|  |
| --- |
| Assignment 1: Calculator |
| CST8221 – JAP Due: July 5th, 2011 Submitted: June 21, 2011 |
| By: William Collins 040652633 For: Svillen Ranev |

**Contents**

* CalculatorView.java
* Testing

/\*\*

\* File name: CalculatorView.java

\* Author: William Collins, 040652633

\* Course: CST8221 – JAP

\* Assignment: 1

\* Date: June 20, 2011

\* Professor: Svillen (the Villain) Ranev

\* Purpose: Controls the main functionality of the calculator

\* Class List: CalculatorView - The main panel display

\* Controller - handles events generated by the calculator buttons

\*

\*/

**package** Calculator;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**import** javax.swing.border.BevelBorder;

/\*\*

\* The User Interface that will display the calculator to the user.

\* **@author** William Collins (040652633)

\* **@version** 1.1

\* **@see** Calculator

\* **@since** 1.6

\*/

**public** **class** CalculatorView **extends** JPanel{

/\*\*

\* The field that will display the calculator results

\*/

**private** JTextField display;

/\*\*

\* An error indicator used to indicate an error. Obvious, right?

\*/

**private** JLabel error;

/\*\*

\* The decimal button on the calculator. Who would have guessed it?

\*/

**private** JButton dotButton;

/\*\*

\* The model that will do all of the calculations

\*/

**private** CalculatorModel model;

//I had originally planned on using constants for all button names, but the strings themselves

//seem to be more descriptive themselves, and aren't apt to change, so I took out the constants

**private** **static** **final** String *BACKSPACE\_TOOLTIP* = "Backspace (Alt-B)";

//These are the names that will be used to create the main calculator butttons

**private** **final** String buttonNames[] = {

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

"4", "5", "6", "\*", "\u221A",

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

"0", ".", "+/-", "+", "="

};

**public** CalculatorView(){

//Set up our observable class, and add this view as an observer so that we can receive updates

model = **new** CalculatorModel();

Controller handler = **new** Controller();

handler.setEnabled(**true**);

//Get our panel ready

setLayout(**new** BorderLayout());

setBorder(BorderFactory.*createEmptyBorder*(0, 2, 5, 2));

//Create our panels

JPanel northPanel = **new** JPanel(**new** FlowLayout(FlowLayout.*CENTER*, 5, 0)); //The panel that will hold everything above the main calculator buttons

JPanel displayPanel = **new** JPanel(); //The first row of components at the top of the calculator

JPanel radioPanel = **new** JPanel(); // The panel for the row of components under the display panel

//Set up our awesome error label and add it to the view

error = **new** JLabel();

error.setOpaque(**true**);

error.setForeground(Color.*BLACK*);

error.setBackground(Color.*GREEN*);

error.setPreferredSize(**new** Dimension(20, 20));

//Set up our extreme text display

display = **new** JTextField("0.0", 15);

display.setEditable(**false**);

display.setHorizontalAlignment(JTextField.*RIGHT*);

display.setBackground(Color.*WHITE*);

//Set up our super awesome backspace button

JButton backspace = **new** JButton("<<"); //The backspace button

backspace.setSize(20, 20);

backspace.setForeground(Color.*BLACK*);

backspace.setBorder(BorderFactory.*createEmptyBorder*());

backspace.setToolTipText(*BACKSPACE\_TOOLTIP*);

backspace.setMnemonic(KeyEvent.*VK\_B*);

backspace.setOpaque(**false**);

backspace.setContentAreaFilled(**false**);

//Add the display information to the display panel

displayPanel.add(error);

displayPanel.add(display);

displayPanel.add(backspace);

//Time to set up the radio buttons

ButtonGroup radioGroup = **new** ButtonGroup(); //Controls how the radio buttons behave

JRadioButton intButton = **new** JRadioButton("Integer"); //The button specifying that we are calculating integers

