Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

B. Sc. Internationale Medieninformatik
Prof. Dr. Weber-Wulff

Group 1
Tuesday, 19.11.2019

Group members:

Timo Schmidt	571252	timo.schmidt@student.htw-berlin.de
Pavel Tsvyatkov	559632	pavel.tsvyatkov@student.htw-berlin.de

Post-Lab Report

Lab 5: Fun with Calculators 1

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

We discovered that there is no possibility to multiply or divide numbers in this calculator. We decided to implement both since it is done very quickly.

We first added two new methods called multiply() and divide(). They both refer to the applyOperator method with their corresponding character.

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

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

Next we added new cases in the applyOperator() method for the two operators.

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

Now that we have added the functionality we needed to create new buttons for the GUI and made them refer to their corresponding methods. To do so we went into our UserInterface class and changed the GridLayout to have one more row (from 4 to 5).

```
JPanel buttonPanel = new JPanel(new GridLayout( rows: 5,  cols: 4));
```

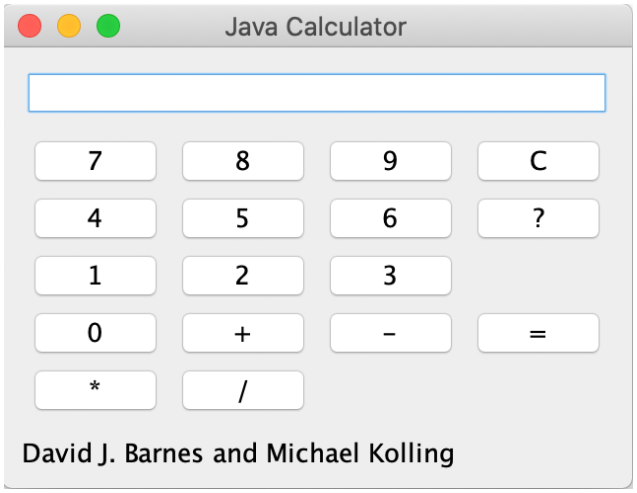
We also added two new buttons:

```
addButton(buttonPanel,  buttonText: "*");
addButton(buttonPanel,  buttonText: "/");
```

Last but not least we added the following lines to our actionPerformed method to make sure the methods get called by clicking on one of the new buttons:

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

The result looked like this:



We tested our new buttons and they worked fine (see later in the test cases of exercise 4).

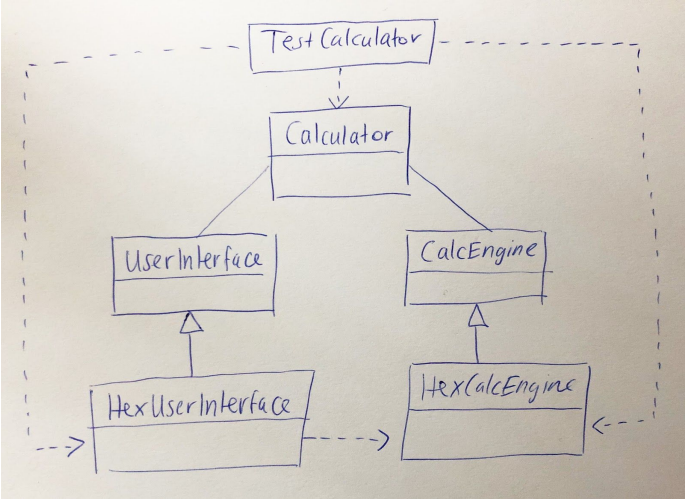
Moreover, we have found another error in this calculator: If you type in a number and press the equals symbol ("=") right after, the value gets changed back to zero instead of keeping the current value. The latter was the case when we tested it on the built-in Windows calculator, so we decided to change it in our calculator as well.

We fixed this by adding a new if statement to the equals() method.

```
// Added functionality: Value stays the same if pressing = right after value
else if(!haveLeftOperand) {
    leftOperand = displayValue;
}
```

Now the value stays the same when clicking "=".

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 first drew a sketch for determining which new classes we would need in order to implement hexadecimal digits. We thought the easiest solution would be to work with two classes, one should extend the UserInterface and one the CalcEngine.

We started by creating a new class called HexUserInterface. This class extends the UserInterface class. In its constructor we called the superclass with CalcEngine as its parameter (we later changed it to our new HexCalcEngine after creating the corresponding subclass).

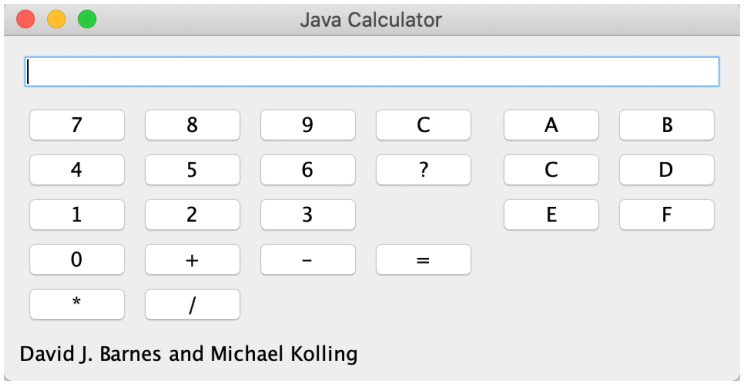
```
public HexUserInterface(HexCalcEngine engine) {
    super(engine);
}
```

We then rewrote the method makeFrame() to include our new buttons. To do so we needed to change private to protected though. We called the same method of the superclass to make sure it is also showing the old buttons. After that we created a new JPanel buttonPanelHex with a new GridLayout of 3 rows and 2 columns.

