МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНЙ УНІВЕРСИТЕТ

кафедра програмного забезпечення автоматизованих систем

**ЗВІТ**

**про виконання**

**практичної роботи з дисципліни**

**"Конструювання програмного забезпечення"**

|  |  |
| --- | --- |
| **Перевірив роботу:** | **Виконав роботу:** |
| доцент кафедри ПЗАС  Володимир МЕТЕЛАП | студент гр. ПЗ-2104  Олександр Ліла |

Черкаси 2023

**Практична робота 2**

**Створення текстового редактору**

**Мета:** Створити текстовий редактор на мові програмування Java з базовим функціоналом для редагування текстових даних, який буде мати наступні функції: відкриття, збереження файлів, копіювання, вставлення та вирізання тексту, а також додаткові можливості форматування, такі як зміна стилю шрифту та визначення розміру шрифту.

**Хід виконання практичної роботи:**

**Технічне завдання на створення простого текстового редактору**

**1)Вимоги завдання:**

**1. Основний інтерфейс:**

* Реалізація текстового редактору на базі класу, що наслідується від **JFrame**.
* Область редагування тексту повинна мати полосу прокрутки, якщо текст не вміщується.
* Використання меню та панелі інструментів для доступу до функціоналу редактору.

**2. Функціональність збереження та відкриття файлів:**

* Можливість відкривати і зберігати тексти в файловій системі.
* Використання діалогового вікна для вибору файлу при відкритті та збереженні.

**3. Функціональність редагування тексту:**

* Можливість копіювання, вставлення та вирізання тексту з використанням області виділення.
* Використання комбінацій клавіш для копіювання, вставлення та вирізання тексту.
* Виведення контекстного меню з опціями копіювання, вставлення та вирізання при натисканні правою кнопкою миші.

**4. Функціональність форматування тексту:**

* Можливість встановлювати жирний, курсивний та підкреслений стиль виділеного тексту.
* Забезпечення можливості вибору розміру шрифту для виділеного тексту.

**5. Статусна інформація:**

* Виведення статусної інформації про кількість слів та літер у поточному тексті.
* Оновлення статусної інформації при зміні тексту в редакторі.

**6. Допоміжна інформація:**

* Додавання пунктів меню для допоміжної інформації, такої як "Довідка" та "Про програму".
* Виведення вікна з довідковою інформацією з файлу "help.txt" при виборі опції "Довідка".

**7. Інтерфейс користувача:**

* Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача. Використання кнопок у панелі інструментів для швидкого доступу до основних опцій редактора.
* Забезпечення можливості закриття редактору з можливістю збереження незбережених змін.

**2) Блок схема роботи текстового редактора**

Так

Ні

1. **3. 2ю -ься Початок: -** Створення екземпляру класу TextEditor. - Встановлення основних параметрів вікна та інтерфейсу.
2. **Очікування подій: -** Програма очікує події від користувача.

Подія сталася?

**3.** **Обробка подій: -** Кнопки та меню реагують на натискання користувача та викликають відповідні методи для виконання операцій.

-Методи взаємодіють з іншими класами, такими як **JFileChooser** для роботи з файлами.

**4. Відображення та оновлення інтерфейсу:**

- Інтерфейс оновлюється в залежності від виконаних операцій та дій користувача.

- Статусна мітка відображає інформацію про кількість слів та літер у введеному тексті.

**5. Закінчення:**

- Програма завершує роботу при закритті вікна.

**3) Ескіз текстового редактора**

**Основне вікно** (рис. 1):

Велика область для введення та редагування тексту.

Полоса прокрутки, якщо текст не вміщується.

Меню з розділами "File", "Edit", "Format", "Help".

**Панель інструментів:**

Кнопки для швидкого доступу до опцій, таких як відкриття файлу, збереження, копіювання, вставлення, вирізання, форматування тексту.

Кнопки для встановлення/зняття стилів тексту (жирний, курсив, підкреслений).

