Lab Report 4 – 08.11.2011 – Sebastian Denzer (533251) – Andy Lindemann (533200)

1.
First we made Calculator.java the main class, having to change the CalcEngine and UserInterface fields to static. Later on, we also recognized that the program did not close correct (javaw.exe still running), so we also set the JFrames' setDefaultCloseOperation().

As we have already done some MVC work now, we know how to use this class configuration: The UserInterface.java gets a reference to the CalcEngine via the constructor, simply pushing forward each command that is entered by the JFrame components from the UI to the CalcEngine.

So to implement new operators, one can simply add the methods to the CalcEngine, e.g. divide() and multiply(). The applyOperator() method (after some checking, and after the next value is entered) starts the calculateResult(). Eventually the final calculation is done here: we added the characters for division and multiplication, and did the calculation.

Now to the UI: we set the GridLayout in the buttonPanel to size (4, 6), and added the buttons we needed for now (multiply, divide). When actionPerformed is called for one of them, the according methods get called.

The functionality for now works great, but when testing it we ackknowledged that there was no integer range check: at some point when entering a number, the number did crazy things like getting negative, or being not the right sequence of numbers anymore. After some trial and error, we found one good method for preventing this: we do the calculation as a long value in the first, and check if it's outside the Integer. MAX_VALUE/MIN_VALUE. We check this on calculateResult() and numberPressed(), and do an System. out.println() (like the keySequenceError()) if it's out of range. If so, the values are not clear()'ed, as we both definitly hate angry customers (angry because they have to enter the numbers again).

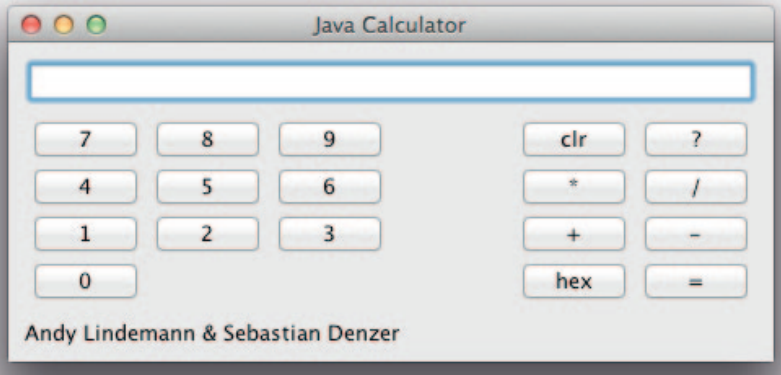
Now we wanted to implement the negative numbers entering: in the first version, it is only possible to get negative numbers by subtraction. We used a boolean isNegative: when the minus button is pushed (the minus() method is called), isNegative is set to true only if buildingDisplayValue is false. When the numberPressed() method determines that a new number building is started, it applies a minus to the value if isNegative is true (of course, isNegative is set to false afterwards).

2.
We first wanted to inherit from the CalcEngine a HexadecimalCalcEngine. But that would have meant to use the CalcEngine as a polymorphic variable from out the UserInterface. Thinking about the problem and looking through the CalcEngine code, we finally realized that only one method has really to manage with decimal system problems: only when entering a number, the algorithm needs a divisor (in the decimal case, 10).

So we added an int divisor, set up two methods to setToDecimal() and setToHexadecimal(), and included the divisor in the numberPressed() calculation.

We added a new getDisplayValueString() method where the CalcEngine figures out what Integer method to use (on base of the divisor): toHexString(), or the normal toString().

In fact, we added the toBinaryString() and toOctalString(), too, but didn't include any octal or binary functionality. But in principle, we only need the UI's for them, and switching between them, to finish excercise 5.



3
Testing hexadecimal calculation through the UserInterface.java, we first made a copy of it and changed the copy to hexadecimal layout. When the actionPerformed() method is called with alphanumerical commands from 0-9 and A-F, we use the Integer.parseInt() method, also passing the divisor (CalcEngine.getDivisor()): it parses for an int in the decimal, or for our use, in the hexadecimal system. The value is passed to CalcEgines' numberPressed() method.

Not to mention: we changed the 'C' button to 'clr' because it collides with the hexadecimal 'C'.

Now we extended the makeFrame() method in UserInterface.java: it asks for wether the CalcEngine is decimal or hexadecimal, and sets up the GridLayout as needed. Of course, the frames' contentPane is removeAll()'ed before.

Here we found a new problem: the hexadecimal system has no negative numbers, as it's is only a representation of the integer value. So we exluded negative number creation for hex numbers in the CalcEngine, and let the CalcEngine clear() when switching to hex and having any negative numbers in memory.

The last thing to do: both GridLayouts got a button for its vice versa partner, 'hex' and 'dec', switching between both CalcEngines' states and rebuilding the JFrame via makeFrame().

4
We wanted to test the basic functionality (+-/*), so we did a who0le lot of tests on that. Plus, cases where one or both operands are negative. Calculation arroung MAX_INT and MIN_INT was also important; as we don't throw exceptions but do System.out "errors", we have to check if, after an illegal operation, the calculator still has the same value as before.

