**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Experiment Title | Date | Sign |
| **6** | **Write a program in java to sort the content of a given text file** |  |  |
| **7** | **Convert the content of a given file into the uppercase content of the same file.** |  |  |
| **8** | **Develop an analog clock using applet/java AWT components.** |  |  |
| **9** | **Develop a scientific calculator using swings.** |  |  |
| **10** | **Create an editor like MS-Word using swings** |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**EXPERIMENT 6**

**Problem statement:** **Write a program in java to sort the content of a given text file**

**Programming Code:**

import java.io.\*;

import java.util.\*;

public class sortTextFile {

public static void main(String[] args) {

List<String> lines = new ArrayList<>();

// Read lines from file

try (BufferedReader reader = new BufferedReader(new FileReader("D:\\docs ND stuff\\2nd yr coding stuff\\JAVA lab\\LAB\\input.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

lines.add(line);

}

} catch (IOException e) {

System.out.println("Error reading file: " + e.getMessage());

}

// Sort lines

Collections.sort(lines);

// Write sorted lines to new file

try (BufferedWriter writer = new BufferedWriter(new FileWriter("D:\\docs ND stuff\\2nd yr coding stuff\\JAVA lab\\LAB\\sorted\_output.txt"))) {

for (String line : lines) {

writer.write(line);

writer.newLine();

}

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

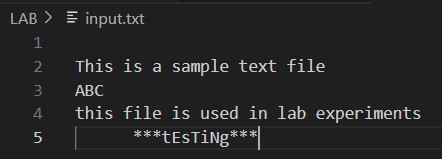
}

}

}

**OUTPUT:**

**Unsorted File**



**Sorted File**

A screenshot of a computer program

Description automatically generated

**LEARNING OUTCOMES:**

* Understanding the importance of sorting algorithms in organizing data.
* Learning how to read lines from a file and store them in a data structure (e.g., an ArrayList).
* Gaining experience in using sorting techniques (e.g., Collections.sort()) to arrange data in a specific order (e.g., alphabetically or numerically).
* Understanding the significance of proper file handling (opening, reading, writing, and closing files) in Java.
* Applying problem-solving skills to create a program that efficiently sorts and writes the content back to the file.

**EXPERIMENT 7**

**Problem statement:** **Convert the content of a given file into the uppercase content of the same file.**

**Programming Code:**

import java.io.\*;

import java.util.\*;

public class fileTextUppercase {

public static void main(String[] args) {

List<String> lines = new ArrayList<>();

// Read lines from file

try (BufferedReader reader = new BufferedReader(new FileReader("D:\\docs ND stuff\\2nd yr coding stuff\\JAVA lab\\LAB\\Lower2Upper.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

lines.add(line.toUpperCase());

}

} catch (IOException e) {

System.out.println("Error reading file: " + e.getMessage());

}

// Write uppercase lines back to the same file

try (BufferedWriter writer = new BufferedWriter(new FileWriter("D:\\docs ND stuff\\2nd yr coding stuff\\JAVA lab\\LAB\\Lower2Upper.txt"))) {

for (String line : lines) {

writer.write(line);

writer.newLine();

}

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

}

}

}

**OUTPUT:**

**Lowercase File**

A black background with white text

Description automatically generated

**Uppercase File**

A black and white text

Description automatically generated

**LEARNING OUTCOMES:**

* Understanding the usage of the toUpperCase() method in Java to transform all characters of a string to uppercase.
* Learning how to read and write file content using Java I/O classes (e.g., FileReader, BufferedReader, FileWriter, and BufferedWriter).
* Gaining practical knowledge of file manipulation by converting lowercase text to uppercase.

