Name: Kim Ngan Le Dang (570940)

Bartholomäus Berresheim (568624)

Date: 19.11.2019

Informatik 2 – 2. Zug – WiSe2019/2020

Laboratory 5: Fun with Calculator 1

Assignment

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

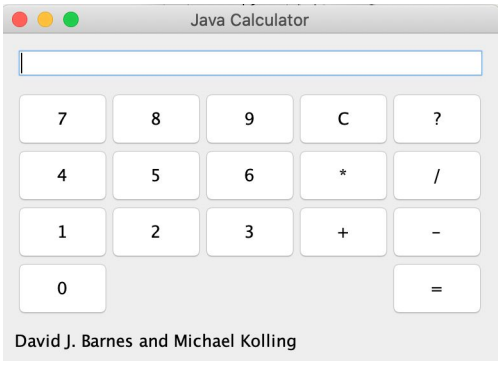
In our prelab, we both found out that the calculator misses the multiplication and division operators and that it cannot work with negative numbers yet. Our teacher said we wouldn't try to do the negative numbers for this task, so we just implemented the two missing operators. We have read through the code to know what happened in which class. We first started with the **`UserInterface`** class. We made changes in the **`makeFrame()`** method and got our new calculator interface like the screenshot below.

```
JPanel buttonPanel = new JPanel(new GridLayout(4, 5));
    addButton(buttonPanel, "7");
    addButton(buttonPanel, "8");
    addButton(buttonPanel, "9");
    addButton(buttonPanel, "C");
    addButton(buttonPanel, "?");

    addButton(buttonPanel, "4");
    addButton(buttonPanel, "5");
    addButton(buttonPanel, "6");
    addButton(buttonPanel, "*");
    addButton(buttonPanel, "/");

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

    addButton(buttonPanel, "0");
    buttonPanel.add(new JLabel(" "));
    buttonPanel.add(new JLabel(" "));
    buttonPanel.add(new JLabel(" "));
    addButton(buttonPanel, "=");
```



Then, we added two else-if-statements within the **`actionPerformed()`** method and of course, we then had to add the two new public methods namely **`multiplication()`** and **`division()`** in the CalcEngine class.

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

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

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

After adding these two methods, we added two more cases to the method **`calculateResult()`** so that our operators could work.

```
protected void calculateResult()
{
    switch(lastOperator) {
        case '*':
            displayValue = leftOperand * displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
        case '/':
            displayValue = leftOperand / displayValue;
            haveLeftOperand = true;
            leftOperand = displayValue;
            break;
```

We tested our calculator and it worked as intended! We actually wrote some lines of **`System.out.println()`** to see in the Terminal window clearly which numbers or operators were pressed and how the results looked like. These lines

of code were placed in the methods: **`numberPressed()`**, **`multiplication()`**, **`division()`**, **`plus()`**, **`minus()`**, and **`calculateResult()`**.

```
Calculator calculat1 = new Calculator();
numberPressed: 4
Operator: +
numberPressed: 3
Result: 7
Operator: *
numberPressed: 3
Result: 21
Operator: /
numberPressed: 7
Result: 3
Operator: -
numberPressed: 3
Result: 0
```

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

The first thing we did to accomplish the given task was to add the missing buttons. To do that, we started by changing the private variables from the **`UserInterface`** class into protected ones, so that we can access them from a subclass.

```
protected CalcEngine calc;
protected boolean showingAuthor;
protected JFrame frame;
protected JTextField display;
protected JLabel status;
```

After that, we created a class called **`HexUserInterface`** that extends the **`UserInterface`** class. In it we created two **`JPanel`** fields, the first one called **`hexaPanel`** and the second one called **`FcontentPane`**, that had **`frame.getContentPane()`** as its value.

```
protected JPanel FcontentPane = (JPanel)frame.getContentPane();
protected JPanel hexaPanel;
```

Then we made a **`makePanel()`** method, that would assign a new **`JPanel`** as value to our **`hexaPanel`** field and then add buttons to **`hexaPanel`** from A to F.

```
public void makePanel(){


    hexaPanel = new JPanel(new GridLayout(4, 2));
          addButton(hexaPanel, "A");
          addButton(hexaPanel, "B");
          addButton(hexaPanel, "C");
          addButton(hexaPanel, "D");

          addButton(hexaPanel, "E");
          addButton(hexaPanel, "F");

}
```

Then we created a constructor, with **`CalcEngine`** engine as its parameter, in which we called the **`super()`** method with engine as its parameter, that calls the constructor from the **`UserInterface`** class, followed by a calling of the **`makePanel()`** method. We then added the hexaPanel to the **`FcontentPane`** field and set its Layout to be on the western side, and finally we called **`frame.pack()`**, so that the frame would adapt its size to its content.

