1. java Create an application that has a text area with scroll bars. Enable horizontal scrolling as needed. Always provide vertical scroll bars. Create two buttons. If the first button is pressed, append a number to the text area. Start the number at 1, and increase it after the button is pressed. If the second button is pressed, append a new line to the text area. Put a nice border around the scroll pane

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

public class ScrollableTextAreaApp extends JFrame {
    private JTextArea textArea;
    private JScrollPane scrollPane;
    private JButton numberButton;
    private JButton newlineButton;
    private int number = 1;

    public ScrollableTextAreaApp() {
        setTitle("Scrollable TextArea App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setPreferredSize(new Dimension(400, 300));
        initComponents();
        pack();
        setLocationRelativeTo(null);
        setVisible(true);
    }

    private void initComponents() {
        textArea = new JTextArea();
        scrollPane = new JScrollPane(textArea);
        scrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        numberButton = new JButton("Append Number");
        numberButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textArea.append(number + " ");
                number++;
            }
        });

        newlineButton = new JButton("Append New Line");
        newlineButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                textArea.append(System.lineSeparator());
```

```
        }
      });

      JPanel buttonPanel = new JPanel();
      buttonPanel.add(numberButton);
      buttonPanel.add(newlineButton);

      setLayout(new BorderLayout());
      add(scrollPane, BorderLayout.CENTER);
      add(buttonPanel, BorderLayout.SOUTH);
   }

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

2. Create an application that models a simple sales terminal. You should be able to sell three kinds of items. Have one button for each item, and attach a picture of the item to the button. Each button should have three labels associated with it. These labels will display the price of the item, the number of that item sold in the current transaction, and a subtotal for that item. Each time a button is pressed, increase the count of that item in the current sale by one and update the subtotal. A separate tenth label should show the total cost of the current sale. An “EndSale” menu item ends the current sale and resets the totals to zero

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

public class SalesTerminalApp extends JFrame {
   private JLabel totalLabel;
   private JButton item1Button;
   private JButton item2Button;
   private JButton item3Button;
   private JLabel item1PriceLabel;
   private JLabel item2PriceLabel;
   private JLabel item3PriceLabel;
   private JLabel item1CountLabel;
   private JLabel item2CountLabel;
   private JLabel item3CountLabel;
   private JLabel item1SubtotalLabel;
   private JLabel item2SubtotalLabel;
   private JLabel item3SubtotalLabel;
```

```
private double item1Price = 10.0;
private double item2Price = 15.0;
private double item3Price = 20.0;
private int item1Count = 0;
private int item2Count = 0;
private int item3Count = 0;
private double item1Subtotal = 0.0;
private double item2Subtotal = 0.0;
private double item3Subtotal = 0.0;
private double total = 0.0;

public SalesTerminalApp() {
   setTitle("Sales Terminal App");
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   initComponents();
   pack();
   setLocationRelativeTo(null);
   setVisible(true);
}

private void initComponents() {
   JPanel panel = new JPanel();
   panel.setLayout(new GridLayout(4, 3));

   item1Button = createItemButton("Item 1", "item1.jpg");
   item2Button = createItemButton("Item 2", "item2.jpg");
   item3Button = createItemButton("Item 3", "item3.jpg");

   item1PriceLabel = new JLabel("Price: $" + item1Price);
   item2PriceLabel = new JLabel("Price: $" + item2Price);
   item3PriceLabel = new JLabel("Price: $" + item3Price);

   item1CountLabel = new JLabel("Count: " + item1Count);
   item2CountLabel = new JLabel("Count: " + item2Count);
   item3CountLabel = new JLabel("Count: " + item3Count);

   item1SubtotalLabel = new JLabel("Subtotal: $" + item1Subtotal);
   item2SubtotalLabel = new JLabel("Subtotal: $" + item2Subtotal);
   item3SubtotalLabel = new JLabel("Subtotal: $" + item3Subtotal);

   totalLabel = new JLabel("Total: $" + total);

   panel.add(item1Button);
   panel.add(item1PriceLabel);
   panel.add(item1CountLabel);
   panel.add(item1SubtotalLabel);

   panel.add(item2Button);
```

```
    panel.add(item2PriceLabel);
    panel.add(item2CountLabel);
    panel.add(item2SubtotalLabel);

    panel.add(item3Button);
    panel.add(item3PriceLabel);
    panel.add(item3CountLabel);
    panel.add(item3SubtotalLabel);

