Lab Report

EXERCISE 5: FUN WITH CALCULATORS 1

Name		Titel des Kurses	Datum
Juri Wiechmann Stepan Burlachenko	571085 571718	Prof. Dr. Weber-Wulff Info 2 Group 2	24.11.2019

Index

Introduction

This week's exercise was about making calculator and the numeric systems. This week we had a lot of fun with coding which was the reason we ran the extra mile. It takes a lot of time to fully understand the given code and then to work with them. But it's interesting to notice that at some point you work with the given code like it would be yours.

PRE-LAB

1. **You all know what a calculator is! Here's some code: calculator-full-solution(S1). Have a look at how it solves the problem of reading in a digit followed by an operator followed by a digit, and how to calculate the value when = is pressed.**

In the UserInterface-class we implements the ActionListener which let us input an int into the "numberPressed" method in the "CalcEngine"-class. The Engine take a look if the pressed digit is the first of a new one and if there is a left operand. It's also saving the last operator that we pressed and the left operand. If we now calculate the result we just calculate the left operand with the display value in corresponds to the operator.

2. **Is there anything important missing in this calculator? Beware: do not expect to read and understand this in the first few minutes of the lab! Read it at home, discuss it with others, perhaps over the class forum so that we can join in and help! How does it work?**

So multiplication and division is missing. Also in the "numberPressed"- method we multiply more incoming digits by ten that they are in the right position. But if we want now to use other numeric systems we need a variable that we get for example by a parameter that gives us the base of the current using system. For example if we but it an A and again A we get a 6E. But we want AA to show up. We can solve this by set the Base to 16 and so on for other numeric systems.

3. **What are hexadecimal numbers, by the way?**

Hexadecimal is one of these numeric systems. Its base is 16 and we get 6 extra digits which are

A for 10
B for 11
C for 12
D for 13
E for 14
F for 15

Assignments

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

First, we added multiplication and division by adding buttons, creating the if-statement in the "actionPerformed"-method and calling the two new methods in the "CalcEngine" that we wrote.

```
addButton(buttonPanel, "x");
addButton(buttonPanel, "/");
```

```
else if(command.equals("x")) {
    calc.multiplication();
}
else if(command.equals("/")) {
    calc.division();
```

```
private void calculateResult()
{
    switch(lastOperator) {
        case '+':
            displayValue = leftOperand + displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
        case '-':
            displayValue = leftOperand - displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
        case 'x':
            displayValue = leftOperand * displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
        case '/':
            displayValue = leftOperand / displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
        default:
            keySequenceError();
            break;
    }
}
```

```
/**
 * The 'multiplication' button was pressed.
 */
public void multiplication()
{
    applyOperator('x');
}
/**
 * The 'division' button was pressed.
 */
public void division()
{
    applyOperator('/');
}
```

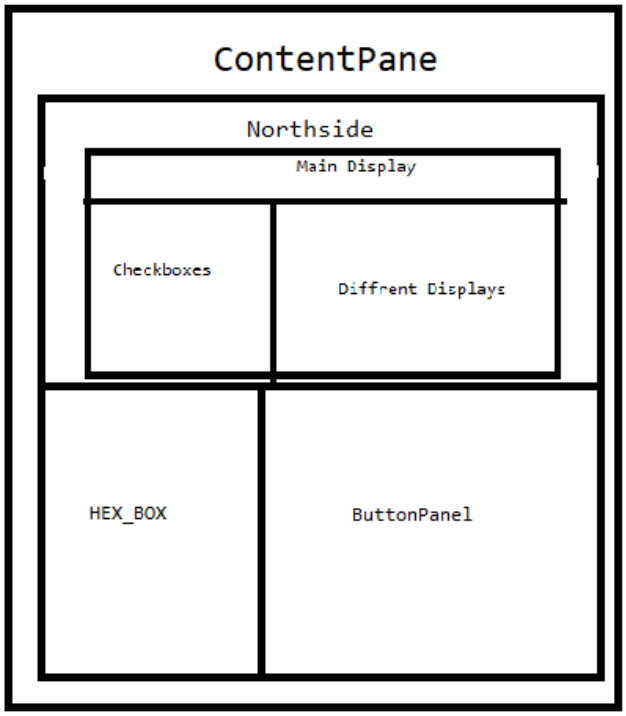
We should mention that the calculation is still based on Integer so we round down to the next Integer if we divide.

For the next change, we just needed to add a new parameter to our "numberPressed"-method: Base which is a variable:

```
public void numberPressed(int number, int Base)
{
    if(buildingDisplayValue) {
        // Incorporate this digit.
        displayValue = displayValue*Base + number;
    }
    else {
        // Start building a new number.
        displayValue = number;
        buildingDisplayValue = true;
    }
}
```

2. **Extend your integer calculator without making any changes to your superclasses except for changing privates to protected to include buttons for entering in the hexadecimal digits. Make the calculator do its calculations and display in hexadecimal notation.**

We knew we weren't allowed to change things but in order to design some things, to sort and to get access to a specific JPanel, we modified it a bit without changing the functionality. First, we split our "ConentPane" into the "ButtonPanel" and the "Northside". In the "ButtonPanel", we hold all the buttons except the HEX-Numbers.



In the "Northside", we hold our displays and the Checkboxes.

To implement these Checkboxes and the "HEX_BOS"-panel with the Hexa-buttons, we created a "UserInterface_HEX"-class.

First, we created an Array of type “JCheckbox”. The length is determined by a Constructor-parameter (which we later extended to 3 or 4 checkboxes. But for now we have only 2. One for DEC and the other for HEX. In method “addLayout_HEX” which we call in the constructor, we create these 2 and add the “ActionListener” for each of them. We also put them into “ButtonGroup” to make sure we can only can select one at a time.

We also wanted the extra displays next to the Checkboxes. Without changing the numeric-system, these displays show the value in these specific numeric-systems. We saw that in the Windows-calculator and really liked this design idea.

The Main-Display is important and we changed also the size of the text.

Now we can get to the “HEX_BOX”. We simply added six buttons and some empty JLabel for a nicer look.

The Code for this looks like this:

```
    protected JCheckBox[] Boxes;
    protected ButtonGroup CheckBoxGrp = new ButtonGroup();
    private JPanel HEX_BOX;

    JPanel Checkboxes = new JPanel(new GridLayout(4, 2));

    JTextField DECdisplay;
    JTextField HEXdisplay;
…
        public UserInterface_HEX(CalcEngine engine, int Amount_Checkboxes) {
               super(engine);
               this.Boxes = new JCheckBox[Amount_Checkboxes];
               addLayout_HEX();
        }
        public void addLayout_HEX() {

         Boxes[0] = new JCheckBox("DEC",true);
         Boxes[1] = new JCheckBox("HEX");

         Boxes[0].addActionListener(this);
         Boxes[1].addActionListener(this);

         DECdisplay = new JTextField("0");
         HEXdisplay = new JTextField("0");

         addCheckbox(Checkboxes, Boxes[0]);
         Checkboxes.add(DECdisplay);
         addCheckbox(Checkboxes, Boxes[1]);
         Checkboxes.add(HEXdisplay);

         CheckBoxGrp.add(Boxes[0]);
         CheckBoxGrp.add(Boxes[1]);

         Northside.add(Checkboxes);

         Font font1 = new Font("SansSerif", Font.BOLD, 20);
         display.setFont(font1);

         contentPane.add(Northside, BorderLayout.NORTH);

         HEX_BOX = new JPanel(new GridLayout(5,2));
         buttonPanel.add(HEX_BOX);

         HEX_BOX.add(new JLabel(" "));
         HEX_BOX.add(new JLabel(" "));
         addButton(HEX_BOX, "A");
         addButton(HEX_BOX, "B");
         addButton(HEX_BOX, "C");
         addButton(HEX_BOX, "D");
         addButton(HEX_BOX, "E");
         addButton(HEX_BOX, "F");
         HEX_BOX.add(new JLabel(" "));
         HEX_BOX.add(new JLabel(" "));

         contentPane.add(HEX_BOX, BorderLayout.WEST);
         frame.pack();

         assignButtonGrps_HEX();
         GreyButtons_HEX();
        }
```

