**HTML Editor (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:  
1. Класс Controller должен быть создан в отдельном файле.  
2. Класс View должен быть создан в отдельном файле.  
3. Класс View должен быть унаследован от JFrame и реализовывать интерфейс ActionListener.  
4. В классе Controller должны быть приватные поля: View view, HTMLDocument document, File currentFile.  
5. В классе Controller должен быть создан метод public static void main (String[] args).  
6. Класс Controller должен содержать конструктор с одним параметром, инициализирующий поле view.  
7. В классе View должно быть приватное поле Controller controller, а также сеттер и геттер к нему.  
8. В классе View должен быть метод public void actionPerformed(ActionEvent e).

### HTML Editor (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:  
1. Класс Controller должен содержать метод public void init().  
2. Класс View должен содержать метод public void init().  
3. Класс Controller должен реализовывать метод public void exit().  
4. Класс View должен реализовывать метод public void exit().  
5. Реализуй, согласно условию, метод public static void main(String[] args) у класса Controller.

### HTML Editor (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:  
1. Класс View должен содержать приватное проинициализированое поле JTabbedPane tabbedPane.  
2. Класс View должен содержать приватное проинициализированое поле JTextPane htmlTextPane.  
3. Класс View должен содержать приватное проинициализированое поле JEditorPane plainTextPane.  
4. Класс FrameListener должен быть создан в отдельном файле.  
5. Класс FrameListener должен содержать приватное поле View view.  
6. Класс FrameListener должен должен содержать конструктор с одним параметром, инициализирующий поле view.  
7. В классе FrameListenerМетод должен быть метод windowClosing(WindowEvent windowEvent).

### HTML Editor (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.

### HTML Editor (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 методtoStringу переданного исключения).

Требования:  
1. Класс View должен содержать метод public void selectedTabChanged().  
2. Класс TabbedPaneChangeListener должен быть создан в отдельном файле.  
3. Класс TabbedPaneChangeListener должен содержать поле View view.  
4. Класс TabbedPaneChangeListener должен содержать конструктор с одним параметром, инициализирующий поле view.  
5. Класс TabbedPaneChangeListener должен содержать метод public void stateChanged(ChangeEvent e).  
6. Класс ExceptionHandler должен быть создан в отдельном файле.  
7. Класс ExceptionHandler должен содержать метод public static void log(Exception e).

### HTML Editor (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**.  
После запуска приложения можно будет увидеть текущие результаты: две независимые закладки (HTML и Текст), в каждой из которых можно набирать свой текст.

Требования:  
1. В методе initEditor() для компонента htmlTextPane должен устанавливаться тип контента "text/html" через сеттер setContentType.  
2. В методе initEditor() должен создаваться новый локальный компонент JScrollPane через конструктор принимающий htmlTextPane.  
3. В методе initEditor() для компонента tabbedPane должна добавляться вкладка с именем "HTML" и созданным компонентом JScrollPane на базе htmlTextPane.  
4. В методе initEditor() должен создаваться новый локальный компонент JScrollPane через конструктор принимающий plainTextPane.  
5. В методе initEditor() для компонента tabbedPane должна добавляться вкладка с именем "Текст" и созданным компонентом JScrollPane на базе plainTextPane.  
6. В методе initEditor() для компонента tabbedPane должен устанавливаться предпочтительный размер панели, через сеттер setPreferredSize.  
7. В методе initEditor() для компонента tabbedPane должен добавляться слушатель TabbedPaneChangeListener через метод addChangeListener.  
8. Метод initEditor() должен добавлять по центру панели контента текущего фрейма нашу панель с вкладками, через getContentPane().add().