In order to create the buttons for the hexadecimal values we needed to change the addButton method to protected first and were then able to create the buttons. Finally we added frame.pack(); to our method to make sure the proportions are right.

The result looked like that:

```
buttonPanelHex = new JPanel(new GridLayout( rows: 5, cols: 2));

addButton(buttonPanelHex,  buttonText: "A");
addButton(buttonPanelHex,  buttonText: "B");

addButton(buttonPanelHex,  buttonText: "C");
addButton(buttonPanelHex,  buttonText: "D");

addButton(buttonPanelHex,  buttonText: "E");
addButton(buttonPanelHex,  buttonText: "F");
```



We discovered that we can no longer differentiate between the clear button (“C”) and our new hexadecimal button (also “C”). We decided to change the clear button to “AC” instead (like on several other calculators).

Now we are hitting the critical part: Make the calculator do its calculations in hexadecimal:

We first created the other class HexCalcEngine which extends the CalcEngine class of our calculator. It also calls its super class in the constructor by using super().

We figured out that we need to multiply the displayValue by 16 instead of 10 (like in decimal system) and add the new number when using a hexadecimal value with multiple digits. Unfortunately this new method was also used for the normal decimal calculations. In order to solve that problem we added a new boolean inHexMode as a field which is false at the beginning.

```
public void numberPressed(int number) {

    if (inHexMode) {
        if (buildingDisplayValue) {
            // Incorporate this digit.
            displayValue = displayValue * 16 + number;
        } else {
            // Start building a new number.
            displayValue = number;
            buildingDisplayValue = true;
        }
    }

    else
        super.numberPressed(number);
}
```

This boolean becomes true when the checkbox is selected (see exercise 3). To do so we needed to create a new method setHexMode with a boolean as parameter which can be called in the HexUserInterface class later on.

```
protected void setHexMode(boolean b) {
    inHexMode = b;
}
```

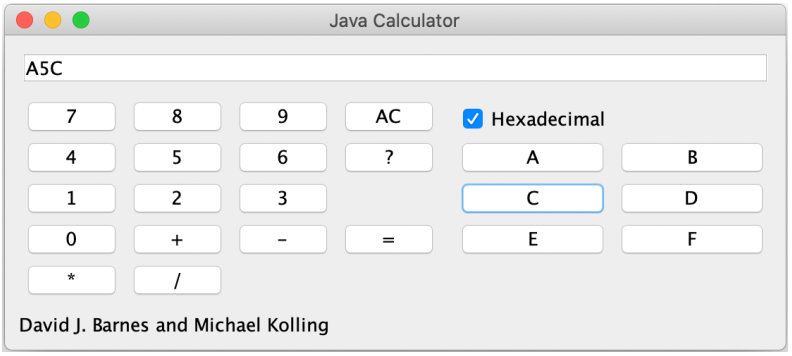
Back in the HexUserInterface class we continued with the actionPerformed method. Once again we called the same method of the superclass in the first line (to make sure the other buttons are still working). Instead of using if-statements like in the UserInterface class we thought it would be a better idea to use switch cases instead. We created cases for “A” to “F” and used our new numberPressed method to type in numbers (A = 10 … F = 15). Now we realized that we would need another method to show the current hexadecimal value on the display.

We created a new method buildHexValue with the command (String) as parameter and also return a String at the end. We also created two new fields: private String displayValueHex and a boolean buildingDisplayValueHex. This is working similar to the numberPressed method in CalcEngine. If any number gets typed in the first time the boolean becomes true and the displayValueHex is our command (e.g. “A” when you press “A”). If you type in more numbers the characters get added to the displayValueHex String. This String gets returned at the end of the method.

```
private String buildHexValue(String command) {
    if (buildingDisplayValueHex) {
        displayValueHex += command;
    }
    else {
        displayValueHex = command;
        buildingDisplayValueHex = true;
    }
    return displayValueHex;
}
```

To make sure the displayValueHex-String is shown on the display we called the method setText of display and made its value the returned String of the method we have just created. We also needed to include all the numbers when you are in hexadecimal mode so we added corresponding switch

cases for them as well. Last but not least we changed the boolean buildingDisplayValueHex to false when the command is one of the operators, equals or clear (AC).



Finally we needed to implement a new redisplay method. Once again we call the same method of the superclass to make sure the functionality for decimal numbers stays the same.

In case we are operating in hexadecimal mode (if checkbox.isSelected()) - see exercise 3) we set the display text to the corresponding hexadecimal result. To do that we wrote a short method with return type String getHexadecimal() which pulls out the current displayValue of our HexCalcEngine and transforms it into a hexadecimal String by using the method .toHexString() of Integer. In order to output the hexadecimal letters in uppercase we also added .toUpperCase().

```
protected String getHexadecimal() {
    int dec = calc.getDisplayValue();
    return Integer.toHexString(dec).toUpperCase();
}
```

We are using this method in our redisplay() method and also check if the value on the display is currently in construction (buildingDisplayValueHex == true). If so we show the displayValueHex String instead. Everything was working as expected.