JRadioButton floatButton = **new** JRadioButton("Float"); //The button specifying that we are calculating floating point numbers

intButton.setBackground(Color.*CYAN*);

floatButton.setBackground(Color.*PINK*);

floatButton.setSelected(**true**);

floatButton.setPreferredSize(intButton.getPreferredSize());

intButton.setVerticalAlignment(SwingConstants.*CENTER*);

radioGroup.add(intButton);

radioGroup.add(floatButton);

radioPanel.add(intButton);

radioPanel.add(floatButton);

radioPanel.setBorder(BorderFactory.*createCompoundBorder*(BorderFactory.*createLineBorder*(Color.*BLUE*, 2), BorderFactory.*createLineBorder*(Color.*BLUE*)));

radioPanel.setPreferredSize(**new** Dimension(215, 40));

//Set up the action listeners for appropriate buttons

backspace.addActionListener(**new** Controller());

intButton.addActionListener(**new** Controller());

floatButton.addActionListener(**new** Controller());

//Add the display and the radio button panel to the NORTH panel

northPanel.add(displayPanel);

northPanel.add(radioPanel);

//Calculate the dimension for the NORTH Panel

Dimension sizeOfNorth = **new** Dimension(260, 73);

northPanel.setPreferredSize(sizeOfNorth);

//Finally add this sweet new panel

add(northPanel, BorderLayout.*NORTH*);

//Now add the array of buttons

JPanel buttonPanel = **new** JPanel(**new** GridLayout(4, 5, 5, 5)); //The panel that will hold the main calculator buttons

**for** (String s: buttonNames){

addButton(buttonPanel, s, handler);

}

add(buttonPanel, BorderLayout.*CENTER*);

}

/\*\*

\* **@param** c: The container that the button will be placed into

\* **@param** s: The name of the button to be added

\* **@param** handler: An action listener that will handle the button's events

\*/

**private** **void** addButton(Container c, String s, AbstractAction handler){

JButton newButton = **new** JButton(s); //the new button to be added

newButton.setBorder(BorderFactory.*createBevelBorder*(BevelBorder.*RAISED*));

//Take care of certain special buttons

**if** (s.equals("C")){

newButton.setForeground(Color.*BLACK*);

newButton.setBackground(Color.*RED*);

}

**else** **if** (s.equals("=")){

newButton.setForeground(Color.*BLACK*);

newButton.setBackground(Color.*YELLOW*);

}

**else** {

newButton.setForeground(Color.*PINK*);

}

newButton.addActionListener(handler);

c.add(newButton);

**if** (s.equals(".")){

dotButton = newButton;

}

}

/\*\*

\* The controller will handle events from the view and send information to the model for processing

\* **@author** William Collins (040652633)

\* **@version** 1.1

\* **@see** Calculator

\* **@since** 1.6

\*/

**private** **class** Controller **extends** AbstractAction {

@Override

**public** **void** actionPerformed(ActionEvent e) {

String command = e.getActionCommand();

**if** (!(e.getSource() **instanceof** AbstractButton)){

**return**;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* WE HAVE AN ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** (model.isError()){

**if** (!command.equals("C") && !command.equals("Float") && !command.equals("Integer")){

**return**;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ARITHMETIC

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** (command.equals("+") || command.equals("-") || command.equals("\*") || command.equals("/") || command.equals("P") || command.equals("\u221A")){

model.setOperand(display.getText());

model.setOperation(command);

//display.setText("0");

**if** (command.equals("\u221A")){

display.setText(model.getResult());

}

model.setReadyForNewNumber(**true**);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EQUALS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals("=")){

model.setOperand(display.getText());

display.setText(model.getResult());

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PLUS/MINUS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals("+/-")){

String temp = display.getText();

//If it is already negative, make it positive

**if** (display.getText().startsWith("-")){

temp = temp.substring(1);

display.setText(temp);