At the bottom, you can see 2 methods that we call: “assignButtonGrps_HEX” and “GreyButtons_HEX”. Let’s start with “assignButtonGrps_HEX”. For a better overview and to Disable/Enable the right buttons for the right Mode that we are in, we wanted to have different “ButtonGroups”. For each Mode (which numeric system is in use), we created a “ButtonGroup”:

```
protected ButtonGroup DECButtons = new ButtonGroup();
protected ButtonGroup HEXButtons = new ButtonGroup();
```

To fill up the “HEXButtons” we simply need to get every component in the “HEX_BOX” and then to add it to the “HEXButtons”. It gets more complicated in “DECButtons”. Because we don’t want all components in the “buttonPanel”, we need to filter them to get only the buttons from 0-9. To do that, we need to check if our component is a “JButton” first, and if this is true, we need to compare the button-text: if it’s between 0 and 9. We created a Field that is an Array of Components. We use this for every “JLabel” we want to get through. `Component[] comp;`Now we use this:

```
//assign all the Buttons that are relevant to their ButtonGroup
protected void assignButtonGrps_HEX() {

   comp = buttonPanel.getComponents();
   //DEC
   for(Component E : comp) {
         if(E instanceof JButton)
               if(isNumber(((JButton)E).getText()))
               if(isBetween(Integer.parseInt(((JButton)E).getText()), 0, 9))
                     DECButtons.add((JButton)E);
   }
   //HEX
   comp = HEX_BOX.getComponents();
   for(Component E : comp)
         if(E instanceof JButton)
               HEXButtons.add((JButton)E);
}
```

Here we need to use the “isNumber”-method and the “isBetween”-method. I think you know what they do but here is the code:

```
//return true if min <= value <= max
protected boolean isBetween(int value, int min, int max) {
   if(value >= min && value <= max)
         return true;
   else
         return false;
}
```

```
//Checks if the String is a single digit Number
protected boolean isNumber(String value) {
   if( value.equals("0") ||
          value.equals("1") ||
          value.equals("2") ||
          value.equals("3") ||
          value.equals("4") ||
          value.equals("5") ||
          value.equals("6") ||
          value.equals("7") ||
          value.equals("8") ||
          value.equals("9") ||
          value.equals("A") ||
          value.equals("B") ||
          value.equals("C") ||
          value.equals("D") ||
          value.equals("E") ||
          value.equals("F"))
                return true;
          else
                return false;
}
```

Now we can use these button groups to handle the Enable/Disable status of these buttons according to the current mode. We looked up how we can iterate a "ButtonGroup"(S2)(S3). It turns out that it wasn't that easy. You need to use the Enumeration type.

Upon launching the calculator and every time we change the mode, we disable all buttons and then turn all buttons on enable which are in the right "ButtonGroup":

```
    /*Disable all Buttons and then enable only these which are in the
      right ButtonGroup that corresponds to the selection mode
    */
protected void GreyButtons_HEX() {
       //Make all Grey
       Enumeration<AbstractButton> HEX = HEXButtons.getElements();
       while(HEX.hasMoreElements()) {
              JButton Button = (JButton) HEX.nextElement();
              Button.setEnabled(false);
       }
       Enumeration<AbstractButton> DEC_theRest = DECButtons.getElements();
       while(DEC_theRest.hasMoreElements()) {
              JButton Button = (JButton) DEC_theRest.nextElement();
              Button.setEnabled(false);
       }
       //DEC-Mode
       if(Boxes[0].isSelected()) {
              Enumeration<AbstractButton> DEC = DECButtons.getElements();
              while(DEC.hasMoreElements()) {
                     JButton Button = (JButton) DEC.nextElement();
                     Button.setEnabled(true);
              }
       }
       //HEX-Mode
       else if(Boxes[1].isSelected()) {
              HEX = HEXButtons.getElements();
              while(HEX.hasMoreElements()) {
                     JButton Button = (JButton) HEX.nextElement();
                     Button.setEnabled(true);
              }
              Enumeration<AbstractButton> DEC = DECButtons.getElements();
              while(DEC.hasMoreElements()) {
                     JButton Button = (JButton) DEC.nextElement();
                     Button.setEnabled(true);
              }
       }
}
```

We also change some things in the “actionPerformed”-method. We create a Boolean variable “BaseChange” which turns true in the following loop if the event is triggered by “JCheckboxes”. If “BaseChange” is evaluated to true, we set the new base and we call the “GreyButtons_HEX”-method:

```
boolean BaseChange = false;

for(JCheckBox E : Boxes)
       if(E.getText().equals(command))
              BaseChange = true;
if(BaseChange) {
       setBase_HEX();
       GreyButtons_HEX();

}
```

If a number gets pressed, we save this number in an variable. In order to get it work for the HEX numbers, we convert the given command to a HexaDecimal-number. This will work for every numeric-system with a Base lower than 17 because Hex contains all numbers that are used in the other that we will implement. That we call the "CalcEngine"- method: "numberPressed" with the number that we initialize and the current base:

```
else if(isNumber(command)) {
      int number = Hexa.HEXToDEC(new Hexa(command));
         calc.numberPressed(number, Base);
 }
```

Now if the pressed button is something else, we call the "ChooseCommand"-method which we wrote to have a better overview in our "actionPerformed"-method:

```
else
        ChooseCommand(command);
…
    protected void ChooseCommand(String command) {
       if(command.equals("+")) {
            calc.plus();
        }
        else if(command.equals("-")) {
            calc.minus();
        }
        else if(command.equals("x")) {
            calc.multiplication();
        }
        else if(command.equals("/")) {
            calc.division();
        }
        else if(command.equals("=")) {
            calc.equals();
        }
        else if(command.equals("Clear")) {
            calc.clear();
        }
        else if(command.equals("?")) {
            showInfo();
        }
      }
```

At the end we call 3 methods to clear our interface.:

```
redisplay_HEX();
updateTextfields_HEX();
GreyButtons_HEX();
```

First we update the Main display:

```
    //in dependence to our selected mode the main Textfield update
      protected void redisplay_HEX()
    {
      if(Boxes[0].isSelected()) {
            display.setText("" + calc.getDisplayValue());
      }
      else if(Boxes[1].isSelected()) {
            display.setText("" + Hexa.DECtoHEX(calc.getDisplayValue()));
      }
    }
```

Than we update the other Textfields:

```
//update our four seperate Textfields
protected void updateTextfields_HEX() {
      DECdisplay.setText("" + calc.getDisplayValue());
      HEXdisplay.setText("" + Hexa.DECtoHEX(calc.getDisplayValue()));
}
```

And at the end we grey the right buttons out.