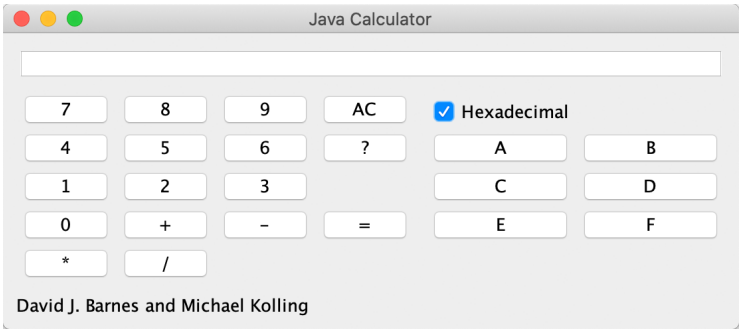
```
protected void redisplay() {
    super.redisplay();

    // as long as the value gets built, show the String instead of number
    if (checkbox.isSelected()) {
        if (buildingDisplayValueHex)
            display.setText(displayValueHex);
        display.setText(getHexadecimal());
    }
}
```

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

To be able to switch between decimal and hexadecimal, we decided to use a JCheckBox. To do this, we first wrote a private field for the checkbox in our HexUserInterface class. In the makeFrame() method we assigned its value to be a new JCheckbox with the text “Hexadecimal”. We added the button to our buttonHexPanel by writing the following:

```
buttonPanelHex.add(checkbox);
```

We also added an empty JLabel to make sure the first hexadecimal button starts in the next row. The result looked like that:



We thought about a solution for graying out the buttons when we work in decimal mode for quite some time. In the Java API (java.awt.Container) we found the method getComponents(). We started by writing a new method and called it checkBoxAction.

Then we decided to create an array of Components, which holds all the components in our buttonPanelHex. After that we used a for loop and checked if the component is an instance of JButton. In the API in java.awt.Label, we found the method getText, which we use to check if the components we loop through match “A-F”. In java.awt.Component we found a useful method setEnabled which takes a boolean value as a parameter, so we wrote a new boolean variable isChecked and passed it. At this point our method looked like this:

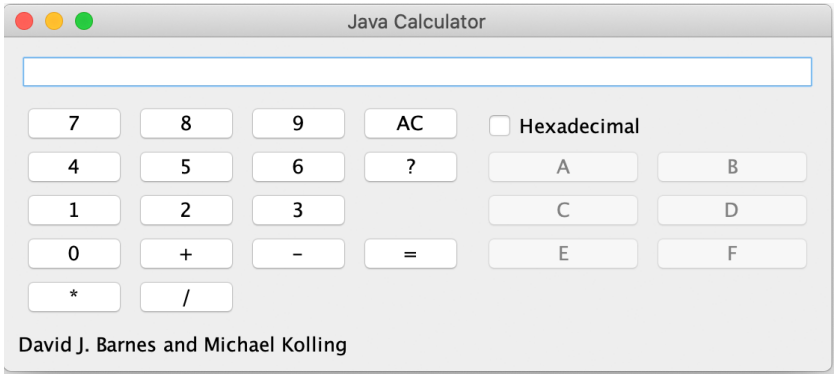
```
private void checkBoxAction() {
    boolean isChecked = checkbox.isSelected();
    ((HexCalcEngine) calc).inHexMode = isChecked;

    Component[] button = buttonPanelHex.getComponents();
    for (Component component : button) {
        if (component instanceof JButton) {
            if (((JButton) component).getText().matches( regex: "[A-F]")) {
                component.setEnabled(isChecked);
            }
        }
    }
}
```

The next thing we had to do was to add an ItemListener to our checkbox to gray out the hexadecimal buttons correctly when we click it (by calling the method we have just created). We also needed the itemStateChanged method so we know when the checkbox gets selected or deselected.

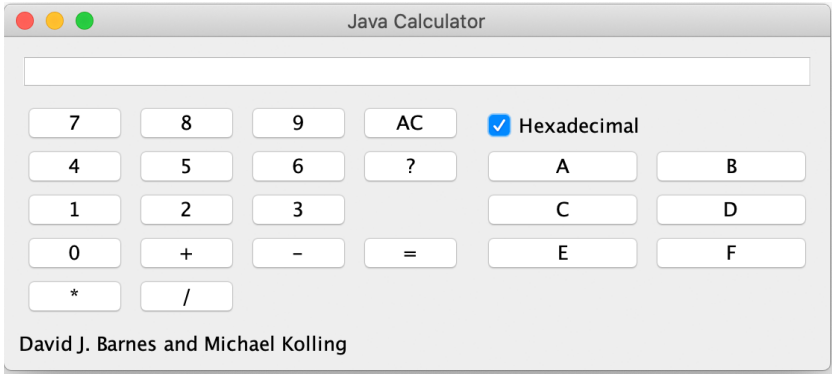
```
checkbox.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        checkBoxAction();
    }
});
```

After that we went on to test how the checkbox works in our calculator. The ItemListener was working properly but the initial state was wrong so we also called the method checkBoxAction() once in the makeFrame() method to make sure its initial state is also grayed out.

Now everything was working.



When we check Hexadecimal to swap to hexadecimal mode, the buttons are no longer grayed out and we are able to use them.



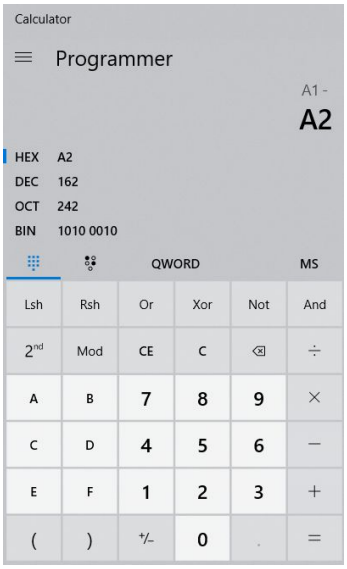
4.) What test cases do you need to test your calculator? If you find errors in your calculator, document them in your report!

We first thought about several tests we would like to execute with our calculator. We created a new test class of Calculator and used JUnit to include several test cases.