}

//If it is positive, make it negative

**else** {

temp = "-" + temp;

display.setText(temp);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CLEAR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals("C")){

model.setError(**false**);

model.clear();

model.setOperation(command);

display.setText(model.getResult());

error.setBackground(Color.*GREEN*);

error.setText("");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FLOAT && INTEGER

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals("Float") || command.equals("Integer")){

**if** (model.getMode().equals(command)){

**return**;

}

model.setMode(command);

dotButton.setEnabled(command.equals("Float"));

//If the number is zero, we want 0.0 when in floating point

**try** {

display.setText(String.*valueOf*(Float.*valueOf*(display.getText()).intValue()));

**if** (Float.*valueOf*(display.getText()) == 0.0){

display.setText(model.getZero());

}

}**catch** (NumberFormatException nfx){

//Do nothing

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BACKSPACE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals("<<")){

**if** ((display.getText().startsWith("-") && display.getText().length() == 2) || Float.*valueOf*(display.getText()) == 0.0 || display.getText().length() == 1){

display.setText(model.getZero());

model.setReadyForNewNumber(**true**);

}

**else** {

display.setText(display.getText().substring(0, display.getText().length()-1));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DECIMAL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** **if**(command.equals(".")){

//If we are starting a new number, then let's put a decimal place

**if** (model.isReadyForNewNumber()){

display.setText(command);

model.setReadyForNewNumber(**false**);

}

//In the case that we are not starting a new number, let's make

//sure no other decimal places exist in the text

**else** **if** (!display.getText().contains(command)){

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

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NUMBERS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**else** {

//This is when we enter the negative sign on a zero

//if (display.getText().startsWith("-") && model.isReadyForNewNumber()){

// display.setText("-" + command);

// model.setReadyForNewNumber(false);

//}

**if** (model.isReadyForNewNumber()){

display.setText(command);

model.setReadyForNewNumber(**false**);

}

**else** {

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

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CALCULATIONS GAVE

\* US AN ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** (model.isError()){

error.setBackground(Color.*RED*);

error.setText("E");

}

} //End of ActionPerformed method

}

}

TESTING

This assignment is an easy combination of UI testing and functionality testing. UI testing is to be done by inspection and comparison to the graceful Calculator UI provided by your highness S~R. Testing of the events generated by the calculator buttons is done by inspection. A button click produces relevant output to the display.

Functionality testing is also made easy, because we have to conform to the functionality specified in the assignment. The following test cases must prove successful:

1) 2 + 2 = must display 4 in Integer mode and 4.0 in Float mode.

2) 2.5 [Integer] + 2 = must display 4

3) 2 \* 2 = + 4 = must display 8 (or 8.0)

4) 2\*= must display 4 (or 4.0)

5) 2\*/= must display 1(or 1.0)

6) 2/\*= must display 4(or 4.0)

7) 2 + 2 = must display 4 or 4.0. If 2 is clicked after =, must display 2

8) 12 [<<]+2 = must display 3

9) 2 [+/-] must display -2 or -2.0. If followed by [+/-], must display 2 or 2.0

10) -123[<<][<<] must display -1

11) -123.0 [<<][<<] must display -123

12) -123.0 [<<][<<][<<][<<][<<] must display 0.0

13) 4 √ must display 2

14) -4 √ must display -- and the label must display E on red background. After

error, the calculator must not respond to any button except C, Integer,

and Float. Pressing C must display 0 or 0.0 depending on the current

mode of operation, and change the label color to green with no text.

15) [Integer] must disable the . button

16) 16 √ √ must display 2 or 2.0

17) 16 √ √ [nteger] √ must display 1

18) -4 / 2 = must display -2 or -2.0

19) 2.5 [nteger] must display 2

20) 2 [Float] must display 2

21) [Integer] 2 [C] must display 0

22) [Float] 2 [C] must display 0.0

23) 4 P .5 = must display 2.0

24) -4 P .5 = must display error

25) 1 / 0 = must display error (see 13).

26) 0 / 0 = must display error (see 13)

Thanks for the easy test plan S~R!