PROGRAM 10

Aim:

To write a Java program that reads a file name from the user and displays information about whether the file exists, is readable, is writable, its type (file or directory), and its length in bytes

Steps:

1. Create a Scanner object to read user input.
2. Prompt the user to enter a file name.
3. Create a File object using the file name.
4. Check if the file exists.
5. If it exists, check if it is readable, writable, a directory, or a regular file.
6. Display the file length if it is a file.
7. If the file does not exist, print a message.

Program:

```
import java.io.File;
import java.util.Scanner;

public class FileInfo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the file name (with path if necessary): ");
        String fileName = scanner.nextLine();

        File file = new File(fileName);

        if (file.exists()) {
            System.out.println("File exists: Yes");

            if (file.canRead()) {
                System.out.println("File is readable: Yes");
            } else {
                System.out.println("File is readable: No");
            }

            if (file.canWrite()) {
                System.out.println("File is writable: Yes");
            } else {
                System.out.println("File is writable: No");
            }

            if (file.isDirectory()) {
                System.out.println("It is a directory.");
            } else {
                System.out.println("It is a regular file.");
            }

            if (file.isFile()) {
                System.out.println("File length: " + file.length() + " bytes");
            }

        } else {
            System.out.println("File exists: No");
        }

        scanner.close();
    }
}
```

Output:

```
Enter the file name (with path if necessary): test.txt
File exists: Yes
File is readable: Yes
File is writable: Yes
It is a regular file.
File length: 128 bytes
```

PROGRAM 11

Aim:

To write a Java program that allows users to enter text and modify its font, size, and style (bold/italic) dynamically using frames and controls.

Steps:

1. Create a JFrame and set its layout.
2. Add a JTextArea for user input and place it inside a JScrollPane.
3. Create JComboBox for font family and size selection.
4. Add JCheckBox for bold and italic style options.
5. Implement an ActionListener to update font settings dynamically.

Program

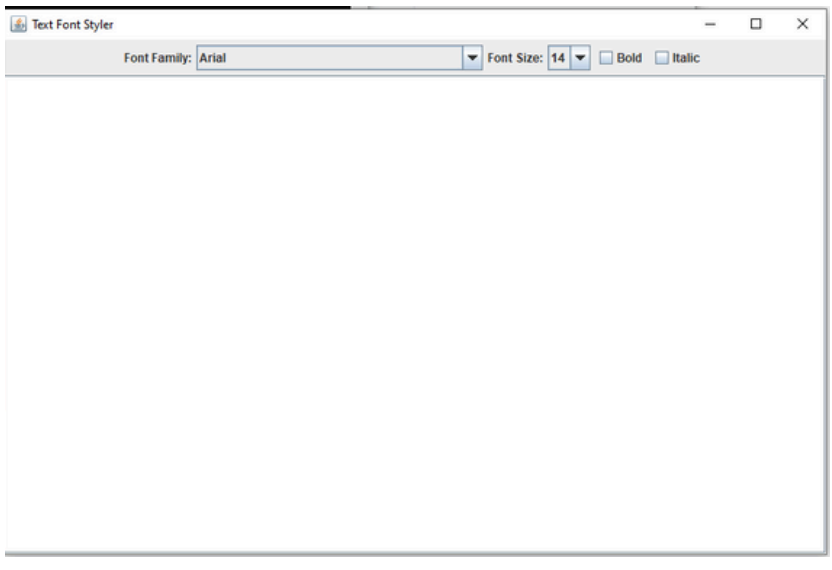
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class FontStyler extends JFrame {
    private JTextArea textArea;
    private JComboBox<String> fontFamilyComboBox;
    private JComboBox<Integer> fontSizeComboBox;
    private JCheckBox boldCheckBox;
    private JCheckBox italicCheckBox;

    public FontStyler() {
        setTitle("Text Font Styler");
        setLayout(new BorderLayout());

        textArea = new JTextArea(10, 30);
        textArea.setFont(new Font("Arial", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(textArea);
        add(scrollPane, BorderLayout.CENTER);

        JPanel controlPanel = new JPanel();
        controlPanel.setLayout(new FlowLayout());

        String[] fontFamilies =
GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyName
s();
        fontFamilyComboBox = new JComboBox<>(fontFamilies);
        fontFamilyComboBox.setSelectedItem("Arial");
        controlPanel.add(new JLabel("Font Family:"));
        controlPanel.add(fontFamilyComboBox);