Here you can see some examples:

```
@Test
public void T03_01_TestHexAddition() {
    HexCalcEngine hexcalc = new HexCalcEngine();
    HexUserInterface gui = new HexUserInterface(hexcalc);
    hexcalc.setHexMode(true);

    // A + A
    hexcalc.numberPressed(10);
    hexcalc.plus();
    hexcalc.numberPressed(10);
    hexcalc.equals();
    gui.redisplay();
    assertEquals( expected: "14", gui.getHexadecimal());

    // FF + AA
    hexcalc.numberPressed(15);
    hexcalc.numberPressed(15);
    hexcalc.plus();
    hexcalc.numberPressed(10);
    hexcalc.numberPressed(10);
    hexcalc.equals();
    gui.redisplay();
    assertEquals( expected: "1A9", gui.getHexadecimal());

    // A5 + C9
    hexcalc.numberPressed(10);
    hexcalc.numberPressed(5);
    hexcalc.plus();
    hexcalc.numberPressed(12);
    hexcalc.numberPressed(9);
    hexcalc.equals();
    gui.redisplay();
    assertEquals( expected: "16E", gui.getHexadecimal());

    // ACD + F96
    hexcalc.numberPressed(10);
    hexcalc.numberPressed(12);
    hexcalc.numberPressed(13);
    hexcalc.plus();
    hexcalc.numberPressed(15);
    hexcalc.numberPressed(9);
    hexcalc.numberPressed(6);
    hexcalc.equals();
    gui.redisplay();
    assertEquals( expected: "1A63", gui.getHexadecimal());
}
```

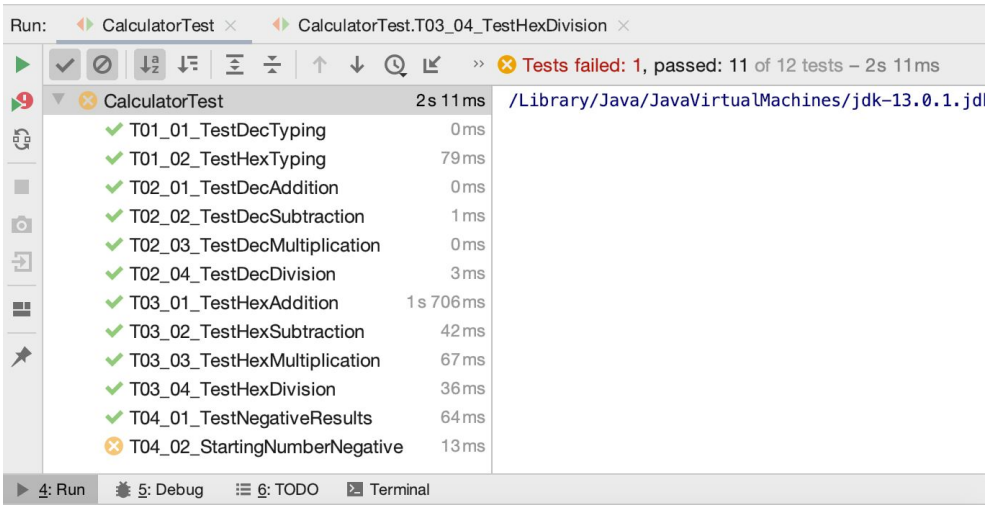
```
@Test
public void T04_01_TestNegativeResults() {
    HexCalcEngine hexcalc = new HexCalcEngine();
    HexUserInterface gui = new HexUserInterface(hexcalc);

    // Decimal mode 10 - 15
    hexcalc.setHexMode(false);
    hexcalc.numberPressed(1);
    hexcalc.numberPressed(0);
    hexcalc.minus();
    hexcalc.numberPressed(1);
    hexcalc.numberPressed(5);
    hexcalc.equals();
    assertEquals( expected: -5, hexcalc.getDisplayValue());

    // Hexidecimal mode A1 - A2
    hexcalc.setHexMode(true);
    hexcalc.numberPressed(10);
    hexcalc.numberPressed(1);
    hexcalc.minus();
    hexcalc.numberPressed(10);
    hexcalc.numberPressed(2);
    hexcalc.equals();
    gui.redisplay();
    assertEquals( expected: "FFFFFFFF", gui.getHexadecimal());
}
```



While we are in hexadecimal mode, we noticed that trying to calculate A1−A2 shows FFFFFFFF as a result.

We were not sure if that behaviour is correct, so we tested it on the built in Windows calculator and got the same result (but with fewer F's because our calculator is using Integers and is therefore limited).

After finishing all the test cases we tested them with our software. The result looked like that:



The last test fails because we are not able to start with negative numbers in our calculator. As we mentioned before our calculator is working with Integers only. That is why it is not possible to type in floating numbers or get any as a result.

When calculating with big values it is also possible to get wrong results due to an Overflow. An integer is limited to the value 2 147 483 647 (decimal) / 7FFF FFFF (hexadecimal).

What did we learn?

Timo Schmidt

After learning about superclasses/subclasses in the first semester I could finally use my knowledge to implement it in a real project. I also gained a better understanding for frames, panels and buttons and how to implement an ActionListener/ItemListener. Last but not least I figured out how to work with JUnit in Eclipse/IntelliJ instead of using BlueJ for that matter.

Pavel Tsvyatkov

Playing around with the calculator and figuring out how it works was a lot of fun. I learned more about the Components and Containers in java.awt. While integrating the checkbox, I learned about ItemListener and the other methods we used to make the checkbox work, like isSelected and setEnabled. I also learned how to use JUnit in Eclipse to be able to write and test methods.