    JPanel totalPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    totalPanel.add(totalLabel);

    setLayout(new BorderLayout());
    add(panel, BorderLayout.CENTER);
    add(totalPanel, BorderLayout.SOUTH);
}

private JButton createItemButton(String name, String imagePath) {
    ImageIcon icon = new ImageIcon(imagePath);
    JButton button = new JButton(name, icon);
    button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == item1Button) {
                item1Count++;
                item1CountLabel.setText("Count: " + item1Count);
                item1Subtotal += item1Price;
                item1SubtotalLabel.setText("Subtotal: $" + item1Subtotal);
            } else if (e.getSource() == item2Button) {
                item2Count++;
                item2CountLabel.setText("Count: " + item2Count);
                item2Subtotal += item2Price;
                item2SubtotalLabel.setText("Subtotal: $" + item2Subtotal);
            } else if (e.getSource() == item3Button) {
                item3Count++;
                item3CountLabel.setText("Count: " + item3Count);
                item3Subtotal += item3Price;
                item3SubtotalLabel.setText("Subtotal: $" + item3Subtotal);
            }

            total = item1Subtotal + item2Subtotal + item3Subtotal;
            totalLabel.setText("Total: $" + total);
        }
    });

    return button;
}
```

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

3. Create an application that overrides the action of the methods windowActivated and windowDeactivated. When the window is activated, change the background color to white. When it is deactivated, change the background to gray. Use at least four components in the GUI

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

public class WindowActivationApp extends JFrame {
    private JPanel panel1;
    private JPanel panel2;
    private JButton button1;
    private JButton button2;

    public WindowActivationApp() {
        setTitle("Window Activation App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        initComponents();
        pack();
        setLocationRelativeTo(null);
        setVisible(true);
    }

    private void initComponents() {
        panel1 = new JPanel();
        panel1.setPreferredSize(new Dimension(200, 100));
        panel1.setBackground(Color.WHITE);

        panel2 = new JPanel();
        panel2.setPreferredSize(new Dimension(200, 100));
        panel2.setBackground(Color.WHITE);

        button1 = new JButton("Button 1");
        button2 = new JButton("Button 2");

        setLayout(new FlowLayout());
        add(panel1);
        add(panel2);
```

```
        add(button1);
        add(button2);

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowActivated(WindowEvent e) {
                panel1.setBackground(Color.WHITE);
                panel2.setBackground(Color.WHITE);
            }

            @Override
            public void windowDeactivated(WindowEvent e) {
                panel1.setBackground(Color.GRAY);
                panel2.setBackground(Color.GRAY);
            }
        });
    }

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

4. Create an application that overrides the action of the method windowIconified. Every time the window is iconified, add one to a counter and display the counter in a label.

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

public class WindowIconifyApp extends JFrame {
    private JLabel counterLabel;
    private int counter = 0;

    public WindowIconifyApp() {
        setTitle("Window Iconify App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        initComponents();
        pack();
        setLocationRelativeTo(null);
        setVisible(true);
    }
```

```
    private void initComponents() {
        counterLabel = new JLabel("Counter: " + counter);

        setLayout(new FlowLayout());
        add(counterLabel);

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowIconified(WindowEvent e) {
                counter++;
                counterLabel.setText("Counter: " + counter);
            }
        });
    }

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

5. Create a GUI that has two buttons on it. When clicked, each button creates a new window. The first button creates a window that has a single button on it. Pressing that button will change the window's background color back and forth between red and green. The second button creates a window that has two buttons. Pressing the first button of these two buttons will change the window's background color to black. Pressing the second button will change the background color to white. Make sure that your windows will just dispose of themselves when they are closed.

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

public class MultipleWindowsApp {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createMainWindow();
            }
        });
    }

    private static void createMainWindow() {
```

```
    JFrame mainWindow = new JFrame("Main Window");
    mainWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JButton button1 = new JButton("Create Single Button Window");
    button1.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            createSingleButtonWindow();
        }
    });

    JButton button2 = new JButton("Create Two Buttons Window");
    button2.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            createTwoButtonsWindow();
        }
    });

    JPanel panel = new JPanel();
    panel.add(button1);
    panel.add(button2);
    mainWindow.add(panel);

    mainWindow.pack();
    mainWindow.setLocationRelativeTo(null);
    mainWindow.setVisible(true);
}

private static void createSingleButtonWindow() {
    JFrame singleButtonWindow = new JFrame("Single Button Window");
    singleButtonWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    JButton button = new JButton("Toggle Color");
    button.addActionListener(new ActionListener() {
        private Color currentColor = Color.RED;

        @Override
        public void actionPerformed(ActionEvent e) {
            if (currentColor == Color.RED) {
                singleButtonWindow.getContentPane().setBackground(Color.GREEN);
                currentColor = Color.GREEN;
            } else {
                singleButtonWindow.getContentPane().setBackground(Color.RED);
                currentColor = Color.RED;
            }
        }
    });
```

```
        singleButtonWindow.add(button);
        singleButtonWindow.pack();
        singleButtonWindow.setLocationRelativeTo(null);
        singleButtonWindow.setVisible(true);
    }

    private static void createTwoButtonsWindow() {
        JFrame twoButtonsWindow = new JFrame("Two Buttons Window");
        twoButtonsWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JButton button1 = new JButton("Black");
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                twoButtonsWindow.getContentPane().setBackground(Color.BLACK);
            }
        });

        JButton button2 = new JButton("White");
        button2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                twoButtonsWindow.getContentPane().setBackground(Color.WHITE);
            }
        });

        JPanel panel = new JPanel();
        panel.add(button1);
        panel.add(button2);
        twoButtonsWindow.add(panel);

        twoButtonsWindow.pack();
        twoButtonsWindow.setLocationRelativeTo(null);
        twoButtonsWindow.setVisible(true);
    }
}
```