**EXPERIMENT 8**

**Problem statement:** **Develop an analog clock using applet/java AWT components.**

**Programming Code:**

import java.awt.\*;

import java.util.\*;

import javax.swing.Timer;

import javax.swing.\*;

public class AnalogClock extends JPanel {

private int hours = 0, minutes = 0, seconds = 0;

public AnalogClock() {

Timer timer = new Timer(1000, e -> repaint());

timer.start();

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Calendar now = Calendar.getInstance();

hours = now.get(Calendar.HOUR\_OF\_DAY);

minutes = now.get(Calendar.MINUTE);

seconds = now.get(Calendar.SECOND);

int clockRadius = Math.min(getWidth(), getHeight()) / 3;

int clockX = getWidth() / 2;

int clockY = getHeight() / 2;

// Draw the clock face

g.setColor(Color.BLACK);

g.drawOval(clockX - clockRadius, clockY - clockRadius, 2 \* clockRadius, 2 \* clockRadius);

// Draw the numbers

g.setFont(new Font("default", Font.BOLD, 14));

for (int i = 1; i <= 12; i++) {

double angle = Math.toRadians(90 - (i \* 30));

int numberX = clockX + (int) ((clockRadius - 20) \* Math.cos(angle));

int numberY = clockY - (int) ((clockRadius - 20) \* Math.sin(angle));

g.drawString(Integer.toString(i), numberX, numberY);

}

// Draw the hour hand

double hourAngle = Math.toRadians(90 - (hours \* 30 + minutes \* 0.5));

int hourHandLength = clockRadius / 2;

g.setColor(Color.BLACK);

g.drawLine(clockX, clockY,

clockX + (int) (hourHandLength \* Math.cos(hourAngle)),

clockY - (int) (hourHandLength \* Math.sin(hourAngle)));

// Draw the minute hand

double minuteAngle = Math.toRadians(90 - (minutes \* 6));

int minuteHandLength = clockRadius \* 2 / 3;

g.setColor(Color.BLUE);

g.drawLine(clockX, clockY,

clockX + (int) (minuteHandLength \* Math.cos(minuteAngle)),

clockY - (int) (minuteHandLength \* Math.sin(minuteAngle)));

// Draw the second hand

double secondAngle = Math.toRadians(90 - (seconds \* 6));

int secondHandLength = clockRadius - 20;

g.setColor(Color.RED);

g.drawLine(clockX, clockY,

clockX + (int) (secondHandLength \* Math.cos(secondAngle)),

clockY - (int) (secondHandLength \* Math.sin(secondAngle)));

}

public static void main(String[] args) {

JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(800, 800);

frame.add(new AnalogClock());

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

}

**OUTPUT:**

A clock with a red and blue hand

Description automatically generated

**LEARNING OUTCOMES:**

* Swing GUI Development: Understanding Swing components (`JFrame`, `JPanel`) and their usage in creating graphical user interfaces in Java applications.
* Event-Driven Programming: Recognizing the usage of event listeners (`Timer`) to trigger actions at regular intervals and update the clock display.
* Graphics Drawing: Introduction to basic graphics drawing operations using the `Graphics` class to create visual elements such as ovals and lines.
* Date and Time Handling: Utilizing Java's `Calendar` class to work with date and time data, extracting hours, minutes, and seconds for display on the clock.

**EXPERIMENT 9**

**Problem statement:** **Develop a scientific calculator using swings.**

**Programming Code:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Calculator extends JFrame implements ActionListener {

private JTextField displayField;

public Calculator() {

setTitle("Scientific Calculator");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setResizable(false);

displayField = new JTextField(20);

displayField.setEditable(false);

displayField.setHorizontalAlignment(JTextField.RIGHT);

displayField.setPreferredSize(new Dimension(400, 50));

JPanel buttonPanel = new JPanel(new GridLayout(5, 4));

String[] buttonLabels = {

"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"0", ".", "=", "+",

"sin", "cos", "sqrt", "C"

};

for (String label : buttonLabels) {

JButton button = new JButton(label);

button.addActionListener(this);

buttonPanel.add(button);