Appendix

Calculator.java

```
/**
* 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
*         extended by Timo Schmidt, Pavel Tsvyatkov
* @version 2019.11.24
*/
public class Calculator
{
   private HexCalcEngine engine;
   private HexUserInterface gui;

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

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

UserInterface.java

```
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
*         extended by Timo Schmidt, Pavel Tsvyatkov
* @version 2019.11.24
*/
public class UserInterface
   implements ActionListener
{
   protected CalcEngine calc;
   protected boolean showingAuthor;

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

   /**
    * 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(8, 8));
       contentPane.setBorder(new EmptyBorder( 10, 10, 10, 10));
```

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

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

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

            addButton(buttonPanel, "1");
            addButton(buttonPanel, "2");
            addButton(buttonPanel, "3");
            buttonPanel.add(new JLabel(""));

            addButton(buttonPanel, "0");
            addButton(buttonPanel, "+");
            addButton(buttonPanel, "-");
            addButton(buttonPanel, "=");

            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);
    }

    /**
     * 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);
        }
        else if(command.equals("+")) {
            calc.plus();
        }
        else if(command.equals("-")) {
            calc.minus();
        }
        else if(command.equals("*")) {
            calc.multiply();
        }
        else if(command.equals("/")) {
            calc.divide();
        }
        else if(command.equals("=")) {
            calc.equals();
        }
            else if(command.equals("AC")) {             // renamed due to double
appearance
             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.
     */
    private void showInfo()
    {
        if(showingAuthor)
            status.setText(calc.getVersion());
        else
             status.setText(calc.getAuthor());

        showingAuthor = !showingAuthor;
    }
}
```

CalcEngine.java

```
/**
* The main part of the calculator doing the calculations.
*
* @author David J. Barnes and Michael Kolling
*         extended by Timo Schmidt, Pavel Tsvyatkov
* @version 2019.11.24
*/
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?
    protected 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.
    protected 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 'plus' button was pressed.
     */
    public void plus()
    {
        applyOperator('+');
    }

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

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

    /**
     * The 'divide' button was pressed.
     */
    public void divide()
    {
        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;
        }
          // Added functionality: Value stays the same if pressing = right after
value
        else if(!haveLeftOperand) {
            leftOperand = displayValue;
        }
        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 1.0";
    }

    /**
     * Combine leftOperand, lastOperator, and the
     * current display value.
     * The result becomes both the leftOperand and
     * the new display value.
     */
    protected 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();
    }
}
```

HexUserInterface.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class HexUserInterface extends UserInterface {

   private String displayValueHex = "";
   private boolean buildingDisplayValueHex;
   private JCheckBox checkbox;
   private JPanel buttonPanelHex;

   public HexUserInterface(HexCalcEngine engine) {
       super(engine);
   }

   protected void makeFrame() {
       super.makeFrame();

       JPanel contentPane = (JPanel)frame.getContentPane();
       checkbox = new JCheckBox("Hexadecimal");

       buttonPanelHex = new JPanel(new GridLayout(5,2));
       buttonPanelHex.add(checkbox);
       buttonPanelHex.add(new JLabel(""));

       addButton(buttonPanelHex, "A");
       addButton(buttonPanelHex, "B");

       addButton(buttonPanelHex, "C");
       addButton(buttonPanelHex, "D");

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

       contentPane.add(buttonPanelHex, BorderLayout.EAST);

         // This method gets called once to grey out the buttons in the initial
state
       checkBoxAction();

       checkbox.addItemListener(new ItemListener() {
           public void itemStateChanged(ItemEvent e) {
               checkBoxAction();
           }
       });

       frame.pack(); // changed to protected
   }

   public void actionPerformed(ActionEvent event) {
       super.actionPerformed(event);
       String command = event.getActionCommand();

       switch (command) {
           case "0":
```

```
            case "1":
            case "2":
            case "3":
            case "4":
            case "5":
            case "6":
            case "7":
            case "8":
            case "9":
                display.setText(buildHexValue(command));
                break;
            case "A":
                display.setText(buildHexValue(command));
                calc.numberPressed(10);
                break;
            case "B":
                display.setText(buildHexValue(command));
                calc.numberPressed(11);
                break;
            case "C":
                display.setText(buildHexValue(command));
                calc.numberPressed(12);
                break;
            case "D":
                display.setText(buildHexValue(command));
                calc.numberPressed(13);
                break;
            case "E":
                display.setText(buildHexValue(command));
                calc.numberPressed(14);
                break;
            case "F":
                display.setText(buildHexValue(command));
                calc.numberPressed(15);
                break;
            case "+":
            case "-":
            case "*":
            case "/":
            case "=":
            case "AC":
                buildingDisplayValueHex = false;
                break;
        }

    }

    private String buildHexValue(String command) {
        if (buildingDisplayValueHex) {
            displayValueHex += command;
        }
        else {
            displayValueHex = command;
            buildingDisplayValueHex = true;
        }
        return displayValueHex;
    }

    protected void redisplay() {
        super.redisplay();
```

```
        // as long as the value gets built, show the String instead of number
        if (checkbox.isSelected()) {
            if (buildingDisplayValueHex)
                display.setText(displayValueHex);
            display.setText(getHexadecimal());
        }
    }

    protected String getHexadecimal() {
        int dec = calc.getDisplayValue();
        return Integer.toHexString(dec).toUpperCase();
    }

    /**
         * This method gets called when the checkbox state changes
(checked/unchecked)
     */
    private void checkBoxAction() {
        boolean isChecked = checkbox.isSelected();
        ((HexCalcEngine) calc).inHexMode = isChecked;

        Component[] button = buttonPanelHex.getComponents();
        for (Component component : button) {
            if (component instanceof JButton) {
                if (((JButton) component).getText().matches("[A-F]")) {
                    component.setEnabled(isChecked);
                }
            }
        }
    }

