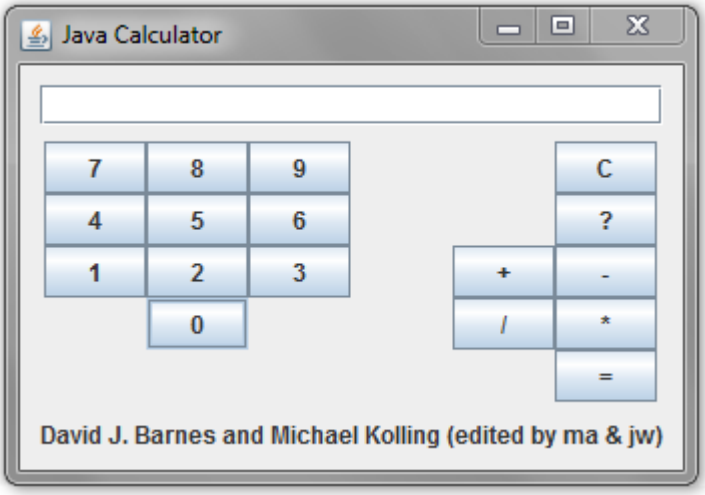
# Laboratory 5: Fun with Calculators – Week 1

([http://people.f4.htw-berlin.de/~weberwu/info2/labs/Exer5.shtml](http://people.f4.htw-berlin.de/~weberwu/info2/labs/Exer5.shtml))

## 1. Implement the missing functionality for single digit numbers and only one operator discovered in the pre-lab.

At first we add two new buttons for "*" and "/" in the class UserInterface. The method makeFrame() creates the existing buttons, so we simply add the new ones and some blank ones (`buttonPanel.add(new JLabel(" "));`) for a better look:




Now the new buttons need a function. In the method actionPerformed() we tell the new buttons to run a method in class CalcEngine:

```
} else if (command.equals("/")) {
    calc.divide();
} else if (command.equals("*")) {
    calc.times();
```

The methods in class CalcEngine simply add the right operator to the calculation:

```
public void divide() {
    applyOperator('/');
}

public void times() {
    applyOperator('*');
}
```

The last step is to modify the method calculateResult(), which deals with the actual calculation formula:

```
case '*':
    displayValue = leftOperand * displayValue;
    haveLeftOperand = true;
    leftOperand = displayValue;
    break;
case '/':
    displayValue = leftOperand / displayValue;
    haveLeftOperand = true;
    leftOperand = displayValue;
    break;
```

Now the code should be complete for the new operators. We test the new buttons and everything works fine!

## 2. Extend your integer calculator to include buttons for entering in the hexadecimal digits. Make the calculator do its calculations and display in hexadecimal notation.

At first we have to implement the buttons. Since the addButton() method implements the buttons anonymously we can't access them later on at exercise 3. So we add them manually

```
aButton = new JButton("A");
buttonPanel.add(aButton);
aButton.addActionListener(this);
```

now we need to say, what the actionPerformed() method has to do with these buttons. So we added them to the if condition:

```
if (command.equals("0") || command.equals("1") || command.equals("2")
                        || command.equals("3") || command.equals("4")
                        || command.equals("5") || command.equals("6")
                        || command.equals("7") || command.equals("8")
                        || command.equals("9") || command.equals("A")
                        || command.equals("B") || command.equals("C")
                        || command.equals("D") || command.equals("E")
                        || command.equals("F"))
```

and save the buttons value as an hexadecimal number

```
int number = Integer.parseInt(command, 16);
```

So now internally we calculate just like with any other kind of number.
To print the result as a hex number, we go to the redisplay() method and change it to

```
display.setText(Integer.toHexString(calc.getDisplayValue()));
```

## 3. Integrate a checkbox for switching between decimal and hexadecimal formats. Do not show the hexadecimal digits (or grey them out) when you have the calculator in decimal mode.

