класс **TabbedPaneChangeListener** реализующий интерфейс **ChangeListener** в пакете listeners. Этот класс будет слушать и обрабатывать изменения состояния панели вкладок.  
Реализуй в этом классе:  
5.1.1. Конструктор, принимающий представление в виде параметра и сохраняющий во внутреннее поле view класса.  
5.1.2. Переопредели метод из интерфейса **ChangeListener**, он должен вызывать метод selectedTabChanged() у представления. Последнего метода еще нет, сделай для него заглушку.  
5.2. Объяви класс **ExceptionHandler**. Это будет наш обработчик исключительных ситуаций, который ты в дальнейшем сможешь переопределить.  
Пока добавь в него статический метод log(Exception e), который будет выводить в консоль краткое описание проблемы (*используй метод*toString*у переданного исключения*).

**Требования:**

* •

Класс View должен содержать метод public void selectedTabChanged().

* •

Класс TabbedPaneChangeListener должен быть создан в отдельном файле.

* •

Класс TabbedPaneChangeListener должен содержать поле View view.

* •

Класс TabbedPaneChangeListener должен содержать конструктор с одним параметром, инициализирующий поле view.

* •

Класс TabbedPaneChangeListener должен содержать метод public void stateChanged(ChangeEvent e).

* •

Класс ExceptionHandler должен быть создан в отдельном файле.

* •

Класс ExceptionHandler должен содержать метод public static void log(Exception e).

**6.**

Реализуй метод инициализации панелей редактора initEditor(). **Он должен:**  
6.1. Устанавливать значение "***text/html***" в качестве типа контента для компонента **htmlTextPane**.  
Найди и используй подходящий метод.  
6.2. Создавать новый локальный компонент **JScrollPane** на базе **htmlTextPane**.  
6.3. Добавлять вкладку в панель tabbedPane с именем "***HTML***" и компонентом из предыдущего пункта.  
6.4. Создавать новый локальный компонент **JScrollPane** на базе **plainTextPane**.  
6.5. Добавлять еще одну вкладку в **tabbedPane** с именем "***Текст***" и компонентом из предыдущего пункта.  
6.6. Устанавливать предпочтительный размер панели **tabbedPane**.  
6.7. Создавать объект класса **TabbedPaneChangeListener** и устанавливать его в качестве слушателя изменений в **tabbedPane**.  
6.8. Добавлять по центру панели контента текущего фрейма нашу панель с вкладками.  
Получить панель контента текущего фрейма можно с помощью метода getContentPane(), его реализация унаследовалась от **JFrame**.  
Подумай, метод с какими параметрами необходимо вызвать, чтобы панель с вкладками отображалась по центру панели контента текущего фрейма.  
После запуска приложения можно будет увидеть текущие результаты: две независимые закладки (*HTML и Текст*), в каждой из которых можно набирать свой текст.

**Требования:**

* •

В методе initEditor() для компонента htmlTextPane должен устанавливаться тип контента "text/html" через сеттер setContentType.

* •

В методе initEditor() должен создаваться новый локальный компонент JScrollPane через конструктор принимающий htmlTextPane.

* •

В методе initEditor() для компонента tabbedPane должна добавляться вкладка с именем "HTML" и созданным компонентом JScrollPane на базе htmlTextPane.

* •

В методе initEditor() должен создаваться новый локальный компонент JScrollPane через конструктор принимающий plainTextPane.

* •

В методе initEditor() для компонента tabbedPane должна добавляться вкладка с именем "Текст" и созданным компонентом JScrollPane на базе plainTextPane.

* •

В методе initEditor() для компонента tabbedPane должен устанавливаться предпочтительный размер панели, через сеттер setPreferredSize.

* •

В методе initEditor() для компонента tabbedPane должен добавляться слушатель TabbedPaneChangeListener через метод addChangeListener.

* •

Метод initEditor() должен добавлять по центру панели контента текущего фрейма нашу панель с вкладками, через getContentPane().add().

**7.**

Добавь класс **MenuHelper**. Это будет вспомогательный класс для инициализации и настройки меню.