Кнопка для вибору розміру шрифту.

**Статусна панель:**

Мітка для відображення статусної інформації (кількість слів та літер).

Оновлення інформації при редагуванні тексту.

**Меню** (рис. 2)**:**

File: "Open", "Save", "Save As", "Exit".

Edit: "Cut", "Copy", "Paste", "Delete".

Format: "Bold", "Italic", "Underline", "Font Size".

Help: "Help", "About".

**Контекстне меню:**

Опції копіювання, вставлення та вирізання при натисканні правою кнопкою миші на текстовій області.

**Допоміжна інформація:**

Діалогове вікно "Help" з інструкціями та іншою корисною інформацією.

Діалогове вікно "About" з інформацією про програму та автора.

**Закриття редактору:**

Вікно закривається за допомогою хрестика або через меню "File -> Exit".

Якщо є незбережені зміни, виводиться діалогове вікно для збереження перед виходом.

**Інтерфейс користувача:**

Інтуїтивно зрозумілі та зручні елементи для доступу до основних опцій.

Зручне розміщення елементів у вікні для ефективного використання простору.

Використання стандартних графічних елементів та стилів.

Інші аспекти:

Забезпечення підтримки розширення можливостей редактора у майбутньому (наприклад, додаткові функції форматування, підтримка різних кодувань тощо).

Статусна мітка (слова, символи)

File

Help

Edit

Команди меню

**. . .**

**Кнопки**

**Тут буде текст**

Команди вікна

Copy

Delete

Cut

Paste

Команди контекстного меню

… .

Рисунок 1 – Головне вікно текстового редактора.

File

Edit

Help

Copy Ctrl+C

Paste Ctrl+V

Cut Ctrl+X

Команди підменю

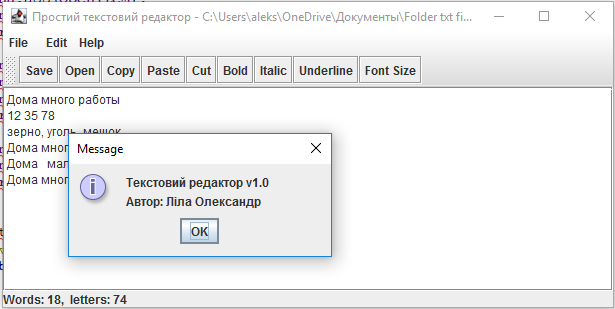
Адресна строка

Рисунок 2 – Ескіз меню

**Код програми на мові Java:**

**import** javax.swing.\*;  
**import** javax.swing.event.DocumentEvent;  
**import** javax.swing.event.DocumentListener;  
**import** javax.swing.text.AttributeSet;  
**import** javax.swing.text.SimpleAttributeSet;  
**import** javax.swing.text.StyleConstants;  
**import** javax.swing.text.StyledDocument;  
  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.\*;  
  
**public class** TextEditor **extends** JFrame {  
 **private** JTextPane **textPane**;  
 **private** JFileChooser **fileChooser**;  
 String **currentFileName**;  
 **private** JLabel **statusLabel**;  
 **private boolean isTextBold** = **false**;  
  
 JMenuBar **menuBar** = **new** JMenuBar();  
 JToolBar **toolBar** = **new** JToolBar(); *// Панель инструментов* JMenu **fileMenu** = **new** JMenu(**"File"**);  
 JMenuItem **openItem** = **new** JMenuItem(**"Open"**);  
 JMenuItem **saveItem** = **new** JMenuItem(**"Save"**);  
  
 JMenu **editMenu** = **new** JMenu(**" Edit"**);  
 JMenuItem **copyItem** = **new** JMenuItem(**"Copy"**);  
 JMenuItem **pasteItem** = **new** JMenuItem(**"Paste"**);  
 JMenuItem **deleteItem** = **new** JMenuItem(**"Cut"**);  
  
