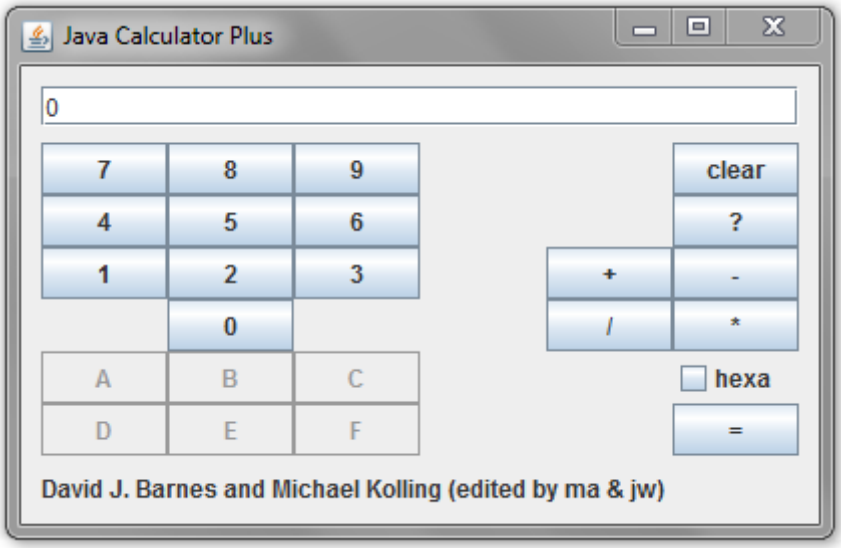
Laboratory 7: Fun with Calculators – Week 2

(http://people.f4.htw-berlin.de/~weberwu/info2/labs/Exer7.shtml)

1. Make a copy of your Calculator. Make sure that it works before you begin!

We copy class CalcEngine, Calculator and UserInterface from our previous lab 5 into our new lab 7. A short test shows that everything works just like before!



2. Rework it to accept a long String of single digits separated by operators that have precedence. The bored may use multi-digit numbers and floating-point or scientific notation, if they please.

For this we introduced a new JCheckBox to activate the String support. In the actionPerformed method, if the "=" button got pressed we check if the checkbox for string got checked, if so we read out the textfield and pass it to the evaluateInfix method of our calc engine.

```
	} else if (command.equals("=")) {
			if(checkBoxString.isSelected()) {
				calc.evaluateInfix(display.getText(),
				checkBoxHexa.isSelected());
			}else {
				calc.equals();
			}
	}
```

in CalcEngine:

```
	public void evaluateInfix(String ifx, boolean hexa) {
		try {
			displayValue = pf.evaluate(pf.infixToPostfix(ifx), hexa);
		}catch(Exception e) {
			//keySequenceError();
			System.out.println("evaluation from infix error");
		}
	}
```

Here we simply use the evaluate method and infixToPostfix method to calculate the String. The result gets stored in the displayValue and the next redisplay() call shows this in the calculator.

3. Once you get a String input, add in calls to convert this expression to postfix and evaluate the postfix when = is pressed. Presto, your calculator now takes care of operator priorities, like magic! Just a little bit of mathematical thought, and you can introduce new functionality!

This was already done in exercise 2 above.

4. What will be good test cases for the precedence of * over +? Find 15 good test cases and try them out, documenting the results.

1. 3*5+14=29
2. 3+5*14=73
3. 1+2*2-1=4
4. 20/4+6=11
5. 20^2/4=100

6. 90*1+9=99
7. 90+1*9=819
8. 1*1+1=2
9. 1+1*1=2
10. 5*5+5=30
11. 5+5*5?30
12. 12*1/12-1=0
13. 4+3*4=28
14. 4*3+4=16
15. 1+1*1+1*1+1= 4

5. Check that this still works with hexadecimal numbers.

This did not work with hexa decimal numbers, because my evaluate method and infixToPostfix method was not able to handle multi-digit numbers. Since hexa decimal numbers were parsed to decimal numbers, so an F (hexa) would be stored as 15. 15 passed into the infixToPostfix() caused an error. So I had to modify my Postfix class in order to be able to evaluate multi-digit numbers.

I made a new class called MultiDigitPostfix and pasted the methods from Postfix and modified them.

Now the infixToPostfix method of MultiDigitNumber returns a postfix String with spaces between each token.

2*3+4*(5+5) -> 2 3 * 4 5 5 + * +

Now evaluate method is able to take multi-digit numbers. But it also is able to take hexa decimal numbers in the postfix String, when the Boolean “hexa” in the parameters is true:

```
if(hexa) {
	rhs = Integer.parseInt(stack.pop().toString(), 16);
	lhs = Integer.parseInt(stack.pop().toString(), 16);
}else {
	rhs = Integer.parseInt(stack.pop().toString());
	lhs = Integer.parseInt(stack.pop().toString());
}
```

6. Now include the exponentiation operator ^ (2^4 = 16). It has a higher priority than either multiplication or division.

To be able to check the priority of two characters I made a method called higherPriority(char c, char c2):

```
private boolean higherPriority(char c, char c2) {
	if(c == '^' && c2 != '^') {
		return true;
	}else if((c == '*' || c == '/') && (c2 == '+' || c2 == '-')) {
		return true;
	}else {
		return false;
	}
}
```

And implemented that in my infixToPostfix() method.

