INDEX

ABSTRACT

The Java Notepad application is a comprehensive text editor developed using Java Swing, providing users with a feature-rich environment for creating, editing, and managing text files. This project aims to offer a versatile tool that meets the needs of users ranging from casual note-taking to more complex document editing tasks.

The application's core functionalities include the ability to create new files, open existing ones, and save changes, ensuring users can easily manage their text documents. Additionally, the Java Notepad application features advanced editing capabilities such as cut, copy, and paste operations, enhancing user productivity.

One of the key highlights of the Java Notepad application is its customization options. Users can personalize their editing experience by changing the background color of the text area and adjusting the font style and font family to suit their preferences. This level of customization enhances user comfort and productivity, making the application more user-friendly.

The user interface of the Java Notepad application is designed to be simple and intuitive, allowing users to navigate the application effortlessly. The use of Java Swing ensures platform independence, enabling users to run the application on various operating systems without compatibility issues.

Overall, the Java Notepad application is a powerful and versatile text editing tool that offers a wide range of features in a user-friendly package. Whether used for simple note-taking or more complex document editing tasks, the Java Notepad application provides users with a reliable and efficient solution for managing text files.

INTRODUCTION

The Java Notepad application is a versatile text editor designed to simplify text file management. It offers a range of essential features, including the ability to create new files, open existing ones, edit content, and save changes. Users can easily manipulate text using standard operations such as cut, copy, and paste.

One of the notable features of this application is its flexibility in customization. Users can personalize their editing experience by changing the background color of the text area and adjusting the font style to suit their preferences. This level of customization enhances user experience and makes the application more user-friendly.

Built using Java Swing, the Notepad application provides a familiar and intuitive user interface. This ensures that users can navigate the application effortlessly, even if they are not experienced with text editors. Overall, the Java Notepad application is a powerful tool that offers a simple and effective solution for managing text files.

HARDWARE AND SOFTWARE REQ.

Hardware Requirements:

Processor: Intel Core i3 or equivalent

RAM: 2GB or higher

Storage: 100MB of available disk space

Software Requirements:

Java Development Kit (JDK) 8 or later.

Operating System: Windows 7 or later, macOS, or Linux.

Integrated Development Environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans(Any One).

Java Swing library (included in JDK).

Text editor or IDE for viewing and editing the source code.

ALGORITHM

1. **Class Definition:** Create a class named `Notepad` that extends `JFrame` and implements `ActionListener`.

2. **Instance Variables:** Declare `JFrame f` and `JTextArea t` as instance variables inside the `Notepad` class.

3. **Constructor:** Create a constructor for the `Notepad` class to initialize the application.
   - Initialize `f` with a title "Notepad By Zaid & Sameeh".
   - Set the look and feel to NimbusLookAndFeel for a modern appearance.
   - Create a new `JTextArea` `t` for text editing.
   - Create a `JMenuBar` `mb` for the application's menu bar.
   - Create two menus `m1` ("File") and `m2` ("Edit") within the menu bar.
   - Add menu items for "New", "Open", "Save", "Print", "Cut", "Copy", "Paste", and "Close" to the `m1` and `m2` menus.
   - Add "Change Background" and "Change Font Style" menu items to `mb` for customizing the text area.
   - Set action listeners for all menu items to handle user actions.
   - Set the `JMenuBar` of `f` to `mb` and add `t` to `f` for display.
   - Set the size of `f` to 600x500 pixels and center it on the screen for a standard window size.
   - Make `f` visible to the user.

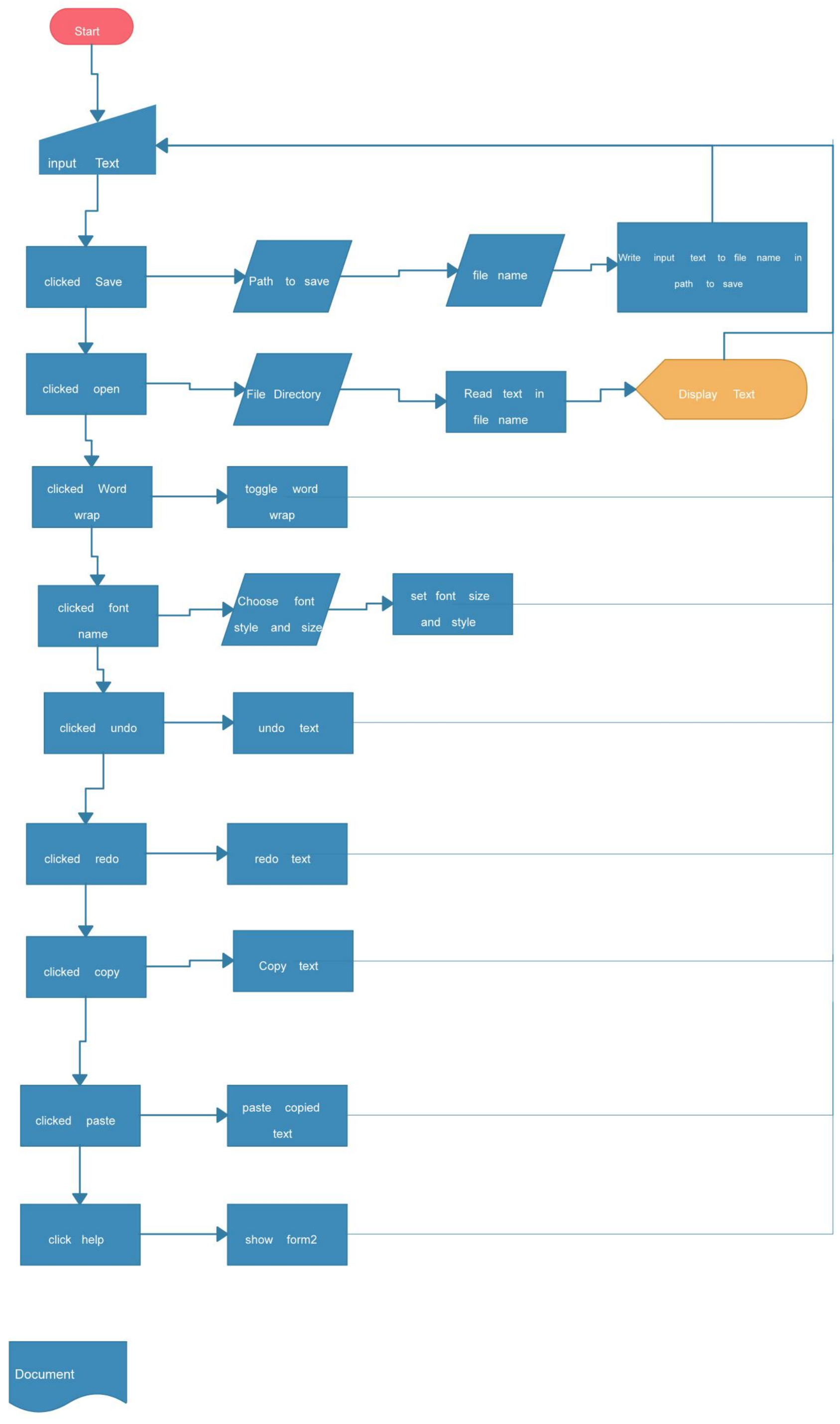
4. **Utility Method:** Create a `setButtonPreferredSize(AbstractButton button)` method to set the preferred size of buttons in the menu bar for consistent appearance.

5. **Action Handling:** Implement the `actionPerformed(ActionEvent e)` method to handle actions performed by the user on the menu items.
   - If the action is "Cut", "Copy", or "Paste", perform the respective operation on the text area `t`.

- If the action is "Save", prompt the user to choose a file location and save the text from `t` to that file.
- If the action is "Print", print the content of `t`.
- If the action is "Open", prompt the user to select a file to open and load its content into `t`.
- If the action is "New", clear the text area `t`.
- If the action is "Close", hide the application window `f`.

6. **Main Method:** Create a `main` method to instantiate a new `Notepad` object and start the application.

FLOWCHART


Start
input Text
clicked Save
Path to save
file name
Write input text to file name in path to save
clicked open
File Directory
Read text in file name
Display Text
clicked Word wrap
toggle word wrap
clicked font name
Choose font style and size
set font size and style
clicked undo
undo text
clicked redo
redo text
clicked copy
Copy text
clicked paste
paste copied text
click help
show form2
Document

SOURCE CODE

```
// Import necessary packages
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;

// Notepad class extending JFrame and implementing ActionListener
public class Notepad extends JFrame implements ActionListener {
   // Declare JTextArea and JFrame variables
   private JTextArea t;
   private JFrame f;

   // Flag to track if text has been changed
   private boolean textChanged = false;

   // Constructor for Notepad class
   Notepad() {
      try {
         // Set the look and feel to the system's look and feel
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
// Create a new JFrame
f = new JFrame("Notepad By Zaid & Sameeh");
// Add a window listener to handle window closing event
f.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        onClose(); // Call onClose method
    }
});

// Create a new JTextArea for editing
t = new JTextArea();
// Set the background color of the text area
t.setBackground(Color.LIGHT_GRAY);
// Add a document listener to track text changes
t.getDocument().addDocumentListener(new DocumentListener() {
    @Override
    public void insertUpdate(DocumentEvent e) {
        textChanged = true; // Text has been inserted
    }

    @Override
    public void removeUpdate(DocumentEvent e) {
        textChanged = true; // Text has been removed
    }

    @Override
    public void changedUpdate(DocumentEvent e) {
        // Style changes, ignored for now
    }
});

// Create a new menu bar
JMenuBar mb = new JMenuBar();
// Create a new "File" menu
JMenu m1 = new JMenu("File");
// Create menu items for "File" menu
JMenuItem mi1 = new JMenuItem("New");
JMenuItem mi2 = new JMenuItem("Open");
JMenuItem mi3 = new JMenuItem("Save");
JMenuItem mi9 = new JMenuItem("Print");

// Add action listeners to menu items
mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi9.addActionListener(this);

// Set preferred size and properties for menu items
setButtonPreferredSize(mi1);
setButtonPreferredSize(mi2);
setButtonPreferredSize(mi3);
setButtonPreferredSize(mi9);

// Add menu items to "File" menu
m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.add(mi9);

// Create a new "Edit" menu
```