**У меню будет следующая структура:**  
*- Файл*  
*- Новый*  
*- Открыть*  
*- Сохранить*  
*- Сохранить как...*  
*- Выход*

*- Редактировать*  
*- Отменить*  
*- Вернуть*  
*- Вырезать*  
*- Копировать*  
*- Вставить*

*- Стиль*  
*- Полужирный*  
*- Подчеркнутый*  
*- Курсив*  
*- Подстрочный знак*  
*- Надстрочный знак*  
*- Зачеркнутый*

*- Выравнивание*  
*- По левому краю*  
*- По центру*  
*- По правому краю*

*- Цвет*  
*- Красный*  
*- Оранжевый*  
*- Желтый*  
*- Зеленый*  
*- Синий*  
*- Голубой*  
*- Пурпурный*  
*- Черный*

*- Шрифт*  
*- Шрифт*  
*- SansSerif, Serif, Monospaced, Dialog, DialogInput,*  
*- Размер шрифта*  
*- 6, 8, 10, 12, 14, 16, 20, 24, 32, 36, 48, 72*

*- Помощь*  
*- О программе*

7.1. Реализуй в **MenuHelper** статический метод JMenuItem addMenuItem(JMenu **parent**, String **text**, ActionListener **actionListener**), где parent - меню в которое мы добавляем пункт, text - текст добавляемого пункта, actionListener - слушатель действий добавляемого пункта меню. Он должен:  
7.1.1. Создавать новый пункт меню **JMenuItem**, используя **text**.  
7.1.2. Устанавливать этому пункту слушателя действий с помощью метода addActionListener().  
7.1.3. Добавлять в parent созданный пункт меню.  
7.1.4. Возвращать созданный пункт меню.  
7.2. Реализуй статический метод JMenuItem addMenuItem(JMenu **parent**, Action **action**). Этот метод аналогичен предыдущему, но в качестве параметра принимает действие action, которое необходимо выполнить при выборе пункта меню.

**Подсказка:** *при создании***JMenuItem***передай в конструктор***action***. Разберись откуда возьмется имя пункта меню, если его не устанавливать дополнительно.*

7.3. Реализуй статический метод JMenuItem addMenuItem(JMenu **parent**, String **text**, Action **action**), который добавляет в parent новый пункт меню с текстом text и действием action при выборе этого метода. При реализации используй вызов метода из предыдущего пункта.  
7.4. Добавь в **MenuHelper** заглушки для следующих статический методов:  
7.4.1. void initHelpMenu(View **view**, JMenuBar **menuBar**) - инициализация меню помощи.  
7.4.2. void initFontMenu(View **view**, JMenuBar **menuBar**) - инициализация меню выбора шрифта.  
7.4.3. void initColorMenu(View **view**, JMenuBar **menuBar**) - инициализация меню выбора цвета.  
7.4.4. void initAlignMenu(View **view**, JMenuBar **menuBar**) - инициализация меню выравнивания.  
7.4.5. void initStyleMenu(View **view**, JMenuBar **menuBar**) - инициализация меню выбора стиля текста.  
7.4.6. void initEditMenu(View **view**, JMenuBar **menuBar**) - инициализация меню редактирования текста.  
7.4.7. void initFileMenu(View **view**, JMenuBar **menuBar**) - инициализация меню Файл.

**Требования:**

* •

Класс MenuHelper должен быть создан в отдельном файле.

* •

Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, String text, ActionListener actionListener).

* •

Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, Action action).

* •

Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, String text, Action action).

* •

Класс MenuHelper должен содержать публичный статический метод void initHelpMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initFontMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initColorMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initAlignMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initStyleMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initEditMenu(View view, JMenuBar menuBar).

* •

Класс MenuHelper должен содержать публичный статический метод void initFileMenu(View view, JMenuBar menuBar).

**8.**

Пришло время добавить все необходимые пункты меню и написать классы действий, выполняемые при нажатии.  
Это довольно трудоемкая и рутинная работа, а ты отличный ученик и не хотелось бы тебя огорчать. Поэтому, в качестве бонуса ты получаешь готовый класс **MenuHelper**! **А ты:**  
8.1. Напиши в пакете **actions** заглушки для следующих классов:  
8.1.1. Класс отмены действия UndoAction. Он должен наследоваться от **AbstractAction** и содержать конструктор UndoAction(View **view**).  
8.1.2. Класс возврата действия RedoAction. Требования аналогичны требованиям к UndoAction.  
8.1.3. Класс **StrikeThroughAction**, который отвечает за стиль текста "***Зачеркнутый***". Унаследуй его от StyledEditorKit.StyledTextAction.  
8.1.4. Класс **SubscriptAction**, который отвечает за стиль текста "***Подстрочный знак***". Его также унаследуй его от StyledEditorKit.StyledTextAction.  
8.1.5. Класс **SuperscriptAction**. Он будет отвечать за стиль "***Надстрочный знак***". Добавь ему правильный родительский класс.  
8.2. Напиши в пакете **listeners** заглушки для классов:  
8.2.1. **UndoMenuListener**, он должен реализовывать интерфейс **MenuListener** и иметь конструктор UndoMenuListener(View view, JMenuItem undoMenuItem, JMenuItem redoMenuItem).  
Чем он будет заниматься узнаешь чуть позже.  
8.2.2. TextEditMenuListener. Этот класс также должен реализовывать интерфейс **MenuListener**.  
Добавь ему конструктор TextEditMenuListener(View view). В следующих заданиях мы рассмотрим его детальнее.