In my evaluate() method I now added the "^" operator:

```
} else if(token.equals("^")) {
	result = (int) Math.pow(lhs, rhs);
}
```

In this lab we learned to add more function to our existing code.

All in the entire lab report took us about 5 hours.

Source code:

CalcEngine:

```
/**
 * 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;

    private MultiDigitPostfix pf = new MultiDigitPostfix();

    /**
     * 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();
    }

    public void evaluateInfix(String ifx, boolean hexa) {
        try {
                displayValue = pf.evaluate(pf.infixToPostfix(ifx), hexa);
        }catch(Exception e) {
                keySequenceError();
        }
    }
}
```

Calculator:

```
/**
 * 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);
    }
}
```

MultiDigitPostfix:

```
import java.util.StringTokenizer;

public class MultiDigitPostfix {

        public int evaluate(String pfx, boolean hexa) throws Exception{
                Stack<Object> stack = new Stack<Object>();
                StringTokenizer st = new StringTokenizer(pfx);
                int result = 0;
                int rhs = 0;
                int lhs = 0;

                //So lange der String noch mehr Token hat
                while (st.hasMoreTokens()) {
```

```
                        String token = st.nextToken();
                        //Wenn der naechste Token eine Zahl ist
                        if (isNumber(token) || token.equalsIgnoreCase("A")
                                                        || token.equalsIgnoreCase("B")
                                                        || token.equalsIgnoreCase("C")
                                                        || token.equalsIgnoreCase("D")
                                                        || token.equalsIgnoreCase("E")
                                                        || token.equalsIgnoreCase("F")) {
                                stack.push(token);
                        } else{
                                //Nehme die ersten beiden Token (Zahlen) vom Stack
                                if(hexa) {
                                        rhs = Integer.parseInt(stack.pop().toString(), 16);
                                        lhs = Integer.parseInt(stack.pop().toString(), 16);
                                }else {
                                        rhs = Integer.parseInt(stack.pop().toString());
                                        lhs = Integer.parseInt(stack.pop().toString());
                                }

                                //Identifiziere den Operator und berechne das Ergebnis
                                if(token.equals("*")) {
                                        result = lhs*rhs;
                                } else if(token.equals("/")) {
                                        result = lhs/rhs;
                                } else if(token.equals("+")) {
                                        result = lhs+rhs;
                                } else if(token.equals("-")) {
                                        result = lhs-rhs;
                                } else if(token.equals("^")) {
                                        result = (int) Math.pow(lhs, rhs);
                                }
                                stack.push(result);
                        }
                }
                System.out.println("Evaluation Ergebnis: " + result);
                return result;
        }

        //Aufgabe 3 ( http://geekswithblogs.net/venknar/archive/2010/07/09/algorithm-for-infix-to-
postfix.aspx )
        public String infixToPostfix(String ifx) throws Exception {
                Stack<Character> stack = new Stack<Character>();
                char[] characters = ifx.toCharArray();
                String postfix = "";

                for(char t:characters) {
                        if(Character.isDigit(t) || Character.isAlphabetic(t)) {
                                postfix += t;
                        }else if(t == '(') {
                                stack.push(t);
                        }else if(t == '+' || t == '-' || t == '*' || t == '/' || t == '^') {
                                if(stack.empty()) {
                                        postfix += " ";
                                        stack.push(t);
                                }else {
                                        while(!stack.empty() && higherPriority(stack.top(), t)) {
                                                postfix += " ";
                                                postfix += stack.pop();
                                        }
                                        postfix += " ";
                                        stack.push(t);
                                }
                        }else if(t == ')') {
                                while(!stack.empty() && stack.top() != '(') {
                                        postfix += " ";
                                        postfix += stack.pop();
                                }
                                stack.pop();
                        }
                }
                while(!stack.empty()) {
```

```
                        postfix += " ";
                        postfix += stack.pop();
                }
                System.out.println("Infix to postfix Ergebnis: " + postfix);
                return postfix;
        }

        //Returns true if c is higher priority than c2
        private boolean higherPriority(char c, char c2) {
                if(c == '^' && c2 != '^') {
                        return true;
                }else if((c == '*' || c == '/') && (c2 == '+' || c2 == '-')) {
                        return true;
                }else {
                        return false;
                }
        }

        //Returns true if input is a number
        private boolean isNumber(String input) {
           try {
                   Integer.parseInt( input );
              return true;
           }catch(NumberFormatException nfe){
              return false;
           }
        }
}
```

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 checkBoxHexa;
        private JCheckBox checkBoxString;

        /**
         * 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());

        checkBoxHexa = new JCheckBox("hexa", false);
        checkBoxString = new JCheckBox("string", 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(checkBoxString);
		checkBoxString.addActionListener(this);
		buttonPanel.add(checkBoxHexa);
		checkBoxHexa.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(checkBoxHexa.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("=")) {
                        if(checkBoxString.isSelected()) {
                                calc.evaluateInfix(display.getText(), checkBoxHexa.isSelected());
                        }else {
                                calc.equals();
                        }
                } else if (command.equals("clear")) {
                        calc.clear();
                } else if (command.equals("?")) {
                        showInfo();
                } else if(command.equals("hexa")) {
                        if(checkBoxHexa.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(checkBoxHexa.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;
        }
}
```