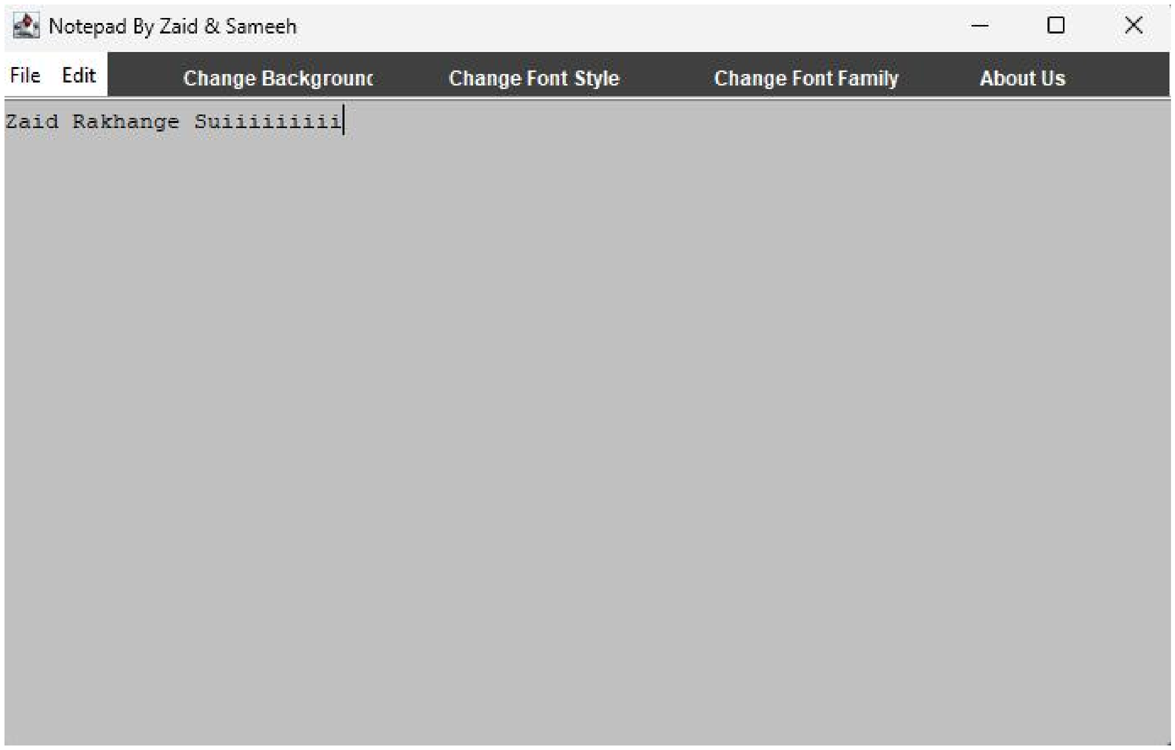
```
JMenu m2 = new JMenu("Edit");
// Create menu items for "Edit" menu
JMenuItem mi4 = new JMenuItem("Cut");
JMenuItem mi5 = new JMenuItem("Copy");
JMenuItem mi6 = new JMenuItem("Paste");

// Add action listeners to menu items
mi4.addActionListener(this);
mi5.addActionListener(this);
mi6.addActionListener(this);

// Set preferred size and properties for menu items
setButtonPreferredSize(mi4);
setButtonPreferredSize(mi5);
setButtonPreferredSize(mi6);

// Add menu items to "Edit" menu
m2.add(mi4);
m2.add(mi5);
m2.add(mi6);

// Create a menu item for "Select All"
JMenuItem mi7 = new JMenuItem("Select All");
mi7.addActionListener(this);
// Set preferred size and properties for the menu item
setButtonPreferredSize(mi7);
// Add the menu item to "Edit" menu
m2.add(mi7);

// Add "File" and "Edit" menus to the menu bar
mb.add(m1);
mb.add(m2);

// Create menu items for additional functionality
JMenuItem changeBgButton = new JMenuItem("Change Background");
JMenuItem changeFontButton = new JMenuItem("Change Font Style");
JMenuItem changeFontFamilyButton = new JMenuItem("Change Font Family");
JMenuItem aboutUsMenuItem = new JMenuItem("About Us");

// Add action listeners to additional menu items
changeBgButton.addActionListener(e -> changeBackground());
changeFontButton.addActionListener(e -> changeFontStyle());
changeFontFamilyButton.addActionListener(e -> changeFontFamily());
aboutUsMenuItem.addActionListener(e -> showAboutUsDialog());

// Set preferred size and properties for additional menu items
setButtonPreferredSize(changeBgButton);
setButtonPreferredSize(changeFontButton);
setButtonPreferredSize(changeFontFamilyButton);
setButtonPreferredSize(aboutUsMenuItem);

// Add additional menu items to the menu bar
mb.add(changeBgButton);
mb.add(changeFontButton);
mb.add(changeFontFamilyButton);
mb.add(aboutUsMenuItem);

// Set the menu bar for the JFrame
f.setJMenuBar(mb);

// Add the text area to the JFrame
f.add(new JScrollPane(t));
```

```
        // Set the size of the JFrame
        f.setSize(710, 450);
        // Set the default close operation for the JFrame
        f.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        // Center the JFrame on the screen
        f.setLocationRelativeTo(null);
        // Make the JFrame visible
        f.setVisible(true);
    }

    // Method to set preferred size and properties for buttons
    private void setButtonPreferredSize(AbstractButton button) {
        Dimension dim = new Dimension(120, button.getPreferredSize().height);
        button.setPreferredSize(dim);
        button.setFont(new Font("Arial", Font.BOLD, 12));
        button.setBorderPainted(true);
        button.setForeground(Color.WHITE);
        button.setBackground(Color.DARK_GRAY);
        button.setFocusPainted(true);
        button.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));
        button.setOpaque(true);
        button.repaint();
    }

    // ActionListener implementation
    public void actionPerformed(ActionEvent e) {
        // Handle action commands for menu items
        switch (e.getActionCommand()) {
            case "Cut" -> t.cut();
            case "Copy" -> t.copy();
            case "Paste" -> t.paste();
            case "Select All" -> onSelectAll();
            case "Save" -> onSave();
            case "Print" -> onPrint();
            case "Open" -> onOpen();
            case "New" -> onNew();
            case "Close" -> onClose();
        }
    }

    // Method to handle window closing event
    private void onClose() {
        if (textChanged) {
            // If text has been changed, ask user to save changes
            switch (askAboutUnsavedChanges()) {
                case JOptionPane.CLOSED_OPTION, JOptionPane.CANCEL_OPTION -> {
                    // Do nothing if user cancels or closes the dialog
                }
                case JOptionPane.YES_OPTION -> {
                    // Save the file and close the frame
                    if (!onSave())
                        return;

                    f.dispose();
                }
                case JOptionPane.NO_OPTION -> f.dispose();
            }
        } else
            f.dispose();
    }

    // Method to save the file
```