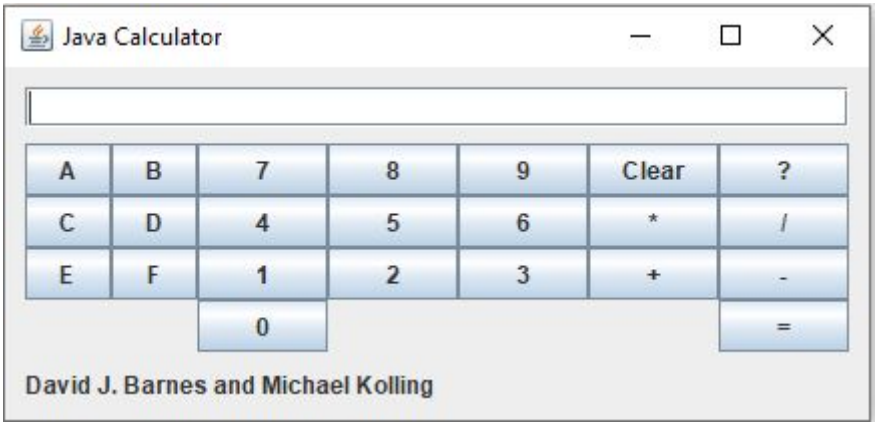
```
public HexUserInterface(CalcEngine engine) {
    super(engine);
    makePanel();
    FcontentPane.add(hexaPanel, BorderLayout.WEST);
    frame.pack();
}
```

The only thing left to do was to change the gui Object in the **`Calculator`** class from an **`UserInterface`** one to a **`HexUserInterface`** one.

```
protected HexUserInterface gui;

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

Then once executed, we got the following result:



With the first part of the task being completed, we moved on to the part of making the calculator do its calculation and displaying in Hexadecimal. To do that we had to add a few lines to the **`actionPerformed()`** and **`redisplay()`** methods contained in the **`UserInterface`** class. Within the **`actionPerformed()`** method, we added an else-if statement that would check if one of the A to F buttons were pushed. If true, it would then assign the first character from the command String minus 55 to a local char variable (for example, in ASCII A=65, then c = 65-55 = 10 and we got 10 as the value of A in Hexadecimal system). Then the chars value is assigned to a local int variable, which is then used as parameters to call the **`calc.numberPressed()`** method.

```
else if(command.equals("A") ||
        command.equals("B") ||
        command.equals("C") ||
        command.equals("D") ||
        command.equals("E") ||
        command.equals("F")) {
    char c = (char) (command.charAt(0) -55);
    int number = c;
    calc.numberPressed(number);
}
```

In the **`redisplay()`** method, we created a local **`String`** variable **`hex`** to which we assigned the into Hexadecimal converted displayValue. To do that we first called the displayValue using the **`getDisplayValue()`** method, which we then converted using the Integer method **`toHexString()`** and then we used the **`toUpperCase()`** method to turn the converted displayValue into uppercase. The **`String hex`** was then used to call the **`setText()`** method.

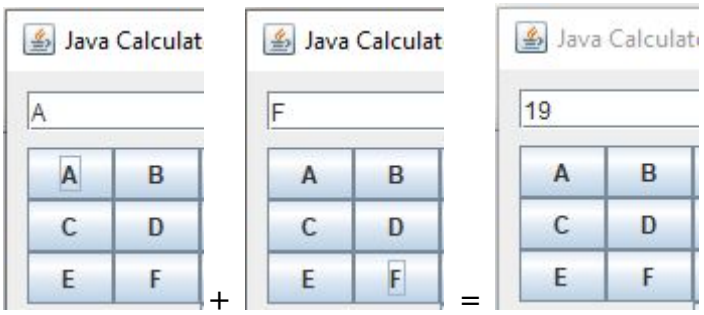
```
protected void redisplay()
{