New set

1.Create an application that has a text area, a text field and a button. In the text field you can enter a name. When the button is pressed get the name from the text field and create a new window. Use the name as the title of the new window. Also program the window so that when it is closed, it will append its name to the text area of your application and then dispose of itself

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

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class NameWindowApp {
    private JFrame mainFrame;
    private JTextArea textArea;

    public NameWindowApp() {
        mainFrame = new JFrame("Main Application");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(new BorderLayout());

        textArea = new JTextArea(10, 30);
        textArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(textArea);

        JTextField textField = new JTextField(20);
        JButton button = new JButton("Create Window");
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String name = textField.getText();
                createNameWindow(name);
                textField.setText("");
            }
        });

        JPanel inputPanel = new JPanel();
        inputPanel.add(new JLabel("Name: "));
        inputPanel.add(textField);
        inputPanel.add(button);

        mainFrame.add(inputPanel, BorderLayout.NORTH);
        mainFrame.add(scrollPane, BorderLayout.CENTER);

        mainFrame.pack();
        mainFrame.setLocationRelativeTo(null);
        mainFrame.setVisible(true);
    }

    private void createNameWindow(String name) {
        JFrame nameWindow = new JFrame(name);
        nameWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        nameWindow.addWindowListener(new java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                textArea.append(name + "\n");
                nameWindow.dispose();
            }
        });
        nameWindow.pack();
```

```
        nameWindow.setLocationRelativeTo(mainFrame);
        nameWindow.setVisible(true);
    }

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

2. Create a GUI that has three buttons, each of which contains the name of a song. When a button is pressed, create a new window. Title the window with the name of the song. In a text area in the window, display the lyrics to the song. When the window is closed, it should dispose of itself, but don't quit the application.

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


public class SongLyricsApp {

    private JFrame mainFrame;


    public SongLyricsApp() {

        mainFrame = new JFrame("Song Lyrics App");

        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JButton button1 = new JButton("Song 1");

        button1.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                createSongWindow("Song 1", "Lyrics for Song 1...");

            }

        });
```

```
    JButton button2 = new JButton("Song 2");
    button2.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            createSongWindow("Song 2", "Lyrics for Song 2...");
        }
    });

    JButton button3 = new JButton("Song 3");
    button3.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            createSongWindow("Song 3", "Lyrics for Song 3...");
        }
    });

    JPanel panel = new JPanel();
    panel.add(button1);
    panel.add(button2);
    panel.add(button3);

    mainFrame.add(panel);
    mainFrame.pack();
    mainFrame.setLocationRelativeTo(null);
    mainFrame.setVisible(true);
}

private void createSongWindow(String songName, String lyrics) {
    JFrame songWindow = new JFrame(songName);
    songWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
        JTextArea lyricsArea = new JTextArea(10, 30);
        lyricsArea.setEditable(false);
        lyricsArea.setText(lyrics);

