Full Documentation for Console-Based Text Editor Project

Project Overview

This project is a **console-based text editor** that allows users to create, edit, and save text files using string manipulation techniques in Java. The implementation focuses on emulating basic functionalities of a notepad editor while leveraging Java's StringBuilder and String methods for efficient string operations.

The primary features include:

1. **Displaying Content**: Viewing the current state of the document.
2. **Adding Text**: Appending new text to the document.
3. **Cut/Copy/Paste**: Managing clipboard operations.
4. **Undo/Redo**: Navigating between previous and current states of the document.
5. **Find and Replace**: Searching for specific text and replacing it.
6. **Saving to File**: Storing the document content into a file.

Features and Implementation

1. Display Content

- **Description**: This feature prints the current document content to the console. If the document is empty, a placeholder [Empty Document] is displayed.
- **Purpose**: Allows the user to review their work at any point.

- **Method**:
    - content.length() is used to check if the document is empty.
    - Prints the content if available.
- **Enhancement Opportunity**:
    - Add support for **syntax highlighting** (using ANSI escape codes for colour formatting).
    - Add a **line-numbering system** to improve clarity.

---

2. Add Text

- **Description**: Appends user-input text to the current document.
- **Purpose**: Allows users to add new information to the document.
- **Method**:
    - content.append(text) is used to add the text to the StringBuilder.
    - Changes are stored in the undoStack before being applied.
- **Enhancement Opportunity**:
    - Add a feature to insert text at specific positions, not just the end.
    - Support for **text alignment** options (left, right, center).

---

3. Cut Text

- **Description**: Removes a portion of the text from the document based on start and end indices and stores it in the clipboard.
- **Purpose**: Enables users to relocate text.
- **Method**:
    - content.substring(start, end) extracts the specified range of text into clipboard.
    - content.delete(start, end) removes the extracted portion.
    - Error handling ensures indices are valid and within bounds.
- **Enhancement Opportunity**:
    - Support for **multiple selections**.
    - Provide a **visual preview** of the selected portion before the operation.

---

4. Copy Text

- **Description**: Copies a portion of the text into the clipboard without altering the document.
- **Purpose**: Helps users duplicate text efficiently.
- **Method**:
    - content.substring(start, end) is used to copy the specified range into clipboard.
    - Error handling ensures indices are valid and within bounds.
- **Enhancement Opportunity**:

    - Support for copying multiple non-contiguous ranges.

---

5. Paste Text

- **Description**: Inserts the content of the clipboard at a user-specified position.
- **Purpose**: Completes the cut-copy-paste workflow.
- **Method**:
    - content.insert(position, clipboard) is used to paste the clipboard text at the specified position.
    - Error handling ensures the position is valid.
- **Enhancement Opportunity**:
    - Support for pasting into **multiple locations** simultaneously.

---

6. Undo

- **Description**: Reverts the document to its last state.
- **Purpose**: Enables users to recover from mistakes.
- **Method**:
    - The current state is saved into undoStack before any operation.
    - On undo, the content is reverted to the state stored in undoStack, and the current state is stored in redoStack.
- **Enhancement Opportunity**:
    - Maintain a history of multiple undo levels for deeper restoration.

---

7. Redo

- **Description**: Re-applies the last undone change.
- **Purpose**: Complements the undo functionality.
- **Method**:
  - On redo, the content is reverted to the state stored in redoStack, and the current state is pushed back into undoStack.
- **Enhancement Opportunity**:
  - Add the ability to view the redo/undo stack history.

---

8. Find and Replace

- **Description**: Searches for a specified substring and replaces it with another string.
- **Purpose**: Allows users to modify text efficiently in bulk.
- **Method**:
  - content.toString().replace(find, replace) replaces all occurrences of the find string with the replace string.
  - Error handling ensures the text to find exists before proceeding.
- **Enhancement Opportunity**:
  - Add an option for **case-sensitive or case-insensitive** searches.
  - Support for **regex-based search** patterns.

---

9. Save to File

- **Description**: Saves the current document content to a user-specified file.
- **Purpose**: Enables users to store their work for future use.
- **Method**:
    - BufferedWriter is used to write the content to a file.
    - Error handling ensures proper file permissions and existence.
- **Enhancement Opportunity**:
    - Add support for saving in different formats (e.g., .txt, .md, .html).
    - Add an auto-save feature.

Additional Features for Future Development

1. **Search Navigation**:
    - Highlight all occurrences of the search term in the document.
    - Provide next/previous navigation to move between matches.
2. **Text Formatting**:
    - Add features for bold, italic, underline, or color-coded text (via markup or ANSI codes).
3. **Auto-save**:
    - Periodically save the document to a temporary file to prevent data loss.
4. **Version History**:

    - Maintain a full version history of the document to allow users to view and restore previous versions.
5. **Advanced Clipboard Management**:
    - Allow users to view the clipboard history and choose from multiple copied items.
6. **UI Upgrade**:
    - Transition from a console-based application to a GUI-based editor using libraries like Swing or JavaFX.

Error Handling and Validation

Error handling is implemented at every stage of the program:

1. **Invalid Input**: Ensures all user inputs are valid and within expected ranges.
2. **File Operations**: Catches file-related exceptions like IOException to handle permission issues or invalid filenames.
3. **Clipboard Operations**: Ensures clipboard operations (cut/copy/paste) are performed only when appropriate.
4. **Undo/Redo**: Prevents undo/redo operations if no corresponding actions exist in the stacks.

Conclusion

This project demonstrates a robust console-based text editor with essential text editing functionalities. It uses Java's string manipulation features to manage and manipulate text efficiently while providing a simple user interface through a console menu.

By incorporating the suggested enhancements, such as syntax highlighting, regex search, and advanced formatting options, this project can evolve into a feature-rich text editor suitable for real-world use cases.