    String hex = (Integer.toHexString(calc.getDisplayValue())).toUpperCase();
    display.setText("" + hex);

}
```

Now when pushed, the display would show the letters and not their decimal value, and the result would also be in Hexadecimal.

Ex:



A= 10; F = 15; A+F = 25; 19 = 1*16 + 9 = 25

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.**

For this task, we started by creating a new **`JPanel`** called **`checkboxPanel in the makeFrame() method`**. We then created 2 new **`JCheckBoxes`** called **`decBox`** and **`hexBox`** and labelled them accordingly. We then added an **`ActionListener`** to each of them using the **`addActionListener()`** method. After that, we created a new **`ButtonGroup`** called **`buttonG`** and added the 2 checkboxes to it. And finally we added the checkboxes to the **`checkboxPanel`**, which in turn was added to the contentPane with its **`BorderLayout`** set to North. This caused the issue of it overlapping with the display, so we chose to change the grid of the **`checkboxPanel`** and add the display to it, instead of directly adding the display to the contentPane.

```
JPanel checkboxPanel = new JPanel(new GridLayout(3,1));
display = new JTextField();
checkboxPanel.add(display);

JCheckBox decBox = new JCheckBox ("DEC");
JCheckBox hexBox = new JCheckBox ("HEX", true);

decBox.addActionListener(this);
hexBox.addActionListener(this);

ButtonGroup buttonG = new ButtonGroup();
buttonG.add(decBox);
buttonG.add(hexBox);

checkboxPanel.add(decBox);
checkboxPanel.add(hexBox);

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

After some deliberation, we chose to create an **`addCheckbox()`** method that would work like the **`addButton()`** method, with the exception of having a boolean as third parameter, creating checkboxes instead of buttons and adding the new checkboxes to **`buttonG`**. For it to work, we had to change **`buttonG`** into a field.

```
protected ButtonGroup buttonG = new ButtonGroup();

addCheckbox(checkboxPanel, "DEC", false);
addCheckbox(checkboxPanel, "HEX", true);

protected void addCheckbox(Container panel, String checkboxText, boolean selected)
{
    JCheckBox checkbox = new JCheckBox(checkboxText, selected);
    checkbox.addActionListener(this);
    buttonG.add(checkbox);
    panel.add(checkbox);
}
```

We then added some code to the **`actionPerformed()`** method, so that it would hide the Hexadecimal digits when you have the calculator in decimal mode. To do that, we added an if-else statement to check whether command is equal to DEC and one checking if its equal to HEX. If the first one is true, then hexaPanel's visibility is set to false using the **`setVisible()`** method, and if the other one is true, then hexaPanel's visibility is set to true. For this we had to change **`hexaPanel`** into an **`UserInterface`** field.

```
if(command.equals("DEC"))
{hexaPanel.setVisible(false);}

else if(command.equals("HEX"))
{hexaPanel.setVisible(true);}
```

In the **`redisplay()`** method, we added one if and one else-statement. The if statement would check whether hexaPanel is visible, if true then it would continue with the Hexadecimal conversion mentioned previously. If false, it would continue with the original content of the **`redisplay()`** method, which was to directly set the **`displayValue`** as the display text, using the **`setText()`** method.

```
protected void redisplay()
{
    if(hexaPanel.isVisible() == true)
    {
    String hex = (Integer.toHexString(calc.getDisplayValue())).toUpperCase();
    display.setText("" + hex);
    }
    else{
        display.setText("" + calc.getDisplayValue());
    }
}
```