 JMenu **helpMenu** = **new** JMenu(**"Help"**);  
 JMenuItem **helpItem** = **new** JMenuItem(**"help"**);  
 JMenuItem **aboutItem** = **new** JMenuItem(**"About program"**);  
  
 *//Метод для відкриття файлу* **public void** openFile(File selectedFile) {  
  
 **currentFileName**=selectedFile.getAbsolutePath();  
 **try** (BufferedReader reader = **new** BufferedReader(**new** FileReader(selectedFile))) {  
 String line;  
 StringBuilder content = **new** StringBuilder();  
 **while** ((line = reader.readLine()) != **null**) {  
 content.append(line).append(**"\n"**);  
 }  
 **textPane**.setText(content.toString());  
 setTitle(**"Простий текстовий редактор - "**+ **currentFileName**);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 **public** String getTextAreaContent() {  
 **return textPane**.getText();  
 }  
  
 *// Метод для збереження файлу* **public void** saveFile(File selectedFile) {  
 **try** (BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(selectedFile))) {  
 writer.write(**textPane**.getText());  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 **public void** setTextAreaContent(String contentToSave) {  
 String savedContent = contentToSave;  
 **textPane**.setText(savedContent);  
 }  
  
 *// Метод для підрахунку кількості слів* **public int** countWords(String text) {  
 String[] words = text.split(**"\\s+"**);  
 **return** words.**length**;  
 }  
  
 *// Метод для підрахунку кількості букв (без пробілів і символів пунктуації)* **public int** countLetters(String text) {  
 text = text.replaceAll(**"[^\\p{L}]"**, **""**); *// Видаляємо всі символи, крім букв* **return** text.length();  
 }  
  
 *// Метод для оновлення статусного мітки* **public void** updateStatusLabel() {  
 String content = **textPane**.getText();  
 **int** wordCount = countWords(content);  
 **int** letterCount = countLetters(content);  
 **statusLabel**.setText(**"Words: "** + wordCount + **", letters: "** + letterCount);  
 }  
  
 **public** TextEditor() {  
 setTitle(**"no name - simple text editor"**);  
 setSize(800, 600);  
 setDefaultCloseOperation(JFrame.***DO\_NOTHING\_ON\_CLOSE***);  
 addWindowListener(**new** WindowAdapter() {  
 @Override  
 **public void** windowClosing(WindowEvent e) {  
 String text = **textPane**.getText().trim(); *// Удаляем начальные и конечные пробелы* **if** (text.isEmpty()) {  
 dispose();  
 } **else** {  
 **int** option = JOptionPane.*showConfirmDialog*(  
 TextEditor.**this**,  
 **"Save file before closing?"**,  
 **"closing"**,  
 JOptionPane.***YES\_NO\_CANCEL\_OPTION***);  
  
 **if** (option == JOptionPane.***YES\_OPTION***) {  
 saveFilePrompt();  
 dispose();  
 } **else if** (option == JOptionPane.***NO\_OPTION***) {  
 dispose();  
 } **else if** (option == JOptionPane.***CANCEL\_OPTION***) {  
  
 }  
 }  
 }  
  
  
 });  
  
 **textPane** = **new** JTextPane();  
 JScrollPane scrollPane = **new** JScrollPane(**textPane**);  
 add(scrollPane, BorderLayout.***CENTER***);  
 add(scrollPane, BorderLayout.***CENTER***);  
 **statusLabel**= **new** JLabel(**"words: 0, letters: 0"**);  
 add(**statusLabel**, BorderLayout.***SOUTH***);  
 JButton saveButton = **new** JButton(**"Save"**);  
 JButton openButton = **new** JButton(**"Open"**);  
 JButton copyButton = **new** JButton(**"Copy"**);  
 JButton pasteButton = **new** JButton(**"Paste"**);  
 JButton cutButton = **new** JButton(**"Cut"**);  
 **toolBar**.add(saveButton);  
 **toolBar**.add(openButton);  
 **toolBar**.add(copyButton);  
 **toolBar**.add(pasteButton);  
 **toolBar**.add(cutButton);  
 add(**toolBar**, BorderLayout.***NORTH***);  
  