**Требования:**

* •

Класс UndoAction должен быть создан в отдельном файле.

* •

Класс RedoAction должен быть создан в отдельном файле.

* •

Класс StrikeThroughAction должен быть создан в отдельном файле.

* •

Класс SubscriptAction должен быть создан в отдельном файле.

* •

Класс SuperscriptAction должен быть создан в отдельном файле.

* •

Класс UndoMenuListener должен быть создан в отдельном файле.

* •

Класс TextEditMenuListener должен быть создан в отдельном файле.

**9.**

9.1. Реализуй метод initMenuBar(). **Он должен:**  
9.1.1. Создавать новый объект типа **JMenuBar**. Это и будет наша панель меню.  
9.1.2. С помощью **MenuHelper** инициализировать меню в следующем порядке: **Файл**, **Редактировать**, **Стиль**, **Выравнивание**, **Цвет**, **Шрифт** и **Помощь**.  
9.1.3. Добавлять в верхнюю часть панели контента текущего фрейма нашу панель меню, аналогично тому, как это мы делали с панелью вкладок.  
9.2. Добавь конструктор класса **View**. Он должен устанавливать внешний вид и поведение (*look and feel*) нашего приложения такими же, как это определено в системе.  
Конструктор не должен кидать исключений, только логировать их с помощью **ExceptionHandler**.

**Подсказа:** *для реализации задания используй класс* **UIManager**.

Запусти приложение, теперь ты должен видеть панель с меню вверху окна. Некоторые из пунктов меню (*например:* **Вырезать**, **Копировать**, **Вставить**, **Стиль** (*частично*), **Выравнивание**, **Цвет**, **Шрифт**) должны уже работать. Убедись, что все работает и только затем продолжи разработку.

**Требования:**

* •

В методе initMenuBar() должно создаваться новое меню (объект типа JMenuBar).

* •

В методе initMenuBar() c помощью MenuHelper должно быть проинициализировано меню в следующем порядке: Файл, Редактировать, Стиль, Выравнивание, Цвет, Шрифт и Помощь.

* •

В методе initMenuBar() должно добавляться новосозданное меню в верхнюю часть панели контента текущего фрейма, используя метод getContentPane().

* •

В конструкторе класса View, через класс UIManager, должен устанавливаться внешний вид и поведение (look and feel).

**10.**

Наш редактор будет поддерживать механизм **отмены**/**возврата** (*undo/redo*) действий в редакторе.  
Реализуй класс **UndoMenuListener**. Этот слушатель будет следить за меню, а если конкретнее, то за моментом, когда меню редактирования будет выбрано пользователем.  
В этот момент он будет запрашивать у представления можем ли мы сейчас отменить или вернуть какое-то действие, и в зависимости от этого делать доступными или не доступными пункты меню "***Отменить***" и "***Вернуть***".  
10.1. Добавь в класс **UndoMenuListener** следующие поля:  
10.1.1. Представление **View** view.  
10.1.2. Пункт меню "***Отменить***" **JMenuItem undoMenuItem**.  
10.1.3. Пункт меню "***Вернуть***" **JMenuItem redoMenuItem**.  
10.2. Реализуй конструктор UndoMenuListener(View **view**, JMenuItem **undoMenuItem**, JMenuItem **redoMenuItem**), он должен инициализировать поля класса.  
10.3. Реализуй метод menuSelected(MenuEvent **menuEvent**). Он будет вызываться перед показом меню. **Он должен:**  
10.3.1. Спрашивать у представления можем ли мы отменить действие с помощью метода boolean canUndo().  
Пока у представления нет такого метода, поэтому добавь заглушку, которая всегда возвращает **false**.  
10.3.2. Делать доступным или не доступным пункт меню **undoMenuItem** в зависимости от того, что нам вернуло представление.

**Подсказка:** *используй метод* setEnabled().

10.3.3. Аналогично поступи и для пункта меню **redoMenuItem**, добавив метод-заглушку canRedo() в представление.  
Запусти программу и убедись, что пункты меню Отменить и Вернуть недоступны.