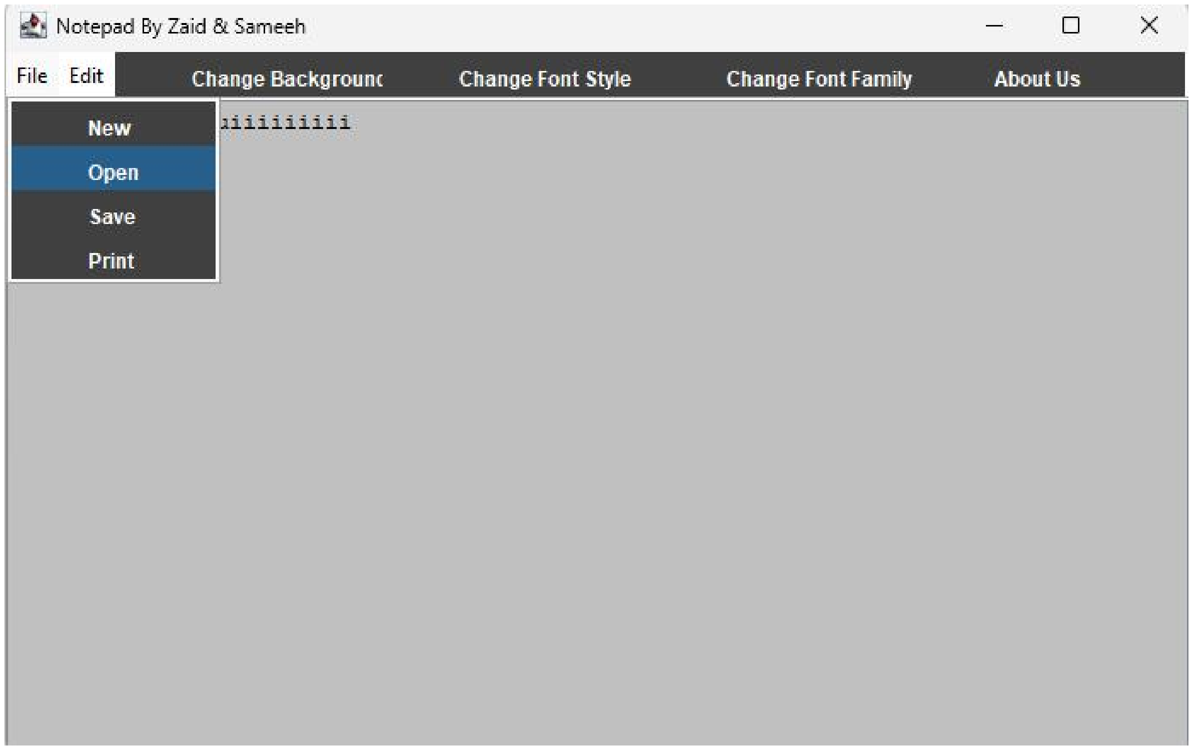
```
private boolean onSave() {
    JFileChooser j = new JFileChooser("f:");
    int r = j.showSaveDialog(null);
    if (r == JFileChooser.APPROVE_OPTION) {
        File fi = new File(j.getSelectedFile().getAbsolutePath());
        try (FileWriter wr = new FileWriter(fi, false); BufferedWriter w = new BufferedWriter(wr)) {
            w.write(t.getText());
            textChanged = false;
            return true;
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(f, ex.getMessage());
        }
    }

    return false;
}

// Method to print the text
private void onPrint() {
    try {
        t.print();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(f, ex.getMessage());
    }
}

// Method to select all text
private void onSelectAll() {
    t.selectAll();
}
```

```
// Method to open a file
private void onOpen() {
    JFileChooser j = new JFileChooser("f:");
    int r = j.showOpenDialog(null);
    if (r == JFileChooser.APPROVE_OPTION) {
        File fi = new File(j.getSelectedFile().getAbsolutePath());
        try (FileReader fr = new FileReader(fi); BufferedReader br = new BufferedReader(fr)) {
            StringBuilder sb = new StringBuilder();
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line).append("\n");
            }
            t.setText(sb.toString());
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(f, ex.getMessage());
        }
    }
}

// Method to create a new document
private void onNew() {
    if (textChanged) {
        switch (askAboutUnsavedChanges()) {
            case JOptionPane.CLOSED_OPTION, JOptionPane.CANCEL_OPTION -> {
                return;
            }
            case JOptionPane.YES_OPTION -> {
                if (!onSave())
                    return;
            }
        }
        }

        t.setText("");
        textChanged = false;
    }

    // Method to ask the user about unsaved changes
    private int askAboutUnsavedChanges() {
        return JOptionPane.showConfirmDialog(
                f,
                "New file has been modified, save changes?",
                "Save changes?",
                JOptionPane.YES_NO_CANCEL_OPTION);
    }

    // Method to show the about us dialog
    private void showAboutUsDialog() {
        String message = "<html>" +
                "Made with    by Zaid and Team<br><br>" +
                "Copyright @2024-2025<br>" +
                "This project is under GitHub's MIT License.<br><br>" +
                "The Product is available for fair use<br><br>" +
                "Illegal Use of Product can lead to Copyright Infringements<br><br>" +
                "This Product is Open Source on Github feel free to contribute<br><br><br>" +
                "<a href=\"https://github.com/zaid-commits/Notepad-JavaApplication\">Contribute on
GitHub</a></html>";

        JLabel label = new JLabel(message);
        label.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        label.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                try {
                    Desktop.getDesktop().browse(new URI("https://github.com/zaid-commits/Notepad-JavaApplication"));
                } catch (IOException | URISyntaxException ex) {
                    ex.printStackTrace();
                }
            }
        });

        JOptionPane.showMessageDialog(f, label, "About Us", JOptionPane.INFORMATION_MESSAGE);
    }

    // Main method to create an instance of Notepad
    public static void main(String[] args) {
        new Notepad();
    }
}
```