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

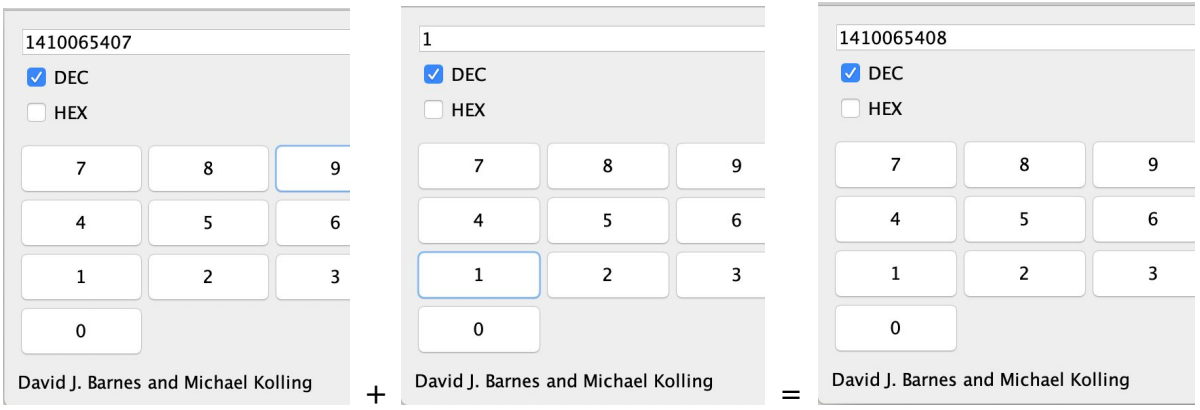
Test Decimal Calculator:

- Test plus
- Test minus
- Test division
- Test multiplication

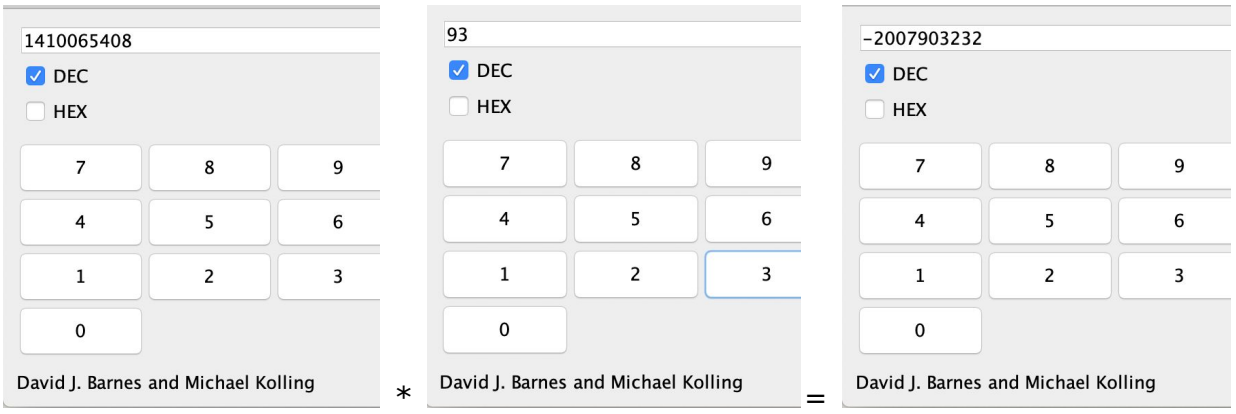
We went through the first 4 test cases and the calculator worked pretty well.

- In this calculator, we can't start the calculation with a negative number but we can use the negative result to do the calculation (for example, we can't write -1 from the beginning but we can use 0-1=-1 and use the result to do the next calculation).
- When we clicked **`Clear`** , the calculator clears all the calculation and displays a 0, but this 0 can't be taken as a digit to directly do the calculation. If we try to do so, it will return a key sequence error.
- Test Overflow/Underflow: when we entered the number that are out of range of integer number, the number will be shown as a negative number or some weird positive number and if we tried to use these number to do calculation, sometimes it returned the right result (in simple plus or minus calculations) sometimes it did not (in multiplication or division).

For example, when we tried to enter 9999999999 we had this overflow number below and we tried to add 1 to it, it still return the right answer.



Then used the result multiply 93, we got an overflow problem.