**Требования:**

* •

Класс UndoMenuListener должен содержать поле представления View view.

* •

Класс UndoMenuListener должен содержать поле JMenuItem undoMenuItem.

* •

Класс UndoMenuListener должен содержать поле JMenuItem redoMenuItem.

* •

Конструктор UndoMenuListener(View view, JMenuItem undoMenuItem, JMenuItem redoMenuItem) должен корректно инициализировать поля класса.

* •

Класс View должен содержать public boolean метод canUndo(), возвращающий false.

* •

Класс View должен содержать public boolean метод canRedo(), возвращающий false.

* •

Метод menuSelected(MenuEvent menuEvent) должен делать доступными или не доступными пункты меню undoMenuItem и redoMenuItem.

**11.**

11.1. Добавь в представление поле **UndoManager** undoManager. Разберись для чего используется этот класс. Проинициализируй поле класса новым объектом.  
11.2. Добавь класс **UndoListener** реализующий интерфейс **UndoableEditListener** в пакет **listeners**. Этот класс будет следить за правками, которые можно отменить или вернуть.  
11.3. Добавь в класс **UndoListener**:  
11.3.1. Поле **UndoManager** undoManager.  
11.3.2. Конструктор, который принимает **UndoManager** и инициализирует поле класса.  
11.3.3. Метод undoableEditHappened(UndoableEditEvent e). Он должен из переданного события получать правку и добавлять ее в **undoManager**.  
11.4. Добавь в представление поле **UndoListener** undoListener, проинициализируй его используя **undoManager**.  
11.5. Добавь в представление методы:  
11.5.1. void undo() - отменяет последнее действие. Реализуй его используя undoManager.  
Метод не должен кидать исключений, логируй их.  
11.5.2. void redo() - возвращает ранее отмененное действие. Реализуй его по аналогии с предыдущим пунктом.  
11.5.3. Реализуй методы boolean canUndo() и boolean canRedo() используя **undoManager**.  
11.5.4. Реализуй геттер для **undoListener**.  
11.5.5. Добавь и реализуй метод void resetUndo(), который должен сбрасывать все правки в менеджере **undoManager**.

**Требования:**

* •

Класс View должен содержать приватное поле UndoManager undoManager, которое должно быть сразу проинициализовано.

* •

Класс View должен содержать приватное поле UndoListener undoListener, которое должно быть сразу проинициализовано через поле undoManager.

* •

Класс View должен содержать геттер к полю UndoListener undoListener.

* •

Класс View должен содержать public void метод undo().

* •

Класс View должен содержать public void метод redo().

* •

Реализация метода canUndo() класса View должна использовать undoManager.

* •

Реализация метода canRedo() класса View должна использовать undoManager.

* •

Класс View должен содержать public void метод resetUndo(), который должен сбрасывать все правки в менеджере undoManager.

* •

Класс UndoListener должен быть создан в отдельном файле.

* •

Класс UndoListener должен содержать приватное поле UndoManager undoManager.

* •

Конструктор UndoListener(UndoManager undoManager) должен корректно инициализировать поле undoManager.

* •

Метод undoableEditHappened(UndoableEditEvent e) в классе UndoListener должен из переданного события получать правку и добавлять ее в undoManager.

**12.**

12.1. Напиши реализацию класса **RedoAction**:  
12.1.1. Добавь в класс поле **View**. Добавь его инициализацию в конструкторе.  
12.1.2. Реализуй метод actionPerformed(ActionEvent **actionEvent**), он должен вызывать метод redo() у представления.  
12.2. Напиши реализацию класса **UndoAction** по аналогии с **RedoAction**.  
12.3. Изучи реализацию класса **StrikeThroughAction**, которую ты получил вместе с заданием и реализуй аналогичным образом классы:  
12.3.1. **SubscriptAction**  
12.3.2. **SuperscriptAction**  
Запусти программу и убедись, что пункты меню Подстрочный знак, Надстрочный знак и Зачеркнутый работают.  
Пункты, отвечающие за отмену и возврат действия пока не подключены к контроллеру и мы не сможем их проверить.

**Требования:**

* •

Класс RedoAction должен содержать поле View view.

* •

Конструктор класса RedoAction(View view) должен инициализировать поле view.

* •

Метод actionPerformed(ActionEvent actionEvent) класса RedoAction должен вызывать метод redo() у представления.

* •

Класс UndoAction должен содержать поле View view.