 **fileMenu**.add(**openItem**);  
 **fileMenu**.add(**saveItem**);  
 **menuBar**.add(**fileMenu**);  
  
 **editMenu**.add(**copyItem**);  
 **editMenu**.add(**pasteItem**);  
 **editMenu**.add(**deleteItem**);  
 **menuBar**.add(**editMenu**);  
  
 **helpMenu**.add(**helpItem**);  
 **helpMenu**.add(**aboutItem**);  
 **menuBar**.add(**helpMenu**);  
  
  
 saveButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveFilePrompt();  
 }  
 });  
  
 openButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 openFilePrompt();  
 }  
 });  
 copyButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.copy();  
 }  
 });  
 pasteButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.paste();  
 }  
 });  
 cutButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.cut();  
 }  
 });  
  
 JButton boldButton = **new** JButton(**"Bold"**);  
 boldButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 toggleBoldText();  
 }  
 });  
 **toolBar**.add(boldButton);  
  
 JButton italicButton = **new** JButton(**"Italic"**);  
 italicButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 toggleItalicText();  
 }  
 });  
 **toolBar**.add(italicButton);  
  
 JButton underlineButton = **new** JButton(**"Underline"**);  
 underlineButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 toggleUnderlineText();  
 }  
 });  
 **toolBar**.add(underlineButton);  
  
 JButton fontSizeButton = **new** JButton(**"Font Size"**);  
 **toolBar**.add(fontSizeButton);  
 fontSizeButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 changeFontSize();  
 }  
 });  
  
  
 **textPane**.getDocument().addDocumentListener(**new** DocumentListener() {  
 @Override  
 **public void** insertUpdate(DocumentEvent e) {  
 updateStatusLabel();  
 }  
  
 @Override  
 **public void** removeUpdate(DocumentEvent e) {  
 updateStatusLabel();  
 }  
  
 @Override  
 **public void** changedUpdate(DocumentEvent e) {  
 updateStatusLabel();  
 }  
 });  
 **textPane**.addMouseListener(**new** MouseAdapter() {  
 @Override  
 **public void** mouseClicked(MouseEvent e) {  
 **if** (SwingUtilities.*isRightMouseButton*(e)) {  
 *// Показываем контекстное меню* showContextMenu(e);  
 }  
 }  
 });  
**openItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 openFilePrompt();  
 }  
 });  
 **saveItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** returnValue = **fileChooser**.showSaveDialog(**saveItem**);  
 **if** (returnValue == JFileChooser.***APPROVE\_OPTION***) {  
 File selectedFile = **fileChooser**.getSelectedFile();  
 saveFile(selectedFile);  
 }  
 }  
 });  
 **copyItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.copy();  
 *//updateStatusLabel();* }  
 });  
 **copyItem**.setAccelerator(KeyStroke.*getKeyStroke*(KeyEvent.***VK\_C***, Toolkit.*getDefaultToolkit*().getMenuShortcutKeyMask()));  
  
 **pasteItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.paste();  
 *//updateStatusLabel();* }  
 });  
 **pasteItem**.setAccelerator(KeyStroke.*getKeyStroke*(KeyEvent.***VK\_P***, Toolkit.*getDefaultToolkit*().getMenuShortcutKeyMaskEx()));  
 **deleteItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.replaceSelection(**""**);  
 }  
 });  
  
 **helpItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JFrame frame = **new** JFrame(**"Help"**);  
 frame.setSize(400, 300);  
 frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 frame.setVisible(**false**);  
 **try** {  
 String filePath = **"src/help.txt"**;  
 BufferedReader reader = **new** BufferedReader(**new** FileReader(filePath));  
 StringBuilder content = **new** StringBuilder();  
 String line;  
 **while** ((line = reader.readLine()) != **null**) {  
 content.append(line).append(**"\n"**);  
 }  
 reader.close();  
  