You can see that at some points we used a “Hexa”-class. We thought we would need a “Hexa”-class that contains some helpful methods. Like we learned in a lab before we started our ADT with an Interface to than implement it into a Class:

```
public interface HexaDecimal {

      public Hexa Add(Hexa HEX);
      public Hexa Subtract(Hexa HEX);
      public Hexa Multiplycation(Hexa HEX);
      public Hexa Division(Hexa HEX);
      public String toString();
}
```

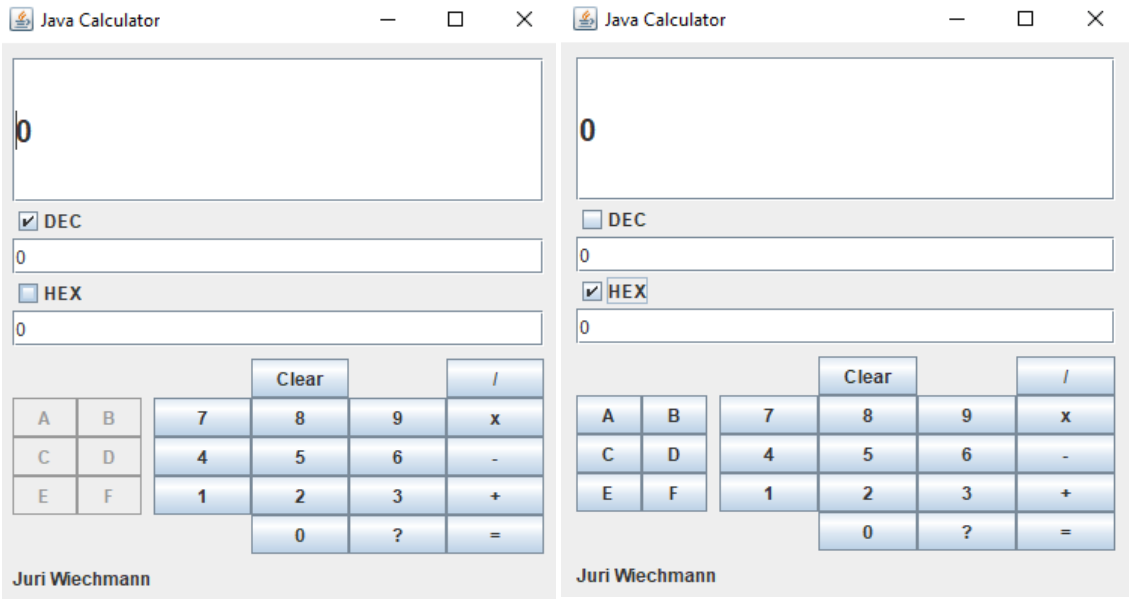
We implemented it in the “Hexa”-class:

```java
public class Hexa implements HexaDecimal {
	private String value;

	public Hexa(String value) {
		setValue(value);
	}
	public String getValue() {
		return value;
	}
	public void setValue (String value) {
		char[] Cvalue = value.toCharArray();
		for(char E : Cvalue)
			if((E < 48 && E > 57) || (E < 65 && E > 70))
			throw new IllegalArgumentException("only Chars from 1-9 and A-F");
		this.value = value;
	}
	@Override
	public Hexa Add(Hexa HEX) {

		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a+b);
	}

	@Override
	public Hexa Subtract(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a-b);
	}

	@Override
	public Hexa Multiplycation(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a*b);
	}

	@Override
	public Hexa Division(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a/b);
	}
	public static int HEXToDEC(Hexa HEX) {
		return Integer.parseInt(HEX.getValue(), 16);
	}
	public static Hexa DECtoHEX(int DEC) {
		return new Hexa(Integer.toHexString(DEC).toUpperCase());
	}
	public String toString() {
		return value;
	}
}
```

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.**

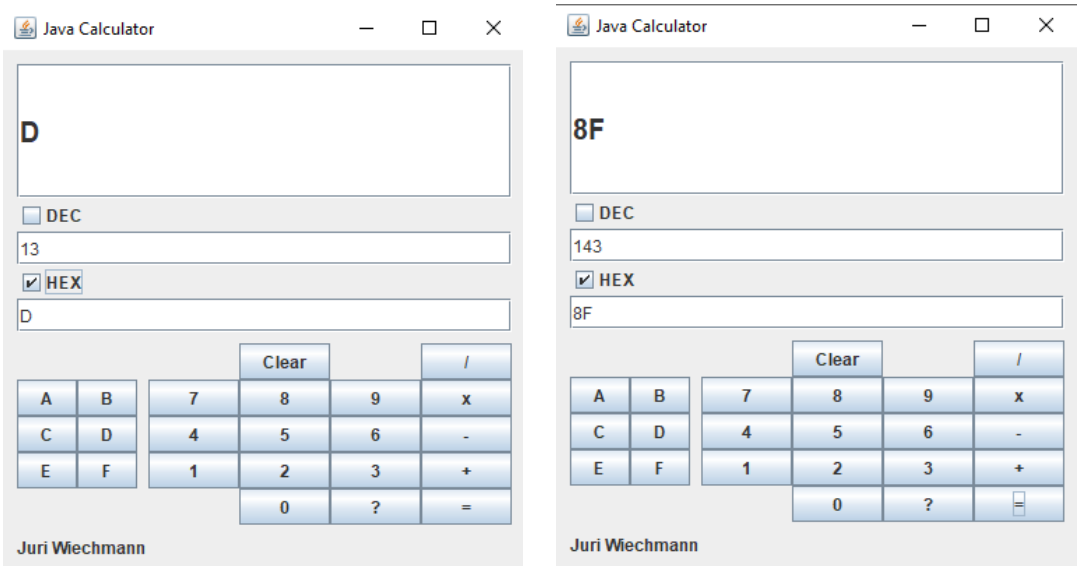
Well, we accidentally did this in the second task. Because our first idea was to use "JCheckboxes" and not buttons.

Till now our Calculator looks like this:

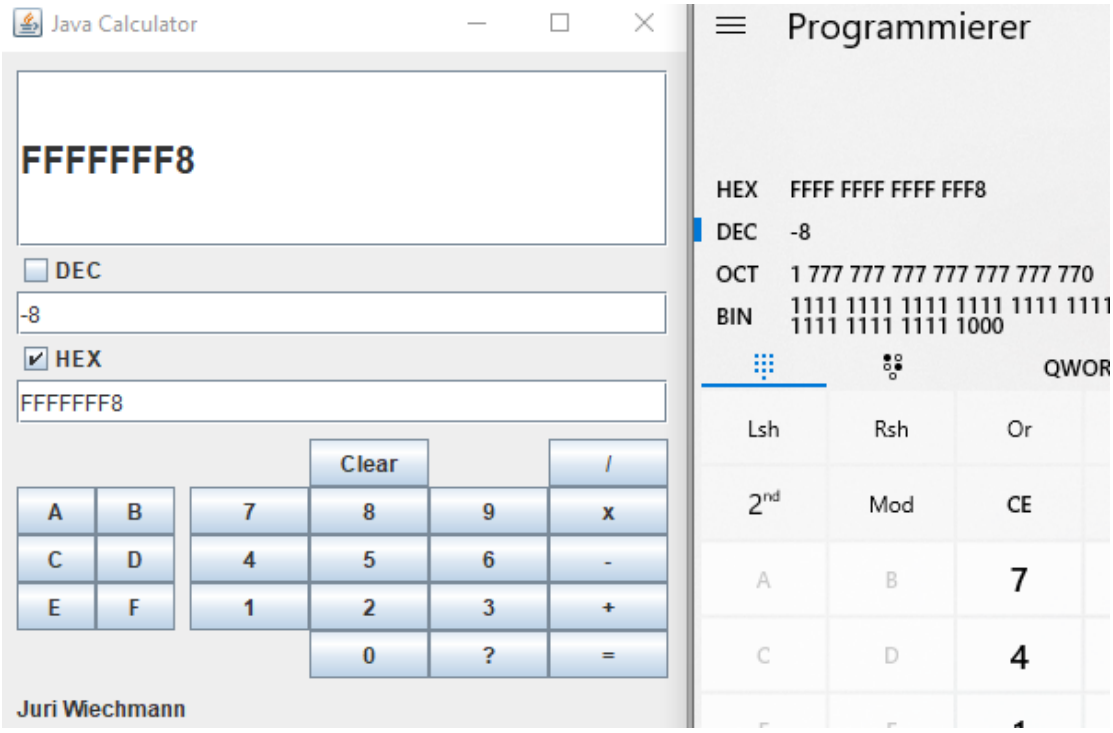


4. **What test cases do you need to test your calculator? If you find errors in your calculator, document them in your report!**

We don't really know what we needed to do here. After all, don't we test our calculator by creating it and pressing the numbers and checking the results? We tested all operators with all numbers (0-9 and A-F). Everything was correct. The displays also showed the right value for each numeric-systems: (First picture: D; second picture D*B)



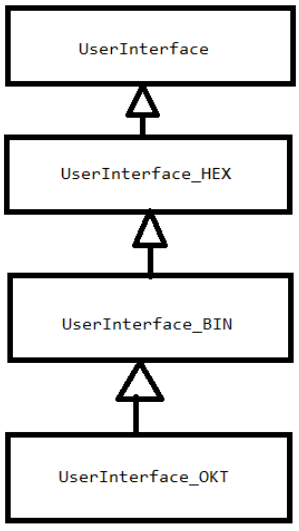
We found two things that weren't working right. First, we can get an overflow. Simply because we use Integer. We could use BigInteger but for that we need to change all the types in the calculator and more on. The second thing were, that if we get a negative number our Hex-Textfield shows some crazy number. It turns out that the number is some sort of right by comparing it with the Windows-calculator. We got the some crazy results in the other numeric-systems later:



It's because Hex-and Oktsystems are based on the Binsystem in which negative numbers got represented by a leading 1.

5. **Make a whole collection of calculators: binary, octal, decimal, hexadecimal. Maybe make a CalculatorFactory that takes the base as a parameter to the constructor, figures out how to layout the buttons nicely, and calculates in the selected base mode?**

Because in Java you can't extend a class by more than one class, we need some kind of hierarchy for the different numeric-systems:



First, we did the same thing as we did in the "UserInterface_HEX" and created an Interface for the Binary-system and the "BIN"-class. Same with the Okta-system. There is nothing really different than to the "Hexa"-class that's why we don't show the code here. If you want to take a look scroll down to the appendix.

We followed the same structure as in the “UserInterface_HEX”. In the constructor, we called the “addLayout_BIN”-method which add a “JCheckbox” and the “Bindisplay” to the “Checkboxes”-JLabel. We also created the “assignButtonGrps_BIN”, ”GreyButtons_BIN”, “setBase_BIN”, “redisplay_BIN” and the “updateTextfields_BIN”-methods which all call the similar …_HEX-methods first and then do the same things for the binary-system:

```
public class UserInterface_HEX_BIN extends UserInterface_HEX {

    protected ButtonGroup BINButtons = new ButtonGroup();

    JTextField BINdisplay;

    public UserInterface_HEX_BIN(CalcEngine engine, int Amount_Checkboxes) {
    private void addLayout_BIN() {
    protected void assignButtonGrps_BIN() {
    protected void GreyButtons_BIN() {
    public void actionPerformed(ActionEvent event)
    protected void updateTextfields_BIN() {
    protected void redisplay_BIN() {
    protected void setBase_BIN() {
}
```

Now we did the exact same with the Okta-system:

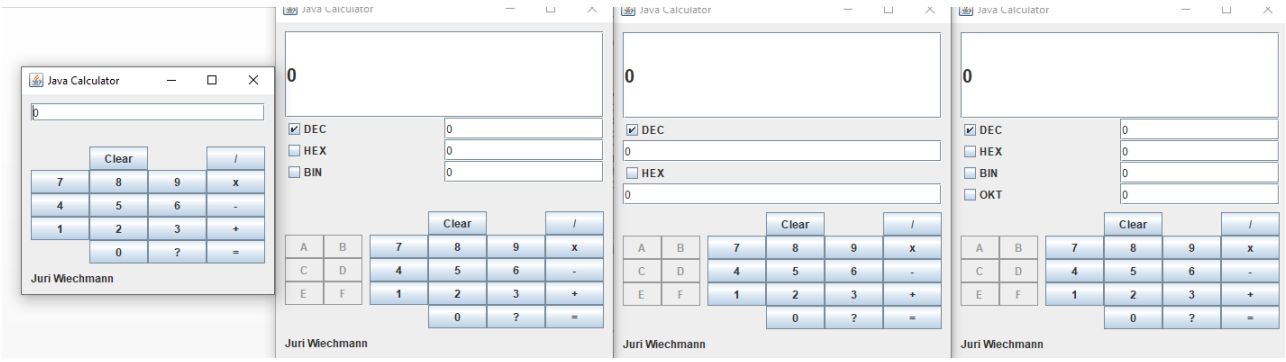
```
public class UserInterface_HEX_BIN_OKT extends UserInterface_HEX_BIN {

    protected ButtonGroup OKTButtons = new ButtonGroup();

    JTextField OKTdisplay;

    public UserInterface_HEX_BIN_OKT(CalcEngine engine, int Amount_Checkboxes) {
    private void addLayout_OKT() {
    protected void assignButtonGrps_OKT() {
    protected void GreyButtons_OKT() {
    public void actionPerformed(ActionEvent event)
    private void setBase_OKT() {
    private void updateTextfields_OKT() {
    private void redisplay_OKT() {
}
```

At the end our Calculator works perfect except the Overflow exception. We wrote a Class that calls the Calculator. In order to still be able to use all the User Interfaces we gave the Calculator-constructor a parameter which determines the Interface that we want:

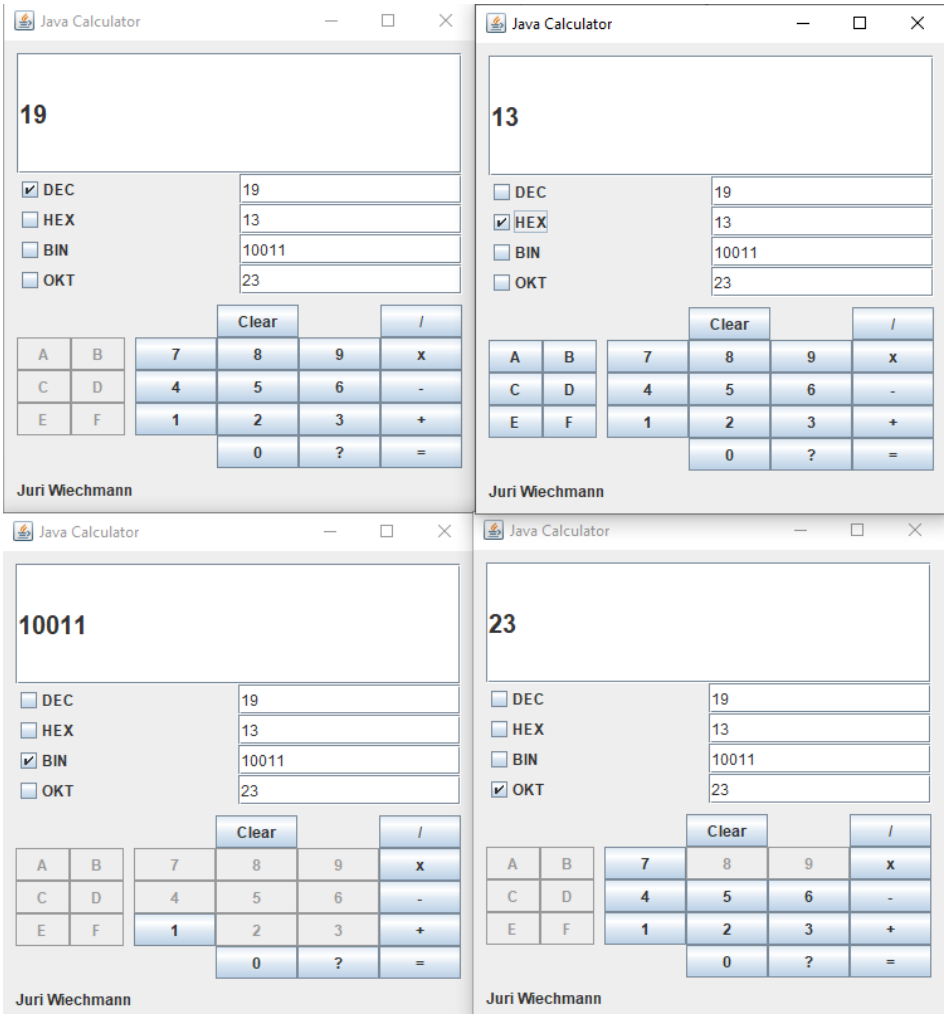
```
private CalcEngine engine;
private UserInterface gui;

/**
 * Create a new calculator and show it.
 * @param Set the Numeric System
 * mode == 1: Basic Calc
 * mode == 2: HexCalc
 * mode == 3: BIN_HEX Calc
 * mode == 4: OKT_BIN_HEX Calc
 */
public Calculator(int mode)
{
    engine = new CalcEngine();
    if(mode == 1) gui = new UserInterface(engine);
    if(mode == 2) gui = new UserInterface_HEX(engine, mode);
    if(mode == 3) gui = new UserInterface_HEX_BIN(engine, mode);
    if(mode == 4) gui = new UserInterface_HEX_BIN_OKT(engine, mode);

}
```

We wrote a Test-Class to create 4 different Objects with each possible parameter:

```
public class Create_Calc {
    public static Calculator Test;
    public static void main(String[] args) {
        Test = new Calculator(1);
        Test = new Calculator(2);
        Test = new Calculator(3);
        Test = new Calculator(4);
    }
}
```



Each one works and for mode = 4 we will show you the functionality of each Numeric-system:


Java Calculator
19
DEC 19
HEX 13
BIN 10011
OKT 23
Clear /
A B 7 8 9 x
C D 4 5 6 -
E F 1 2 3 +
0 ? =
Juri Wiechmann
Java Calculator
13
DEC 19
HEX 13
BIN 10011
OKT 23
Clear /
A B 7 8 9 x
C D 4 5 6 -
E F 1 2 3 +
0 ? =
Juri Wiechmann
Java Calculator
10011
DEC 19
HEX 13
BIN 10011
OKT 23
Clear /
A B 7 8 9 x
C D 4 5 6 -
E F 1 2 3 +
0 ? =
Juri Wiechmann
Java Calculator
23
DEC 19
HEX 13
BIN 10011
OKT 23
Clear /
A B 7 8 9 x
C D 4 5 6 -
E F 1 2 3 +
0 ? =
Juri Wiechmann

Reflections

STEPAN

It took us quite some time to understand the initial code. Even though it was a part of the pre-lab, I still needed time to think how we can add other operations. And even then, we had some issues which were mainly caused by our inattentiveness. It was interesting to think about the way we should place all the buttons to make it easier for the future users to use although the user interface part should not have been our main concern according to the task.
It was good we managed to do Task 1 right in the lab as well as a good part of Task 2. The following tasks were mainly done by Juri and I only tried to follow his way of thinking, hence his name as the main author of the thing.

JURI

In this week`s lab I had problems with understanding the assignments. It wasn't clear for me what Class we should extend and how we far we are allowed to change the Super-classes. At the End I did some major changes in the structure to make it fit into my subclasses. On the other hand I lost myself again into the "Lightsout-pizza-coding-for-hours" experience and can´t stop coding. For example all these CDT´s that I create just because it somehow matches and I could need them some when. But one thing drove me crazy till today. Java in combination with the given code forced me to build a Hierarchy between the numeric-system. In the real world there isn't a Hierarchy in these systems but because in java you can only extend by one class I needed to.

PS: I finally managed to create a github Account. I'm not really comfortable at the moment with it but I hope this way it's safe and more easy to implement this code if you want to test it:

https://github.com/LuffyGM/Calculator1.git

If this works and if it's ok for you I would like to give my code this way because in the Appendix it's getting a really bad overview.

Source:

S1: https://moodle.htw-berlin.de/mod/resource/view.php?id=382750

S2: `https://coderanch.com/t/336958/java/disable-buttonGroup`

S3: `ttps://stackoverflow.com/questions/7160568/iterating-through-enumeration-of-hastable-keys-throws-nosuchelementexception-err`

Appendix:

Java Code:

```
package Übung5_2;

public class Create_Calc {
	public static Calculator Test;
	public static void main(String[] args) {
		//Test = new Calculator(1);
		//Test = new Calculator(2);
		//Test = new Calculator(3);
		Test = new Calculator(4);
	}
}
package Übung5_2;

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
 * @version 2008.03.30
 */
public class UserInterface
    implements ActionListener
{
    protected CalcEngine calc;
    protected boolean showingAuthor;

    protected JFrame frame;
    protected JTextField display;
    protected JLabel status;

    protected JPanel buttonPanel;
    protected JPanel Northside;
    protected JPanel contentPane;

    /**
     * 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());

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

        display = new JTextField("0");
        Northside = new JPanel(new GridLayout(2, 1));

        Northside.add(display);
        contentPane.add(Northside, BorderLayout.NORTH);

        buttonPanel = new JPanel(new GridLayout(5, 3));


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


            addButton(buttonPanel, "7");
            addButton(buttonPanel, "8");
            addButton(buttonPanel, "9");
            addButton(buttonPanel, "x");


            addButton(buttonPanel, "4");
            addButton(buttonPanel, "5");
            addButton(buttonPanel, "6");
            addButton(buttonPanel, "-");


            addButton(buttonPanel, "1");
            addButton(buttonPanel, "2");
            addButton(buttonPanel, "3");
            addButton(buttonPanel, "+");

            buttonPanel.add(new JLabel(" "));
            addButton(buttonPanel, "0");
            addButton(buttonPanel, "?");
            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.
 */
protected void addButton(Container panel, String buttonText)
{
    JButton button = new JButton(buttonText);
    button.addActionListener(this);
    panel.add(button);
}
protected void addCheckbox(Container panel, JCheckBox JCB) {
   panel.add(JCB);
}

/**
 * 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")) {
        int number = Integer.parseInt(command);
        calc.numberPressed(number , 10);
    }
    else if(command.equals("+")) {
        calc.plus();
    }
    else if(command.equals("-")) {
        calc.minus();
    }
    else if(command.equals("x")) {
        calc.multiplication();
    }
    else if(command.equals("/")) {
        calc.division();
```

```
        }
        else if(command.equals("=")) {
            calc.equals();
        }
        else if(command.equals("Clear")) {
            calc.clear();
        }
        else if(command.equals("?")) {
            showInfo();
        }
        // else unknown command.

        redisplay();
    }

    /**
     * Update the interface display to show the current value of the
     * calculator.
     */
    protected void redisplay()
    {
        display.setText("" + calc.getDisplayValue());
    }

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

        showingAuthor = !showingAuthor;
    }
}

package Übung5_2;
/**
 * 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
 * @version 2008.03.30
 */
public class Calculator
{
    private CalcEngine engine;
    private UserInterface gui;

    /**
     * Create a new calculator and show it.
     * @param Set the Numeric System
     * mode == 1: Basic Calc
     * mode == 2: HexCalc
```

```
     * mode == 3: BIN_HEX Calc
     * mode == 4: OKT_BIN_HEX Calc
     */
    public Calculator(int mode)
    {
        engine = new CalcEngine();
        if(mode == 1) gui = new UserInterface(engine);
        if(mode == 2) gui = new UserInterface_HEX(engine, mode);
        if(mode == 3) gui = new UserInterface_HEX_BIN(engine, mode);
        if(mode == 4) gui = new UserInterface_HEX_BIN_OKT(engine, mode);

    }

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

```
package Übung5_2;
/**
 * The main part of the calculator doing the calculations.
 *
 * @author  David J. Barnes and Michael Kolling
 * @version 2008.03.30
 */
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.
 * Nummeric Base
 */
public void numberPressed(int number, int Base)
{
    if(buildingDisplayValue) {
        // Incorporate this digit.
        displayValue = displayValue*Base + 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('-');
}
/**
 * The 'multiplication' button was pressed.
 */
public void multiplication()
{
    applyOperator('x');
}
/**
 * The 'division' button was pressed.
 */
public void division()
{
    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";
}

/**
 * @return The author of this engine.
 */
public String getAuthor()
{
    return "Juri Wiechmann";
}

/**
 * @return The version number of this engine.
 */
public String getVersion()
{
   return "Version 2.1";
}
```

```
/**
 * 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 'x':
            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();
    }

}
```

```
package Übung5_2;


import java.awt.BorderLayout;

import java.awt.Component;

import java.awt.Font;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.Enumeration;


import javax.swing.AbstractButton;

import javax.swing.ButtonGroup;

import javax.swing.JButton;

import javax.swing.JCheckBox;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;


/**
```

```
 * 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 Juri Wiechmann
 * @version 2019.11.25
 */
public class UserInterface_HEX extends UserInterface
    implements ActionListener
{
        protected JCheckBox[] Boxes;
    protected ButtonGroup CheckBoxGrp = new ButtonGroup();

    private JPanel HEX_BOX;

    JPanel Checkboxes = new JPanel(new GridLayout(4, 2));

    JTextField DECdisplay;
    JTextField HEXdisplay;

    protected ButtonGroup DECButtons = new ButtonGroup();
    protected ButtonGroup HEXButtons = new ButtonGroup();

    protected int Base = 10;

    Component[] comp;

        public UserInterface_HEX(CalcEngine engine, int Amount_Checkboxes) {
```

```
            super(engine);
            this.Boxes = new JCheckBox[Amount_Checkboxes];
            addLayout_HEX();
    }
    public void addLayout_HEX() {

Boxes[0] = new JCheckBox("DEC",true);
Boxes[1] = new JCheckBox("HEX");

Boxes[0].addActionListener(this);
Boxes[1].addActionListener(this);

DECdisplay = new JTextField("0");
HEXdisplay = new JTextField("0");

addCheckbox(Checkboxes, Boxes[0]);
Checkboxes.add(DECdisplay);
addCheckbox(Checkboxes, Boxes[1]);
Checkboxes.add(HEXdisplay);

CheckBoxGrp.add(Boxes[0]);
CheckBoxGrp.add(Boxes[1]);

Northside.add(Checkboxes);

Font font1 = new Font("SansSerif", Font.BOLD, 20);
display.setFont(font1);
```

```
    contentPane.add(Northside, BorderLayout.NORTH);

    HEX_BOX = new JPanel(new GridLayout(5,2));
                buttonPanel.add(HEX_BOX);

                HEX_BOX.add(new JLabel(" "));
                HEX_BOX.add(new JLabel(" "));
                addButton(HEX_BOX, "A");
    addButton(HEX_BOX, "B");
    addButton(HEX_BOX, "C");
    addButton(HEX_BOX, "D");
    addButton(HEX_BOX, "E");
    addButton(HEX_BOX, "F");
    HEX_BOX.add(new JLabel(" "));
    HEX_BOX.add(new JLabel(" "));

    contentPane.add(HEX_BOX, BorderLayout.WEST);
    frame.pack();

    assignButtonGrps_HEX();
    GreyButtons_HEX();
        }
public void actionPerformed(ActionEvent event)
{
        String command = event.getActionCommand();
        boolean BaseChange = false;
```

```
        for(JCheckBox E : Boxes)
                if(E.getText().equals(command))
                        BaseChange = true;
        if(BaseChange) {
                setBase_HEX();
                GreyButtons_HEX();
        }
        else if(isNumber(command)) {
        int number = Hexa.HEXToDEC(new Hexa(command));
          calc.numberPressed(number, Base);
    }
    else
        ChooseCommand(command);

    redisplay_HEX();
    updateTextfields_HEX();
    GreyButtons_HEX();
}
//in dependence to our selected mode the main Textfield update
        protected void redisplay_HEX()
{
        if(Boxes[0].isSelected()) {
                display.setText("" + calc.getDisplayValue());
        }
        else if(Boxes[1].isSelected()) {
                display.setText("" + Hexa.DECtoHEX(calc.getDisplayValue()));
        }
}
```