* •

Конструктор класса UndoAction(View view) должен инициализировать поле view.

* •

Метод actionPerformed(ActionEvent actionEvent) класса UndoAction должен вызывать метод undo() у представления.

* •

Конструктор без параметров класса SubscriptAction должен вызывать конструктор суперкласса с параметром StyleConstants.Subscript.

* •

Метод actionPerformed(ActionEvent actionEvent) класса SubscriptAction должен использовать метод setSubscript у StyleConstants с параметрами: SimpleAttributeSet и !StyleConstants.isSubscript(mutableAttributeSet).

* •

Конструктор без параметров класса SuperscriptAction должен вызывать конструктор суперкласса с параметром StyleConstants.Superscript.

* •

Метод actionPerformed(ActionEvent actionEvent) класса SuperscriptAction должен использовать метод setSuperscript у StyleConstants с параметрами: SimpleAttributeSet и !StyleConstants.isSuperscript(mutableAttributeSet).

**13.**

Реализуем класс **TextEditMenuListener** в пакет **listeners**.  
Этот класс будет работать аналогично классу **UndoMenuListener** только для других пунктов меню.  
Пункты меню, отвечающие за стиль, шрифт, цвет и т.д. должны быть доступны только тогда, когда в нашем редакторе выбрана первая вкладка.  
13.1. Добавь в представление метод boolean isHtmlTabSelected().  
Он должен возвращать **true**, если выбрана вкладка, отображающая **html** в панели вкладок (*подсказка: ее индекс 0*).  
13.2. Добавь в класс **TextEditMenuListener** поле **View**, проинициализируй его в конструкторе класса.  
13.3. В классе **TextEditMenuListener** переопредели метод menuSelected(MenuEvent **menuEvent**). **Он должен:**  
13.3.1. Из переданного параметра получать объект, над которым было совершено действие. В нашем случае это будет объект с типом **JMenu**.  
13.3.2. У полученного меню получать список компонентов (*пунктов меню*).  
13.3.3. Для каждого пункта меню вызывать метод setEnabled, передав в качестве параметра результат вызова метода isHtmlTabSelected() из представления.  
Запусти приложение и убедись, что пункты меню стиль, выравнивание, цвет и шрифт доступны только, когда активна закладка **HTML** и не доступны для закладки Текст.

**Требования:**

* •

Класс View должен содержать публичный метод boolean isHtmlTabSelected(), определяющий текущую вкладку.

* •

Класс TextEditMenuListener должен содержать поле View view.

* •

Конструктор класса TextEditMenuListener(View view) должен инициализировать поле view.

* •

Метод menuSelected(MenuEvent menuEvent) класса TextEditMenuListener должен устанавливать доступность пунктов меню стиль, выравнивание, цвет и шрифт в зависимости от выбранной вкладки.

**14.**

14.1. Добавь в класс представления метод selectHtmlTab(). **Он должен:**  
14.1.1. Выбирать **html** вкладку (*переключаться на нее*).  
14.1.2. Сбрасывать все правки с помощью метода, который ты реализовал ранее.  
14.2. Добавь в класс контроллера геттер для модели, в нашем случае это поле **document**.  
14.3. Добавь в представление метод update(), который должен получать документ у контроллера и устанавливать его в панель редактирования **htmlTextPane**.  
14.4. Добавь в представление метод showAbout(), который должен показывать диалоговое окно с информацией о программе. Информацию придумай сам, а вот тип сообщения должен быть **JOptionPane.INFORMATION\_MESSAGE**.

**Требования:**

* •

Класс View должен содержать публичный метод selectHtmlTab(), который должен выбирать вкладку и сбрасывать все правки.

* •

Класс Controller должен содержать геттер для поля document.

* •

Класс View должен содержать публичный метод update(), который должен устанавливать документ в панель редактирования.

* •

Класс View должен содержать публичный метод showAbout(), который должен показывать диалоговое окно с информацией о программе.

**15.**

Добавь в контроллер метод resetDocument(), который будет сбрасывать текущий документ. **Он должен:**  
15.1. Удалять у текущего документа **document** слушателя правок которые можно отменить/вернуть (*найди подходящий для этого метод, унаследованный от* **AbstractDocument**).  
Слушателя нужно запросить у представления (*метод* getUndoListener()).  
Не забудь проверить, что текущий документ существует (*не* **null**).  
15.2. Создавать новый документ по умолчанию и присваивать его полю document.

