DEV-J120. Задача №4. Разработка пользовательского графического интерфейса

Напишите реализацию класса SimpleEditor, который представляет собой графический интерфейс простого текстового редактора. Класс должен обеспечивать следующую функциональность:

- Возможность открытия или создания текстового файла в простом формате, например, .txt;
- Возможность вывода в окно редактирования всего содержимого открытого или созданного файла;
- Просмотр и изменение текса в окне редактирования;
- Сохранение в файле сделанных изменений;
- Закрытие файла без сохранения сделанных изменений.

Обработку событий пользовательского интерфейса типа ActionEvent и WindowEvent (опционально) обеспечивает класс SimpleEditorListener.

Шаблоны классов SimpleEditor и SimpleEditorListener:

```
/*
 * DEV-J120.Задача №4. Главное окно приложения.
 */
package ru.spbstu.hse.gui;


import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
```

```
/**
 * Класс представляет главное окно приложения, реализующего простой текстовый
 * редактор.Класс обеспечивает возможность открытия или создания файла,
 * содержащего простой текст, например, в формате .txt, .html или xml, с
 * возможностью редактирования его содержания.
 *
 * @author (C)Y.D.Zakovryashin, 11.11.2020
 */
public class SimpleEditor extends JFrame {

    private Container cp;
    /**
     * Надпись, отображающая имя текущего/редактируемого файла, а также метку,
     * отражающую его текущее состояние. Например, если текст изменён, но не
     * сохранён, то в данном поле ставиться специальный маркер (по выбору
     * программиста) или меняется шрифт отображения имени файла.
     */
    private JLabel fileName;
    /**
     * Окно редактирования, в котором отображается и редактируется текст файла.
     */
    private JTextArea text;
    /**
     * Главное меню приложения.
     */
    private JMenuBar bar;
    /**
     * Массив пунктов главного меню приложения, который должен включать "File" и
     * "Edit".
     */
    private JMenu[] menu;
```

```
/**
 * Массив команд главного меню приложения, который должен включать команды
 * "Open", "Save", "Cancel" и "Exit".
 */
private JMenuItem[] commandMenu;
/**
 * Массив кнопок интерфейса "Open", "Save", "Cancel" и "Exit", которые
 * дублируют команды главного меню приложения .
 */
private JButton[] commandButton;
/**
 * Ссылка на обработчик команд пользовательского интерфейса.
 */
private SimpleEditorListener listener;

/**
 * Конструктор приложения.
 */
protected SimpleEditor() {
   setTitle("Simple text editor");
   init();
   createMenu();
   setVisible(true);
}

/**
 * Стартовый метод приложения.
 *
 * @param args параметры командной строки.
 */
public static void main(String[] args) {
   SimpleEditor simpleEditor = new SimpleEditor();
```

```
}

/**
 * Метод инициализирует обработчик событий listener, создаёт и настраивает
 * все элементы пользовательского интерфейса.
 */
private void init() {
   // Реализация метода
}

/**
 * Метод полностью создаёт главное меню приложения и добавляет его в главное
 * окно приложения.
 */
private void createMenu() {
   // Реализация метода
}

/**
 * Метод обеспечивает добавление или замену текста в окне редактирования.
 *
 * @param str добавляемый текст.
 * @param append значение true означает, что текст добавляется к тексту,
 * который содержится в поле. Значение false означает, что текст в поле
 * полностью заменяется на значение str.
 */
void appendText(String str, boolean append) {
   // Реализация метода
}

/**
 * Метод возвращает весь текст, содержащийся в окне редактирования.
```

```
     *
     * @return текст из окна редактирования.
     */
    String getText() {
        // Реализация метода
        return null;
    }
}
```

```
/*
 * DEV-J120.Задача №4. Обработчик событий для класса SimpleEdotor.
 */
package ru.spbstu.hse.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
// Импорт классов, которые используются при решении данной задачи.
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 * Класс поддерживает обработку событий WindowEvent и ActionEvent, возникающих в
 * классе SimpleEditor.
 *
 * @author (C)Y.D.Zakovryashin, 11.11.2020
 */
public class SimpleEditorListener extends WindowAdapter
        implements ActionListener, AutoCloseable {
```

```
/**
 * Ссылка на главное окно приложения.
 */
private SimpleEditor editor;
/**
 * Ссылка на текущий файл, открытый в приложении.
 */
private File file;
/**
 * Ссылки на потоки ввода/вывода, связанные с текущим файлом.
 */
private FileReader reader;
private FileWriter writer;

/**
 * Конструктор класса, определяющий ссылку на главное окно приложения.
 *
 * @param editor ссылка на главное окно приложения.
 */
public SimpleEditorListener(SimpleEditor editor) {
   this.editor = editor;
}

/**
 * Метод обеспечивает обработку событий ActionEvent, связанных с кнопками и
 * пунктами меню класса SimpleEditor.
 *
 * @param ae ссылка на объект, описывающий событие.
 */
@Override
public void actionPerformed(ActionEvent ae) {
   switch (ae.getActionCommand()) {
```

```
        case "open":
            // Обработка команды на открытие редактируемого файла. При
            // обработке этой команды следует использовать классы:
            // java.io.FileReader;
            // javax.swing.JFileChooser;
            // javax.swing.JOptionPane.
            // Следует учесть, что при подаче этой команды в приложении
            // уже может быть открыт какой-то файл.
            // Все выброшенные исключения в данном фрагменте должны быть
            // обработаны, при этом пользователю должны сообщаться причины
            // исключения с помощью стандартных диалоговых окон класса
            // JOptionPane
            break;
        case "save":
            // Обработка команды на сохранение результатов редактирования
            // в открытом файле.
            break;
        case "cancel":
            // Обработка команды на закрытие открытого файла без сохранения
            // сделанных изменений.
            break;
        case "exit":
        // Обработка команды на закрытие приложения. При обработке этой
        // команды следует учесть, что в момент её подачи в приложении
        // может быть открыт какой-то файл. В этом случае пользователю
        // с помощью стандартного диалогового окна класса JOptionPane
        // должен предлагаться выбор между закрытием этого файла с
        // сохранением или без сохранения сделанных в этом файле
        // изменений.
    }
}
```

```
    /**
     * Метод, автоматически вызываемый при удалении объекта из памяти (обычно
     * при закрытии приложения). Следует учесть, что в в момент его вызова в
     * приложении может оставаться открытым какой-то файл.
     *
     */
    @Override
    public void close() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```