}
```

HexCalcEngine.java

```
public class HexCalcEngine extends CalcEngine {

   protected boolean inHexMode = false;

   public HexCalcEngine() {
       super();
   }

   public void numberPressed(int number) {

       if (inHexMode) {
           if (buildingDisplayValue) {
               // Incorporate this digit.
               displayValue = displayValue * 16 + number;
           } else {
               // Start building a new number.
               displayValue = number;
              buildingDisplayValue = true;
           }
       }

       else
           super.numberPressed(number);
   }

   protected void setHexMode(boolean b) {
       inHexMode = b;
   }


}
```

CalculatorTest.java

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

/**
* CalculatorTest class.
*
* @author  Timo Schmidt, Pavel Tsvyatkov
* @version 2019.11.24
*/
public class CalculatorTest
{
   public CalculatorTest()
   {
   }

   @Before
   public void setUp()
   {
   }

   @After
   public void tearDown()
   {
   }

   @Test
   public void T01_01_TestDecTyping()
   {
       HexCalcEngine hexcalc = new HexCalcEngine();

       // 255
       hexcalc.numberPressed(2);
       hexcalc.numberPressed(5);
       hexcalc.numberPressed(5);
       assertEquals(255, hexcalc.getDisplayValue());
   }

   @Test
   public void T01_02_TestHexTyping()
   {
       HexCalcEngine hexcalc = new HexCalcEngine();
       HexUserInterface gui = new HexUserInterface(hexcalc);
       hexcalc.setHexMode(true);

       // DAD
       hexcalc.numberPressed(13);
       hexcalc.numberPressed(10);
       hexcalc.numberPressed(13);
       gui.redisplay();
       assertEquals("DAD", gui.getHexadecimal()); // F - A = 5
   }

   @Test
   public void T02_01_TestDecAddition()
   {
```

```
        HexCalcEngine hexcalc = new HexCalcEngine();

        // 2 + 5
        hexcalc.numberPressed(2);
        hexcalc.plus();
        hexcalc.numberPressed(5);
        hexcalc.equals();
        assertEquals(7, hexcalc.getDisplayValue());

        // 20 + 60
        hexcalc.numberPressed(2);
        hexcalc.numberPressed(0);
        hexcalc.plus();
        hexcalc.numberPressed(6);
        hexcalc.numberPressed(0);
        hexcalc.equals();
        assertEquals(80, hexcalc.getDisplayValue());


        // 122 + 188
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(2);
        hexcalc.numberPressed(2);
        hexcalc.plus();
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(8);
        hexcalc.numberPressed(8);
        hexcalc.equals();
        assertEquals(310, hexcalc.getDisplayValue());
    }

    @Test
    public void T02_02_TestDecSubtraction()
    {
        HexCalcEngine hexcalc = new HexCalcEngine();

        // 9 - 5
        hexcalc.numberPressed(9);
        hexcalc.minus();
        hexcalc.numberPressed(5);
        hexcalc.equals();
        assertEquals(4, hexcalc.getDisplayValue());

        // 20 - 13
        hexcalc.numberPressed(2);
        hexcalc.numberPressed(0);
        hexcalc.minus();
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(3);
        hexcalc.equals();
        assertEquals(7, hexcalc.getDisplayValue());
    }

    @Test
    public void T02_03_TestDecMultiplication()
    {
        HexCalcEngine hexcalc = new HexCalcEngine();

        // 3 * 3
        hexcalc.numberPressed(3);
```

```
        hexcalc.multiply();
        hexcalc.numberPressed(3);
        hexcalc.equals();
        assertEquals(9, hexcalc.getDisplayValue());

        // 10 * 12
        hexcalc.numberPressed(10);
        hexcalc.multiply();
        hexcalc.numberPressed(12);
        hexcalc.equals();
        assertEquals(120, hexcalc.getDisplayValue());
    }

    @Test
    public void T02_04_TestDecDivision()
    {
        HexCalcEngine hexcalc = new HexCalcEngine();

        // 9 : 3
        hexcalc.numberPressed(9);
        hexcalc.divide();
        hexcalc.numberPressed(3);
        hexcalc.equals();
        assertEquals(3, hexcalc.getDisplayValue());

        // 40 : 10
        hexcalc.numberPressed(40);
        hexcalc.divide();
        hexcalc.numberPressed(10);
        hexcalc.equals();
        assertEquals(4, hexcalc.getDisplayValue());
    }


    @Test
    public void T03_01_TestHexAddition() {
        HexCalcEngine hexcalc = new HexCalcEngine();
        HexUserInterface gui = new HexUserInterface(hexcalc);
        hexcalc.setHexMode(true);

        // A + A
        hexcalc.numberPressed(10);
        hexcalc.plus();
        hexcalc.numberPressed(10);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("14", gui.getHexadecimal());

        // FF + AA
        hexcalc.numberPressed(15);
        hexcalc.numberPressed(15);
        hexcalc.plus();