        JScrollPane scrollPane = new JScrollPane(lyricsArea);

        songWindow.add(scrollPane);
        songWindow.pack();
        songWindow.setLocationRelativeTo(mainFrame);
        songWindow.setVisible(true);
    }

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

3. Create an application that will compute the average of a list of numbers. Begin with two windows. The first window, titled “Display,” should have a label and a vertically scrollable text area. The second window, titled “Data entry,” should have a text field and an Enter button. When a user enters an integer value into the text field and clicks the Enter button, copy the value in the text field to the text area in the display window. Also, update the label in this window to indicate the number of values, their sum, and average. If either window is closed, quit the application.

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

```
public class AverageCalculatorApp {
    private JFrame displayWindow;
    private JTextArea textArea;
    private JLabel infoLabel;
    private int valueCount;
    private int sum;

    public AverageCalculatorApp() {
        displayWindow = new JFrame("Display");
        displayWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        textArea = new JTextArea(10, 30);
        textArea.setEditable(false);

        JScrollPane scrollPane = new JScrollPane(textArea);

        infoLabel = new JLabel("Number of values: 0, Sum: 0, Average: 0");

        displayWindow.setLayout(new BorderLayout());
        displayWindow.add(infoLabel, BorderLayout.NORTH);
        displayWindow.add(scrollPane, BorderLayout.CENTER);
        displayWindow.pack();
        displayWindow.setLocationRelativeTo(null);
        displayWindow.setVisible(true);

        JFrame dataEntryWindow = new JFrame("Data Entry");
        dataEntryWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JTextField textField = new JTextField(10);
        JButton enterButton = new JButton("Enter");

        enterButton.addActionListener(new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    int value = Integer.parseInt(textField.getText());
                    appendValue(value);
                    textField.setText("");
                    textField.requestFocus();
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(dataEntryWindow, "Invalid input. Please enter an integer.");
                }
            }
        });

        JPanel panel = new JPanel();
        panel.add(textField);
        panel.add(enterButton);

        dataEntryWindow.add(panel);
        dataEntryWindow.pack();
        dataEntryWindow.setLocationRelativeTo(displayWindow);
        dataEntryWindow.setVisible(true);
    }

    private void appendValue(int value) {
        textArea.append(value + "\n");
        valueCount++;
        sum += value;
        double average = (double) sum / valueCount;
        infoLabel.setText("Number of values: " + valueCount + ", Sum: " + sum + ", Average: " + average);
    }
```

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

4. Create an application that will display a string of text using different fonts. Your application should have three menus. The font-name menu will have five menu items: Serif, SansSerif, Monospaced, Dialog, and DialogInput. The style menu will have three menu items: Plain, Bold, and Italic. The size menu will have four menu items: 10, 12, 18, and 24. When a menu item is chosen, change the font and then repaint the window. *Hint*: Use the method drawString to display the sample string. Use the expression new Font(type, style, size) to create the font you will display. Refer to the API documentation for the class Font for additional details

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

public class FontDisplayApp extends JFrame {
    private String sampleText = "Sample Text";
    private String fontName = "Serif";
    private int fontStyle = Font.PLAIN;
    private int fontSize = 12;

    public FontDisplayApp() {
        setTitle("Font Display App");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JMenuBar menuBar = new JMenuBar();
```

```
JMenu fontNameMenu = new JMenu("Font Name");
JMenuItem serifItem = new JMenuItem("Serif");
JMenuItem sansSerifItem = new JMenuItem("SansSerif");
JMenuItem monospacedItem = new JMenuItem("Monospaced");
JMenuItem dialogItem = new JMenuItem("Dialog");
JMenuItem dialogInputItem = new JMenuItem("DialogInput");

serifItem.addActionListener(new FontNameActionListener());
sansSerifItem.addActionListener(new FontNameActionListener());
monospacedItem.addActionListener(new FontNameActionListener());
dialogItem.addActionListener(new FontNameActionListener());
dialogInputItem.addActionListener(new FontNameActionListener());

fontNameMenu.add(serifItem);
fontNameMenu.add(sansSerifItem);
fontNameMenu.add(monospacedItem);
fontNameMenu.add(dialogItem);
fontNameMenu.add(dialogInputItem);

