Programmieren 2
Übung Semesterwoche 4

1. Aufgabe: RechnerGUI
In diesem Schritt geht es darum, das GUI zu implementieren. Der Rechner macht noch nichts!
Ein paar Hinweise:

- Calculator ist Unterklasse von Frame.
- Der Rechner hat eine Breite von 250 und eine Höhe von 210 Pixeln. Diese Grösse darf nicht veränderbar sein.
- Erzeugen Sie 17 Buttons. Sie sollen wie oben dargestellt beschriftet sein.
- Die Buttons sind auf ein Panel mit GridLayout zu positionieren. Mit add() kann man einen Button dem Panel hinzufügen. Das Gitter füllt man damit zeilenweise auf.
- Für die Anzeige verwenden wir ein TextField. Mit setEditable(false) machen wir Textfeld nicht mehr editierbar; mit setBackground(Color.white) setzen wir seinen
- Hintergrund auf weiss.
- Im Frame positionieren wir das Textfeld im Norden und das Panel im Süden.

2. Aufgabe: "mit Ende"
Wir erweitern die Klasse Calculator so, dass man die Anwendung ordentlich beenden kann. Rechnen ist weiterhin noch nicht möglich!
Ein paar Hinweise:

- Calculator implementiert nun das Interface WindowListener bzw. die 6 entsprechenden

Methoden. Für uns ist aber einzig die Methode windowClosing() relevant.

- Das Frame kann nun auch als WindowListener agieren bzw. seine eigenen Events verarbeiten.

Entsprechendes Registrieren ist aber vorgängig notwendig.

3. Aufgabe: jetzt mit RechenPower
Die Klasse Calculator wird nun definitiv zu einem Rechner ausgebaut. Auch dazu ein paar Hinweise:

- Der Rechner soll seinen Zustand festhalten können. Dazu führen wir die Instanzvariablen operand1, operand2, result und operator ein, alle vom Typ int.
- Die Werte von operator interpretieren wir folgt:
- 0 → "Operator unbekannt", 1 → +, 2 → , 3 → *, 4 → /
- Calculator muss zusätzlich das Interface ActionListener implementieren.
- Die Methode actionPerformed() realisiert schlussendlich die Rechenlogik. Sie bedingt einige Zeilen Code, die wir ebenfalls "step by step" implementieren. In einem ersten Schritt kann der Methodenrumpf ja noch leer bleiben.
- Das Frame muss sich bei allen Buttons als ActionListener registrieren.
- Mit folgender Anweisung kann man aus dem ActionEvent e die EventSource bzw. das erste Zeichen der korrespondierenden ButtonBeschriftung extrahieren:
  char ch = (e.getActionCommand()).charAt(0);
- Mit Hilfe einer switch-Anweisung nehmen wir uns den verschiedenen Tastendrücken an. Die 10 Ziffern behandelt man vorteilhaft am Schluss bzw. im default-Teil.
- Überlegen Sie sich für verschiedene Eingaben, wie sich beim Eintippen der einzelnen Zeichen die Anzeige des Rechners fortlaufend verändern muss. Im nachfolgenden Beispiel wird ein eingetipptes Zeichen in " " und die resultierende Anzeige in [ ] dargestellt:
  "C" [0]
  "1" [1] "2" [12] "+" [12] "7" [7] "=" [19]
  "9" [9] "/" [9] "0" [0] "=" [Error]
  "5" [5] "1" [51] "S" [51] "*" [51] "3" [3] "=" [153]
- optional: Kettenrechnungen
  "1" [1] "+" [1] "8" [8] "8" [88] "" [89] "9" [9] "=" [80] "C" [0]

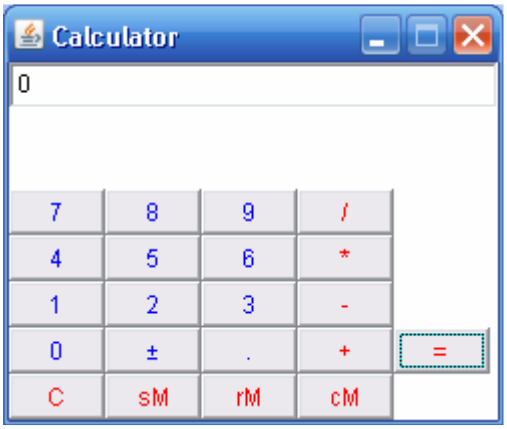
Vielleicht hilft Ihnen noch folgender Tipp:

```
char ch = '7';
int digit = ch '0'; // > digit mit intWert 7
```

Selbstverständlich testen Sie Ihren Rechner nach jedem Entwicklungsschritt wieder "live" aus!