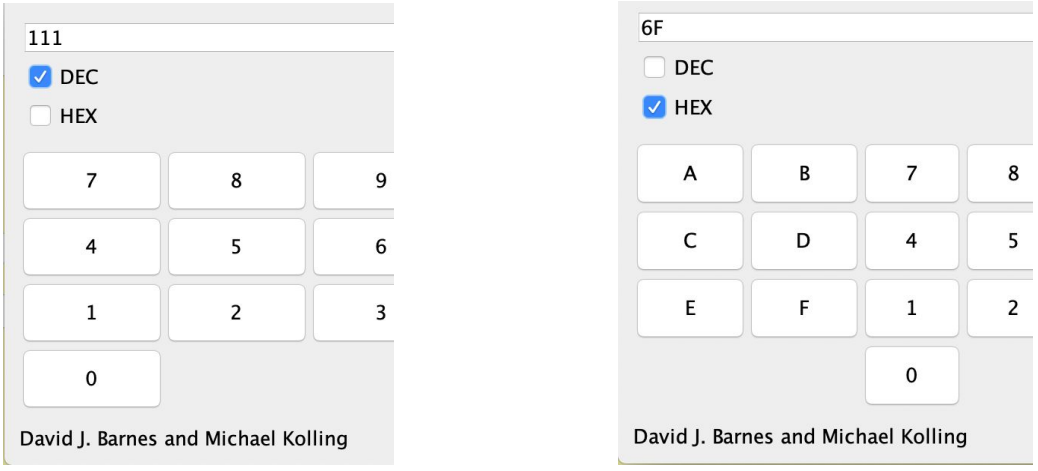


Test Hexadecimal Calculator:

- Test number display (switching the calculators just to see if the display value is the correct hex number)

We tested the number 111 in decimal and see what is the value in hexadecimal. It showed the correct answer.

$6F = 6.\ 16^1 + 15\ .\ 16^0 = 96 + 15 = 111$

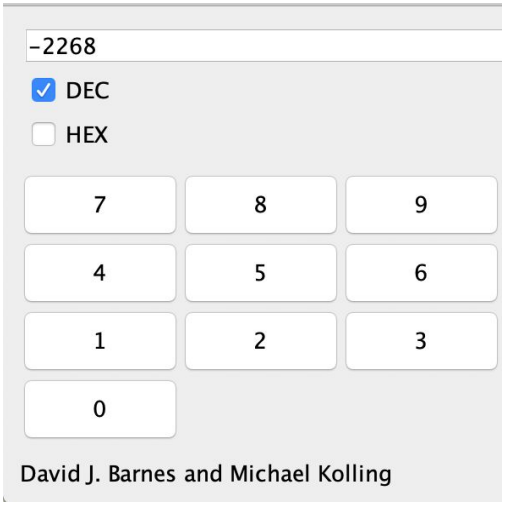


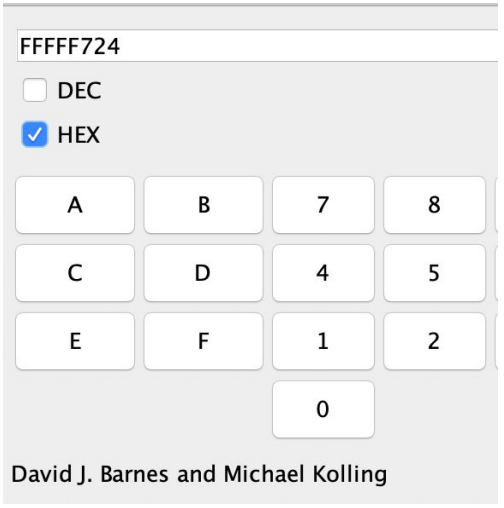
Then we converted a negative decimal number to hexadecimal and the result was a bit surprising. When it is a negative number, the first hex digits were FFFFF and followed by a hex number which was actually the result that has been converted from the negative binary result. We were first confused by the result but we read on websites and found out that those first digits just meant that it was a negative number. For example: the number -2268 is FFFFF724 in hexadecimal. We made some calculations to see what it really is.

So we had the number 2268 which is 1000110011100 in binary. We learned how to convert this binary number to its negative number (flip the value and add 1 to the new number)

1000110011100 (Flip) = 011100100011 (add 1) = 011100100100

011100100100 is the number -2268 in binary system and when we convert this binary number to hexadecimal, the result is 724, which is exactly the same as the last 3 digits of the calculator's result. Thus, the calculator worked as intended.





- Test Overflow/Underflow: We could not recognize any overflow or underflow in the hexadecimal calculator no matter how large the number is

The following 4 test cases are similar to the decimal calculator' tests and there were no problems during testing.