JMenu styleMenu = new JMenu("Style");
JMenuItem plainItem = new JMenuItem("Plain");
JMenuItem boldItem = new JMenuItem("Bold");
JMenuItem italicItem = new JMenuItem("Italic");

plainItem.addActionListener(new StyleActionListener());
boldItem.addActionListener(new StyleActionListener());
italicItem.addActionListener(new StyleActionListener());

styleMenu.add(plainItem);
styleMenu.add(boldItem);
styleMenu.add(italicItem);
```

```
    JMenu sizeMenu = new JMenu("Size");
    JMenuItem size10Item = new JMenuItem("10");
    JMenuItem size12Item = new JMenuItem("12");
    JMenuItem size18Item = new JMenuItem("18");
    JMenuItem size24Item = new JMenuItem("24");

    size10Item.addActionListener(new SizeActionListener());
    size12Item.addActionListener(new SizeActionListener());
    size18Item.addActionListener(new SizeActionListener());
    size24Item.addActionListener(new SizeActionListener());

    sizeMenu.add(size10Item);
    sizeMenu.add(size12Item);
    sizeMenu.add(size18Item);
    sizeMenu.add(size24Item);

    menuBar.add(fontNameMenu);
    menuBar.add(styleMenu);
    menuBar.add(sizeMenu);

    setJMenuBar(menuBar);
}

@Override
public void paint(Graphics g) {
    super.paint(g);

    Font font = new Font(fontName, fontStyle, fontSize);
    g.setFont(font);
    FontMetrics metrics = g.getFontMetrics();
    int x = (getWidth() - metrics.stringWidth(sampleText)) / 2;
```

```
    int y = (getHeight() + metrics.getHeight()) / 2;
    g.drawString(sampleText, x, y);
}

private class FontNameActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        fontName = e.getActionCommand();
        repaint();
    }
}

private class StyleActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        String style = e.getActionCommand();
        switch (style) {
            case "Plain":
                fontStyle = Font.PLAIN;
                break;
            case "Bold":
                fontStyle = Font.BOLD;
                break;
            case "Italic":
                fontStyle = Font.ITALIC;
                break;
        }
        repaint();
    }
}

private class SizeActionListener implements ActionListener {
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
            fontSize = Integer.parseInt(e.getActionCommand());
            repaint();
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                FontDisplayApp app = new FontDisplayApp();
                app.setVisible(true);
            }
        });
    }
}
```

5. Write an application that creates a list of names. You will need a text area in the center of the screen capable of holding ten lines that cannot be edited. Place a text field and a menu. After a user enters a name into the text field and selects the “Add Name” menu option, take the name from the text field and add it to the text area. Then clear the text field.

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

public class NameListApp extends JFrame {
    private JTextArea textArea;
    private JTextField textField;

    public NameListApp() {
        setTitle("Name List App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        textArea = new JTextArea(10, 20);
        textArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(textArea);
        add(scrollPane, BorderLayout.CENTER);
```

```
    textField = new JTextField(20);
    JButton addButton = new JButton("Add Name");
    addButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String name = textField.getText();
            addNameToList(name);
            textField.setText("");
        }
    });

    JPanel inputPanel = new JPanel();
    inputPanel.add(textField);
    inputPanel.add(addButton);
    add(inputPanel, BorderLayout.SOUTH);

    JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("Menu");
    JMenuItem addNameMenuItem = new JMenuItem("Add Name");
    addNameMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String name = textField.getText();
            addNameToList(name);
            textField.setText("");
        }
    });
    menu.add(addNameMenuItem);
    menuBar.add(menu);
    setJMenuBar(menuBar);

    pack();
    setLocationRelativeTo(null);
}

private void addNameToList(String name) {
    String currentText = textArea.getText();
    if (currentText.isEmpty()) {
        textArea.setText(name);
    } else {
        textArea.setText(currentText + "\n" + name);
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
```

```
            NameListApp app = new NameListApp();
            app.setVisible(true);
        }
    });
  }
}
```