**Подсказка:** *воспользуйся подходящим методом класса* **HTMLEditorKit**.

15.3. Добавлять новому документу слушателя правок.  
15.4. Вызывать у представления метод update().

**Требования:**

* •

Класс Controller должен содержать публичный метод resetDocument(), который будет сбрасывать текущий документ.

* •

Метод resetDocument() должен удалять у текущего документа document слушателя правок через метод removeUndoableEditListener().

* •

Метод resetDocument() должен создавать новый документ по умолчанию через метод createDefaultDocument().

* •

Метод resetDocument() должен добавлять новому документу слушателя правок через метод addUndoableEditListener().

* •

Метод resetDocument() должен вызывать у представления метод update().

**16.**

Добавь метод setPlainText(String text) в контроллер.  
Он будет записывать переданный текст с **html** тегами в документ document. **При его реализации:**  
16.1. Сбрось документ.  
16.2. Создай новый реадер StringReader на базе переданного текста.  
16.3. Вызови метод read() из класса **HTMLEditorKit**, который вычитает данные из реадера в документ **document**.  
16.4. Проследи, чтобы метод не кидал исключения. Их необходимо просто логировать.

**Требования:**

* •

Класс Controller должен содержать публичный метод setPlainText(String text), который будет записывать переданный текст с html тегами в документ document.

* •

Метод setPlainText(String text) должен сбрасывать документ через метод resetDocument().

* •

Метод setPlainText(String text) должен создавать новый StringReader на базе переданного параметра.

* •

Метод setPlainText(String text) должен вызывать метод read() у объекта HTMLEditorKit.

* •

Если в методе setPlainText(String text) возникнет исключительная ситуация, то исключение должно логироваться через метод log у класса ExceptionHandler.

**17.**

Добавь метод String getPlainText() в контроллер. Он должен получать текст из документа со всеми html тегами.  
17.1. Создай объект **StringWriter**.  
17.2. Перепиши все содержимое из документа document в созданный объект с помощью метода write класса **HTMLEditorKit**.  
17.3. Как обычно, метод не должен кидать исключений.

**Требования:**

* •

Класс Controller должен содержать публичный метод String getPlainText().

* •

В методе getPlainText() должен создаваться объект класса StringWriter.

* •

Метод getPlainText() должен вызывать метод write() у объекта HTMLEditorKit.

* •

Если в методе getPlainText() возникнет исключительная ситуация, то исключение должно логироваться через метод log у класса ExceptionHandler.

* •

Метод getPlainText() должен возвращать текст из документа со всеми html тегами.

**18.**

Реализуй метод selectedTabChanged() представления. Этот метод вызывается, когда произошла смена выбранной вкладки. **Итак:**  
18.1. Метод должен проверить, какая вкладка сейчас оказалась выбранной.  
18.2. Если выбрана вкладка с индексом **0** (**html** *вкладка*), значит нам нужно получить текст из **plainTextPane** и установить его в контроллер с помощью метода **setPlainText**.  
18.3. Если выбрана вкладка с индексом **1** (*вкладка с***html***текстом*), то необходимо получить текст у контроллера с помощью метода getPlainText() и установить его в панель **plainTextPane**.  
18.4. Сбросить правки (*вызвать метод* resetUndo *представления*).

**Требования:**

* •

Метод selectedTabChanged() должен проверить, какая вкладка сейчас оказалась выбранной.

* •

Если индекс вкладки равен 0 - метод selectedTabChanged() должен получить текст из plainTextPane и установить его в контроллер с помощью метода setPlainText().

* •

Если индекс вкладки равен 1 - метод selectedTabChanged() должен получить текст у контроллера с помощью метода getPlainText() и установить его в панель plainTextPane.

* •

Метод selectedTabChanged() должен сбросить правки.

**19.**

Реализуем метод actionPerformed(ActionEvent actionEvent) у представления, этот метод наследуется от интерфейса **ActionListener** и будет вызваться при выборе пунктов меню, у которых наше представление указано в виде слушателя событий.  
19.1. Получи из события команду с помощью метода getActionCommand(). Это будет обычная строка.  
По этой строке ты можешь понять какой пункт меню создал данное событие.  
19.2. Если это команда "***Новый***", вызови у контроллера метод createNewDocument().  
В этом пункте и далее, если необходимого метода в контроллере еще нет - создай заглушки.  
19.3. Если это команда "***Открыть***", вызови метод openDocument().  
19.4. Если "***Сохранить***", то вызови saveDocument().  
19.5. Если "***Сохранить как...***" - saveDocumentAs().  
19.6. Если "***Выход***" - exit().  
19.7. Если "***О программе***", то вызови метод showAbout() у представления.  
Проверь, что заработали пункты меню Выход и О программе.