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(10);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("1A9", gui.getHexadecimal());

        // A5 + C9
        hexcalc.numberPressed(10);
```

```
        hexcalc.numberPressed(5);
        hexcalc.plus();
        hexcalc.numberPressed(12);
        hexcalc.numberPressed(9);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("16E", gui.getHexadecimal());

        // ACD + F96
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(12);
        hexcalc.numberPressed(13);
        hexcalc.plus();
        hexcalc.numberPressed(15);
        hexcalc.numberPressed(9);
        hexcalc.numberPressed(6);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("1A63", gui.getHexadecimal());
    }

    @Test
    public void T03_02_TestHexSubtraction() {
        HexCalcEngine hexcalc = new HexCalcEngine();
        HexUserInterface gui = new HexUserInterface(hexcalc);
        hexcalc.setHexMode(true);

        // F - A
        hexcalc.numberPressed(15);
        hexcalc.minus();
        hexcalc.numberPressed(10);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("5", gui.getHexadecimal());

        // CC - AA
        hexcalc.numberPressed(12);
        hexcalc.numberPressed(12);
        hexcalc.minus();
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(10);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("22", gui.getHexadecimal());

        // F9 - D5
        hexcalc.numberPressed(15);
        hexcalc.numberPressed(9);
        hexcalc.minus();
        hexcalc.numberPressed(13);
        hexcalc.numberPressed(5);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("24", gui.getHexadecimal());

        // C1D - A5
        hexcalc.numberPressed(12);
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(13);
        hexcalc.minus();
```

```
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(5);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("B78", gui.getHexadecimal());
    }

    @Test
    public void T03_03_TestHexMultiplication() {

        HexCalcEngine hexcalc = new HexCalcEngine();
        HexUserInterface gui = new HexUserInterface(hexcalc);
        hexcalc.setHexMode(true);

        // A*B
        hexcalc.numberPressed(10);
        hexcalc.multiply();
        hexcalc.numberPressed(11);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("6E", gui.getHexadecimal());

        // AC*BD
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(12);
        hexcalc.multiply();
        hexcalc.numberPressed(11);
        hexcalc.numberPressed(13);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("7EFC", gui.getHexadecimal());

        // AAA*BBB
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(10);
        hexcalc.multiply();
        hexcalc.numberPressed(11);
        hexcalc.numberPressed(11);
        hexcalc.numberPressed(11);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("7D182E", gui.getHexadecimal());
    }

    @Test
    public void T03_04_TestHexDivision() {

        HexCalcEngine hexcalc = new HexCalcEngine();
        HexUserInterface gui = new HexUserInterface(hexcalc);
        hexcalc.setHexMode(true);

        // C/4
        hexcalc.numberPressed(12);
        hexcalc.divide();
        hexcalc.numberPressed(4);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("3", gui.getHexadecimal());
```

```
        // FF/5
        hexcalc.numberPressed(15);
        hexcalc.numberPressed(15);
        hexcalc.divide();
        hexcalc.numberPressed(5);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("33", gui.getHexadecimal());
    }

    @Test
    public void T04_01_TestNegativeResults() {
        HexCalcEngine hexcalc = new HexCalcEngine();
        HexUserInterface gui = new HexUserInterface(hexcalc);

        // Decimal mode 10 - 15
        hexcalc.setHexMode(false);
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(0);
        hexcalc.minus();
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(5);
        hexcalc.equals();
        assertEquals(-5, hexcalc.getDisplayValue());

        // Hexidecimal mode A1 - A2
        hexcalc.setHexMode(true);
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(1);
        hexcalc.minus();
        hexcalc.numberPressed(10);
        hexcalc.numberPressed(2);
        hexcalc.equals();
        gui.redisplay();
        assertEquals("FFFFFFFF", gui.getHexadecimal());
    }

    @Test
    public void T04_02_StartingNumberNegative() {
        HexCalcEngine hexcalc = new HexCalcEngine();

        // -5 + 10
        hexcalc.minus();
        hexcalc.numberPressed(5);
        hexcalc.plus();
        hexcalc.numberPressed(1);
        hexcalc.numberPressed(0);
        hexcalc.equals();
        assertEquals(5, hexcalc.getDisplayValue());
    }

}
```