1.Write a program that converts the Java source code from the next-line brace style to the end-of-line brace style.

```
import java.io.*;


public class BraceStyleConverter {
   public static void main(String[] args) {
      if (args.length != 2) {
         System.out.println("Usage: java BraceStyleConverter <inputFile> <outputFile>");
         return;
      }

      String inputFile = args[0];
      String outputFile = args[1];

      try (BufferedReader reader = new BufferedReader(new FileReader(inputFile));
          BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
         String line;
         while ((line = reader.readLine()) != null) {
            line = line.trim();
            if (line.equals("{")) {
               writer.write(" {");
            } else if (line.equals("}")) {
               writer.write("} ");
            } else {
               writer.write(line);
            }
```

```
            writer.newLine();
        }
        System.out.println("Conversion completed successfully!");
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
  }
}
```

2.Write a program that will count the number of characters, words, and lines in a file. Words are separated by whitespace characters. The file name should be passed as a command-line argument

```
import java.io.*;

public class FileStatsCounter {
  public static void main(String[] args) {
    if (args.length != 1) {
      System.out.println("Usage: java FileStatsCounter <inputFile>");
      return;
    }

    String inputFile = args[0];

    int charCount = 0;
    int wordCount = 0;
    int lineCount = 0;

    try (BufferedReader reader = new BufferedReader(new FileReader(inputFile))) {
      String line;
      while ((line = reader.readLine()) != null) {
        charCount += line.length();
        wordCount += line.split("\\s+").length;
```

```
            lineCount++;
        }
        System.out.println("Character count: " + charCount);
        System.out.println("Word count: " + wordCount);
        System.out.println("Line count: " + lineCount);
      } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
      }
    }
}
```

3. Display a greeting and then prompt for and read the name of the input file. It should then try to connect an input stream to that file so that we can read from it, and print a diagnostic message if the stream does not open correctly. It should then prompt for and read the name of the output file. It should then try to connect an output stream to that file so that we can write to it, and print a diagnostic message if the stream does not open correctly. For each character in the input file, our program should read the character, encode it using the Caesar cipher, and output the encoded character to the output file. Our program should conclude by disconnecting the streams from the files, and then print a "success" message

```
import java.io.*;


public class CaesarCipher {
   public static void main(String[] args) {
      System.out.println("Welcome to the Caesar Cipher program!");


      // Prompt for and read the input file name
      System.out.print("Enter the input file name: ");
      String inputFileName = getInput();


      // Try to connect an input stream to the input file
      BufferedReader reader = null;
      try {
         reader = new BufferedReader(new FileReader(inputFileName));
      } catch (FileNotFoundException e) {
```

```
    System.out.println("Error opening input file: " + e.getMessage());
    System.exit(1);
}

// Prompt for and read the output file name
System.out.print("Enter the output file name: ");
String outputFileName = getInput();

// Try to connect an output stream to the output file
BufferedWriter writer = null;
try {
    writer = new BufferedWriter(new FileWriter(outputFileName));
} catch (IOException e) {
    System.out.println("Error opening output file: " + e.getMessage());
    System.exit(1);
}

try {
    int character;
    while ((character = reader.read()) != -1) {
        char encodedCharacter = encode(character);
        writer.write(encodedCharacter);
    }
    System.out.println("Encoding completed successfully!");
} catch (IOException e) {
    System.out.println("Error reading from/writing to file: " + e.getMessage());
} finally {
    try {
        reader.close();
        writer.close();
    } catch (IOException e) {
```

```
            System.out.println("Error closing file: " + e.getMessage());
        }
    }
}

private static char encode(int character) {
    // Apply Caesar cipher by shifting each character by 3 positions
    if (character >= 'a' && character <= 'z') {
        character = (character - 'a' + 3) % 26 + 'a';
    } else if (character >= 'A' && character <= 'Z') {
        character = (character - 'A' + 3) % 26 + 'A';
    }
    return (char) character;
}

private static String getInput() {
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        return reader.readLine();
    } catch (IOException e) {
        System.out.println("Error reading input: " + e.getMessage());
        System.exit(1);
    }
    return null;
}
}
```