**Требования:**

* •

Метод actionPerformed(ActionEvent actionEvent) должен получать из события команду с помощью метода getActionCommand().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "Новый", метод должен вызывать у контроллера createNewDocument().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "Открыть", метод должен вызывать у контроллера openDocument().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "Сохранить", метод должен вызывать у контроллера saveDocument().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "Сохранить как...", метод должен вызывать у контроллера saveDocumentAs().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "Выход", метод должен вызывать у контроллера exit().

* •

Если в метод actionPerformed(ActionEvent actionEvent) передано событие с командой "О программе", метод должен вызывать у представления showAbout().

**20.**

20.1. Реализуй метод создания нового документа createNewDocument() в контроллере. **Он должен:**  
20.1.1. Выбирать **html** вкладку у представления.  
20.1.2. Сбрасывать текущий документ.  
20.1.3. Устанавливать новый заголовок окна, например: "***HTML редактор***". Воспользуйся методом setTitle(), который унаследован в нашем представлении.  
20.1.4. Сбрасывать правки в **Undo** менеджере. Используй метод resetUndo представления.  
20.1.5. Обнулить переменную currentFile.  
20.2. Реализуй метод инициализации init() контроллера.  
Он должен просто вызывать метод создания нового документа.  
Проверь работу пункта меню Новый.

**Требования:**

* •

Метод createNewDocument() в контроллере должен выбирать html вкладку у представления.

* •

Метод createNewDocument() в контроллере должен сбрасывать текущий документ.

* •

Метод createNewDocument() в контроллере должен устанавливать новый заголовок окна.

* •

Метод createNewDocument() в контроллере должен сбрасывать правки в Undo менеджере.

* •

Метод createNewDocument() в контроллере должен обнулить currentFile.

* •

Метод init() в контроллере должен вызывать метод создания нового документа.

**21.**

Для открытия или сохранения файла мы будем использовать **JFileChooser** из библиотеки **swing**.  
Объекты этого типа поддерживают фильтры, унаследованные от **FileFilter**.  
Сейчас мы напишем свой фильтр:  
21.1. Создай публичный класс **HTMLFileFilter** унаследованный от **FileFilter**.  
21.2. Мы хотим, чтобы окно выбора файла отображало только **html/htm** файлы или папки.  
Переопредели метод accept(File file), чтобы он возвращал **true**, если переданный файл директория или содержит в конце имени "***.html***" или "***.htm***" без учета регистра.  
21.3. Чтобы в окне выбора файла в описании доступных типов файлов отображался текст "***HTML и HTM файлы***" переопредели соответствующим образом метод getDescription().

**Требования:**

* •

Класс HTMLFileFilter должен быть создан в отдельном файле.

* •

Класс HTMLFileFilter должен наследоваться от FileFilter.

* •

Метод accept(File f) класса HTMLFileFilter должен возвращать true, если переданный файл - директория или содержит в конце имени ".html" или ".htm" без учета регистра.

* •

Метод accept(File f) класса HTMLFileFilter должен возвращать false, если переданный файл - НЕ директория или НЕ содержит в конце имени ".html" или ".htm" без учета регистра.

* •

Метод getDescription() класса HTMLFileFilter должен возвращать строку "HTML и HTM файлы".

**22.**

Реализуем в контроллере метод для сохранения файла под новым именем saveDocumentAs().  
Реализация должна:  
22.1. Переключать представление на **html** вкладку.  
22.2. Создавать новый объект для выбора файла **JFileChooser**.  
22.3. Устанавливать ему в качестве фильтра объект **HTMLFileFilter**.  
22.4. Показывать диалоговое окно "***Save File***" для выбора файла.  
22.5. Если пользователь подтвердит выбор файла:  
22.5.1. Сохранять выбранный файл в поле **currentFile**.  
22.5.2. Устанавливать имя файла в качестве заголовка окна представления.  
22.5.3. Создавать **FileWriter** на базе **currentFile**.  
22.5.4. Переписывать данные из документа **document** в объекта **FileWriter**-а аналогично тому, как мы это делали в методе getPlainText().  
22.6. Метод не должен кидать исключения.  
Проверь работу пункта меню Сохранить как...

**Требования:**

* •

Метод saveDocumentAs() в контроллере должен переключать представление на html вкладку.

* •

Метод saveDocumentAs() в контроллере должен создавать новый объект для выбора файла JFileChooser.