}

getContentPane().add(displayField, BorderLayout.NORTH);

getContentPane().add(buttonPanel, BorderLayout.CENTER);

pack();

setSize(400, 500);

setLocationRelativeTo(null);

}

private String operator = "";

private double firstOperand = 0.0;

private double secondOperand = 0.0;

@Override

public void actionPerformed(ActionEvent e) {

String command = e.getActionCommand();

switch (command) {

case "+":

case "-":

case "\*":

case "/":

operator = command;

firstOperand = Double.valueOf(displayField.getText());

displayField.setText("");

break;

case "=":

secondOperand = Double.valueOf(displayField.getText());

switch (operator) {

case "+":

displayField.setText(String.valueOf(firstOperand + secondOperand));

break;

case "-":

displayField.setText(String.valueOf(firstOperand - secondOperand));

break;

case "\*":

displayField.setText(String.valueOf(firstOperand \* secondOperand));

break;

case "/":

if (secondOperand != 0) {

displayField.setText(String.valueOf(firstOperand / secondOperand));

} else {

displayField.setText("Error");

}

break;

}

operator = "";

break;

case "sin":

displayField.setText(String.valueOf(Math.sin(Double.valueOf(displayField.getText()))));

break;

case "cos":

displayField.setText(String.valueOf(Math.cos(Double.valueOf(displayField.getText()))));

break;

case "sqrt":

displayField.setText(String.valueOf(Math.sqrt(Double.valueOf(displayField.getText()))));

break;

case "C":

operator = "";

firstOperand = 0.0;

secondOperand = 0.0;

displayField.setText("");

break;

default:

displayField.setText(displayField.getText() + command);

break;

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

Calculator calculator = new Calculator();

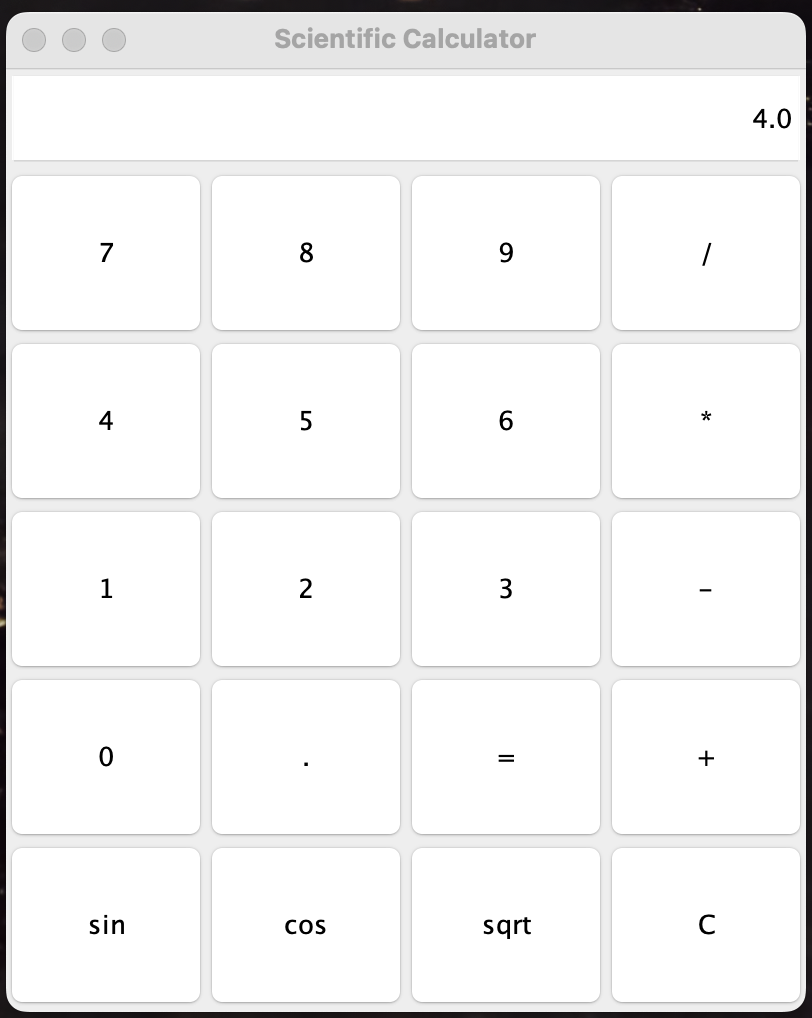
calculator.setVisible(true);

