

## COMPUTER SCIENCE DEPARTMENT

**CS0053**

(PROGRAMMING TOOLS AND TECHNIQUES)

EXERCISE

1

GUI-Based Calculator

|  |  |
| --- | --- |
| Name: **John Paul L. Besagas** | Professor: **Dr. Beau Habal** |
| Date Performed : **9/17/2024** | Date Submitted: **9/18/2024** |

1. **OBJECTIVES**

At the end of this exercise, students must be able to:

Cognitive

1. Understand the topics they have learned from lesson 1.

.

Psychomotor:

1. Apply single and multidimensional array.
2. Construct a modular Java program.
3. Create Java GUI-based program.

Affective

1. Appreciate the concept behind this exercise.
2. **BACKGROUND INFORMATION**

In order to accomplish this exercise, the student must have a clear understanding of the following topics:

* Arrays
* Java Swing Components
* Import packages or libraries

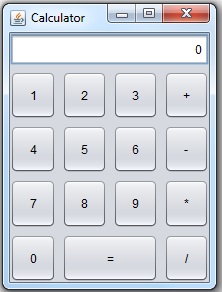
1. **LABORATORY PROCEDURE**

1. Create a new Java class file.

Project Name: SUBJECTCODE-SECTION (i.e CSSSPEC2-N21)

Program Name: Calculator.java

2. Design your layout as shown below



3. Requirements

- Must validate input.

* Apply coding conventions.
* The program must be free from any errors.

|  |
| --- |
| import javax.swing.\*;  import java.awt.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  public class Calculator implements ActionListener {      // GUI      JFrame frameCalculator;      JTextField fieldDisplay;      JButton[] buttonsDigits = new JButton[10];      JButton buttonAdd, buttonSubtract, buttonMultiply, buttonDivide, buttonEquals, buttonClear;      JPanel panelButtons;      Font fontDisplay = new Font("Arial", Font.BOLD, 24);      Color backgroundColor = new Color(30, 30, 30);      Color buttonColor = new Color(60, 60, 60);      Color equalsButtonColor = new Color(0, 122, 204);      Color textColor = Color.WHITE;      StringBuilder expressionInput = new StringBuilder();      public Calculator() {          // Initialize components and show the frame          initializeFrame();          initializeDisplay();          initializeButtons();          initializePanel();          frameCalculator.setVisible(true);      }      // Initialize the main frame      public void initializeFrame() {          frameCalculator = new JFrame("Calculator");          frameCalculator.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);          frameCalculator.setSize(400, 550);          frameCalculator.setLayout(null);          frameCalculator.getContentPane().setBackground(backgroundColor);      }      // Initialize the display field      public void initializeDisplay() {          fieldDisplay = new JTextField("0");          fieldDisplay.setBounds(30, 25, 340, 50);          fieldDisplay.setFont(fontDisplay);          fieldDisplay.setEditable(false);          fieldDisplay.setHorizontalAlignment(JTextField.RIGHT);          fieldDisplay.setBackground(Color.WHITE);          fieldDisplay.setForeground(Color.BLACK);          fieldDisplay.setBorder(BorderFactory.createEmptyBorder());          frameCalculator.add(fieldDisplay);      }      // Initialize all buttons      public void initializeButtons() {          buttonAdd = createButton("+");          buttonSubtract = createButton("-");          buttonMultiply = createButton("\*");          buttonDivide = createButton("/");          buttonEquals = createButton("=");          buttonClear = createButton("C");          buttonEquals.setBackground(equalsButtonColor);          // Create digit buttons (0-9)          for (int i = 0; i < 10; i++) {              buttonsDigits[i] = createButton(String.valueOf(i));          }      }      // Initialize the panel to hold all buttons      public void initializePanel() {          panelButtons = new JPanel();          panelButtons.setBounds(30, 100, 340, 340);          panelButtons.setLayout(new GridLayout(4, 4, 10, 10));          panelButtons.setBackground(backgroundColor);          // Add buttons to the panel in order          panelButtons.add(buttonsDigits[1]);          panelButtons.add(buttonsDigits[2]);          panelButtons.add(buttonsDigits[3]);          panelButtons.add(buttonAdd);          panelButtons.add(buttonsDigits[4]);          panelButtons.add(buttonsDigits[5]);          panelButtons.add(buttonsDigits[6]);          panelButtons.add(buttonSubtract);          panelButtons.add(buttonsDigits[7]);          panelButtons.add(buttonsDigits[8]);          panelButtons.add(buttonsDigits[9]);          panelButtons.add(buttonMultiply);          panelButtons.add(buttonsDigits[0]);          panelButtons.add(buttonEquals);          panelButtons.add(buttonClear);          panelButtons.add(buttonDivide);          frameCalculator.add(panelButtons);      }      // Method to create a button with specified text and styling      public JButton createButton(String text) {          JButton button = new JButton(text);          button.setFont(fontDisplay);          button.setFocusable(false);          button.setBackground(buttonColor);          button.setForeground(textColor);          button.setBorder(BorderFactory.createLineBorder(buttonColor));          button.addActionListener(this);          return button;      }      // Handle button click events      @Override      public void actionPerformed(ActionEvent e) {          // Check if a digit button was clicked          for (int i = 0; i < 10; i++) {              if (e.getSource() == buttonsDigits[i]) {                  appendDigit(i);                  return;              }          }          // Check if an operator button was clicked          if (e.getSource() == buttonAdd) {              appendOperator('+');          } else if (e.getSource() == buttonSubtract) {              appendOperator('-');          } else if (e.getSource() == buttonMultiply) {              appendOperator('\*');          } else if (e.getSource() == buttonDivide) {              appendOperator('/');          } else if (e.getSource() == buttonEquals) {              calculateResult();          } else if (e.getSource() == buttonClear) {              clear();          }      }      // Append a digit to the current input      public void appendDigit(int digit) {          if (fieldDisplay.getText().equals("0")) {              fieldDisplay.setText("");              expressionInput.setLength(0);          }          fieldDisplay.setText(fieldDisplay.getText() + digit);          expressionInput.append(digit);      }      // Append an operator to the current input      public void appendOperator(char operator) {          if (expressionInput.length() > 0) {              if (!isOperator(expressionInput.charAt(expressionInput.length() - 1))) {                  fieldDisplay.setText(fieldDisplay.getText() + operator);                  expressionInput.append(operator);              }          }      }      // Check if a character is an operator      public boolean isOperator(char character) {          return character == '+' || character == '-' || character == '\*' || character == '/';      }      // Calculate the result of the current expression      public void calculateResult() {          try {              double result = evaluate(expressionInput.toString());              fieldDisplay.setText(format(result));              expressionInput.setLength(0);              expressionInput.append(result);          } catch (Exception ex) {              fieldDisplay.setText("Error");              expressionInput.setLength(0);          }      }      // Clear the display and reset the expression      public void clear() {          fieldDisplay.setText("0");          expressionInput.setLength(0);      }      // Evaluate a mathematical expression represented as a string      public double evaluate(String expression) {          return new Object() {              int index = -1, currentCharacter;              void nextCharacter() {                  index++;                  if (index < expression.length()) {                      currentCharacter = expression.charAt(index);                  } else {                      currentCharacter = -1;                  }              }              boolean consume(int characterToEat) {                  while (currentCharacter == ' ')                      nextCharacter();                  if (currentCharacter == characterToEat) {                      nextCharacter();                      return true;                  }                  return false;              }              double parse() {                  nextCharacter();                  double value = parseExpression();                  if (index < expression.length())                      throw new RuntimeException("Unexpected: " + (char) currentCharacter);                  return value;              }              double parseExpression() {                  double value = parseTerm();                  while (true) {                      if (consume('+'))                          value += parseTerm();                      else if (consume('-'))                          value -= parseTerm();                      else                          break;                  }                  return value;              }              double parseTerm() {                  double value = parseFactor();                  while (true) {                      if (consume('\*'))                          value \*= parseFactor();                      else if (consume('/'))                          value /= parseFactor();                      else                          break;                  }                  return value;              }              double parseFactor() {                  if (consume('+'))                      return parseFactor();                  if (consume('-'))                      return -parseFactor();                  double value;                  int startIndex = this.index;                  if ((currentCharacter >= '0' && currentCharacter <= '9') || currentCharacter == '.') {                      while ((currentCharacter >= '0' && currentCharacter <= '9') || currentCharacter == '.')                          nextCharacter();                      value = Double.parseDouble(expression.substring(startIndex, this.index));                  } else {                      throw new RuntimeException("Unexpected: " + (char) currentCharacter);                  }                  return value;              }          }.parse();      }      // Format the result for display, removing decimal if it's a whole number      public String format(double result) {          if (result == (int) result) {              return String.valueOf((int) result);          } else {              return String.valueOf(result);          }      }      public static void main(String[] args) {          new Calculator();      }  } |

1. **QUESTION AND ANSWER**
2. What part of your program use array concept?

**The concept of array is utilized in my program through the management of digit buttons, which represent the numbers 0-9. By storing these buttons in an array, I can efficiently iterate over them to add each button to the GUI panel and handle their events collectively.**

1. How do you able to use those SWING?

**Java Swing components are utilized throughout the calculator program after importing the necessary packages. I used JFrame for creating the main window of the calculator, JTextField for the display where results and inputs are shown, JButton for creating interactive buttons, and JPanel for organizing the buttons into a grid layout.**

1. What are the event handlings you used in the program?

**I used it by implementing the ActionListener and overriding the actionPerformed method. This method is designed to handle various actions based on the source of the event whether a digit, operator, or function button is pressed. ActionListener allows the program to react dynamically to user inputs, executing specific calculator operations like addition, subtraction, multiplication, division, or resetting the display.**

1. What are the coding conventions you applied in your program?

**I applied the class name should be a noun and is properly capitalized. My methods are also verbs then I keep my variable names short and meaningful, both my methods and variables follow the camelCase convention, I also put comments for self- documentation and understanding of each block of code.**

1. **ASSESSMENT**

|  |  |
| --- | --- |
| Department | Computer Science |
| Subject Code | CSSSPEC2 |
| Description | Programming Tools and Techniques |
| Term/Academic Year | 1st Term SY 2016-2017 |

|  |  |
| --- | --- |
| Topic | Java Fundamentals and Coding Convention |
| Lab Activity No | 1 |
| Lab Activity | **GUI-Based Calculator** |
| CLO | **1, 2** |

**Note: The following rubrics/metrics will be used to grade students’ output in the lab exercise 1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | Exceptional | Acceptable | Amateur | Unsatisfactory |
| Specifications  (40%) | The program works and meets all of the specifications. (40) | The program works and produces the correct results and displays them correctly. It also meets most of the other specifications. (35-39) | The program produces correct results but does not display them correctly. (30-34) | The program is producing incorrect results. (20-29) |
| Design  (15 %) | The design is exceptionally attractive. Program is "user-friendly" with informative and consistent prompts and messages. (15) | The design is fairly attractive. Program is "user-friendly" with informative and consistent prompts and messages. (13-14) | The design is fairly attractive. Program is not "user-friendly" but still provide informative and consistent prompts and messages. (10-12) | The design is unattractive and not user-friendly (8-9) |
| Efficiency (20%) | The code is extremely efficient without sacrificing readability and understanding. (20) | The code is fairly efficient without sacrificing readability and understanding. (17-19) | The code is brute force and unnecessarily long. (14-16) | The code is huge and appears to be patched together. (10-13) |
| Readability  (10 %) | The code is exceptionally well organized and very easy to follow. (10) | The code is fairly easy to read. (8-9) | The code is readable only by someone who knows what it is supposed to be doing. (6-7) | The code is poorly organized and very difficult to read. (4-5) |
| Delivery  (15%) | The program was delivered on time. (15) | The program was delivered within a day of the due date. (13-14) | The code was within 2 days of the due date. (10-12) | The code was within a week of the due date. (8-9) |
| Total: 100% |  |  |  |  |