Pre-Lab - Timo Schmidt

P1 ***You all know what a calculator is! Here's some code: calculator-full-solution. 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.***

This calculator uses two booleans as fields in its CalcEngine class: buildingDisplayValue (to indicate whether the next digit will be a new number or is part of the current one) and haveLeftOperand that shows if the operator was already entered.

- numberPressed() Number gets pressed, boolean buildingDisplayValue becomes true
  if there's another number pressed the display value gets multiplied by 10
  and the new number gets added to it (e.g. 2 and 2 -> 2 * 10 = 20 + 2 = 22)
- plus/minus() displayValue gets handed to leftOperand (leftOperand becomes displayValue)
  buildingDisplayValue becomes false and the operator (+/-) gets set in
  previousOperator
- numberPressed() Second operand gets typed in (just like the first one) —> displayValuie
- equals leftOperand & operator (+/-) & displayValue gets calculated
  if both booleans are true + lastOperator != ?, then the calculateResult()
method
  gets called which does the calculating with switch cases

P2 ***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?***

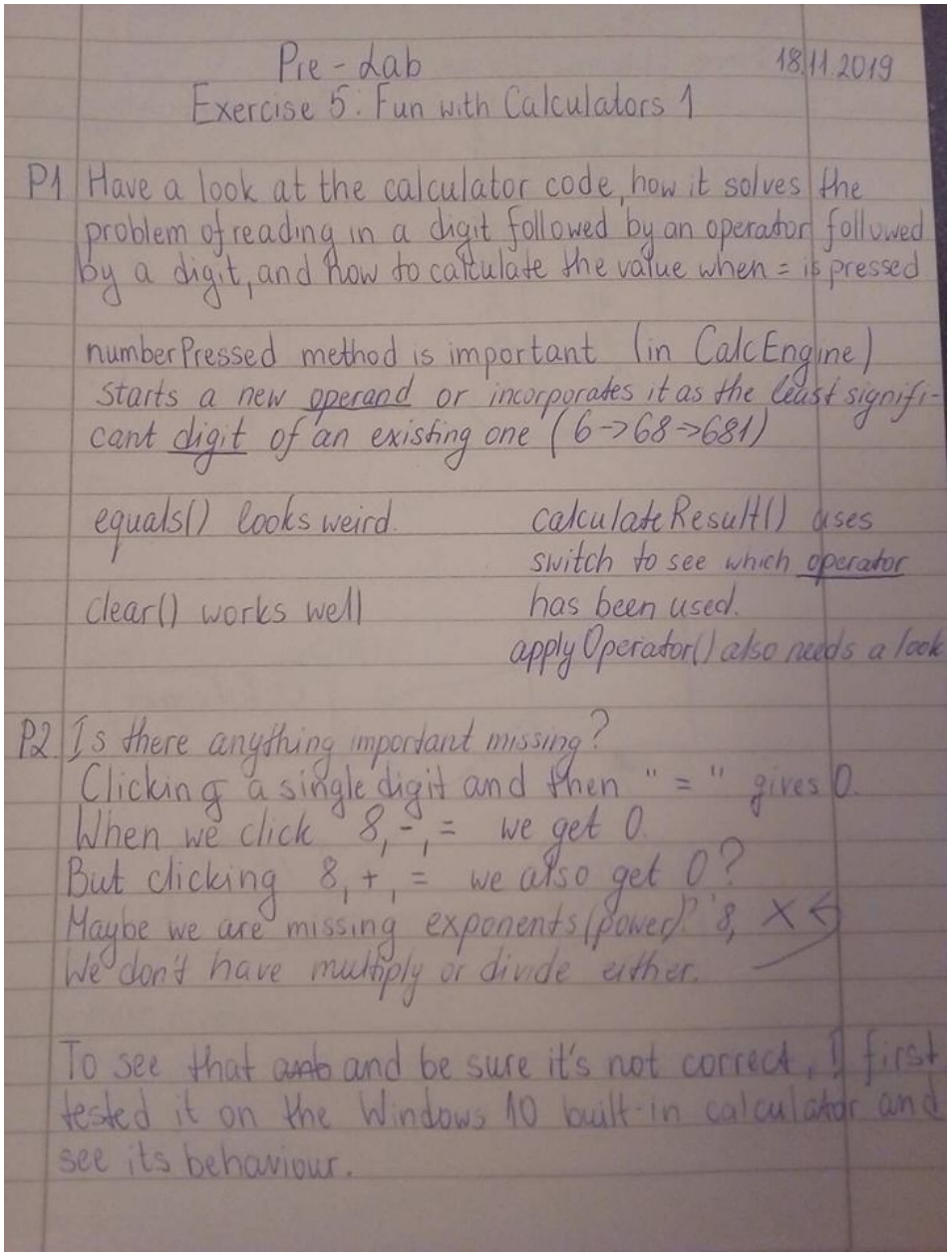
- You can only add and subtract numbers in this calculator. No multiplication or division possible
- the clear method is wrong, it does set the operator to '?' instead of ' '.
- you can't change the operator once executed (like on a real calculator)
- the equals() method is only checking if the operand is not ? instead of + and -
  and it calls another method (instead of doing it straight away) + it bizarrely changes the operator to ? afterwards
- no negative numbers possible

P3 ***What are hexadecimal numbers, by the way?***

In the hexadecimal system numbers are represented using a base of 16. It uses 16 symbols, 0 to 9 (representing the values from 0 to 9) and A to F (representing the values from 10 to 15).

Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Pre-Lab - Pavel Tsvyatkov



Pre-Lab 18.11.2019
Exercise 5: Fun with Calculators 1

P1 Have a look at the calculator code, 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

numberPressed method is important (in CalcEngine)
Starts a new operand or incorporates it as the least significant digit of an existing one (6->68->681)

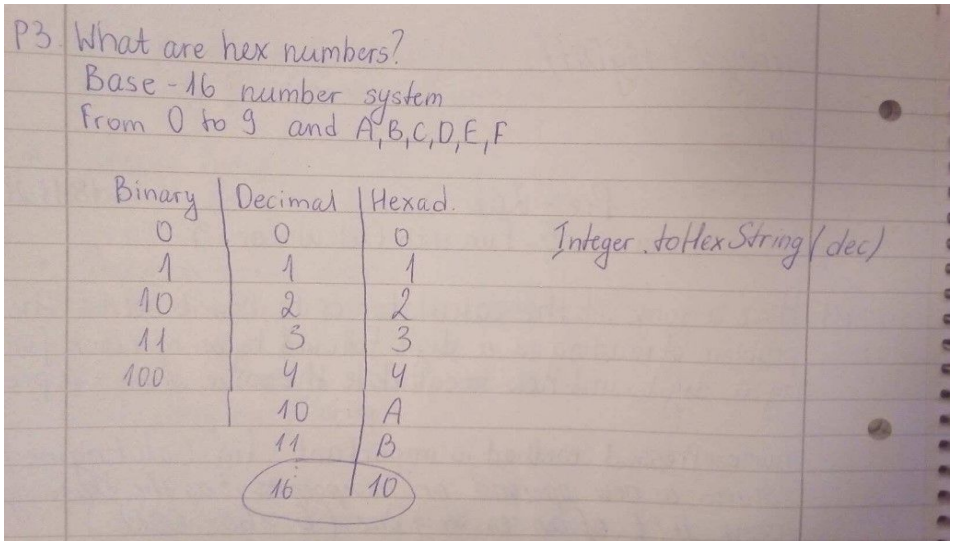
equals() looks weird.

clear() works well

calculateResult() uses switch to see which operator has been used.
applyOperator() also needs a look

P2 Is there anything important missing?
Clicking a single digit and then "=" gives 0.
When we click 8, -, = we get 0.
But clicking 8, +, = we also get 0?
Maybe we are missing exponents (power) ^ 8, ×
We don't have multiply or divide either.

To see that and be sure it's not correct, I first tested it on the Windows 10 built-in calculator and see its behaviour.



P3 What are hex numbers?
Base-16 number system
From 0 to 9 and A, B, C, D, E, F

Binary	Decimal	Hexad.
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
	10	A
	11	B
	16	10

Integer.toHexString(dec)