});

}

}

**OUTPUT:**

****

**LEARNING OUTCOMES:**

* Understanding Event-Driven Java Programming: Passive comprehension of event-driven programming in Java Swing, enabling effective handling of user interactions like button clicks.
* Proficiency in GUI Development: Competence in designing and building GUIs using Java Swing components such as JFrame, JTextField, and JButton, fostering creation of visually appealing and functional interfaces.
* Data Handling and Manipulation Mastery: Ability to handle and manipulate data in Java for accurate calculations and dynamic results.
* Effective Error Handling Implementation: Application of techniques for error handling and input validation to ensure stability and reliability.
* UX Design Principles Understanding: Insight into UX design principles for creating user-friendly applications.

**EXPERIMENT 10**

**Problem statement:** **Create an editor like MS-Word using swings**

**Programming Code:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

public class TextEditor extends JFrame implements ActionListener {

private JTextArea textArea;

private JFileChooser fileChooser;

public TextEditor() {

setTitle("Text Editor");

setSize(800, 600);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

textArea = new JTextArea();

JScrollPane scrollPane = new JScrollPane(textArea);

add(scrollPane, BorderLayout.CENTER);

JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");

JMenuItem newMenuItem = new JMenuItem("New");

JMenuItem openMenuItem = new JMenuItem("Open");

JMenuItem saveMenuItem = new JMenuItem("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");

newMenuItem.addActionListener(this);

openMenuItem.addActionListener(this);

saveMenuItem.addActionListener(this);

exitMenuItem.addActionListener(this);

fileMenu.add(newMenuItem);

fileMenu.add(openMenuItem);

fileMenu.add(saveMenuItem);

fileMenu.add(exitMenuItem);

menuBar.add(fileMenu);

setJMenuBar(menuBar);

fileChooser = new JFileChooser();

setVisible(true);

}

public void actionPerformed(ActionEvent e) {

if (e.getActionCommand().equals("New")) {

textArea.setText("");

}

else if (e.getActionCommand().equals("Open")) {

int returnVal = fileChooser.showOpenDialog(this);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

try {

BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

StringBuilder content = new StringBuilder();

while ((line = reader.readLine()) != null) {

content.append(line).append("\n");

}

reader.close();

textArea.setText(content.toString());

} catch (IOException ex) {

ex.printStackTrace();

}

}

} else if (e.getActionCommand().equals("Save")) {

int returnVal = fileChooser.showSaveDialog(this);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

try {

BufferedWriter writer = new BufferedWriter(new FileWriter(file));

writer.write(textArea.getText());

writer.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

} else if (e.getActionCommand().equals("Exit")) {

System.exit(0);

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

new TextEditor();