```
        //update our four seperate Textfields
    protected void updateTextfields_HEX() {
        DECdisplay.setText("" + calc.getDisplayValue());
        HEXdisplay.setText("" + Hexa.DECtoHEX(calc.getDisplayValue()));
    }
    //Set the Base based on the selected mode
    protected void setBase_HEX() {

        if(Boxes[0].isSelected()) {
                System.out.println("DEC");
                Base = 10;
        }
        else if(Boxes[1].isSelected()) {
                System.out.println("HEX");
                Base = 16;
        }
        redisplay_HEX();
    }
    /*Disable all Buttons and then enable only these which are in the
      right ButtonGroup that corresponds to the selection mode
    */
    protected void GreyButtons_HEX() {
        //https://coderanch.com/t/336958/java/disable-buttonGroup
        //https://stackoverflow.com/questions/7160568/iterating-through-
enumeration-of-hastable-keys-throws-nosuchelementexception-err
        //https://stackoverflow.com/questions/1625855/how-to-disable-javax-swing-
jbutton-in-java
```

```
//Make all Grey
Enumeration<AbstractButton> HEX = HEXButtons.getElements();
while(HEX.hasMoreElements()) {
        JButton Button = (JButton) HEX.nextElement();
        Button.setEnabled(false);
}
Enumeration<AbstractButton> DEC_theRest = DECButtons.getElements();
while(DEC_theRest.hasMoreElements()) {
        JButton Button = (JButton) DEC_theRest.nextElement();
        Button.setEnabled(false);
}
//DEC-Mode
if(Boxes[0].isSelected()) {
        Enumeration<AbstractButton> DEC = DECButtons.getElements();
while(DEC.hasMoreElements()) {
        JButton Button = (JButton) DEC.nextElement();
        Button.setEnabled(true);
}
}
//HEX-Mode
else if(Boxes[1].isSelected()) {
        HEX = HEXButtons.getElements();
while(HEX.hasMoreElements()) {
        JButton Button = (JButton) HEX.nextElement();
        Button.setEnabled(true);
}
Enumeration<AbstractButton> DEC = DECButtons.getElements();
while(DEC.hasMoreElements()) {
```

```
                JButton Button = (JButton) DEC.nextElement();
                Button.setEnabled(true);
        }
        }
        }
    //assign all the Buttons that are relevant to their ButtonGroup
    protected void assignButtonGrps_HEX() {

        comp = buttonPanel.getComponents();
        //DEC
        for(Component E : comp) {
                if(E instanceof JButton)
                        if(isNumber(((JButton)E).getText()))
                                if(isBetween(Integer.parseInt(((JButton)E).getText()), 0,
9))
                        DECButtons.add((JButton)E);
        }
        //HEX
        comp = HEX_BOX.getComponents();
        for(Component E : comp) {
                if(E instanceof JButton) {
                        HEXButtons.add((JButton)E);
                }
        }
    }
    //return true if min <= value <= max
    protected boolean isBetween(int value, int min, int max) {
        if(value >= min && value <= max)
```

```
                return true;
        else
                return false;
}
//Checks if the String is a single digit Number
protected boolean isNumber(String value) {
        if( value.equals("0") ||
                value.equals("1") ||
                value.equals("2") ||
                value.equals("3") ||
                value.equals("4") ||
                value.equals("5") ||
                value.equals("6") ||
                value.equals("7") ||
                value.equals("8") ||
                value.equals("9") ||
                value.equals("A") ||
                value.equals("B") ||
                value.equals("C") ||
                value.equals("D") ||
                value.equals("E") ||
                value.equals("F"))
                return true;
        else
                return false;
}
protected void ChooseCommand(String command) {
         if(command.equals("+")) {
```

```
            calc.plus();
        }
        else if(command.equals("-")) {
            calc.minus();
        }
        else if(command.equals("x")) {
            calc.multiplication();
        }
        else if(command.equals("/")) {
            calc.division();
        }
        else if(command.equals("=")) {
            calc.equals();
        }
        else if(command.equals("Clear")) {
            calc.clear();
        }
        else if(command.equals("?")) {
            showInfo();
        }
            }
}
package Übung5_2;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.util.Enumeration;
```

```
import javax.swing.AbstractButton;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JTextField;

public class UserInterface_HEX_BIN extends UserInterface_HEX {

	protected ButtonGroup BINButtons = new ButtonGroup();

	JTextField BINdisplay;

	public UserInterface_HEX_BIN(CalcEngine engine, int Amount_Checkboxes) {
		super(engine, Amount_Checkboxes);
		addLayout_BIN();
	}
	private void addLayout_BIN() {

		Boxes[2] = new JCheckBox("BIN");

		Boxes[2].addActionListener(this);

		BINdisplay = new JTextField("0");

		CheckBoxGrp.add(Boxes[2]);

	addCheckbox(Checkboxes, Boxes[2]);
	Checkboxes.add(BINdisplay);
```

```
        frame.pack();

        assignButtonGrps_BIN();
        GreyButtons_BIN();
            }
            protected void assignButtonGrps_BIN() {

                    assignButtonGrps_HEX();

                    comp = buttonPanel.getComponents();

            for(Component E : comp) {
                    if(E instanceof JButton)
                            if(isNumber(((JButton)E).getText()))
                                    if(isBetween(Integer.parseInt(((JButton)E).getText()), 0,
1))
                            BINButtons.add((JButton)E);
            }
            }
            protected void GreyButtons_BIN() {

                    GreyButtons_HEX();

                    if(Boxes[2].isSelected()) {
                    Enumeration<AbstractButton> BIN = BINButtons.getElements();
            while(BIN.hasMoreElements()) {
                    JButton Button = (JButton) BIN.nextElement();
```

```
                Button.setEnabled(true);
        }
        }
        }
public void actionPerformed(ActionEvent event)
{
        String command = event.getActionCommand();
        boolean BaseChange = false;

        for(JCheckBox E : Boxes) {
                if(E.getText().equals(command))
                        BaseChange = true;
        }
        if(BaseChange) {
                setBase_BIN();
                GreyButtons_BIN();
        }
        else if(isNumber(command)) {
        int number = Hexa.HEXToDEC(new Hexa(command));
          calc.numberPressed(number, Base);
    }
    else
        ChooseCommand(command);

    redisplay_BIN();
    updateTextfields_BIN();
    GreyButtons_BIN();
}
```

```
        protected void updateTextfields_BIN() {
                updateTextfields_HEX();

                BINdisplay.setText("" + Bin.DECtoBIN(calc.getDisplayValue()));
        }
        protected void redisplay_BIN() {
                redisplay_HEX();
                if(Boxes[2].isSelected()) {
                display.setText("" + Bin.DECtoBIN(calc.getDisplayValue()));
        }
        }
    protected void setBase_BIN() {
        setBase_HEX();

        if(Boxes[2].isSelected()) {
                System.out.println("BIN");
                Base = 2;
        }

        redisplay_BIN();
    }
}
package Übung5_2;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.util.Enumeration;
```

```
import javax.swing.AbstractButton;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JTextField;

public class UserInterface_HEX_BIN_OKT extends UserInterface_HEX_BIN {

	protected ButtonGroup OKTButtons = new ButtonGroup();

	JTextField OKTdisplay;

	public UserInterface_HEX_BIN_OKT(CalcEngine engine, int
Amount_Checkboxes) {
		super(engine, Amount_Checkboxes);
		addLayout_OKT();
	}
	private void addLayout_OKT() {

		Boxes[3] = new JCheckBox("OKT");

		Boxes[3].addActionListener(this);