 JOptionPane.*showMessageDialog*(frame, content.toString(), **"Help"**, JOptionPane.***INFORMATION\_MESSAGE***);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(frame, **"Ошибка при чтении файла справки"**, **"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 });  
  
 **aboutItem**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JOptionPane.*showMessageDialog*(TextEditor.**this**, **"Текстовий редактор v1.0\nАвтор: Ліла Олександр"**);  
 }  
 });  
  
 setJMenuBar(**menuBar**);  
 **fileChooser** = **new** JFileChooser();  
  
 setVisible(**true**);  
 }  
 *// Метод для показа контекстного меню* **private void** showContextMenu(MouseEvent e) {  
 JPopupMenu contextMenu = **new** JPopupMenu();  
 JMenuItem copyItem = **new** JMenuItem(**"Copy"**);  
 copyItem.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent actionEvent) {  
 **textPane**.copy();  
 }  
 });  
 contextMenu.add(copyItem);  
  
 JMenuItem pasteItem = **new** JMenuItem(**"Paste"**);  
 pasteItem.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent actionEvent) {  
 **textPane**.paste();  
 }  
 });  
 contextMenu.add(pasteItem);  
 *// пункты контекстного меню по необходимости  
 // Добавьте новый элемент "Видалити" в контекстное меню* JMenuItem deleteContextMenu = **new** JMenuItem(**"Cut"**);  
 deleteContextMenu.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **textPane**.replaceSelection(**""**);  
 }  
 });  
 contextMenu.add(deleteContextMenu);  
  
 contextMenu.show(e.getComponent(), e.getX(), e.getY());  
 }  
 **private void** saveFilePrompt() {  
 **int** returnValue = **fileChooser**.showSaveDialog(**saveItem**);  
 **if** (returnValue == JFileChooser.***APPROVE\_OPTION***) {  
 File selectedFile = **fileChooser**.getSelectedFile();  
 saveFile(selectedFile);  
 }  
 }  
  
 **private void** openFilePrompt() {  
 String text = **textPane**.getText().trim();  
 **if** (!text.isEmpty()) {  
 **int** option = JOptionPane.*showConfirmDialog*(  
 TextEditor.**this**,  
 **"Зберегти поточний файл перед відкриттям нового?"**,  
 **"Зберегення"**,  
 JOptionPane.***YES\_NO\_CANCEL\_OPTION***);  
  
 **if** (option == JOptionPane.***YES\_OPTION***) {  
 saveFilePrompt();  
 } **else if** (option == JOptionPane.***CANCEL\_OPTION***) {  
 **return**;  
 }  
 }  
  
 **int** returnValue = **fileChooser**.showOpenDialog(**openItem**);  
 **if** (returnValue == JFileChooser.***APPROVE\_OPTION***) {  
 File selectedFile = **fileChooser**.getSelectedFile();  
 openFile(selectedFile);  
 }  
 }  
  
 **private void** toggleBoldText() {  
 **int** selectionStart = **textPane**.getSelectionStart();  
 **int** selectionEnd = **textPane**.getSelectionEnd();  
  
 **if** (selectionStart != selectionEnd) {  
 SimpleAttributeSet attributeSet = **new** SimpleAttributeSet();  
 StyledDocument doc = **textPane**.getStyledDocument();  
  
 *// Получаем атрибуты текста в начальной позиции выделения* AttributeSet startAttributes = doc.getCharacterElement(selectionStart).getAttributes();  
  
 *// Проверяем, жирный ли текст в начальной позиции выделения* **boolean** isBold = StyleConstants.*isBold*(startAttributes);  
  