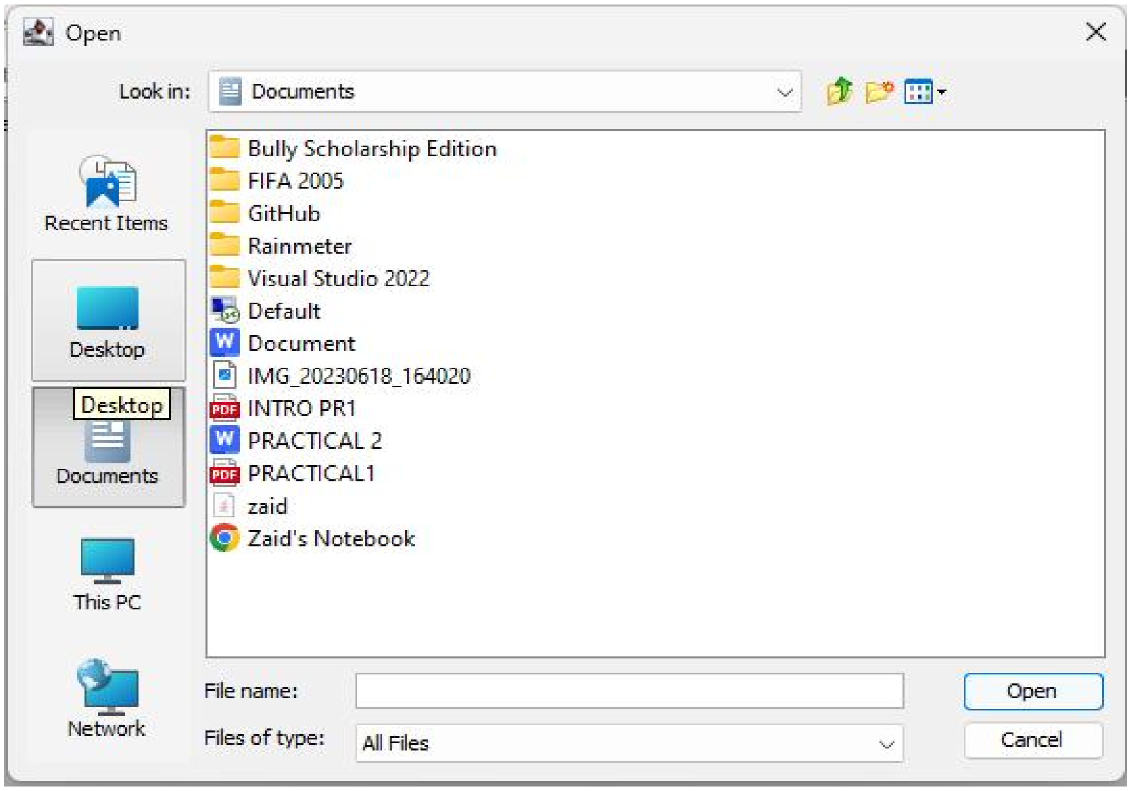
OUTPUT

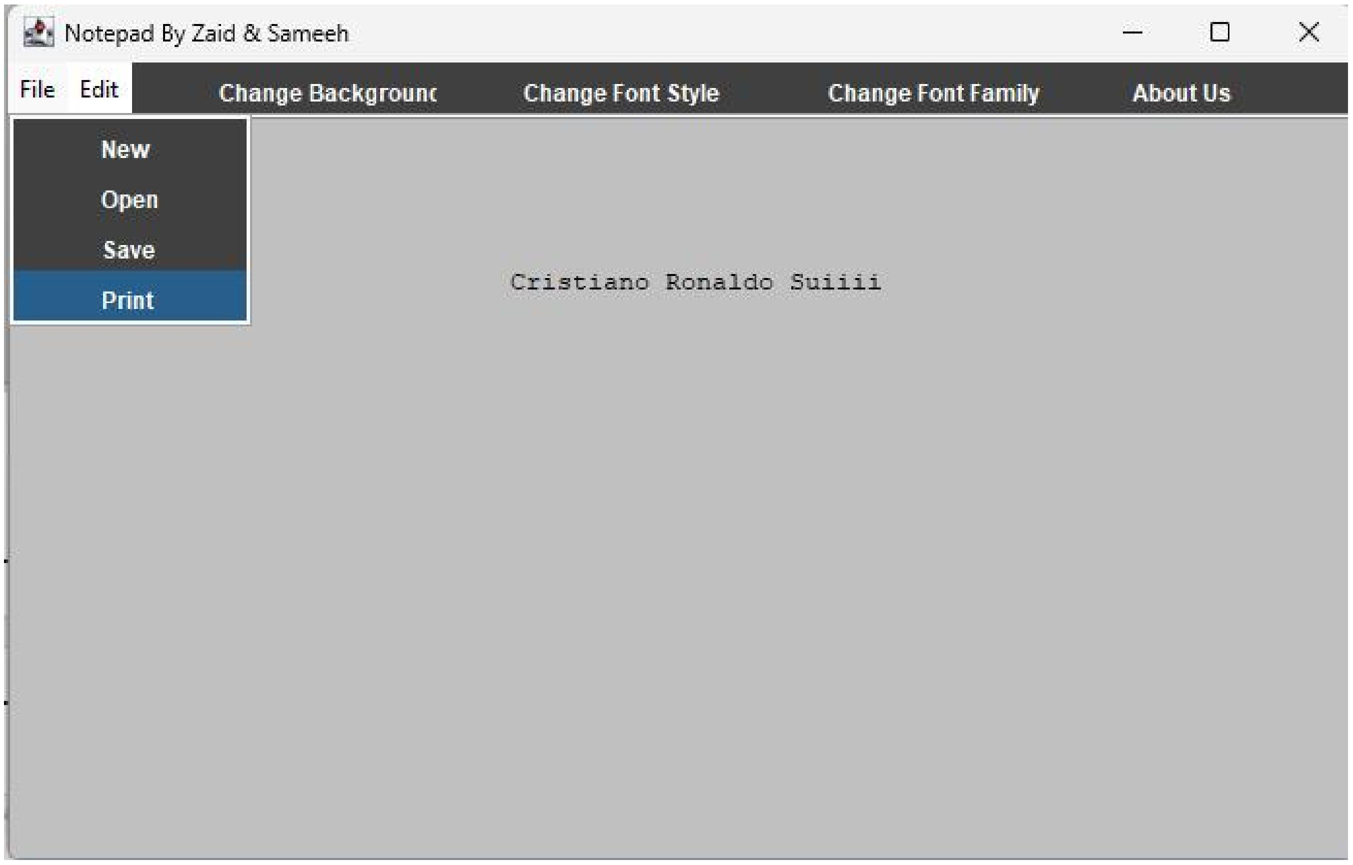


```
Zaid Rakhange Suiiiiiiii
```

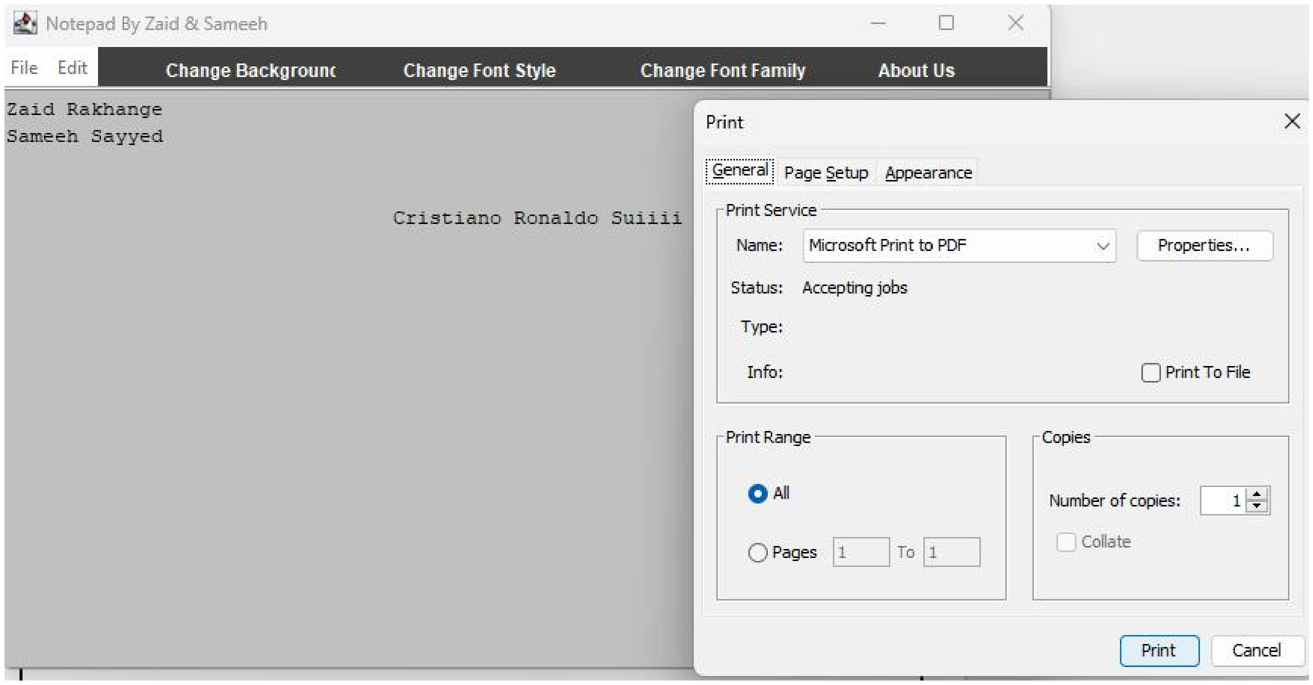


```
uiiiiiiii
```





```
Cristiano Ronaldo Suiiii
```



```
Zaid Rakhange
Sameeh Sayyed

                                        Cristiano Ronaldo Suiiii
```

Print Service

Name: Microsoft Print to PDF

Status: Accepting jobs

Type:

Info:

[ ] Print To File

Print Range

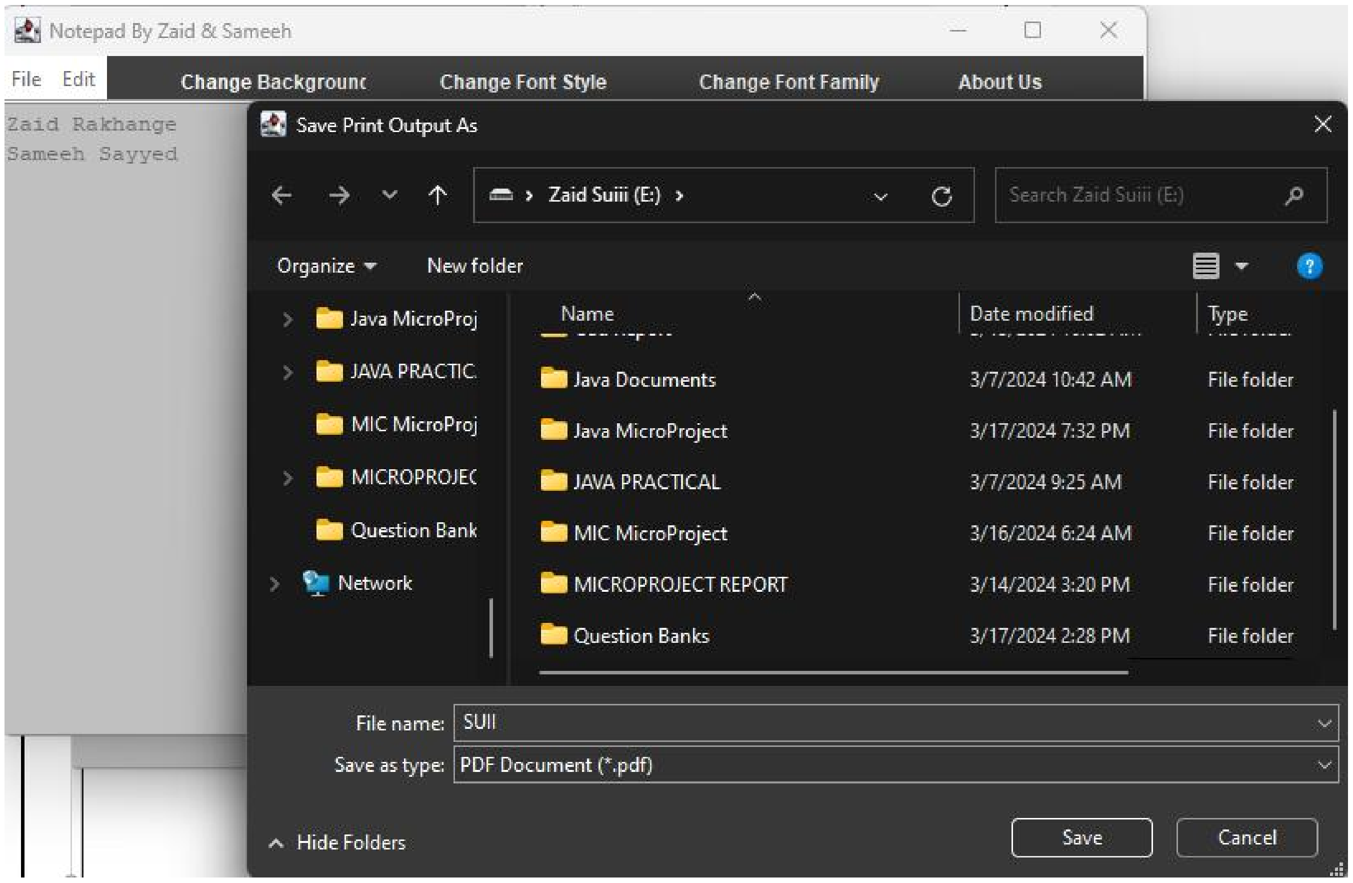
[✓ All]