        Integer[] fontSizes = {10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30};
        fontSizeComboBox = new JComboBox<>(fontSizes);
        fontSizeComboBox.setSelectedItem(14);
        controlPanel.add(new JLabel("Font Size:"));
        controlPanel.add(fontSizeComboBox);
```

```
boldCheckBox = new JCheckBox("Bold");
controlPanel.add(boldCheckBox);

italicCheckBox = new JCheckBox("Italic");
controlPanel.add(italicCheckBox);

add(controlPanel, BorderLayout.NORTH);

ActionListener updateFontActionListener = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String fontFamily = (String) fontFamilyComboBox.getSelectedItem();
        int fontSize = (int) fontSizeComboBox.getSelectedItem();
        int style = Font.PLAIN;
        if (boldCheckBox.isSelected()) style |= Font.BOLD;
        if (italicCheckBox.isSelected()) style |= Font.ITALIC;
        textArea.setFont(new Font(fontFamily, style, fontSize));
    }
};

fontFamilyComboBox.addActionListener(updateFontActionListener);
fontSizeComboBox.addActionListener(updateFontActionListener);
boldCheckBox.addActionListener(updateFontActionListener);
italicCheckBox.addActionListener(updateFontActionListener);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setSize(500, 400);
setVisible(true);
}

public static void main(String[] args) {
SwingUtilities.invokeLater(new Runnable() {
@Override
public void run() {
new FontStyler();
}
});
}
}
```

Output:

C:\Users\ADMIN\Desktop\java>javac FontStyler.java

C:\Users\ADMIN\Desktop\java>java FontStyler



Program 12

Aim:

To create a Java program that handles all mouse events and displays the event name at the center of the window when a mouse event occurs, using adapter classes.

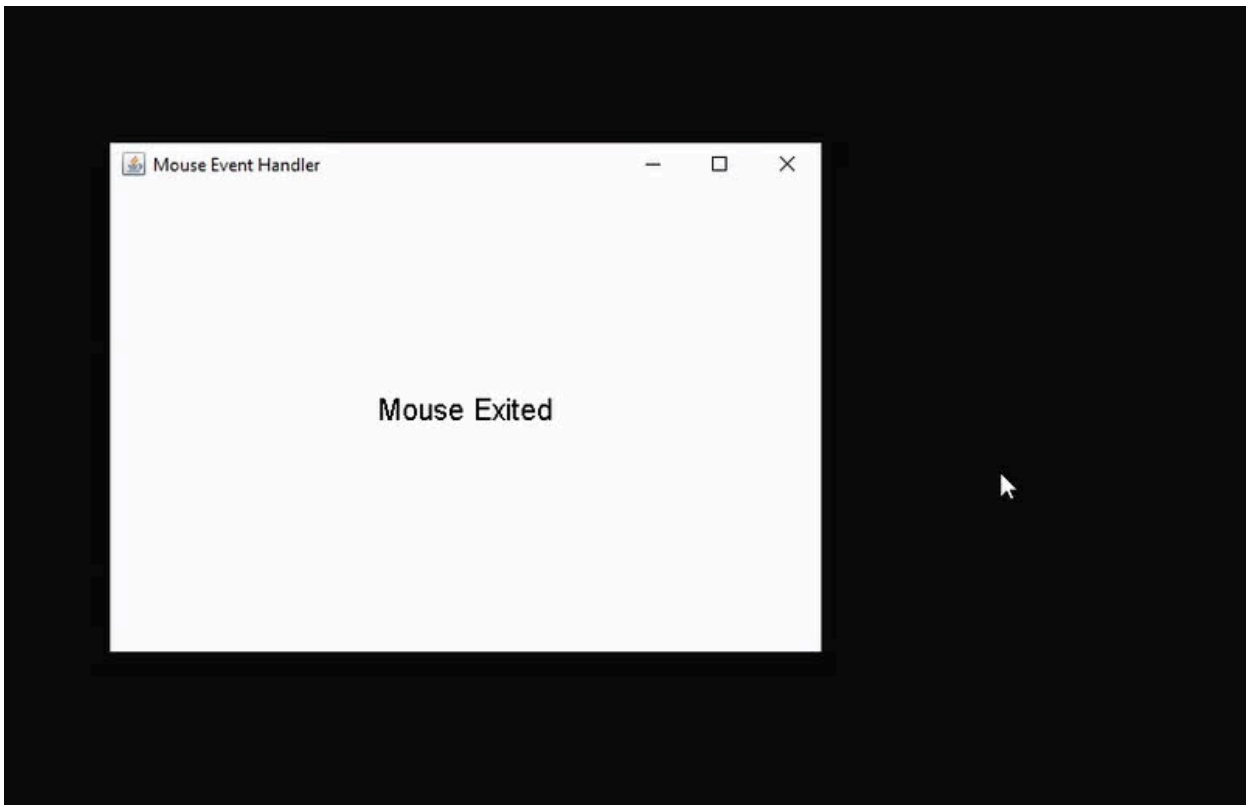
Steps:

1. Create a Frame window and set its size.
2. Use MouseAdapter to handle mouse events such as Clicked, Pressed, Released, Entered, and Exited.
3. Use MouseMotionAdapter to handle MouseMoved and MouseDragged events.
4. Override the paint method to display the event name at the center of the window.
5. Add a WindowAdapter to close the application when the window is closed