4. Read all the bytes from the file. Ignoring repetitions, sort them by byte-code in ascending order. Display the result on the screen. Close the IO stream. Example of bytes in the input file: 44 83 44 Example output: 44 83

```
import java.io.*;

import java.util.*;
```

```
public class ByteSorter {
    public static void main(String[] args) {
        System.out.println("Welcome to the Byte Sorter program!");

        // Prompt for and read the input file name
        System.out.print("Enter the input file name: ");
        String inputFileName = getInput();

        // Try to connect an input stream to the input file
        FileInputStream inputStream = null;
        try {
            inputStream = new FileInputStream(inputFileName);
        } catch (FileNotFoundException e) {
            System.out.println("Error opening input file: " + e.getMessage());
            System.exit(1);
        }

        Set<Byte> byteSet = new TreeSet<>();

        try {
            int byteValue;
            while ((byteValue = inputStream.read()) != -1) {
                byteSet.add((byte) byteValue);
            }
            System.out.print("Sorted bytes: ");
            for (Byte b : byteSet) {
                System.out.print(b + " ");
            }
            System.out.println();
        } catch (IOException e) {
```

```
            System.out.println("Error reading from file: " + e.getMessage());
        } finally {
            try {
                inputStream.close();
            } catch (IOException e) {
                System.out.println("Error closing file: " + e.getMessage());
            }
        }
    }

    private static String getInput() {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            return reader.readLine();
        } catch (IOException e) {
            System.out.println("Error reading input: " + e.getMessage());
            System.exit(1);
        }
        return null;
    }
}
```

5. Suppose that a text file contains an unspecified number of scores separated by blanks. Write a program that prompts the user to enter the file, reads the scores from the file, and displays their total and average.

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

public class ScoreCalculator {
    public static void main(String[] args) {
        System.out.println("Welcome to the Score Calculator program!");

        // Prompt for and read the input file name
```

```
System.out.print("Enter the input file name: ");
String inputFileName = getInput();

// Try to connect a file input stream to the input file
FileInputStream inputStream = null;
try {
   inputStream = new FileInputStream(inputFileName);
} catch (FileNotFoundException e) {
   System.out.println("Error opening input file: " + e.getMessage());
   System.exit(1);
}

// Read scores from the file and calculate total and average
Scanner scanner = new Scanner(inputStream);
int total = 0;
int count = 0;

while (scanner.hasNext()) {
   if (scanner.hasNextInt()) {
      int score = scanner.nextInt();
      total += score;
      count++;
   } else {
      scanner.next(); // Skip non-integer value
   }
}

// Close the input stream
try {
   inputStream.close();
} catch (IOException e) {
```

```
            System.out.println("Error closing file: " + e.getMessage());
        }

        // Calculate average
        double average = count > 0 ? (double) total / count : 0.0;

        // Display total and average
        System.out.println("Total score: " + total);
        System.out.println("Average score: " + average);
    }

    private static String getInput() {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            return reader.readLine();
        } catch (IOException e) {
            System.out.println("Error reading input: " + e.getMessage());
            System.exit(1);
        }
        return null;
    }
}
```

6. Create a data file with 100 lines. Each line in the file consists of a faculty member's first name, last name, rank, and salary. The faculty member's first name and last name for the ith line are FirstName and LastName. The rank is randomly generated as assistant, associate, and full. The salary is randomly generated as a number with two digits after the decimal point. The salary for an assistant professor should be in the range from 50,000 to 80,000, for associate professor from 60,000 to 110,000, and for full professor from 75,000 to 130,000. Save the file in Salary.txt.

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
```

```
public class DataFileGenerator {
    public static void main(String[] args) {
        System.out.println("Generating data file...");

        // Generate and write data to the file
        try {
            PrintWriter writer = new PrintWriter(new FileWriter("Salary.txt"));

            for (int i = 1; i <= 100; i++) {
                String firstName = "FirstName";
                String lastName = "LastName";
                String rank = generateRandomRank();
                double salary = generateRandomSalary(rank);

                String line = firstName + " " + lastName + " " + rank + " " + String.format("%.2f", salary);
                writer.println(line);
            }

            writer.close();
            System.out.println("Data file generated successfully!");
        } catch (IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }

    private static String generateRandomRank() {
        String[] ranks = {"assistant", "associate", "full"};
        Random random = new Random();
        int index = random.nextInt(ranks.length);
        return ranks[index];
```

```
    }

    private static double generateRandomSalary(String rank) {
        Random random = new Random();

        switch (rank) {
            case "assistant":
                return 50000 + random.nextDouble() * (80000 - 50000 + 1);
            case "associate":
                return 60000 + random.nextDouble() * (110000 - 60000 + 1);
            case "full":
                return 75000 + random.nextDouble() * (130000 - 75000 + 1);
            default:
                return 0.0;
        }
    }
}
```