Pages 1 To 1

Copies

Number of copies: 1

[ ] Collate

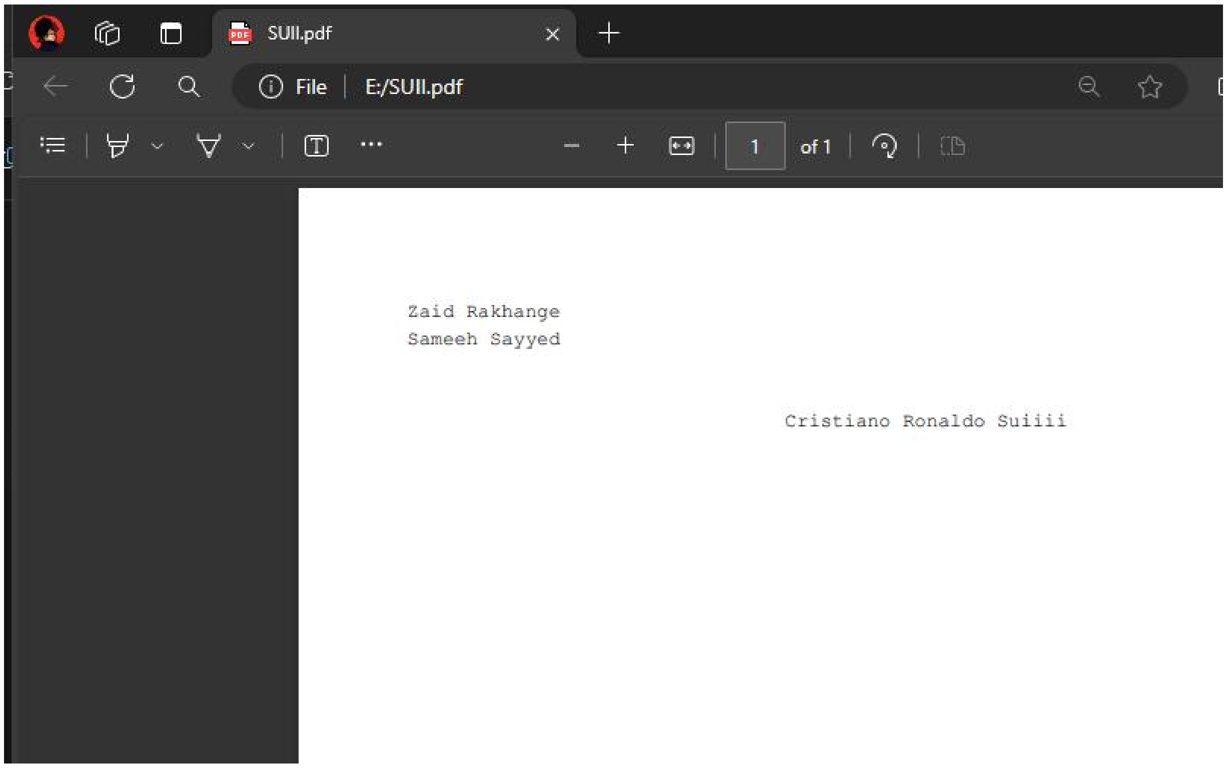


```
Zaid Rakhange
Sameeh Sayyed
```

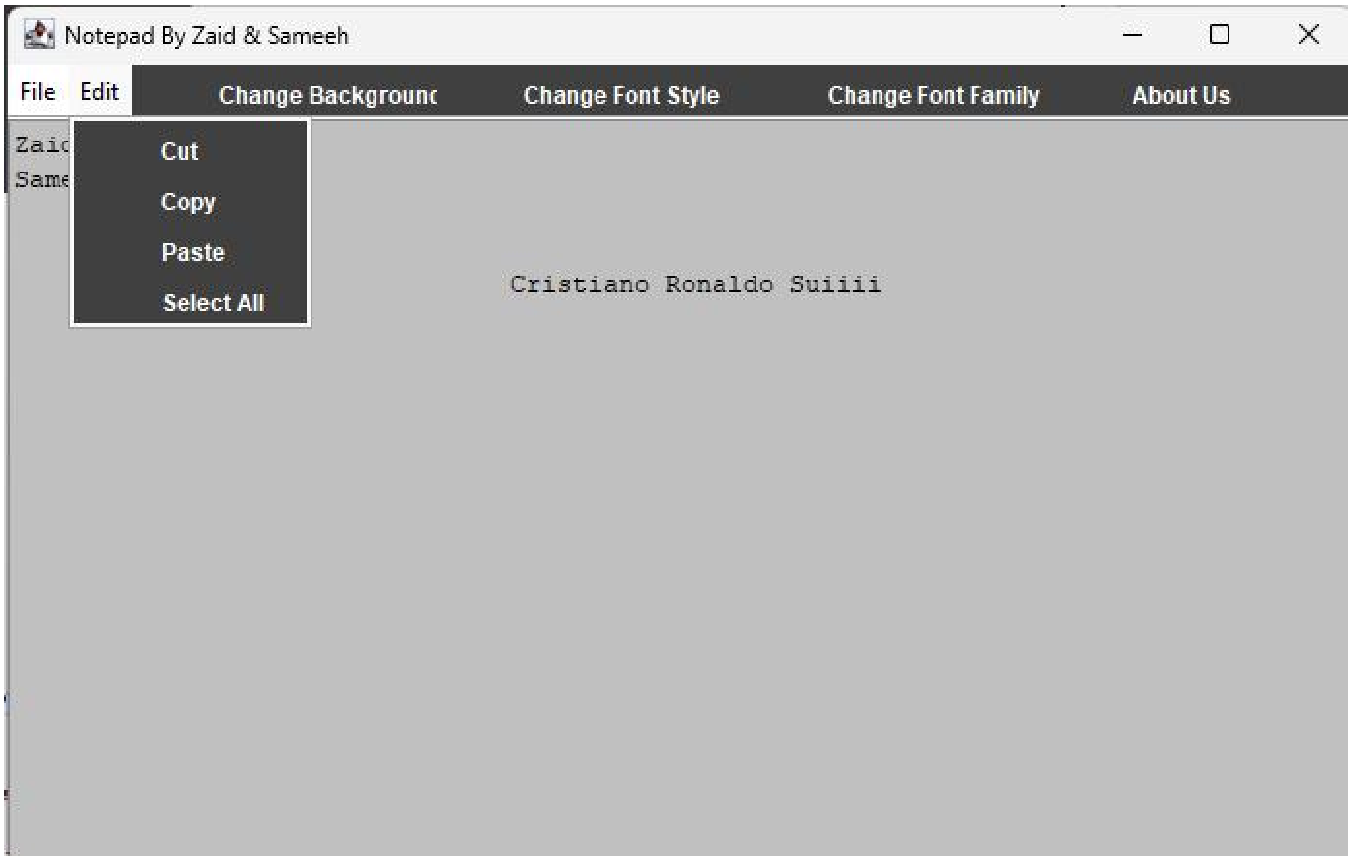
Name	Date modified	Type
Java Documents	3/7/2024 10:42 AM	File folder
Java MicroProject	3/17/2024 7:32 PM	File folder
JAVA PRACTICAL	3/7/2024 9:25 AM	File folder
MIC MicroProject	3/16/2024 6:24 AM	File folder
MICROPROJECT REPORT	3/14/2024 3:20 PM	File folder
Question Banks	3/17/2024 2:28 PM	File folder

File name: SUII

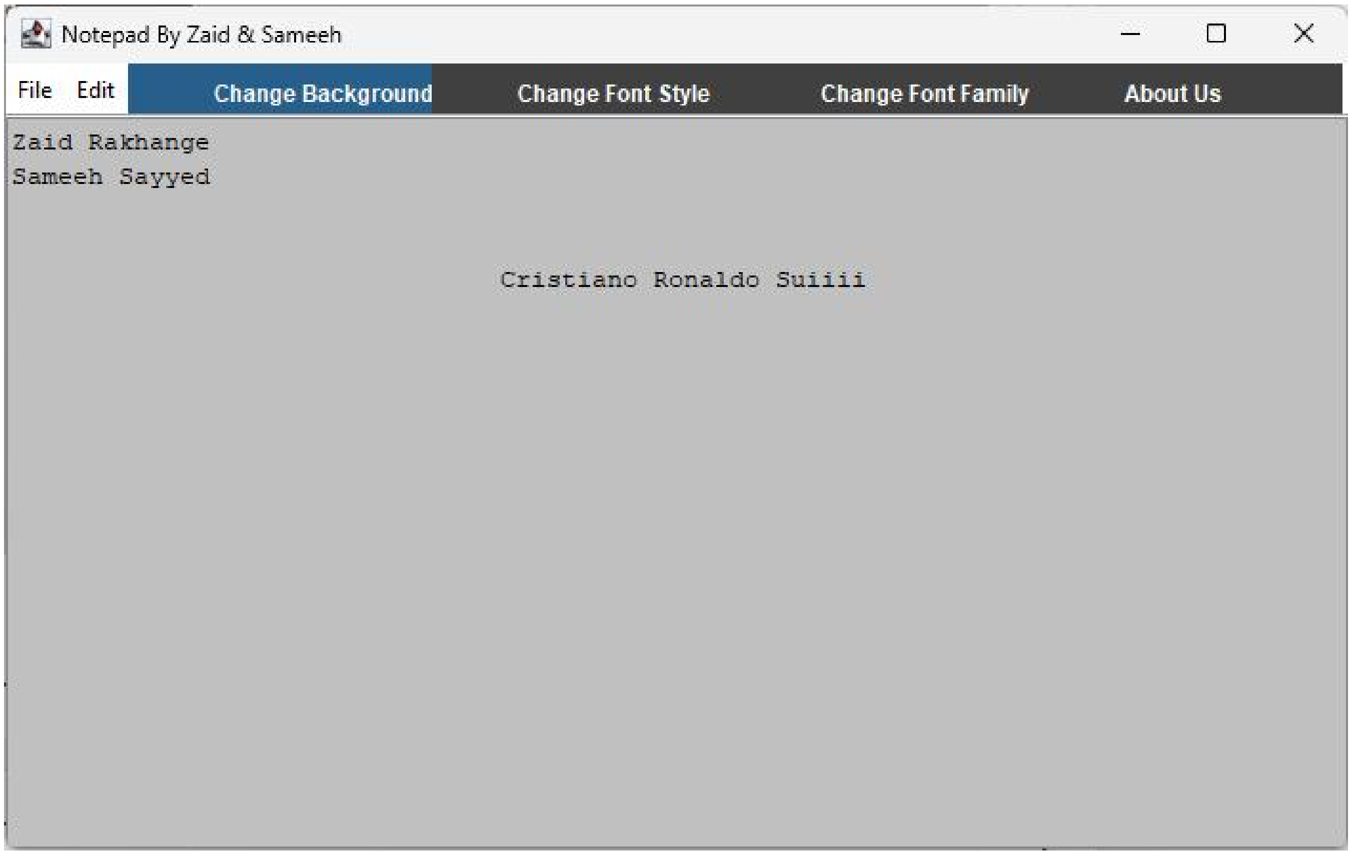
Save as type: PDF Document (*.pdf)



```
Zaid Rakhange
Sameeh Sayyed

                    Cristiano Ronaldo Suiiii
```