To integrate a checkbox we have to declare a new field of type JcheckBox

```
private JCheckBox checkBox;
```

in the makeFrame() method of the UserInterface class we now initialize the variable and give the checkBox a name („hexa“) and make it deselected

```
checkBox = new JCheckBox("hexa", false);
```

then add it to the buttonPanel and add the actionListener to it.

```
buttonPanel.add(checkBox);
checkBox.addActionListener(this);
```

In the actionPerformed() method we check if the checkbox got selected or deselected with the isSelected() method of the JcheckBox class. If so we enable all buttons, if not we disable them.

```
else if(command.equals("hexa")) {
        if(checkBox.isSelected()) {
                aButton.setEnabled(true);
                bButton.setEnabled(true);
                cButton.setEnabled(true);
                dButton.setEnabled(true);
                eButton.setEnabled(true);
                fButton.setEnabled(true);
        } else {
                aButton.setEnabled(false);
                bButton.setEnabled(false);
                cButton.setEnabled(false);
                dButton.setEnabled(false);
                eButton.setEnabled(false);
                fButton.setEnabled(false);
        }
}
```

## 4. What test cases do you need to test your calculator?

Test1
Input: Hexa off, 5+5
Output: 10
Test2
Input: Hexa off, 10-5
Output: 5
Test3
Input: Hexa off, 5*5
Output: 25
Test4:
Input: Hexa off, 10/5
Output: 2

Test5:
Input: Hexa on, A+5
Output: F
Test6:
Input: Hexa on, B-3
Output: 8
Test7:
Input: Hexa on, C*5
Output: 3C
Test8:
Input: Hexa on, E/2
Output: 7

In this lab we learned to study the code of an existing program and to modify it. We also learned how to add buttons to a GUI.
All in all the lab took us round about 4 hours.

## Source Code:

Class ClacEngine:

```
/**
 * The main part of the calculator doing the calculations.
 *
 * @author  David J. Barnes and Michael Kolling (edited by ma & jw)
 * @version 2012.11.13
 */
public class CalcEngine
{
    // The calculator's state is maintained in three fields:
    //     buildingDisplayValue, haveLeftOperand, and lastOperator.

    // Are we already building a value in the display, or will the
    // next digit be the first of a new one?
    private boolean buildingDisplayValue;
    // Has a left operand already been entered (or calculated)?
    private boolean haveLeftOperand;
    // The most recent operator that was entered.
    private char lastOperator;

    // The current value (to be) shown in the display.
    private int displayValue;
    // The value of an existing left operand.
    private int leftOperand;

    /**
     * Create a CalcEngine.
     */
    public CalcEngine()
    {
        clear();
    }

    /**
     * @return The value that should currently be displayed
     * on the calculator display.
     */
    public int getDisplayValue()
    {
        return displayValue;
    }

    /**
     * A number button was pressed.
     * Either start a new operand, or incorporate this number as
     * the least significant digit of an existing one.
     * @param number The number pressed on the calculator.
     */
    public void numberPressed(int number)
    {
        if(buildingDisplayValue && number < 10) {
            // Incorporate this digit.
            displayValue = displayValue*10 + number;
        }
        else {
```

```
            // Start building a new number.
            displayValue = number;
            buildingDisplayValue = true;
        }
    }

    /**
     * The 'plus' button was pressed.
     */
    public void plus()
    {
        applyOperator('+');
    }

    /**
     * The 'minus' button was pressed.
     */
    public void minus()
    {
        applyOperator('-');
    }

    public void divide() {
       applyOperator('/');
    }

    public void times() {
       applyOperator('*');
    }
    /**
     * The '=' button was pressed.
     */
    public void equals()
    {
        // This should completes the building of a second operand,
        // so ensure that we really have a left operand, an operator
        // and a right operand.
        if(haveLeftOperand &&
                lastOperator != '?' &&
                buildingDisplayValue) {
            calculateResult();
            lastOperator = '?';
            buildingDisplayValue = false;
        }
        else {
            keySequenceError();
        }
    }

    /**
     * The 'C' (clear) button was pressed.
     * Reset everything to a starting state.
     */
    public void clear()
    {
        lastOperator = '?';
        haveLeftOperand = false;
```

```
        buildingDisplayValue = false;
        displayValue = 0;
    }

    /**
     * @return The title of this calculation engine.
     */
    public String getTitle()
    {
        return "Java Calculator Plus";
    }

    /**
     * @return The author of this engine.
     */
    public String getAuthor()
    {
        return "David J. Barnes and Michael Kolling (edited by ma & jw)";
    }

    /**
     * @return The version number of this engine.
     */
    public String getVersion()
    {
       return "Version 1.0";
    }

    /**
     * Combine leftOperand, lastOperator, and the
     * current display value.
     * The result becomes both the leftOperand and
     * the new display value.
     */
    private void calculateResult()
    {
        switch(lastOperator) {
            case '+':
                displayValue = leftOperand + displayValue;
                haveLeftOperand = true;
                leftOperand = displayValue;
                break;
            case '-':
                displayValue = leftOperand - displayValue;
                haveLeftOperand = true;
                leftOperand = displayValue;
                break;
            case '*':
                displayValue = leftOperand * displayValue;
                haveLeftOperand = true;
                leftOperand = displayValue;
                break;
            case '/':
                displayValue = leftOperand / displayValue;
                haveLeftOperand = true;
                leftOperand = displayValue;
                break;
```

```
            default:
                keySequenceError();
                break;
        }
    }

    /**
     * Apply an operator.
     * @param operator The operator to apply.
     */
    private void applyOperator(char operator)
    {
        // If we are not in the process of building a new operand
        // then it is an error, unless we have just calculated a
        // result using '='.
        if(!buildingDisplayValue &&
                    !(haveLeftOperand && lastOperator == '?')) {
            keySequenceError();
            return;
        }

        if(lastOperator != '?') {
            // First apply the previous operator.
            calculateResult();
        }
        else {
            // The displayValue now becomes the left operand of this
            // new operator.
            haveLeftOperand = true;
            leftOperand = displayValue;
        }
        lastOperator = operator;
        buildingDisplayValue = false;
    }

    /**
     * Report an error in the sequence of keys that was pressed.
     */
    private void keySequenceError()
    {
        System.out.println("A key sequence error has occurred.");
        // Reset everything.
        clear();
    }
}
```