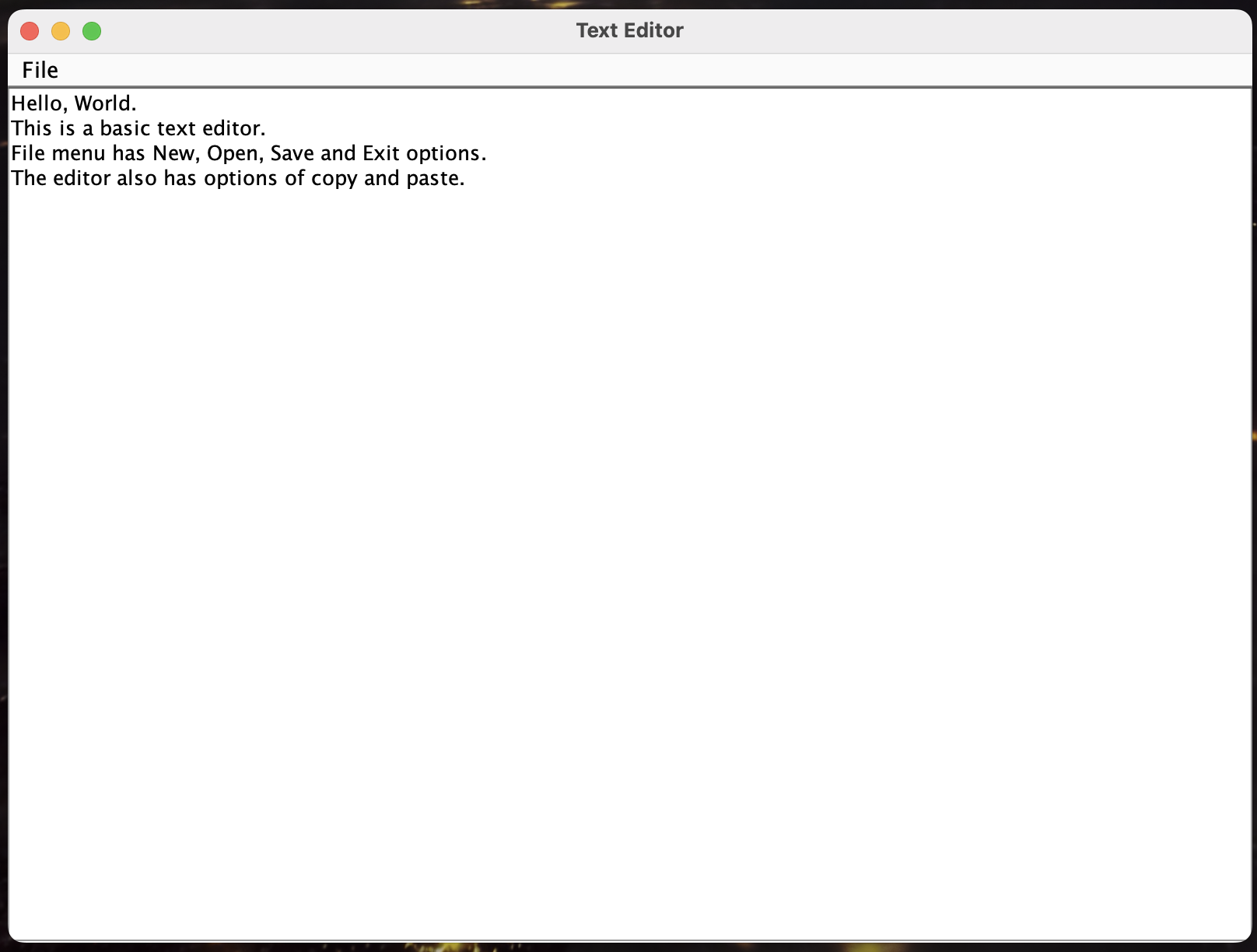
}

});

}

}

**OUTPUT:**

****

**LEARNING OUTCOMES:**

* Swing Component Understanding Achieved: Comprehension of Swing components like JTextArea, JMenuBar, JMenu, and JMenuItem for building interactive GUIs in Java applications is attained.
* File Handling Proficiency Acquired: Proficiency in file operations such as reading from and writing to files using BufferedReader, FileReader, BufferedWriter, and FileWriter classes is achieved.
* Event Handling Techniques Demonstrated: The code showcases event handling techniques using ActionListener interfaces to respond to user interactions with menu items, providing insights into managing event-driven programming in Java.
* UI Design Principles Understood: Designing a basic text editor interface imparts understanding of UI design principles like layout management and menu creation to enhance usability and functionality.