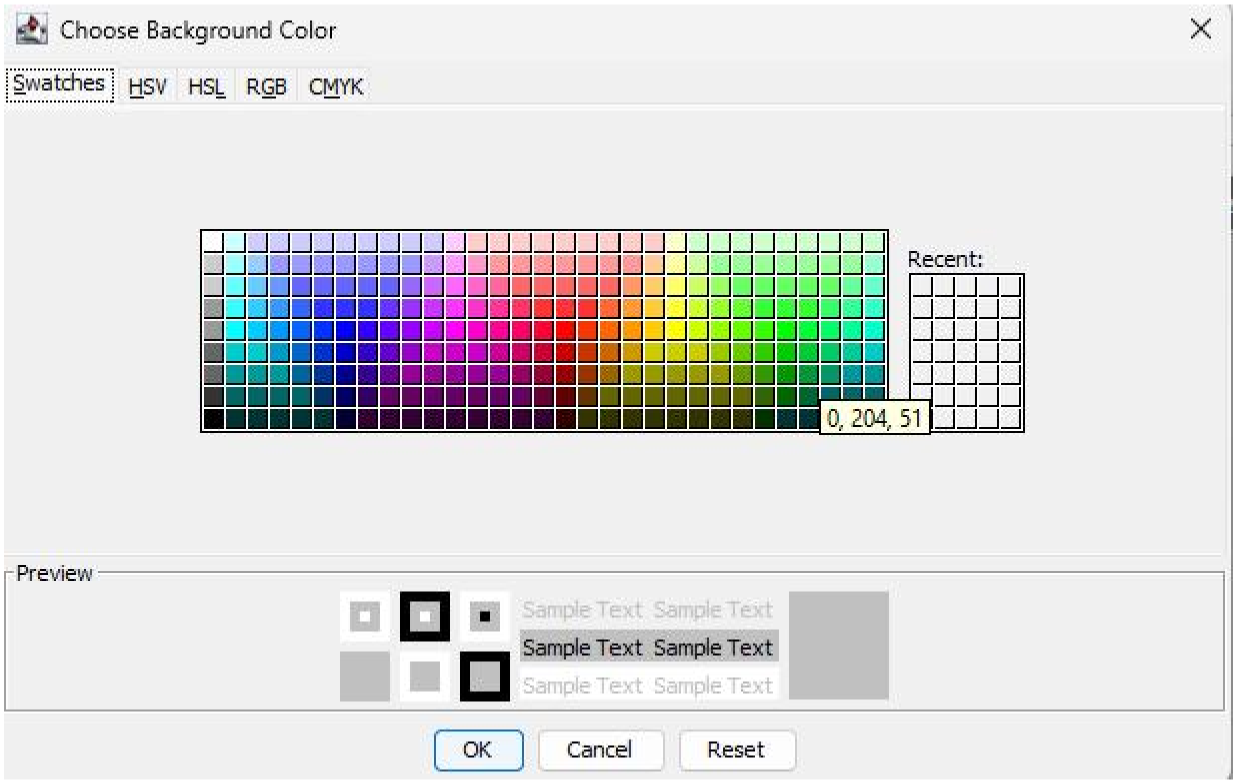


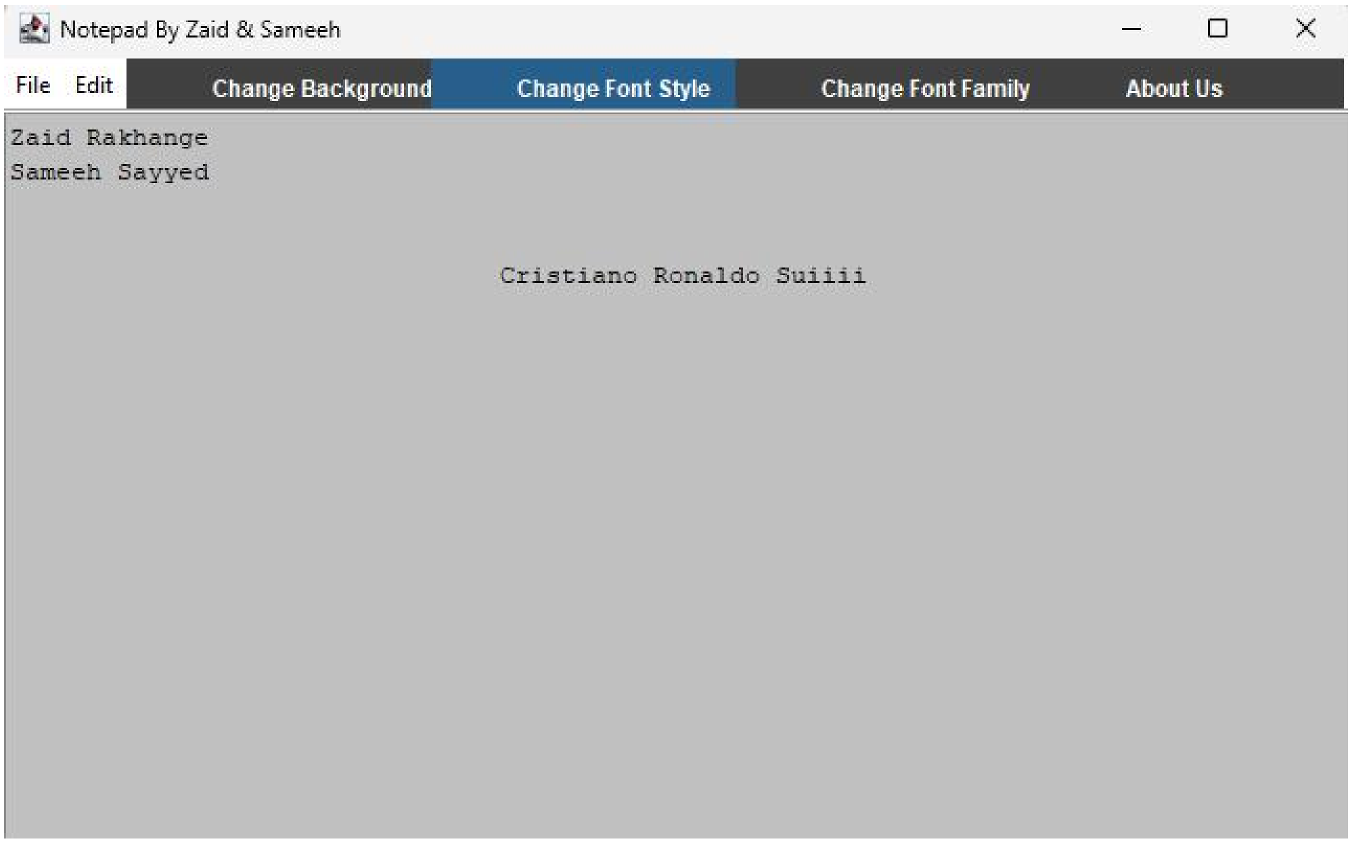
```
Zaic
Same

                    Cristiano Ronaldo Suiiii
```