Optional: 4. Aufgabe – noch ein Speicher dazu!
Drei zusätzliche Buttons "sM" (store Memory), "rM" (recall Memory) und "cM" (clear Memory) ermöglichen das Handling eines einfachen Speichers. Ein Label signalisiert mit "M", dass ein Wert gespeicherter ist.

Optional: 5. Aufgabe – mit Kommazahlen wäre der Rechner perfekt!




```
package calculator;

/**
 * @author Thomas Galliker
 */
public class CalcApp
{
    //for testing purpose only
    public static void main(String[] args)
    {
        new Calculator();
    }
}
```

```
package calculator;

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

public class Calculator extends Frame implements WindowListener, ActionListener
{
    private final int WIDTH=250;
    private final int HEIGHT=210;
    private TextField text1;
    private Label lbl1;
    private double operand1, operand2, memory, result;
    private boolean isDotSet = false;
    private boolean isNewOperand = false;
    private Operation operator = null;

    public Calculator()
    {
        setTitle("Calculator");
        setSize(WIDTH, HEIGHT);
        setResizable(false);

        BorderLayout layout1 = new BorderLayout();
        setLayout(layout1);

        Panel p1 = new Panel(new GridLayout(5,5));
```

```
ExtButton b = new ExtButton("btn7", "7");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn8", "8");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn9", "9");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnDivide", "/");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnEmpty1", "");
b.setVisible(false);
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn4", "4");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn5", "5");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn6", "6");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnTimes", "*");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnEmpty2", "");
b.setVisible(false);
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn1", "1");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn2", "2");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn3", "3");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnMinus", "-");
```

```
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnEmpty3", "");
b.setVisible(false);
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btn0", "0");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnSign", "+/-");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnDot", ".");
b.setForeground(Color.blue);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnPlus", "+");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnEqual", "=");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnClear", "C");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnMemStore", "sM");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnMemRecall", "rM");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

b = new ExtButton("btnMemClear", "cM");
b.setForeground(Color.red);
b.addActionListener(this);
p1.add(b);

text1 = new TextField();
text1.setEditable(false);
text1.setBackground(Color.white);

lbl1 = new Label();
lbl1.setBackground(Color.white);

add(text1, BorderLayout.NORTH);
add(lbl1, BorderLayout.CENTER);
add(p1, BorderLayout.SOUTH);

setVisible(true);
addWindowListener(this);
```

```
        clear();
    }

    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }

    public void windowIconified(WindowEvent e) { }
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }

    public void actionPerformed(ActionEvent e)
    {
        ExtButton button = (ExtButton)e.getSource();
        String name = button.getName();

        // Debug Hash Codes of Buttons:
        // System.out.println(name.hashCode());

        switch(name.hashCode())
        {
            //btn0
            case 3034452:
                setOperand(0);
                break;

            //btn1
            case 3034453:
                setOperand(1);
                break;

            //btn2
            case 3034454:
                setOperand(2);
                break;

            //btn3
            case 3034455:
                setOperand(3);
                break;

            //btn4
            case 3034456:
                setOperand(4);
                break;

            //btn5
            case 3034457:
                setOperand(5);
                break;

            //btn6
            case 3034458:
                setOperand(6);
                break;

            //btn7
            case 3034459:
                setOperand(7);
                break;

            //btn8
            case 3034460:
                setOperand(8);
                break;
```

```
            //btn9
            case 3034461:
                setOperand(9);
                break;

            //btnDivide
            case 154186229:
                setOperation(Operation.DIVIDE);
                break;

            //btnTimes
            case 2097951306:
                setOperation(Operation.TIMES);
                break;

            //btnMinus
            case 2091488116:
                setOperation(Operation.MINUS);
                break;

            //btnPlus
            case 206107158:
                setOperation(Operation.PLUS);
                break;

            //btnEqual
            case 2084344376:
                getResult();
                break;

            //btnSign
            case 206193209:
                changeSign();
                break;

            //btnDot
            case -1378836147:
                setDot();
                break;

            //btnClear
            case 2082333009:
                clear();
                break;

            //btnMemStore
            case 1116964488:
                storeMemory();
                break;

            //btnMemRecall
            case 223305290:
                recallMemory();
                break;

            //btnMemClear
            case 1101939700:
                clearMemory();
                break;

            default:
                break;
        }
    }

    private void setOperand(int value)
    {
        //flush field "text1" if "isNewOperand=true" flag is set
        if(isNewOperand)
```

```
            {
                text1.setText("");
                isNewOperand = false;
            }

        if(operator == null)
        {
            String s = text1.getText() + String.valueOf(value);

            if(!isDotSet)
            {
                Integer i = new Integer(s);
                text1.setText(i.toString());
            }
            else
            {
                text1.setText(s);
            }

            operand1 = Double.valueOf(text1.getText());
        }
        else
        {
            String s = text1.getText() + String.valueOf(value);

            if(!isDotSet)
            {
                Integer i = new Integer(s);
                text1.setText(i.toString());
            }
            else
            {
                text1.setText(s);
            }

            operand2 = Double.valueOf(text1.getText());
        }
    }

    private void setOperation(Operation o)
    {
        //perform calculation before the new operator is set
        calculate();

        //set new operator
        operator = o;

        //flag "isNewOperand=true" declares that the current screen output
        //has to be cleared as soon as a new input was set
        isNewOperand = true;
    }

    public enum Operation
    {
        PLUS, MINUS, TIMES, DIVIDE;

        double eval(double x, double y)
        {
            switch(this)
            {
                case PLUS:   return x + y;
                case MINUS:  return x - y;
                case TIMES:  return x * y;
                case DIVIDE: return x / y;
            }
            throw new AssertionError("Unknown operator: " + this);
        }
    }
```

```
    private double calculate()
    {
        //Calculate operand1 and operand2 with selected operator
        Operation o = operator;

        if(o != null)
        {
            //division by zero is not allowed
            if(operand2 == 0 && o == Operation.DIVIDE)
            {
                JOptionPane.showMessageDialog(null,        "Division        by        Zero
Exception!","Exception", JOptionPane.ERROR_MESSAGE);
                clear();
            }
            else
            {
                //eval method performs the defined calculation
                result = o.eval(operand1,operand2);
            }

            //exchange operand1 with result to allow linked calculations
            operand1 = result;
            operand2 = 0;
        }
        else
        {
            result = operand1;
        }

        //flush current operator sign
        operator = null;

        //flush decimal dot
        isDotSet = false;

        //flag "isNewOperand=true" declares that the current screen output
        //has to be cleared as soon as a new input was set
        isNewOperand = true;

        return result;
    }

    private void getResult()
    {
        result = calculate();
        text1.setText(Double.toString(result));
    }

    private void clear()
    {
        operand1 = 0;
        operand2 = 0;
        operator = null;
        result = 0;
        setOperand(0);
    }

    private void changeSign()
    {
        if(operator == null)
        {
            //if no operator is set we negate operand1
            operand1 = operand1 * (-1);
            text1.setText(Double.toString(operand1));
        }
        else
        {
            //if an operator is set, we negate the result
            result = result * (-1);
```

```
            text1.setText(Double.toString(result));
        }
    }

    private void storeMemory()
    {
        memory = operand1;
        refreshMemoryContent(true);
    }

    private double recallMemory()
    {
        text1.setText(Double.toString(memory));

        if(operator == null)
        {
            operand1 = memory;
        }
        else
        {
            operand2 = memory;
        }
        return memory;
    }

    private void clearMemory()
    {
        memory = 0;
        lbl1.setText("");
    }

    private void refreshMemoryContent(boolean showMemoryContent)
    {
        if(showMemoryContent)
        {
            lbl1.setText("M("+Double.toString(memory)+")");
        }
        else
        {
            lbl1.setText("M");
        }
    }

    private void setDot()
    {
        if(!isDotSet)
        {
            text1.setText(text1.getText() + ".");
            isDotSet = true;
        }
    }


}
```

```
package calculator;

import java.awt.*;

public class ExtButton extends Button
{
    private String name = "";

    public ExtButton(String label)
    {
        this.setName(label);
        this.setLabel(label);
    }

    public ExtButton(String name, String label)
    {
        this.setName(name);
        this.setLabel(label);
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

}
```

Weitere Informationen:
http://de.wikibooks.org/wiki/Java_Standard:_Grafische_Oberfl%C3%A4chen_mit_AWT
http://www.saar.de/~awa/jgui.htm
http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html
http://www.cs.ubc.ca/local/computing/software/j2sdk-1.4.2/docs/api/javax/swing/JOptionPane.html
http://www.myfaq.com.cn/api/jdk142/java/awt/Label.html
http://www.daniweb.com/forums/thread16127.html
http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html