```
import java.awt.*;
import java.awt.event.*;

public class MouseEventExample extends Frame {

// Variable to hold the event name
private String eventName = "No Event";

public MouseEventExample() {
   // Set the title of the window
   setTitle("Mouse Event Handler");

   // Set the layout to null for custom positioning
   setLayout(null);

   // Set the size of the window
   setSize(400, 300);

   // Add mouse event listeners using MouseAdapter
   addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
         eventName = "Mouse Clicked";
         repaint(); // Requesting a repaint when an event occurs
   }

@Override
public void mousePressed(MouseEvent e) {
   eventName = "Mouse Pressed";
   repaint();
}

@Override
public void mouseReleased(MouseEvent e) {
   eventName = "Mouse Released";
   repaint();
}

@Override
public void mouseEntered(MouseEvent e) {
   eventName = "Mouse Entered";
   repaint();
}
```

```
    @Override
    public void mouseExited(MouseEvent e) {
        eventName = "Mouse Exited";
        repaint();
    }
});

// Add mouse motion event listeners using MouseAdapter
addMouseMotionListener(new MouseAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        eventName = "Mouse Dragged";
        repaint();
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        eventName = "Mouse Moved";
        repaint();
    }
});

    // Set the window to be visible
    setVisible(true);

    // Set the close operation
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            System.exit(0); // Exit the application when the window is closed
        }
    });
}

// Override the paint method to draw the event name at the center
@Override
public void paint(Graphics g) {
    // Get the width and height of the window
    Dimension size = getSize();
    int width = size.width;
    int height = size.height;

    // Set the font and draw the event name at the center of the window
    g.setFont(new Font("Arial", Font.PLAIN, 20));
    FontMetrics metrics = g.getFontMetrics();
    int x = (width - metrics.stringWidth(eventName)) / 2; // Center X position
    int y = (height + metrics.getHeight()) / 2; // Center Y position
    g.drawString(eventName, x, y);
  }