### HTML Editor (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:  
1. Класс MenuHelper должен быть создан в отдельном файле.  
2. Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, String text, ActionListener actionListener).  
3. Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, Action action).  
4. Класс MenuHelper должен реализовывать публичный статический метод JMenuItem addMenuItem(JMenu parent, String text, Action action).  
5. Класс MenuHelper должен содержать публичный статический метод void initHelpMenu(View view, JMenuBar menuBar).  
6. Класс MenuHelper должен содержать публичный статический метод void initFontMenu(View view, JMenuBar menuBar).  
7. Класс MenuHelper должен содержать публичный статический метод void initColorMenu(View view, JMenuBar menuBar).  
8. Класс MenuHelper должен содержать публичный статический метод void initAlignMenu(View view, JMenuBar menuBar).  
9. Класс MenuHelper должен содержать публичный статический метод void initStyleMenu(View view, JMenuBar menuBar).  
10. Класс MenuHelper должен содержать публичный статический метод void initEditMenu(View view, JMenuBar menuBar).  
11. Класс MenuHelper должен содержать публичный статический метод void initFileMenu(View view, JMenuBar menuBar).

### HTML Editor (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 AbstractActionи 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:  
1. Класс UndoAction должен быть создан в отдельном файле.  
2. Класс RedoAction должен быть создан в отдельном файле.  
3. Класс StrikeThroughAction должен быть создан в отдельном файле.  
4. Класс SubscriptAction должен быть создан в отдельном файле.  
5. Класс SuperscriptAction должен быть создан в отдельном файле.  
6. Класс UndoMenuListener должен быть создан в отдельном файле.  
7. Класс TextEditMenuListener должен быть создан в отдельном файле.

### HTML Editor (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:  
1. В методе initMenuBar() должно создаваться новое меню (объект типа JMenuBar).  
2. В методе initMenuBar() c помощью MenuHelper должно быть проинициализировано меню в следующем порядке: Файл, Редактировать, Стиль, Выравнивание, Цвет, Шрифт и Помощь.  
3. В методе initMenuBar() должно добавляться новосозданное меню в верхнюю часть панели контента текущего фрейма, используя метод getContentPane().  
4. В конструкторе класса View, через класс UIManager, должен устанавливаться внешний вид и поведение (look and feel).

### HTML Editor (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:  
1. Класс UndoMenuListener должен содержать поле представления View view.  
2. Класс UndoMenuListener должен содержать поле JMenuItem undoMenuItem.  
3. Класс UndoMenuListener должен содержать поле JMenuItem redoMenuItem.  
4. Конструктор UndoMenuListener(View view, JMenuItem undoMenuItem, JMenuItem redoMenuItem) должен корректно инициализировать поля класса.  
5. Класс View должен содержать public boolean метод canUndo(), возвращающий false.  
6. Класс View должен содержать public boolean метод canRedo(), возвращающий false.  
7. Метод menuSelected(MenuEvent menuEvent) должен делать доступными или не доступными пункты меню undoMenuItem и redoMenuItem.

### HTML Editor (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:  
1. Класс View должен содержать приватное поле UndoManager undoManager, которое должно быть сразу проинициализовано.  
2. Класс View должен содержать приватное поле UndoListener undoListener, которое должно быть сразу проинициализовано через поле undoManager.  
3. Класс View должен содержать геттер к полю UndoListener undoListener.  
4. Класс View должен содержать public void метод undo().  
5. Класс View должен содержать public void метод redo().  
6. Реализация метода canUndo() класса View должна использовать undoManager.  
7. Реализация метода canRedo() класса View должна использовать undoManager.  
8. Класс View должен содержать public void метод resetUndo(), который должен сбрасывать все правки в менеджере undoManager.  
9. Класс UndoListener должен быть создан в отдельном файле.  
10. Класс UndoListener должен содержать приватное поле UndoManager undoManager.  
11. Конструктор UndoListener(UndoManager undoManager) должен корректно инициализировать поле undoManager.  
12. Метод undoableEditHappened(UndoableEditEvent e) в классе UndoListener должен из переданного события получать правку и добавлять ее в undoManager.