- Test plus
- Test minus
- Test multiplication
- Test division

Collaboration

In the second assignment, it was stated that we weren't allowed to change the superclass except for changing privates to protected. We tried to abide by these rules and looked for a way to solve the problem but ultimately we failed. After asking into the group chat if anyone knew the solution, Tony pointed out that it wasn't possible under the given conditions, since we needed access to the contentPane or buttonPanel, which were local variables of the makeFrame()

method in the UserInterface class, which meant that we had to change them into protected fields, so that we could access them from a sub-class. We did that, until we got the idea of creating a field in the HexUserInterface class with the value frame.getContentPane(), so we tried it out and it worked.

Reflection

Kim: Working with the calculator was cool. After this exercise, I understand more about the inheritance . For task 2, there was an issue at the beginning, within the constructor of the subclass because we forgot to call super(), so it returned an error message but then we could figure it out and solved the problem. It was a cool idea to use the number of characters from ASCII to get the actual integer value of the Hexadecimal digits. I wasn't sure how to do the task 3 but with the help of my partner I could finally understand it. I got to know the class ButtonGroup this time, we used it so that it creates a set of buttons but only one of them is checked at a time. I had fun working with Bartho, I did not have much time for doing this lab but he helped me a lot with the code and explained it to me.

Bartho: This week's lab had its ups and downs. The positive parts are that it helped with my understanding of inheritances, java.awt and java.swing. The downside was trying to find a way of accessing the contentPane from the subclass without changing it, although it made it much more gratifying, when I thought of using the frame.getContentPane() and it worked. For you it might have been the obvious solution but it sure wasn't for me. I guess that is the most important thing I learned in this lab, that I need to be more careful/attentive when reading through a pre-existing program. Although I ended up not using the idea from Tony, it was still helpful with getting my mind of that part of the task, so that I could continue with the rest of it.

Appendix A

Source code of the project

1. **Calculator**

```
package ubung_5;

/**
* 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
```

```
* @edited by Kim & Bartho 24.11.2019
*/
public class Calculator
{
   protected CalcEngine engine;
   protected HexUserInterface gui;

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

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

2. UserInterface

```
package ubung_5;

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
 * @edited by Kim & Bartho 24.11.2019
 */
public class UserInterface
    implements ActionListener
{
    protected CalcEngine calc;
    protected boolean showingAuthor;
    protected JFrame frame;
    protected JTextField display;
    protected JLabel status;

    protected JPanel hexaPanel;
```

```
protected ButtonGroup buttonG = new ButtonGroup();
/**
 * 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.
 */
protected void makeFrame()
{
    frame = new JFrame(calc.getTitle());
    JPanel contentPane = (JPanel)frame.getContentPane();

    contentPane.setLayout(new BorderLayout(0, 8));
    contentPane.setBorder(new EmptyBorder( 10, 10, 10, 10));

    JPanel buttonPanel = new JPanel(new GridLayout(4, 5));

    addButton(buttonPanel, "7");
    addButton(buttonPanel, "8");
    addButton(buttonPanel, "9");
    addButton(buttonPanel, "Clear");
    addButton(buttonPanel, "?");

    addButton(buttonPanel, "4");
    addButton(buttonPanel, "5");
    addButton(buttonPanel, "6");
    addButton(buttonPanel, "*");
    addButton(buttonPanel, "/");