  public static void main(String[] args) {
    // Create an instance of MouseEventExample to show the window
    new MouseEventExample();
  }
}
```

Output:



//Skip this 👆

Program 13:

Aim:

To create a simple calculator in Java using JFrame and JButton, which performs basic arithmetic operations (+, -, *, /).

Steps:

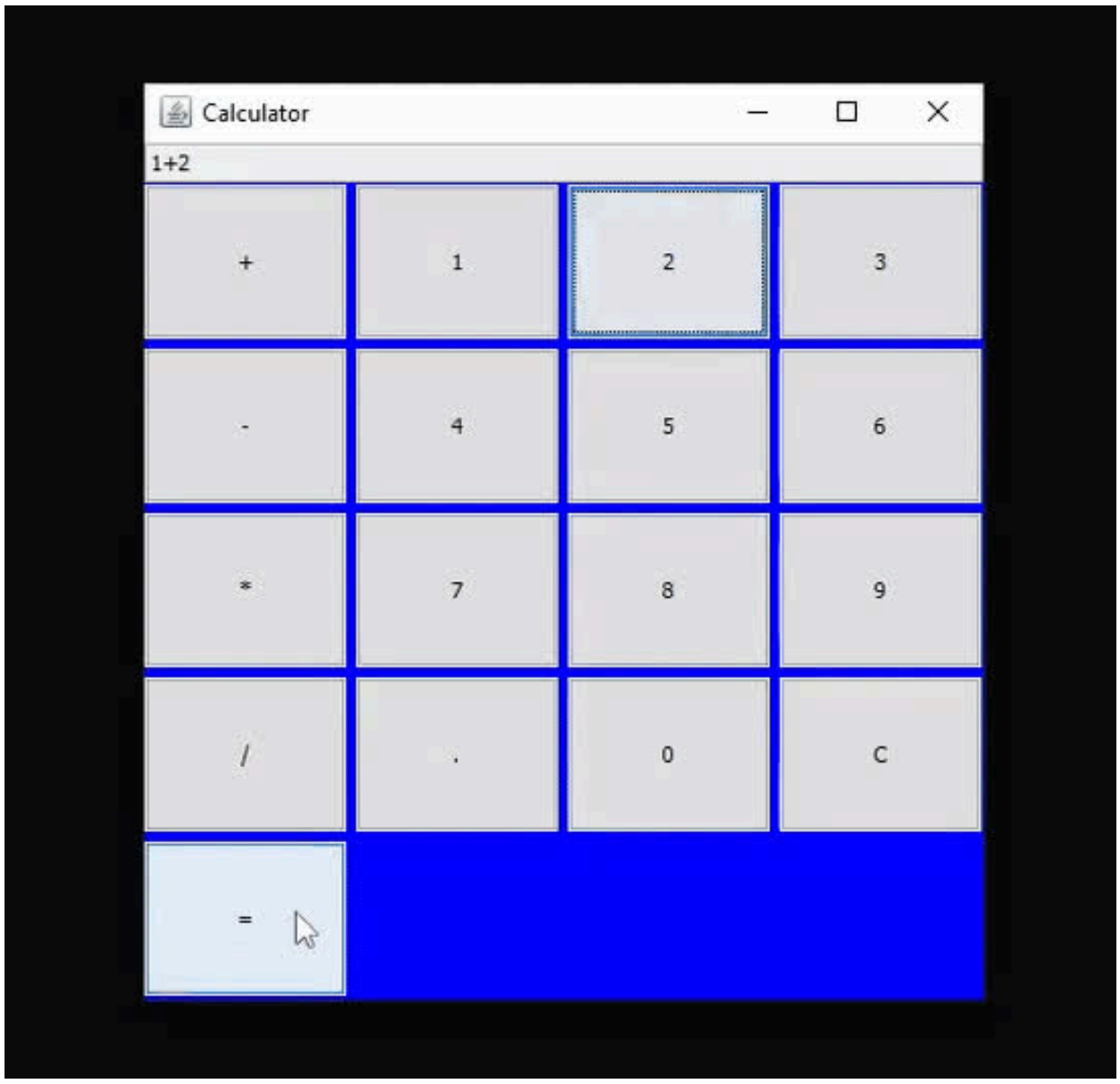
1. Create a JFrame window and a JTextField to display input and results.
2. Add buttons for digits (0-9), arithmetic operations (+, -, *, /), clear (C), and equals (=).
3. Implement ActionListener to handle button clicks and update the display.
4. Store operands and operators in variables and perform calculations on = press.
5. Display the result in the text field and handle clearing using the "C" button.

Program

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;

class calculator extends JFrame implements ActionListener {
   static JFrame f;
   static JTextField l;
   String s0, s1, s2;

   calculator() {
      s0 = s1 = s2 = "";
   }

   public static void main(String args[]) {
      f = new JFrame("Calculator");

      try {
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) {
   System.err.println(e.getMessage());
}

calculator c = new calculator();
l = new JTextField(16);
l.setEditable(false);

JButton b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, ba, bs, bd, bm, be, beq, beq1;
b0 = new JButton("0");
b1 = new JButton("1");
b2 = new JButton("2");
b3 = new JButton("3");
b4 = new JButton("4");
b5 = new JButton("5");
b6 = new JButton("6");
b7 = new JButton("7");
b8 = new JButton("8");
b9 = new JButton("9");
      beq1 = new JButton("=");
      ba = new JButton("+");
      bs = new JButton("-");
      bd = new JButton("/");
      bm = new JButton("*");
      beq = new JButton("C");
      be = new JButton(".");

      JPanel p = new JPanel();
      p.setLayout(new GridLayout(5, 4, 5, 5)); // Added layout for better arrangement

      JButton[] buttons = { ba, b1, b2, b3, bs, b4, b5, b6, bm, b7, b8, b9, bd, be, b0,
beq, beq1 };
      for (JButton button : buttons) {
         button.addActionListener(c);
         p.add(button);
      }

      p.setBackground(Color.blue);
      f.add(l, BorderLayout.NORTH);
      f.add(p);
```

```
    f.setSize(250, 300);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    String s = e.getActionCommand();

    if ((s.charAt(0) >= '0' && s.charAt(0) <= '9') || s.charAt(0) == '.') {
        if (!s1.equals(""))
            s2 += s;
        else
            s0 += s;

        l.setText(s0 + s1 + s2);
    } else if (s.equals("C")) {
        s0 = s1 = s2 = "";
        l.setText("");
    } else if (s.equals("=")) {
        if (!s0.isEmpty() && !s2.isEmpty()) {
            double te = 0;
            double num1 = Double.parseDouble(s0);
            double num2 = Double.parseDouble(s2);

            switch (s1) {
                case "+": te = num1 + num2; break;
                case "-": te = num1 - num2; break;
                case "*": te = num1 * num2; break;
                case "/": te = (num2 != 0) ? num1 / num2 : 0; break; // Prevent division by
zero
            }

            l.setText(s0 + s1 + s2 + "=" + te);
            s0 = Double.toString(te);
            s1 = s2 = "";
        }
    } else {
        if (s1.isEmpty() || s2.isEmpty())
            s1 = s;
        else {
            double te = 0;
            double num1 = Double.parseDouble(s0);
            double num2 = Double.parseDouble(s2);

            switch (s1) {
                case "+": te = num1 + num2; break;
                case "-": te = num1 - num2; break;
                case "*": te = num1 * num2; break;
                case "/": te = (num2 != 0) ? num1 / num2 : 0; break;
            }

            s0 = Double.toString(te);
            s1 = s;
            s2 = "";
        }

        l.setText(s0 + s1 + s2);
    }
  }
}
```