* •

Метод saveDocumentAs() в контроллере должен устанавливать объекту класса JFileChooser в качестве фильтра объект HTMLFileFilter.

* •

Метод saveDocumentAs() в контроллере должен, используя метод showSaveDialog() у JFileChooser показывать диалоговое окно "Save File" для выбора файла.

* •

Метод saveDocumentAs() в контроллере должен сохранять выбранный файл в поле currentFile, если пользователь подтвердит выбор файла.

* •

Метод saveDocumentAs() в контроллере должен устанавливать имя файла в качестве заголовка окна представления, если пользователь подтвердит выбор файла.

* •

Метод saveDocumentAs() в контроллере должен создавать FileWriter на базе currentFile, если пользователь подтвердит выбор файла.

* •

Метод saveDocumentAs() в контроллере должен используя HTMLEditorKit переписывать данные из документа document в объект FileWriter-а, если пользователь подтвердит выбор файла.

* •

Метод saveDocumentAs() в контроллере не должен кидать исключения, а логировать через ExceptionHandler.

**23.**

23.1. Напишем метод для сохранения открытого файла saveDocument().  
Метод должен работать также, как saveDocumentAs(), но не запрашивать файл у пользователя, а использовать **currentFile**.  
Если **currentFile** равен **null**, то вызывать метод saveDocumentAs().  
23.2. Пришло время реализовать метод openDocument().  
Метод должен работать аналогично методу saveDocumentAs(), в той части, которая отвечает за выбор файла.

**Подсказка:** *Обрати внимание на имя метода для показа диалогового окна.*

**Когда файл выбран, необходимо:**  
23.2.1. Установить новое значение **currentFile**.  
23.2.2. Сбросить документ.  
23.2.3. Установить имя файла в заголовок у представления.  
23.2.4. Создать **FileReader**, используя **currentFile**.  
23.2.5. Вычитать данные из **FileReader**-а в документ document с помощью объекта класса **HTMLEditorKit**.  
23.2.6. Сбросить правки (*вызвать метод*resetUndo*представления*).  
23.2.7. Если возникнут исключения - залогируй их и не пробрасывай наружу.  
Проверь работу пунктов меню Сохранить и Открыть.

**Требования:**

* •

Метод saveDocument() в контроллере должен переключать представление на html вкладку.

* •

Метод saveDocument() в контроллере должен создавать FileWriter на базе currentFile, если currentFile != null.

* •

Метод saveDocument() в контроллере должен используя HTMLEditorKit переписывать данные из документа document в объект FileWriter-а, если currentFile != null.

* •

Метод saveDocument() в контроллере должен вызывать метод saveDocumentAs(), если currentFile == null.

* •

Метод saveDocument() в контроллере не должен кидать исключения, а логировать через ExceptionHandler.

* •

Метод openDocument() в контроллере должен переключать представление на html вкладку.

* •

Метод openDocument() в контроллере должен создавать новый объект для выбора файла JFileChooser.

* •

Метод openDocument() в контроллере должен устанавливать объекту класса JFileChooser в качестве фильтра объект HTMLFileFilter.

* •

Метод openDocument() в контроллере должен, используя метод showOpenDialog() у JFileChooser показывать диалоговое окно "Open File" для выбора файла.

* •

Метод openDocument() в контроллере должен установить новое значение currentFile, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере должен сбросить документ, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере должен устанавливать имя файла в качестве заголовка окна представления, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере должен создавать FileReader на базе currentFile, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере должен используя HTMLEditorKit вычитать данные из FileReader-а в документ document, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере должен сбросить правки, если пользователь подтвердит выбор файла.

* •

Метод openDocument() в контроллере не должен кидать исключения, а логировать через ExceptionHandler.

**24.**

Твой **html** редактор готов!  
Ты научился:  
- Создавать приложения с графическим интерфейсом.  
- Работать с диалоговыми окнами.  
- Пользоваться классами из библиотеки **Swing**.  
- Реализовывать взаимодействие компонентов программы с помощью событий, слушателей, действий.  
- Усилил свои знания в области **MVC**.

Что можно улучшить в разработанном редакторе:  
- Добавить панель инструментов, повторяющую функционал меню.  
- Добавить подсветку **html** тегов на второй вкладке.  
- Добавить возможность загрузки документа из Интернет.  
- Расширить возможности редактора (*вставка картинки, ссылки и т.д.*)

**Поздравляю, так держать!**

**Требования:**

* •

html редактор готов!