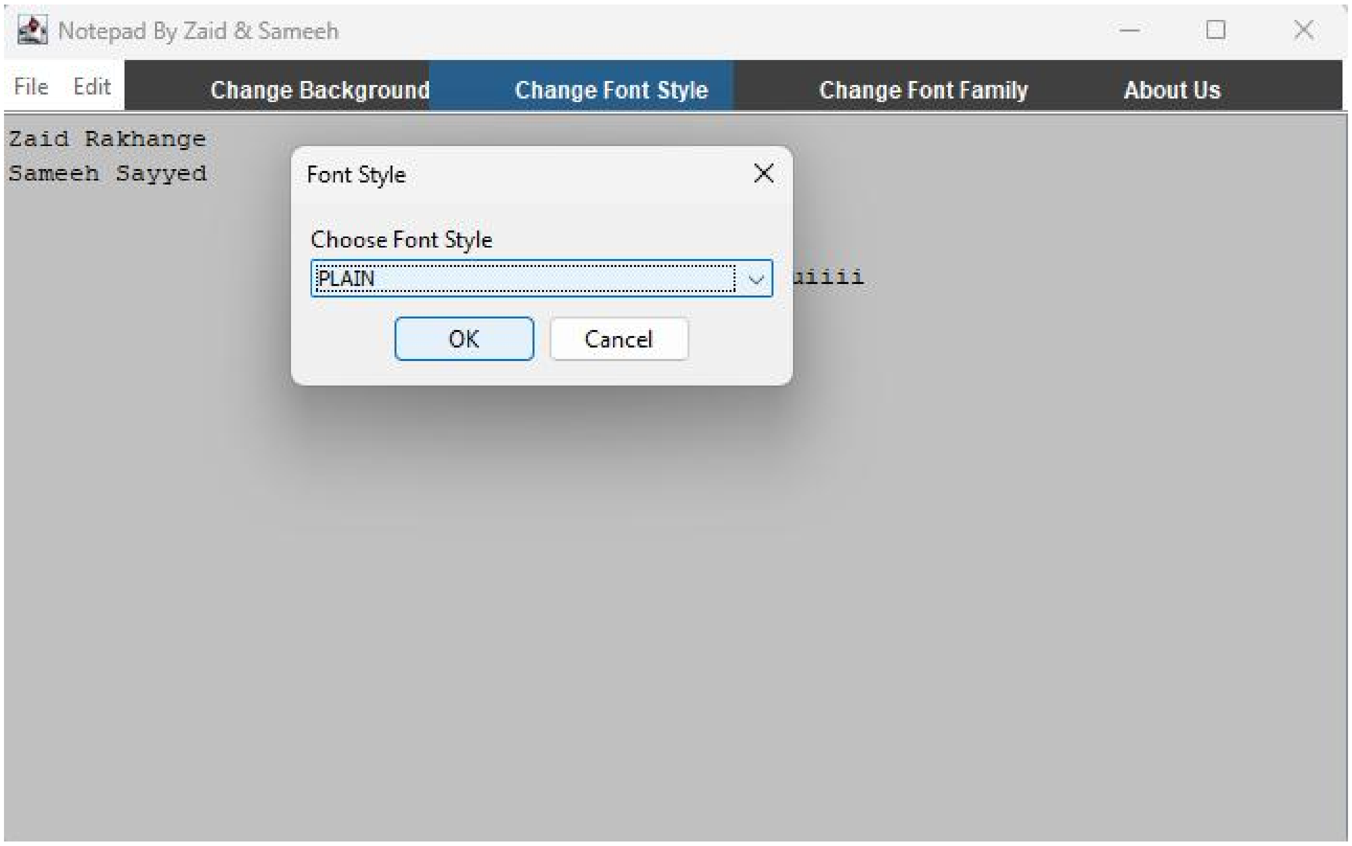
Zaid Rakhange
Sameeh Sayyed

Cristiano Ronaldo Suiiii



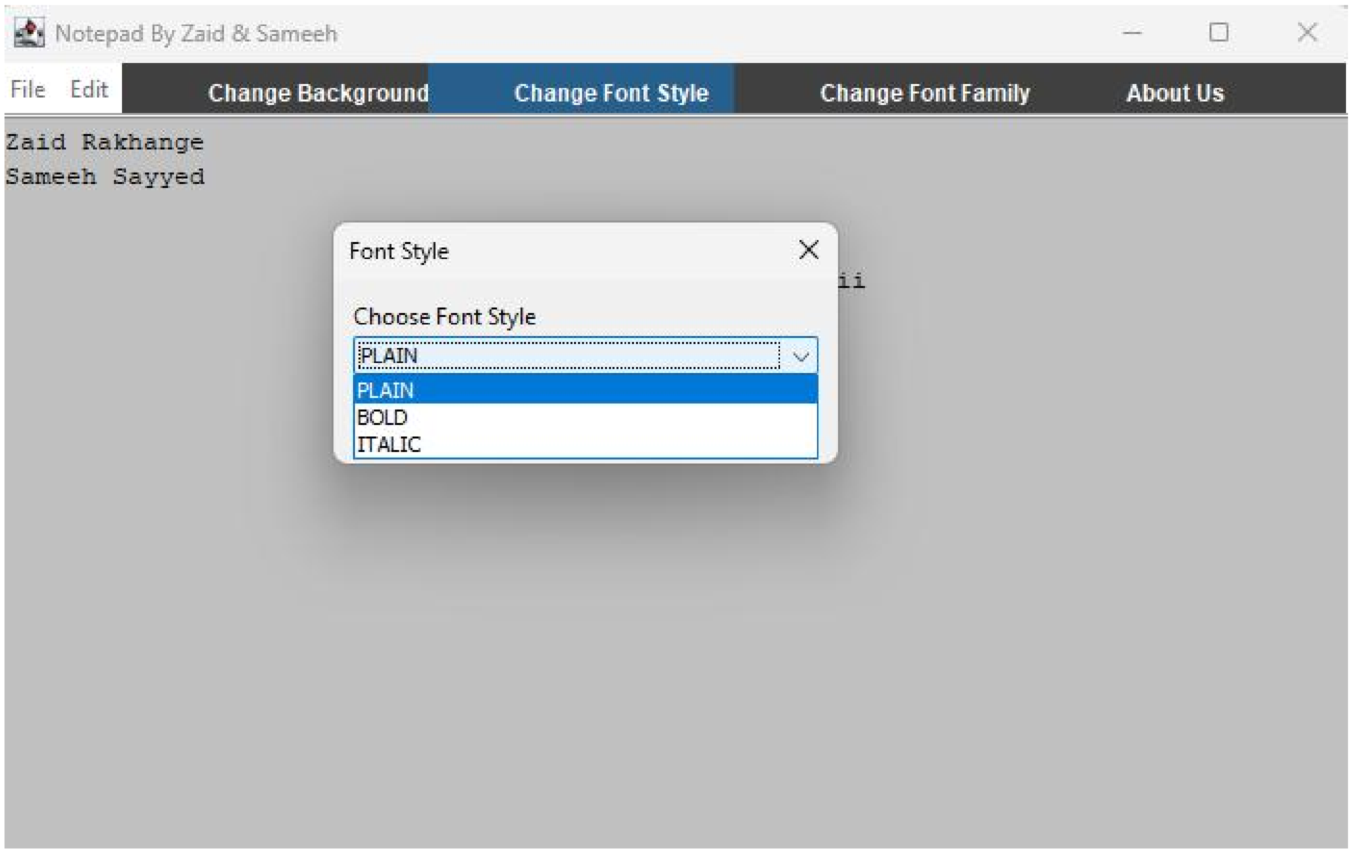


```
Zaid Rakhange
Sameeh Sayyed

                    Cristiano Ronaldo Suiiii
```