Output:



Program 14:

Aim:

To develop a Traffic Light Simulator using Java Swing, where selecting Red, Yellow, or Green displays the corresponding messages "Stop," "Ready," and "Go" with appropriate colors.

Steps:

1. Create a JFrame (TrafficLightSimulator) and set properties like title, size, and layout.
2. Create a JLabel (messageLabel) to display messages.
3. Create JRadioButtons (redButton, yellowButton, greenButton) for Red, Yellow, and Green lights.
4. Group the buttons using ButtonGroup to allow only one selection at a time.
5. Add action listeners to each button:
6. redButton → Display "Stop" in Red.
7. yellowButton → Display "Ready" in Yellow.
8. greenButton → Display "Go" in Green.
9. Set the default state with an empty message and black text.
10. Add all components to the JFrame and set it visible.

Program

```
import javax.swing.*;
import java.awt.*;

public class TrafficLightSimulator extends JFrame {
   // Label to display the message (Stop, Ready, Go)
   private JLabel messageLabel;

   public TrafficLightSimulator() {
      // Set up the window (JFrame)
      setTitle("Traffic Light Simulator");
      setSize(300, 200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new FlowLayout());

      // Create the label for displaying the message
      messageLabel = new JLabel();
      messageLabel.setFont(new Font("Arial", Font.PLAIN, 20));
      add(messageLabel); // Add label to the frame

      // Create three radio buttons for the traffic lights
      JRadioButton redButton = new JRadioButton("Red", true);   // Default selected
      JRadioButton yellowButton = new JRadioButton("Yellow");
      JRadioButton greenButton = new JRadioButton("Green");

      // Group the buttons so only one can be selected at a time
      ButtonGroup buttonGroup = new ButtonGroup();
      buttonGroup.add(redButton);
      buttonGroup.add(yellowButton);
      buttonGroup.add(greenButton);

      // Add radio buttons to the frame
      add(redButton);
      add(yellowButton);
      add(greenButton);

      // Define actions when each button is selected
      redButton.addActionListener(e -> updateMessage("Stop", Color.RED));
      yellowButton.addActionListener(e -> updateMessage("Ready", Color.YELLOW));
      greenButton.addActionListener(e -> updateMessage("Go", Color.GREEN));

      // Set initial state (Red light selected)
      updateMessage("Stop", Color.RED);

      // Make the window visible
      setVisible(true);
   }

   // Method to update the message and its color
   private void updateMessage(String text, Color color) {
      messageLabel.setText(text);
      messageLabel.setForeground(color);
   }

   public static void main(String[] args) {
      // Ensure the GUI updates properly in the Swing event thread
      SwingUtilities.invokeLater(TrafficLightSimulator::new);
   }
}
```

Output: