LBYCPEI

**Object-Oriented Programming Laboratory**



*Image by @Criisalys: https://twitter.com/Criisalys/status/1375504259404476418/photo/1*

**Laboratory Module 5**

Java Custom Objects and Encapsulation

By

Jay Michael T. Carlos, LBYCPEI-EQ8

# INTRODUCTION

This module is all about using *encapsulation* to merge data and functions in a custom-bult class to effectively model a problem and easily interact with the necessary parts. In this module, you will use standard good practices in dealing with object data in Java and object-oriented programming in general to make it easier to model more complex problems and applications, such as a calculator.

1. Objectives
2. To understand encapsulation in object-oriented programming.
3. To develop a calculator app using custom objects and encapsulation.
4. To understand good practices in handling data in objects in object-oriented programming.
5. To implement an app with a graphical user interface without using libraries such as Java Swing.
6. Materials
7. IntelliJ IDEA (The Java IDE)
8. Java SE 14
9. cpei.jar (acm.jar and karel.jar)
10. ACM graphics documentation
11. Junit documentation

# PROCEDURES (Individual) / EXPERIMENTAL PLAN

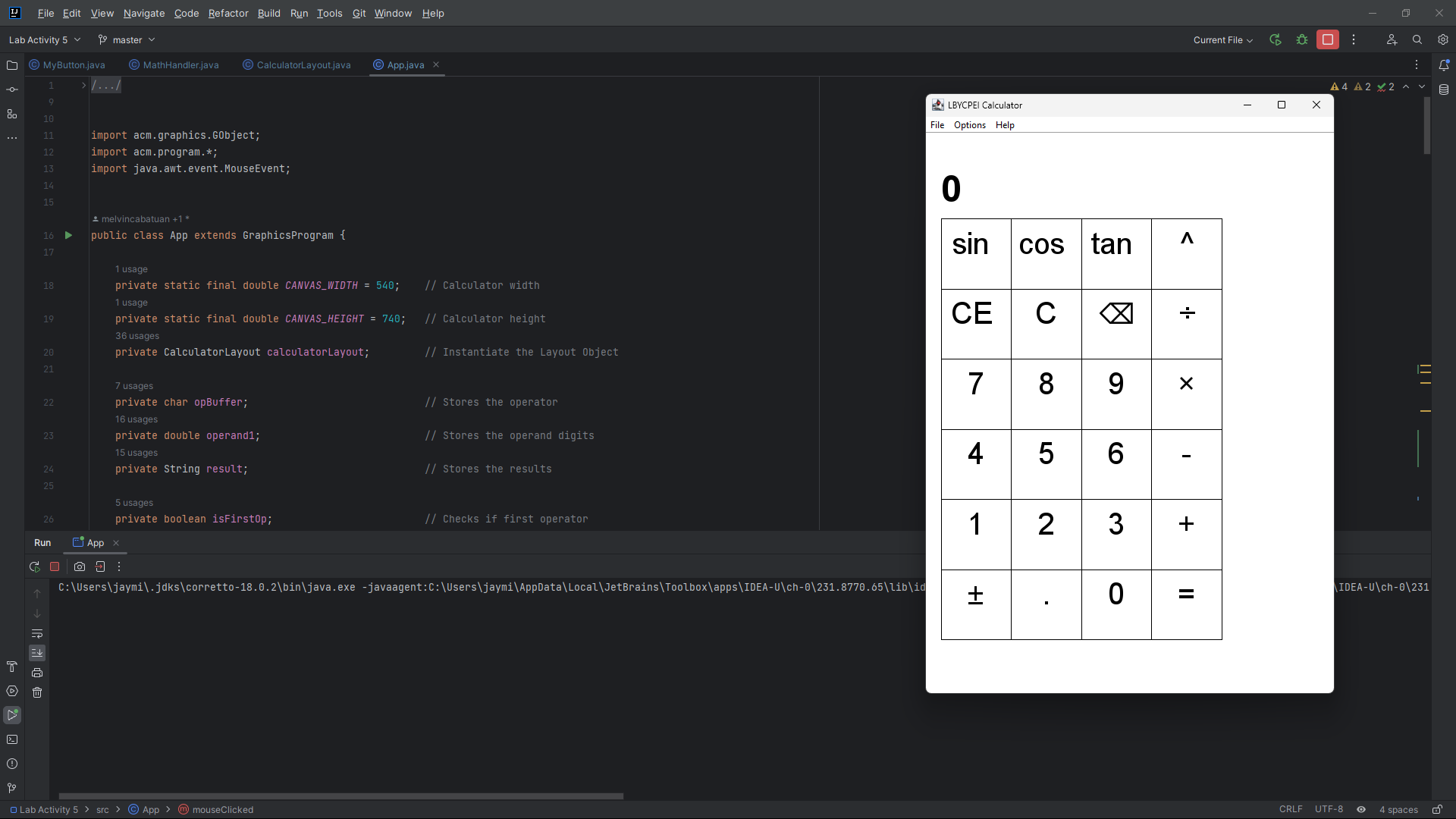
1. In Laboratory Familiarization 1: "Calculator App” First, download the starter file from <https://github.com/LBYCPEI/CalculatorStarter>. You will see four files in the src folder, three of which you will modify: “App.java”, “MyButton.java”, “CalculatorLayout.java”, and “MathHandler.java”. You will not edit MyButton.java. You can start by editing the MathHandler.java file, as it is the simplest to get started with and has a lot missing. This file is responsible for returning the values from the various mathematical operations that the calculator will be able to do. Below the second evaluate() method, there are blank methods that each correspond to an operation. Some of them have a version that returns an integer and another that returns a double, to accommodate both types of values. Replace the return statements in both add() methods with return operand1 + operand2. Then, replace the return statements in both subtract() methods with return operand1 – operand2. Then, replace the return statements in both divide() methods with return operand1 / operand2. Then, replace the return statements in both multiply() methods with return operand1 \* operand2. Then, create four new methods that return a double value. One called sin(), cos(), tan(), and exponent(). sin() cos() and tan() should each have operand1 as the parameter, and exponent() should have operand1 and operand2 as the parameters. For the sin() method, return Math.sin() with loperand1 as the parameter. For the cos() method, return Math.cos() with operand1 as the parameter. For the tan() method, return Math.tan() with operand1 as the parameter. The last one is the exponent() method, which should return Math.pow(operand1, operand2). Then, in the evaluate() methods, add new cases in the switch() statement; one for each of the new methods you created. For sin(), the case should be ‘s’, and it should set result to the return value of sin(operand1). For cos(), the case should be ‘c’, and it should set result to the return value of cos(operand1). For tan(), the case should be ‘t’, and it should set result to the return value of tan(operand1). For exponent(), the case should be ‘^’, and it should set result to the return value of exponent(operand1, operand2).

Next, in CalculatorLayout.java, add three blank (‘ ‘) characters and a ‘^’ symbol to the start of the labels[] array. The blank characters are for when a button symbol is longer than 1 character (sin, cos, and tan), and the ‘^’ is for the exponent. You will then need to reflect this change in the CalculatorLayout() method, inside the nested for loop that draws the buttons. Remove the code to the right of MyButton myButton, as this is a shorthand conditional statement that checks if the code is at the first button and replaces the blank label to “CE”. This shorthand works for simpler conditions, but you are going to be replacing it with multiple statements, which may get confusing in that case; as such, a switch statement with 5 different cases, including the default one. The other 4 cases should be 0, 1, 2, and 4, because those indices correspond to the buttons with labels longer than 1 character. Case 0 should set myButton to be a new instance of MyButton() with the arguments x, y, sqSize, sqSize, “sin” + labels[count++]. Case 1 should set myButton() to be a new instance of MyButton() with the same arguments, except the text should be “cos” instead of “sin”. Case 2 should set myButton() to be a new instance of MyButton() with the same arguments, except the text should be “tan” instead of “cos”. Case 4 should set myButton() to be a new instance of MyButton() with the same arguments, except the text should be “CE”. The default should be the same, but the last argument should be “” + labels[count++]. Then, go to the top of the CalculatorLayout class and change NROWS to 7.

The last file you need to edit is the App.java file, wherein you first need to go to the if condition that checks if isOperator(symbol) is true. Below the condition that checks if isPriorEquals is true, create a new else-if block that checks if symbol is ‘s’, ‘c’, or ‘t’, and that isFirstOp is false. Inside the conditional, set operand1 to the text on the calculator’s main display, like so: Double.parseDouble(calculatorLayout.getMainDisplay()). Then, set operand2 to the return value of the evaluate() method from the MathHandler, giving it operand1, 0, and opBuffer as parameters. Then, set result to be “” + operand2 to convert it to a string. Then, display result on the main display like so: calculatorLayout.setMainDisplay(result). Below the if statement that checks if symbol is “=”, create an if condition that checks if symbol is ‘s’, then another one that checks if symbol is ‘c’, then another one that checks if symbol is ‘t’. In each of the if conditions, call the clearMemoDisplay() and clearMainDisplay() methods, then call setMemoDisplay() and setMainDisplay() with the respective trigonometric function as a string + operand1 as the argument. Lastly, in isOperator(), add || symbol == ‘^’ || symbol == ‘s’ || symbol == ‘c’ || symbol == ‘t’ to the end of the return statement inside of the parentheses to check for the new operators that you added earlier.

# RESULTS AND DISCUSSION

1. Laboratory Familiarization 1: "Calculator App” Result:



**Explanation**:

App.java is responsible for handling user interaction with the app and being a driver program for the rest of the classes that perform the calculator’s necessary functions. run() is the driver program. mouseClicked() is the event listener that listens to user mouse inputs that then checks what button the user clicked on, sets the operand fields to their appropriate value, and calls the appropriate methods to refresh the calculator’s main and memo displays. initBooleans() contains some Boolean values that modify certain behaviors of the calculator, such as determining whether a character can be deleted or not, or if the operator input is the first one. isOperator() returns a Boolean value of true if the character received as a parameter is an operator, otherwise it returns false. Lastly, the main() method starts the app by creating a new instance of it.

MyButton.java contains methods to simplify the process of creating a button. You do not need to touch the methods in this file. The MyButton() constructor accepts an x, y, width, height, and text value, and adds a GLabel with certain predefined characteristics such as its font and a position offset based on the parameters given to it. getText() returns the content of the GLabel’s label as a string. getCenterX() calculates the horizontal center point of a button and compensates for the label’s width. getCenterY() does the same thing but for the vertical center point. Lastly, setText() modifies the text of a button.

MathHandler.java is where the actual calculations happen in the program. Most of the methods you see here will have two versions: one that returns a double and another that returns an integer. This is so that the calculator will return an appropriate value; for example, 2 + 2 should return 4 instead of 4.00, and 4.5 + 9.8 should return 14.3 and not 14 or 15. The evaluate() method accepts both operands and an operator. The operator argument gets passed into a switch case that determines which method gets called; the return value gets saved into the result field. All the other methods in this file simply calculate the appropriate value based on the operands given as parameters and return that value. Sine, cosine, tangent, and exponent functions require the use of Java’s Math functions like Math.sin(), Math.cos(), Math.tan(), and Math.pow().

CalculatorLayout.java handles the layout and display of the calculator buttons. The CalculatorLayout() constructor creates each of the buttons in the calculator and adds the appropriate labels to the buttons. setMemoDisplay() changes the label in the top display of the calculator. setMainDisplay() changes the lower, larger display. getMainDisplay() returns the value of the label on the main display. clearMainDisplay() clears the buffer and sets the main display to 0. clearMemoDisplay() does the same thing but for the memo display. clearNumBuffer() resets the buffer that stores user input. clearMemoBuffer() does the same thing but for the memo display. clearMemoElement() removes the element from the memo display. negateElement() checks a specified character and reverses the symbol. deleteOneCharacter() reduces the length of the buffer and updates the label.

A screenshot of a computer

Description automatically generated with medium confidence

# CONCLUSION:

I was personally able to understand encapsulation. It is the concept of wrapping the data and code together as a single unit. I was able to make the calculator app using this concept. The encapsulation came in the form of the displays containing the methods needed to access, edit, and remove the data. Good practices were upheld in the program because encapsulation ensured that data access is controlled. Lastly, a graphical user interface was created without the use of libraries such as Java Swing. I have learned how to use encapsulation in an object-oriented program and how to manage data with good practices as a result. The most common mistake I made while making this program was not returning proper values for the sine, cosine, and tangent functions. Unlike the operations, the trigonometric functions only use one operand, and the rest of the operations use two operands. As such, adjustments must also be made in App.java to accommodate this properly. I overcame this issue by studying how the App works and how methods connect to one another. I would recommend that anyone study the interaction between methods before actually writing any code, because it will help avoid the common errors I encountered.

# REFERENCES

1. Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). The Java language specification. Oracle America, Inc.
2. Roberts, E. (2005). Karel the robot learns java. Department of Computer Science Stanford University.
3. Roberts, E. (2008). The Art & Science of Java. Pearson.
4. Oracle. Java Documentation. ( https://docs.oracle.com/en/java )
5. Schildt, H. (2017). Java: A Beginner's Guide, 7th Edition. McGraw-Hill Education.
6. Troccoli, N. (2017). CS 106A: Assignment #1: Karel the Robot. Stanford University
7. UML basics: The component diagram. http://www.ibm.com/developerworks/rational/library/dec04/bell/

# APPENDIX

1. Lab Activity 1.1 “App.java”

/\*  
 \* File: Calculator.java (Find and solve bugs if they exist.)  
 \* ---------------------  
 \* This class is the main class for a sample calculator app implementation  
 \* Author: Cobalt mkc  
 \* Date created: July 22, 2019  
 \* Last modified: Aug 3, 2022  
 \*/  
  
  
import acm.graphics.GObject;  
import acm.program.\*;  
import java.awt.event.MouseEvent;  
  
  
public class App extends GraphicsProgram {  
  
 private static final double *CANVAS\_WIDTH* = 540; // Calculator width  
 private static final double *CANVAS\_HEIGHT* = 740; // Calculator height  
 private CalculatorLayout calculatorLayout; // Instantiate the Layout Object  
  
 private char opBuffer; // Stores the operator  
 private double operand1; // Stores the operand digits  
 private String result; // Stores the results  
   
 private boolean isFirstOp; // Checks if first operator  
 private boolean isPriorEquals; // Checks if it is prior to equal sign   
 private boolean isFirstPoint; // Checks if first decimal point  
 private boolean isDeletable; // Checks if it is deletable  
  
  
 public void run() {  
 setTitle("LBYCPEI Calculator");  
 setCanvasSize(*CANVAS\_WIDTH*, *CANVAS\_HEIGHT*);  
 calculatorLayout = new CalculatorLayout(getHeight());  
 add(calculatorLayout);  
 initBooleans();  
 addMouseListeners(); // Adds the program as both a MouseListener and MouseMotionListener to the canvas.  
 // (Refer to: https://cs.stanford.edu/people/eroberts/jtf/javadoc/complete/acm/program/GraphicsProgram.html)  
 }  
  
  
 public void mouseClicked(MouseEvent e) {  
 // This method runs everytime you click the mouse, thus it enables you to access the mouse events  
   
   
 GObject element = calculatorLayout.getElementAt(e.getX(), e.getY());  
 // getElementAt() Returns the topmost graphical object that contains the point (x, y), or null if no such object exists.  
 // Documentation : https://cs.stanford.edu/people/eroberts/jtf/javadoc/complete/acm/program/GraphicsProgram.html  
   
 if (element instanceof MyButton) {  
 String input = ((MyButton) element).getText(); // Gets the text associated with the button. e.g. C, CE, ⌫, ±, 0, 1,...,9, etc.  
 // e.g. input = 0   
   
 // I. Handle special cases: Clear Element, Clear All, and Delete  
 if (input.equals("CE ")) {  
 calculatorLayout.clearMainDisplay();  
 calculatorLayout.clearMemoElement(opBuffer);  
 System.*out*.println("Clear Element");  
 return;  
 }  
 if (input.equals("C")) {  
 calculatorLayout.clearMainDisplay();  
 calculatorLayout.clearMemoDisplay();  
 initBooleans();  
 System.*out*.println("Clear Called");  
 return;  
 }  
 if (input.equals("⌫") && isDeletable) { // ⌫ symbolizes delete  
 calculatorLayout.deleteOneCharacter();  
 System.*out*.println("Delete Called");  
 return;  
 }  
  
 // II. Handle arithmetic symbols and operations  
  
 char symbol = input.charAt(0); // Aiming for 0,1,2,3,4,5,6,7,8,9,.,±,=,+,-,x,÷  
  
 if (symbol == '±' && isDeletable) {  
 calculatorLayout.negateElement(opBuffer);  
 System.*out*.println("Negation Called");  
 return;  
 }  
  
 if ((symbol >= '0' && symbol <= '9') || symbol == '.') {  
 isDeletable = true;  
 if (symbol == '.') {  
 if (!isFirstPoint) {  
 return;  
 } else {  
 isFirstPoint = false;  
 }  
 }  
 if (isPriorEquals) {  
 calculatorLayout.clearMainDisplay();  
 isPriorEquals = false;  
 System.*out*.println("Digit: Prior Equals");  
 }  
 calculatorLayout.setMemoDisplay(symbol);  
 calculatorLayout.setMainDisplay(symbol);  
 System.*out*.println("Digit: Prior Not Equals");  
 return;  
 }  
  
  
 double operand2;  
 if (isOperator(symbol)) {  
 if (isFirstOp && !isPriorEquals) {  
 operand1 = Double.*parseDouble*(calculatorLayout.getMainDisplay());  
 calculatorLayout.setMemoDisplay(symbol);  
 opBuffer = symbol;  
 isFirstOp = false;  
 System.*out*.println("Operator: First Operation and Not prior equals");  
 } else if (isPriorEquals) {  
 calculatorLayout.setMemoDisplay(result + symbol);  
 opBuffer = symbol;  
 isFirstOp = false;  
 System.*out*.println("Operator: Prior equals!");  
 } else if (symbol == 's' || symbol == 'c' || symbol == 't' && !isFirstOp) {  
 operand1 = Double.*parseDouble*(calculatorLayout.getMainDisplay());  
 operand2 = MathHandler.*evaluate*(operand1, 0, opBuffer);  
 result = "" + operand2;  
 calculatorLayout.setMainDisplay(result);  
 } else {  
 operand2 = Double.*parseDouble*(calculatorLayout.getMainDisplay());  
 operand1 = MathHandler.*evaluate*(operand1, operand2, opBuffer);  
 result = "" + operand1;  
 result = result.contains(".") ? result.replaceAll("0\*$", "").replaceAll("\\.$", "") : result;  
 calculatorLayout.setMainDisplay(result);  
 calculatorLayout.setMemoDisplay(symbol);  
 isDeletable = false;  
 System.*out*.println("Operator: Second operator");  
 }  
 calculatorLayout.clearNumBuffer();  
 isFirstPoint = true;  
 }  
 if (symbol == '=') {  
 operand2 = Double.*parseDouble*(calculatorLayout.getMainDisplay());  
 operand1 = MathHandler.*evaluate*(operand1, operand2, opBuffer);  
 result = "" + operand1;  
 result = result.contains(".") ? result.replaceAll("0\*$", "").replaceAll("\\.$", "") : result;  
 calculatorLayout.setMainDisplay(result);  
 calculatorLayout.clearMemoDisplay();  
 initBooleans();  
 System.*out*.println("Equals: evaluated");  
 System.*out*.println("operand1 = " + operand1);  
 System.*out*.println("operand2 = " + operand2);  
 }  
 if (symbol == 's') {  
 calculatorLayout.clearMemoDisplay();  
 calculatorLayout.clearMainDisplay();  
 calculatorLayout.setMemoDisplay("sin " + operand1);  
 calculatorLayout.setMainDisplay("sin " + operand1);  
 }  
 if (symbol == 'c') {  
 calculatorLayout.clearMemoDisplay();  
 calculatorLayout.clearMainDisplay();  
 calculatorLayout.setMemoDisplay("cos " + operand1);  
 calculatorLayout.setMainDisplay("cos " + operand1);  
 }  
 if (symbol == 't') {  
 calculatorLayout.clearMemoDisplay();  
 calculatorLayout.clearMainDisplay();  
 calculatorLayout.setMemoDisplay("tan " + operand1);  
 calculatorLayout.setMainDisplay("tan " + operand1);  
 }  
 }  
 }  
  
 private void initBooleans() {  
 isFirstOp = true;  
 isPriorEquals = true;  
 isDeletable = false;  
 isFirstPoint = true;  
 }  
  
 private boolean isOperator(char symbol) {  
 return (symbol == '+' || symbol == '-' || symbol == '×' || symbol == '÷' || symbol == '^' || symbol == 's' || symbol == 'c' || symbol == 't');  
 }  
  
 // Solves java.lang.NoClassDefFoundError  
 public static void main(String[] args) {  
 (new App()).start(args);  
 }  
}

2. Lab Activity 1.2 “CalculatorLayout.java”

/\*  
 \* File: CalculatorLayout.java  
 \* ---------------------  
 \* This class is the layout class for a sample calculator app implementation  
 \* Author: Cobalt mkc  
 \* Date modified: July 22, 2019  
 \* Last Modified: Aug 5, 2022  
 \*/  
  
  
import acm.graphics.GCompound;  
import acm.graphics.GLabel;  
  
public class CalculatorLayout extends GCompound {  
  
 private static final int NROWS = 7; /\* Number of rows \*/  
 private static final int NCOLS = 4; /\* Number of columns \*/  
 private static final int MARGIN = 20;  
 private static final char[] labels = {  
 ' ', ' ', ' ', '^',  
 ' ', 'C', '⌫', '÷',  
 '7', '8', '9', '×',  
 '4', '5', '6', '-',  
 '1', '2', '3', '+',  
 '±', '.', '0', '='  
 };  
  
 private GLabel memoDisplay;  
 private GLabel mainDisplay;  
 private StringBuilder memoBuffer;  
 private StringBuilder numBuffer;  
 private static final String MAIN\_FONT = "SansSerif-bold-48";  
 private static final String MEMO\_FONT = "SansSerif-bold-28";  
  
  
 public CalculatorLayout(double height) {  
 double sqSize = height / (NROWS + 1);  
 numBuffer = new StringBuilder();  
 clearNumBuffer();  
 mainDisplay = new GLabel(numBuffer.toString(), MARGIN, MARGIN + 70);  
 mainDisplay.setFont(MAIN\_FONT);  
  
 memoBuffer = new StringBuilder();  
 memoDisplay = new GLabel("", MARGIN, 2 \* MARGIN);  
 memoDisplay.setFont(MEMO\_FONT);  
  
 add(mainDisplay);  
 add(memoDisplay);  
  
 int count = 0;  
 for (int i = 1; i < NROWS; i++) {  
 for (int j = 0; j < NCOLS; j++) {  
 double x = MARGIN + j \* sqSize;  
 double y = i \* sqSize + MARGIN;  
 MyButton myButton;  
  
 switch (count) {  
 case 0:  
 myButton = new MyButton(x, y, sqSize, sqSize, "sin" + labels[count++]);  
 break;  
 case 1:  
 myButton = new MyButton(x, y, sqSize, sqSize, "cos" + labels[count++]);  
 break;  
 case 2:  
 myButton = new MyButton(x, y, sqSize, sqSize, "tan" + labels[count++]);  
 break;  
 case 4:  
 myButton = new MyButton(x, y, sqSize, sqSize, "CE" + labels[count++]);  
 break;  
 default:  
 myButton = new MyButton(x, y, sqSize, sqSize, "" + labels[count++]);  
 break;  
 }  
 add(myButton);  
 }  
 }  
 }  
  
 /\* Sample Polymorphic Methods \*/  
 public void setMemoDisplay(char symbol) {  
 memoBuffer.append(symbol);  
 memoDisplay.setLabel(memoBuffer.toString());  
 }  
  
 public void setMemoDisplay(String input) {  
 memoBuffer.append(input);  
 memoDisplay.setLabel(memoBuffer.toString());  
 }  
  
 /\* Sample Polymorphic Methods \*/  
 public void setMainDisplay(char symbol) {  
 if (numBuffer.length() > 0 && numBuffer.charAt(0) == '0') { // Do not append on initial zero  
 numBuffer.setCharAt(0, symbol);  
 } else {  
 numBuffer.append(symbol);  
 }  
 mainDisplay.setLabel(numBuffer.toString());  
 }  
  
 public void setMainDisplay(String input) {  
 mainDisplay.setLabel(input);  
 }  
  
  
 public String getMainDisplay() {  
 return numBuffer.toString();  
 }  
  
 public void clearMainDisplay() {  
 clearNumBuffer();  
 mainDisplay.setLabel("0");  
 }  
  
 public void clearMemoDisplay() {  
 clearMemoBuffer();  
 memoDisplay.setLabel("");  
 }  
  
 public void clearNumBuffer() {  
 numBuffer.setLength(1);  
 numBuffer.setCharAt(0, '0');  
 }  
  
 public void clearMemoBuffer() {  
 memoBuffer.setLength(0);  
 }  
  
 public void clearMemoElement(char operation) {  
 int position = memoBuffer.lastIndexOf("" + operation);  
 memoBuffer.setLength(position + 1);  
 memoDisplay.setLabel(memoBuffer.toString());  
 }  
  
  
 public void negateElement(char operation) {  
 int position = memoBuffer.lastIndexOf("" + operation);  
 if (memoBuffer.charAt(position + 1) != '-') {  
 memoBuffer.insert(position + 1, '-');  
 } else {  
 memoBuffer.deleteCharAt(position + 1);  
 }  
 if (numBuffer.charAt(0) != '-') {  
 numBuffer.insert(0, '-');  
 } else {  
 numBuffer.deleteCharAt(0);  
 }  
 memoDisplay.setLabel(memoBuffer.toString());  
 mainDisplay.setLabel(numBuffer.toString());  
 }  
  
 public void deleteOneCharacter() {  
 if (memoBuffer.length() == 0 || numBuffer.length() == 0) {  
 return;  
 }  
 memoBuffer.setLength(memoBuffer.length() - 1);  
 numBuffer.setLength(numBuffer.length() - 1);  
 memoDisplay.setLabel(memoBuffer.toString());  
 mainDisplay.setLabel(numBuffer.toString());  
 }  
}

3. Lab Activity 1.3 “MyButton.java”

/\*  
 \* File: MyButton.java (!!! TO BE MODIFIED !!! In the YOUR CODE HERE parts)  
 \* ---------------------  
 \* This class is a custom button class for a sample calculator app implementation  
 \* Author: Cobalt mkc  
 \* Date modified: July 22, 2019  
 \* Last modified: Aug 3, 2022  
 \*/  
  
  
import acm.graphics.GCompound; // https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/acm/graphics/GCompound.html   
import acm.graphics.GLabel; // https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/acm/graphics/GLabel.html   
import acm.graphics.GRect; // https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/acm/graphics/GRect.html  
  
  
public class MyButton extends GCompound {  
  
 private static final String *FONT* = "SansSerif-bold-64";  
 private GRect key; // https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/acm/graphics/GRect.html  
 private GLabel keyText; // https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/acm/graphics/GLabel.html   
  
 public MyButton(double x, double y, double width, double height, String text) {  
 // This method draws the Button object which is composed of GRect and GLabel objects  
 key = new GRect(x, y, width, height);  
 keyText = new GLabel(text);  
 keyText.setFont("SansSerif-40");  
  
 add(key);  
 add(keyText, key.getCenterX() - (keyText.getWidth() / 2), key.getCenterY());  
 }  
  
 public String getText() {  
 // This method returns the text associated with the button  
 return keyText.getLabel();  
 }  
  
 private double getCenterX(double width, double labelWidth) {  
 // No need to modify this method. You may use this to center the GLabel in the button  
 return (width - labelWidth) / 2.0f;  
 }  
  
 private double getCenterY(double height, double ascent) {  
 // No need to modify this method. You may use this to center the GLabel in the button   
 return (ascent + (height - ascent) / 2.0f);  
 }  
  
 public void setText(String text) {  
 // This method modifies the keyText of the button  
 keyText.setLabel(text);  
 }  
}

4. Lab Activity 1.4 “MathHandler.java”

/\*  
 \* File: MathHandler.java (!!! TO BE MODIFIED !!! In the YOUR CODE HERE parts)  
 \* ---------------------  
 \* This class is a helper class for a sample calculator app implementation  
 \* Author: Cobalt mkc  
 \* Date created: July 22, 2019  
 \* Last modified: Aug 3, 2022  
 \*/  
  
  
  
  
public class MathHandler {  
  
 public static int evaluate(int operand1, int operand2, char operator) {  
 // No modifications needed here!!!  
 int result;  
 switch (operator) {  
 case '+':  
 result = *add*(operand1, operand2);  
 break;  
 case '-':  
 result = *subtract*(operand1, operand2);  
 break;  
 case '×':  
 result = *multiply*(operand1, operand2);  
 break;  
 case '÷':  
 result = *divide*(operand1, operand2);  
 break;  
 default:  
 result = 0;  
 }  
 return result;  
 }  
  
 public static double evaluate(double operand1, double operand2, char operator) {  
 // No modifications needed here!!!  
 double result;  
 switch (operator) {  
 case '+':  
 result = *add*(operand1, operand2);  
 break;  
 case '-':  
 result = *subtract*(operand1, operand2);  
 break;  
 case '×':  
 result = *multiply*(operand1, operand2);  
 break;  
 case '÷':  
 result = *divide*(operand1, operand2);  
 break;  
 case '^':  
 result = *exponent*(operand1, operand2);  
 System.*out*.println(result);  
 break;  
 case 's':  
 result = *sin*(operand1);  
 System.*out*.println(result);  
 break;  
 case 'c':  
 result = *cos*(operand1);  
 System.*out*.println(result);  
 break;  
 case 't':  
 result = *tan*(operand1);  
 System.*out*.println(result);  
 break;  
 default:  
 result = 0;  
 }  
 return result;  
 }  
  
  
 */\*\*  
 \* Returns the sum of operand1 and operand2  
 \** ***@param*** *operand1  
 \** ***@param*** *operand2  
 \** ***@return*** *\*/* public static int add(int operand1, int operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 + operand2;  
 }  
  
 public static double add(double operand1, double operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 + operand2;  
 }  
  
  
 public static int subtract(int operand1, int operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 - operand2;  
 }  
  
 public static double subtract(double operand1, double operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 - operand2;  
 }  
  
  
 public static int divide(int operand1, int operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 / operand2;  
 }  
  
 public static double divide(double operand1, double operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 / operand2;  
 }  
  
 public static int multiply(int operand1, int operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 \* operand2;  
 }  
  
 public static double multiply(double operand1, double operand2) {  
 /\* YOUR CODE HERE \*/  
 return operand1 \* operand2;  
 }  
  
 public static double sin(double operand1) {  
 return Math.*sin*(operand1);  
 }  
  
 public static double cos(double operand1) {  
 return Math.*cos*(operand1);  
 }  
  
 public static double tan(double operand1) {  
 return Math.*tan*(operand1);  
 }  
  
 public static double exponent(double operand1, double operand2) {  
 return Math.*pow*(operand1, operand2);  
 }  
}