 *// Инвертируем состояние жирного шрифта* StyleConstants.*setBold*(attributeSet, !isBold);  
  
 doc.setCharacterAttributes(selectionStart, selectionEnd - selectionStart, attributeSet, **false**);  
 }  
 }  
  
 **private void** toggleItalicText() {  
 **int** selectionStart = **textPane**.getSelectionStart();  
 **int** selectionEnd = **textPane**.getSelectionEnd();  
  
 **if** (selectionStart != selectionEnd) {  
 SimpleAttributeSet attributeSet = **new** SimpleAttributeSet();  
 StyledDocument doc = **textPane**.getStyledDocument();  
  
 *// Получаем атрибуты текста в начальной позиции выделения* AttributeSet startAttributes = doc.getCharacterElement(selectionStart).getAttributes();  
  
 *// Проверяем, курсивный ли текст в начальной позиции выделения* **boolean** isItalic = StyleConstants.*isItalic*(startAttributes);  
  
 *// Инвертируем состояние курсива* StyleConstants.*setItalic*(attributeSet, !isItalic);  
  
 doc.setCharacterAttributes(selectionStart, selectionEnd - selectionStart, attributeSet, **false**);  
 }  
 }  
  
 **private void** toggleUnderlineText() {  
 **int** selectionStart = **textPane**.getSelectionStart();  
 **int** selectionEnd = **textPane**.getSelectionEnd();  
  
 **if** (selectionStart != selectionEnd) {  
 SimpleAttributeSet attributeSet = **new** SimpleAttributeSet();  
 StyledDocument doc = **textPane**.getStyledDocument();  
  
 *// Получаем атрибуты текста в начальной позиции выделения* AttributeSet startAttributes = doc.getCharacterElement(selectionStart).getAttributes();  
  
 *// Проверяем, подчеркнут ли текст в начальной позиции выделения* **boolean** isUnderline = StyleConstants.*isUnderline*(startAttributes);  
  
 *// Инвертируем состояние подчеркивания* StyleConstants.*setUnderline*(attributeSet, !isUnderline);  
  
 doc.setCharacterAttributes(selectionStart, selectionEnd - selectionStart, attributeSet, **false**);  
 }  
 }  
  
 **private void** changeFontSize() {  
 **int** selectionStart = **textPane**.getSelectionStart();  
 **int** selectionEnd = **textPane**.getSelectionEnd();  
  
 **if** (selectionStart != selectionEnd) {  
 String input = JOptionPane.*showInputDialog*(**"Enter new font size:"**);  
 **try** {  
 **int** newSize = Integer.*parseInt*(input);  
 **if** (newSize > 0) {  
 SimpleAttributeSet attributeSet = **new** SimpleAttributeSet();  
 StyledDocument doc = **textPane**.getStyledDocument();  
  
 *// Устанавливаем новый размер шрифта* StyleConstants.*setFontSize*(attributeSet, newSize);  
  
 doc.setCharacterAttributes(selectionStart, selectionEnd - selectionStart, attributeSet, **false**);  
 } **else** {  
 JOptionPane.*showMessageDialog*(**this**, **"Font size must be a positive integer."**);  
 }  
 } **catch** (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(**this**, **"Invalid font size. Please enter a positive integer."**);  
 }  
 }  
 }  
  
 **public static void** main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> **new** TextEditor());  
 }  
  
}

Результат виконання програми:



**Висновок:** Створено текстовий редактор на базі класу, що наслідується від JFrame. Область редагування тексту має полосу прокрутки для зручності в разі, якщо текст не вміщується. Використовані меню та панель інструментів для доступу до функціоналу редактору.

На мою думку проект вдало виконано, і він відповідає вимогам технічного завдання. Розроблений текстовий редактор є функціональним та зручним у використанні, а його інтерфейс відповідає сучасним стандартам. Робота над проектом дозволила отримати практичні навички в роботі з графічним інтерфейсом, обробкою подій, та взаємодією з файловою системою.