		OKTdisplay = new JTextField("0");

		CheckBoxGrp.add(Boxes[3]);

	addCheckbox(Checkboxes, Boxes[3]);
```

```
        Checkboxes.add(OKTdisplay);

        frame.pack();

        assignButtonGrps_OKT();
        GreyButtons_OKT();
            }
            protected void assignButtonGrps_OKT() {

                    assignButtonGrps_BIN();

                    comp = buttonPanel.getComponents();

            for(Component E : comp) {
                    if(E instanceof JButton)
                            if(isNumber(((JButton)E).getText()))
                                    if(isBetween(Integer.parseInt(((JButton)E).getText()), 0,
7))
                            OKTButtons.add((JButton)E);
            }
            }
            protected void GreyButtons_OKT() {

                    GreyButtons_BIN();

                    if(Boxes[3].isSelected()) {
                    Enumeration<AbstractButton> OKT = OKTButtons.getElements();
            while(OKT.hasMoreElements()) {
```

```
                JButton Button = (JButton) OKT.nextElement();
                Button.setEnabled(true);
        }
        }
        }
public void actionPerformed(ActionEvent event)
{
        String command = event.getActionCommand();
        boolean BaseChange = false;

        for(JCheckBox E : Boxes) {
                if(E.getText().equals(command))
                        BaseChange = true;
        }
        if(BaseChange) {
                setBase_OKT();
                GreyButtons_OKT();
        }
        else if(isNumber(command)) {
        int number = Hexa.HEXToDEC(new Hexa(command));
          calc.numberPressed(number, Base);
    }
    else
        ChooseCommand(command);

    redisplay_OKT();
    updateTextfields_OKT();
    GreyButtons_OKT();
```

```
    }
        private void setBase_OKT() {
            setBase_BIN();

        if(Boxes[3].isSelected()) {
            System.out.println("OKT");
            Base = 8;
        }
        redisplay_OKT();
        }
        private void updateTextfields_OKT() {
            updateTextfields_BIN();

            OKTdisplay.setText("" + Okta.DECtoOKT(calc.getDisplayValue()));
        }
        private void redisplay_OKT() {
            redisplay_BIN();
            if(Boxes[3].isSelected()) {
            display.setText("" + Okta.DECtoOKT(calc.getDisplayValue()));
        }
        }
}
```

```
package Übung5_2;

public interface HexaDecimal {

        public Hexa Add(Hexa HEX);
        public Hexa Subtract(Hexa HEX);
        public Hexa Multiplycation(Hexa HEX);
        public Hexa Division(Hexa HEX);
        public String toString();
}
```

```
package Übung5_2;
```

```
public class Hexa implements HexaDecimal {
	private String value;

	public Hexa(String value) {
		setValue(value);
	}
	public String getValue() {
		return value;
	}
	public void setValue (String value) {

		char[] Cvalue = value.toCharArray();
		for(char E : Cvalue) {
			if((E < 48 && E > 57) || (E < 65 && E > 70)) {
				throw new IllegalArgumentException("only Chars
from 1-9 and A-F");
			}
		}
		this.value = value;
	}
	@Override
	public Hexa Add(Hexa HEX) {

		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a+b);
	}

	@Override
	public Hexa Subtract(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a-b);
	}

	@Override
	public Hexa Multiplycation(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a*b);
	}

	@Override
	public Hexa Division(Hexa HEX) {
		int a = Hexa.HEXToDEC(this);
		int b = Hexa.HEXToDEC(HEX);

		return Hexa.DECtoHEX(a/b);
	}
	public static int HEXToDEC(Hexa HEX) {
		return Integer.parseInt(HEX.getValue(), 16);
	}
	public static Hexa DECtoHEX(int DEC) {
		return new Hexa(Integer.toHexString(DEC).toUpperCase());
```

```
	}
	public String toString() {
		return value;
	}
}
package Übung5_2;

public interface BinSystem {

	public Bin Add(Bin bin);
	public Bin Subtract(Bin bin) throws Exception;
	public Bin Multiplycation(Bin bin);
	public Bin Division(Bin bin);
	public String toString();
}
package Übung5_2;

public class Bin implements BinSystem {

	String value;

	public Bin(String value) {
		setValue(value);
	}
	@Override
	public Bin Add(Bin bin) {
		int a = Bin.BINToDEC(this);
		int b = Bin.BINToDEC(bin);

		return Bin.DECtoBIN(a+b);
	}
	@Override
	public Bin Subtract(Bin bin) throws Exception {
		int a = Bin.BINToDEC(this);
		int b = Bin.BINToDEC(bin);
		if(a<b) throw new Exception("B is bigger then A");

		return Bin.DECtoBIN(a-b);
	}
	@Override
	public Bin Multiplycation(Bin bin) {
		int a = Bin.BINToDEC(this);
		int b = Bin.BINToDEC(bin);

		return Bin.DECtoBIN(a*b);
	}
	@Override
	public Bin Division(Bin bin) {
		int a = Bin.BINToDEC(this);
		int b = Bin.BINToDEC(bin);

		return Bin.DECtoBIN(a/b);
	}
	public static int BINToDEC(Bin bin) {
		return Integer.parseInt(bin.getValue(), 2);
	}
	public static Bin DECtoBIN(int DEC) {
		return new Bin(Integer.toBinaryString(DEC));
```

```
	}
	public String toString() {
		return value;
	}
	public String getValue() {
		return value;
	}
	public void setValue(String value) {
		this.value = value;
	}
}
package Übung5_2;

public interface OktSystem {
	public Okta Add(Okta OKT);
	public Okta Subtract(Okta OKT) throws Exception;
	public Okta Multiplycation(Okta OKT);
	public Okta Division(Okta OKT);
	public String toString();
}
package Übung5_2;

public class Okta implements OktSystem{

	String value;



	public Okta(String value) {
		setValue(value);
	}
	@Override
	public Okta Add(Okta OKT) {
		int a = Okta.OKTToDEC(this);
		int b = Okta.OKTToDEC(OKT);

		return Okta.DECtoOKT(a+b);
	}
	@Override
	public Okta Subtract(Okta OKT) throws Exception {
		int a = Okta.OKTToDEC(this);
		int b = Okta.OKTToDEC(OKT);

		return Okta.DECtoOKT(a-b);
	}
	@Override
	public Okta Multiplycation(Okta OKT) {
		int a = Okta.OKTToDEC(this);
		int b = Okta.OKTToDEC(OKT);

		return Okta.DECtoOKT(a*b);
	}
	@Override
	public Okta Division(Okta OKT) {
		int a = Okta.OKTToDEC(this);
		int b = Okta.OKTToDEC(OKT);
```

```
			return Okta.DECtoOKT(a/b);
		}
		public static int OKTToDEC(Okta OKT) {
			return Integer.parseInt(OKT.getValue(), 8);
		}
		public static Okta DECtoOKT(int DEC) {
			return new Okta(Integer.toOctalString(DEC));
		}
		public String getValue() {
			return value;
		}
		public void setValue(String value) {

			char[] Cvalue = value.toCharArray();
			for(char E : Cvalue) {
				if(E < 48 && E > 55) {
					throw new IllegalArgumentException("only Chars
from 1-7");
				}
			}
			this.value = value;
		}
		public String toString() {
			return value;
		}

}
```