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

```
        addButton(buttonPanel, "0");
        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(new JLabel(" "));
        buttonPanel.add(new JLabel(" "));
        addButton(buttonPanel, "=");

        contentPane.add(buttonPanel, BorderLayout.CENTER);

        JPanel checkboxPanel = new JPanel(new GridLayout(3,1));
        display = new JTextField();
        checkboxPanel.add(display);

        addCheckbox(checkboxPanel, "DEC", false);
        addCheckbox(checkboxPanel, "HEX", true);

        contentPane.add(checkboxPanel, BorderLayout.NORTH);

        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 addCheckbox(Container panel, String checkboxText, boolean
selected)
    {
       JCheckBox checkbox = new JCheckBox(checkboxText, selected);
       checkbox.addActionListener(this);
       buttonG.add(checkbox);
        panel.add(checkbox);
    }

    protected 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("DEC"))
{hexaPanel.setVisible(false);}

if(command.equals("HEX"))
{
    hexaPanel.setVisible(true);}

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);
}
else if(command.equals("A") ||
            command.equals("B") ||
            command.equals("C") ||
            command.equals("D") ||
            command.equals("E") ||
            command.equals("F")) {
     char c = (char) (command.charAt(0) -55);
     int number = c;
    calc.numberPressed(number);
}
else if(command.equals("*")) {
    calc.multiplication();
}
else if(command.equals("/")) {
    calc.division();
}
else if(command.equals("+")) {
    calc.plus();
}
else if(command.equals("-")) {
    calc.minus();
}
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()
    {
       if(hexaPanel.isVisible() == true)
       {
       String hex = (Integer.toHexString(calc.getDisplayValue())).toUpperCase();
        display.setText("" + hex);
       }
       else{
              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;
    }
}
```

3. HexUserInterface

```
package ubung_5;

import java.awt.*;
import javax.swing.*;
/**
 * Write a description of class HexUserInterface here.
 *
 * @author Kim & Bartho
 * @version 24.11.2019
 */
public class HexUserInterface extends UserInterface
{
    protected JPanel FcontentPane = (JPanel)frame.getContentPane();

    public HexUserInterface(CalcEngine engine) {
              super(engine);
              makePanel();
```

```
            FcontentPane.add(hexaPanel, BorderLayout.WEST);
            frame.pack();
        }

    public void makePanel(){

        hexaPanel = new JPanel(new GridLayout(4, 2));
        addButton(hexaPanel, "A");
        addButton(hexaPanel, "B");
        addButton(hexaPanel, "C");
        addButton(hexaPanel, "D");

        addButton(hexaPanel, "E");
        addButton(hexaPanel, "F");

    }
}
```

4. CalcEngine

```
package ubung_5;

/**
* The main part of the calculator doing the calculations.
*
* @author  David J. Barnes and Michael Kolling
* @version 2008.03.30
* @edited by Kim & Bartho 24.11.2019
*/
public class CalcEngine
{
   // The calculator's state is maintained in three fields:
   //     buildingDisplayValue, haveLeftOperand, and lastOperator.

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

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

```
}

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

    return displayValue;
}

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

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

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

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

```
/**
 * The 'minus' button was pressed.
 */
public void minus()
{
    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 "David J. Barnes and Michael Kolling";
}

/**
 * @return The version number of this engine.
 */
public String getVersion()
{
   return "Version 1.1";
}

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

5. Run (Only needed for eclipse use, not for BlueJ)

```
package ubung_5;

public class Run {

	public static void main(String[] args) {
		Calculator calculator = new Calculator();
		calculator.show();
	}

}
```

Appendix B

Kim's Prelab

1. **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?**

There're a lot of things that are missing. For example, operator for division, multiple, parentheses, we can't write negative numbers.

This calculation can only process a single calculation. It solves the simple problems and cannot do the complex calculation. The display screen can only display either the operator or the number.

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

"The **hexadecimal numeral system**, often shortened to **"hex"**, is a numeral system made up of 16 symbols (base 16). The standard numeral system is called decimal(base 10) and uses ten symbols: 0,1,2,3,4,5,6,7,8,9. Hexadecimal uses the decimal numbers and six extra symbols. There are no numerical symbols that represent values greater than nine, so letters taken from the English alphabet are used, specifically A, B, C, D, E and F. Hexadecimal A = decimal 10, and hexadecimal F = decimal 15."

Source: https://simple.wikipedia.org/wiki/Hexadecimal

Bartho's Prelab

2. The things I noticed, that were missing from the calculator, were multiplication, division and negative numbers as input possibility.
3. Simply put, hexadecimal numbers are like binary numbers but with 16 as base instead of 2. Hexadecimal numbers go from 0-9 and then from A-F with A being 10 and F being 15.

References

https://stackoverflow.com/questions/33629416/how-to-tell-if-hex-value-is-negative/33629511

https://stackoverflow.com/questions/5826818/are-hexadecimal-numbers-ever-negative/5827491#5827491

https://coderanch.com/t/246080/java-programmer-OCPJP/certification/Negative-Hexadecimal-number