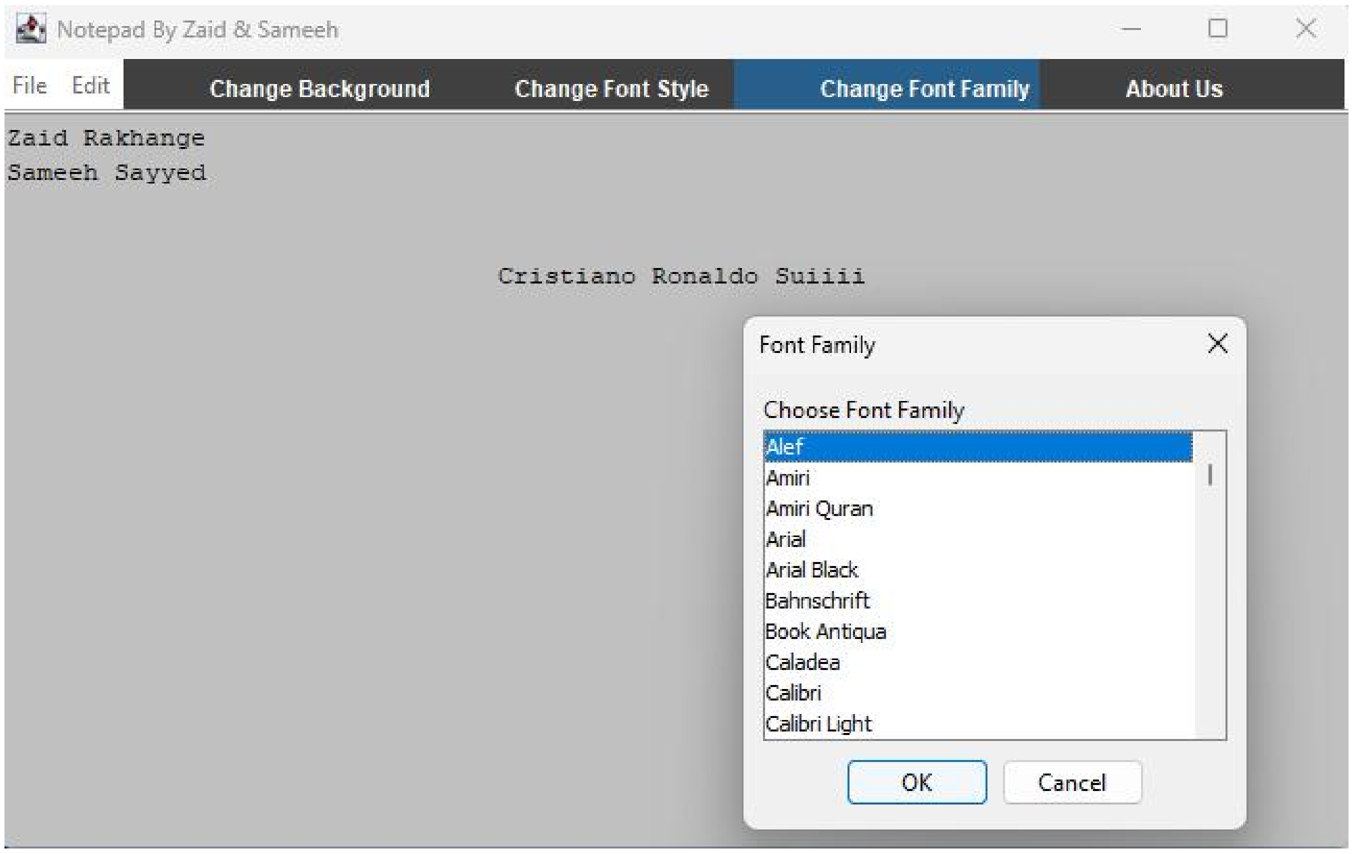


```
Zaid Rakhange
Sameeh Sayyed
```

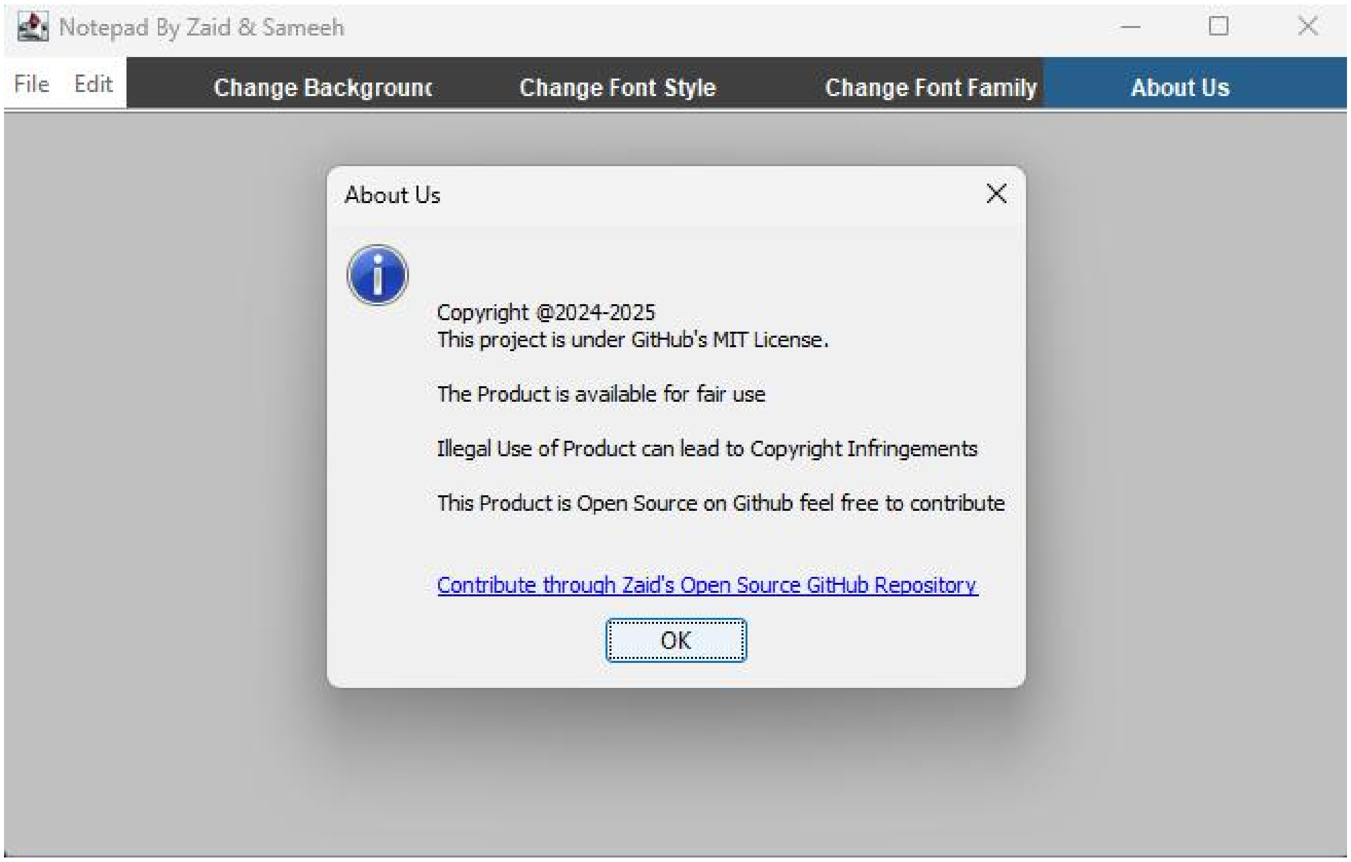
Choose Font Style: PLAIN



```
Zaid Rakhange
Sameeh Sayyed
```



```
Zaid Rakhange
Sameeh Sayyed


                        Cristiano Ronaldo Suiiii
```


Notepad By Zaid & Sameeh

File Edit Change Backgrounc Change Font Style Change Font Family About Us

About Us

Copyright @2024-2025
This project is under GitHub's MIT License.

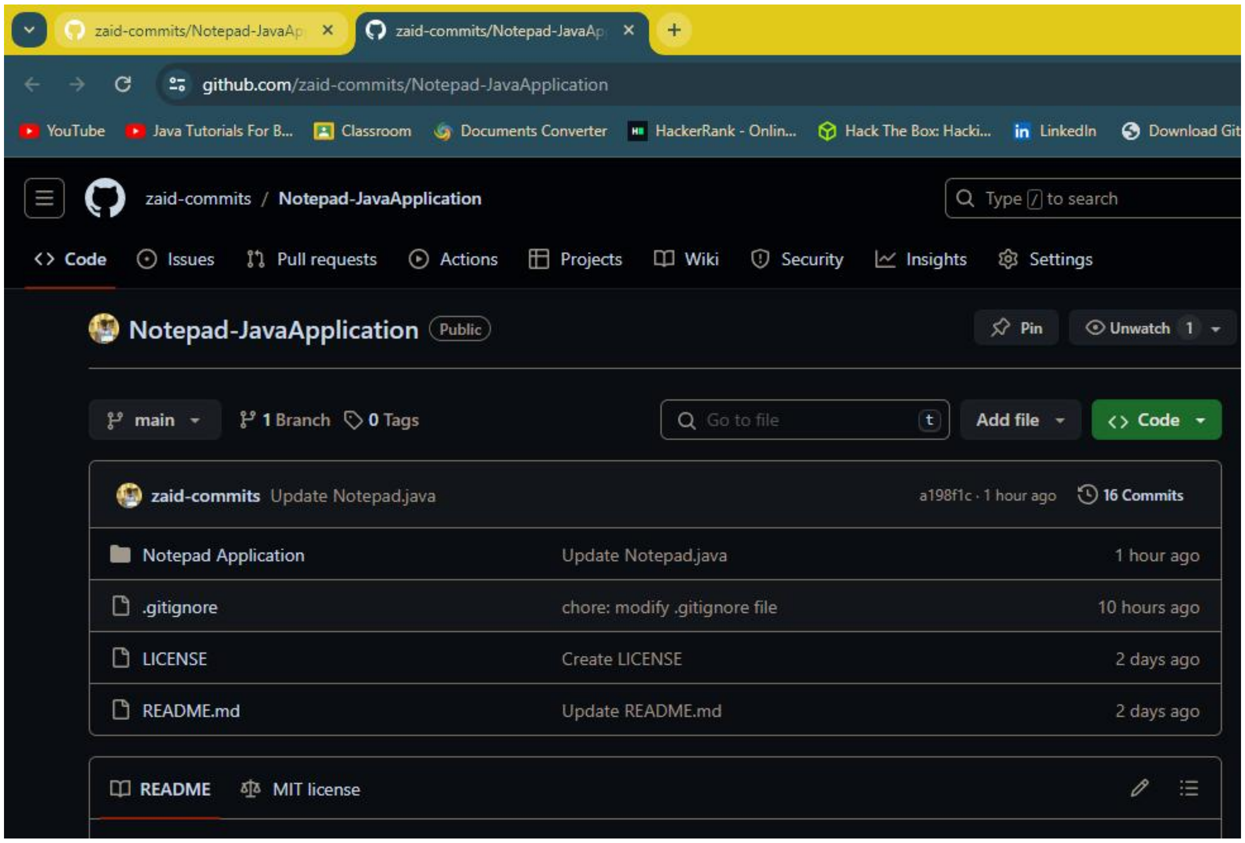
The Product is available for fair use

Illegal Use of Product can lead to Copyright Infringements

This Product is Open Source on Github feel free to contribute

Contribute through Zaid's Open Source GitHub Repository

OK


github.com/zaid-commits/Notepad-JavaApplication

zaid-commits / Notepad-JavaApplication

Notepad-JavaApplication Public

main 1 Branch 0 Tags

zaid-commits Update Notepad.java a198f1c · 1 hour ago 16 Commits

Notepad Application	Update Notepad.java	1 hour ago
.gitignore	chore: modify .gitignore file	10 hours ago
LICENSE	Create LICENSE	2 days ago
README.md	Update README.md	2 days ago

README MIT license

APPLICATIONS

1. **Educational Purposes:** This Notepad application can be used as a teaching tool to demonstrate basic GUI programming concepts in Java using Swing. It covers topics such as creating menus, handling events, and customizing the look and feel of a Swing application.

2. **Personal Use:** The Notepad application can be used as a lightweight text editor for personal use. It provides basic text editing functionalities such as cut, copy, paste, save, and open, along with options to customize the text area's appearance.

3. **Learning Java Swing:** For those learning Java Swing, this code serves as a practical example to understand how to create a simple GUI application and handle user interactions.

4. **Prototype Development:** The code can be used as a starting point for developing more complex text editor applications or integrated development environments (IDEs) with additional features and functionalities.

5. **Code Snippet Manager:** It can be modified to serve as a code snippet manager, where users can store and organize their frequently used code snippets.

6. **Simple Note-taking Application:** The application can be used as a simple note-taking tool, allowing users to quickly jot down and save notes.

CONCLUSION

This Notepad application in Java Swing showcases fundamental GUI programming concepts such as menu creation, event handling, and UI customization. It provides essential text editing features and enables users to customize the appearance of the text area. This code is not only a valuable educational resource for learning Java Swing but also a practical tool for personal use or as a foundation for more advanced applications. It highlights the versatility and user-friendliness of Java Swing for developing graphical user interfaces in Java, making it suitable for a wide range of software development projects.

REFERENCE

1.https://www.javatpoint.com/java-swing

2.https://docs.oracle.com/javase/tutorial/uiswing/

3.https://www.geeksforgeeks.org/introduction-to-java-swing/