Class Claculator:

```
/**
 * The main class of a simple calculator. Create one of these and you'll
 * get the calculator on screen.
 *
 * @author David J. Barnes and Michael Kolling (edited by ma & jw)
 * @version 2012.11.13
 */
public class Calculator
{
    private CalcEngine engine;
```

```
    private UserInterface gui;

    /**
     * Create a new calculator and show it.
     */
    public Calculator()
    {
        engine = new CalcEngine();
        gui = new UserInterface(engine);
    }

    public static void main(String args[]) {
      new Calculator();
    }

    /**
     * In case the window was closed, show it again.
     */
    public void show()
    {
        gui.setVisible(true);
    }
}
```

Class UserInterface:

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

/**
 * A graphical user interface for the calculator. No calculation is being done
 * here. This class is responsible just for putting up the display on screen. It
 * then refers to the "CalcEngine" to do all the real work.
 *
 * @author David J. Barnes and Michael Kolling (edited by ma & jw)
 * @version 2012.11.13
 */

public class UserInterface implements ActionListener {
       private CalcEngine calc;
       private boolean showingAuthor;

       private JFrame frame;
       private JTextField display;
       private JLabel status;

       JButton aButton;
       JButton bButton;
       JButton cButton;
       JButton dButton;
       JButton eButton;
       JButton fButton;

       private JCheckBox checkBox;
```

```
/**
 * Create a user interface.
 *
 * @param engine
 *            The calculator engine.
 */
public UserInterface(CalcEngine engine) {
calc = engine;
showingAuthor = true;
makeFrame();
frame.setVisible(true);
}

/**
 * Set the visibility of the interface.
 *
 * @param visible
 *            true if the interface is to be made visible, false otherwise.
 */
public void setVisible(boolean visible) {
frame.setVisible(visible);
}

/**
 * Make the frame for the user interface.
 */
private void makeFrame() {
frame = new JFrame(calc.getTitle());

checkBox = new JCheckBox("hexa", false);

JPanel contentPane = (JPanel) frame.getContentPane();
contentPane.setLayout(new BorderLayout(8, 8));
contentPane.setBorder(new EmptyBorder(10, 10, 10, 10));

display = new JTextField();
contentPane.add(display, BorderLayout.NORTH);

JPanel buttonPanel = new JPanel(new GridLayout(6, 6));
addButton(buttonPanel, "7");
addButton(buttonPanel, "8");
addButton(buttonPanel, "9");
buttonPanel.add(new JLabel(" "));
buttonPanel.add(new JLabel(" "));
addButton(buttonPanel, "clear");

addButton(buttonPanel, "4");
addButton(buttonPanel, "5");
addButton(buttonPanel, "6");
buttonPanel.add(new JLabel(" "));
buttonPanel.add(new JLabel(" "));
addButton(buttonPanel, "?");

addButton(buttonPanel, "1");
addButton(buttonPanel, "2");
```

```
        addButton(buttonPanel, "3");
        buttonPanel.add(new JLabel(" "));
        addButton(buttonPanel, "+");
        addButton(buttonPanel, "-");

        buttonPanel.add(new JLabel(" "));
        addButton(buttonPanel, "0");
        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(new JLabel(" "));
        addButton(buttonPanel, "/");
        addButton(buttonPanel, "*");

        aButton = new JButton("A");
        buttonPanel.add(aButton);
        aButton.addActionListener(this);
        aButton.setEnabled(false);

        bButton = new JButton("B");
        buttonPanel.add(bButton);
        bButton.addActionListener(this);
        bButton.setEnabled(false);

        cButton = new JButton("C");
        buttonPanel.add(cButton);
        cButton.addActionListener(this);
        cButton.setEnabled(false);

        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(checkBox);
        checkBox.addActionListener(this);

        dButton = new JButton("D");
        buttonPanel.add(dButton);
        dButton.addActionListener(this);
        dButton.setEnabled(false);

        eButton = new JButton("E");
        buttonPanel.add(eButton);
        eButton.addActionListener(this);
        eButton.setEnabled(false);

        fButton = new JButton("F");
        buttonPanel.add(fButton);
        fButton.addActionListener(this);
        fButton.setEnabled(false);

        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(new JLabel(" "));
        addButton(buttonPanel, "=");

        contentPane.add(buttonPanel, BorderLayout.CENTER);

        status = new JLabel(calc.getAuthor());
        contentPane.add(status, BorderLayout.SOUTH);
```

```
frame.pack();
}

/**
 * Add a button to the button panel.
 *
 * @param panel
 *            The panel to receive the button.
 * @param buttonText
 *            The text for the button.
 */
private void addButton(Container panel, String buttonText) {
JButton button = new JButton(buttonText);
button.addActionListener(this);
panel.add(button);
}

/**
 * An interface action has been performed. Find out what it was and handle
 * it.
 *
 * @param event
 *            The event that has occured.
 */
public void actionPerformed(ActionEvent event) {
String command = event.getActionCommand();

if (command.equals("0") || command.equals("1") || command.equals("2")
              || command.equals("3") || command.equals("4")
              || command.equals("5") || command.equals("6")
              || command.equals("7") || command.equals("8")
              || command.equals("9") || command.equals("A")
              || command.equals("B") || command.equals("C")
              || command.equals("D") || command.equals("E")
              || command.equals("F")) {
       int number;
       if(checkBox.isSelected()) {
              number = Integer.parseInt(command, 16);
       } else {
              number = Integer.parseInt(command);
       }
       calc.numberPressed(number);
} else if (command.equals("+")) {
       calc.plus();
} else if (command.equals("-")) {
       calc.minus();
} else if (command.equals("/")) {
       calc.divide();
} else if (command.equals("*")) {
       calc.times();
} else if (command.equals("=")) {
       calc.equals();
} else if (command.equals("clear")) {
       calc.clear();
} else if (command.equals("?")) {
       showInfo();
```

```
	} else if(command.equals("hexa")) {
		if(checkBox.isSelected()) {
			aButton.setEnabled(true);
			bButton.setEnabled(true);
			cButton.setEnabled(true);
			dButton.setEnabled(true);
			eButton.setEnabled(true);
			fButton.setEnabled(true);
		} else {
			aButton.setEnabled(false);
			bButton.setEnabled(false);
			cButton.setEnabled(false);
			dButton.setEnabled(false);
			eButton.setEnabled(false);
			fButton.setEnabled(false);
		}
	}
	// else unknown command.

	redisplay();
	}

	/**
	 * Update the interface display to show the current value of the calculator.
	 */
	private void redisplay() {
	if(checkBox.isSelected()) {
		display.setText(Integer.toHexString(calc.getDisplayValue()));
	} else {
		display.setText("" + calc.getDisplayValue());
	}
	}

	/**
	 * Toggle the info display in the calculator's status area between the
	 * author and version information.
	 */
	private void showInfo() {
	if (showingAuthor)
		status.setText(calc.getVersion());
	else
		status.setText(calc.getAuthor());

	showingAuthor = !showingAuthor;
	}
}
```