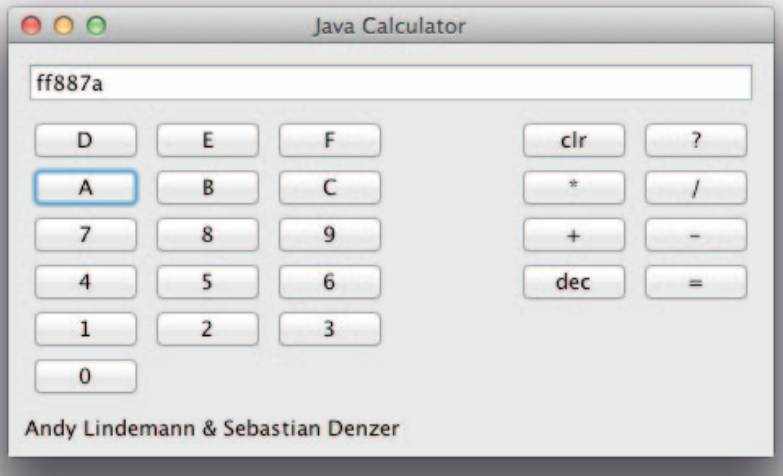
The hexadecimal calculation itself doesn't do anything else than the decimal one, because internally the values remain int values; so here we only did tests on entering hex values where the only additional functionality is found.

When figuring out test cases, we realized that CalcEngine.java doesn't check for zero division problems. We fixed that, of course, and print out an error.

Statistics

Time
~ 8 hrs.

Lines of code
CalcEngine.java: 178
CalcEngineTest.java: 284
Calculator.java: 13
UserInterface.java: 163



Andy did most of the work on the UserInterface.java, Sebastian did so on the CalcEngine.java. We both did the test cases and figured out the problems (or better: their solutions) in teamwork, pushing ideas back and forth.

```java
package calculator_full_solution;

/**
 * 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;

    private boolean isNegative;
    // The most recent operator that was entered.
    private char lastOperator;

    // The current value (to be) shown in the display.
    private int displayValue;
    // The value of an existing left operand.
    private int leftOperand;

    private int divisor;

    public CalcEngine() {
        this.divisor = 10;
        clear();
    }

    public void setToDecimal() {
        this.divisor = 10;
    }

    public void setToHexadecimal() {
        // No negative hexadecimal allowed.
        if (leftOperand<0 || displayValue < 0 || isNegative) {
            clear();
        }

        this.divisor = 16;
    }

    public int getDivisor() {
        return divisor;
    }

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

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

        if (this.divisor == 16)
            return Integer.toHexString(displayValue);
        if (this.divisor == 8)
            return Integer.toOctalString(displayValue);
        if (this.divisor == 2)
            return Integer.toBinaryString(displayValue);

        return Integer.toString(displayValue);
    }

    public boolean isHexadecimal() {
        return (this.divisor == 16);
    }

    public boolean isDecimal() {
        return (this.divisor == 10);
    }
```

```
 83
 84      /**
 85       * A number button was pressed. Either start a new operand, or incorporate this number as the least sigr
 86       *
 87       * @param number
 88       *            The number pressed on the calculator.
 89       */
 90      public void numberPressed(int number) {
 91          if (buildingDisplayValue) {
 92              // Calculate long result for checking int range.
 93              long result;
 94              // Negative number building.
 95              if (displayValue < 0) {
 96                  result = (long) displayValue * divisor - number;
 97              }
 98              // Positive number building.
 99              else {
100                  result = (long) displayValue * divisor + number;
101              }
102              // Check if long calculation is inner int range.
103              if (result <= Integer.MAX_VALUE && result >= Integer.MIN_VALUE) {
104                  displayValue = (int) result;
105              }
106              else {
107                  outOfRangeError();
108              }
109          }
110          else {
111              // Start building a new number.
112              // Negative Numbers only for decimal system.
113              if (isNegative && divisor==10) {
114                  displayValue = -number;
115              }
116              else {
117                  displayValue = number;
118              }
119              isNegative = false;
120              buildingDisplayValue = true;
121          }
122
123      }
124
125      /**
126       * The 'plus' button was pressed.
127       */
128      public void plus() {
129          applyOperator('+');
130      }
131
132      /**
133       * The 'minus' button was pressed.
134       */
135      public void minus() {
136          // If already building a value, apply minus.
137          if (buildingDisplayValue) {
138              applyOperator('-');
139          }
140          // Else set to negative number.
141          else {
142              this.isNegative = true;
143          }
144      }
145
146      /***
147       * The 'multiply' button was pressed.
148       */
149      public void multiply() {
150          applyOperator('*');
151      }
152
153      /***
154       * The 'divide' button was pressed.
155       */
156      public void divide() {
157          applyOperator('/');
158      }
159
160      /***
161       * The 'power' button was pressed.
162       */
163      public void power() {
164          applyOperator('^');
```

```
165     }
166
167     /**
168      * The '=' button was pressed.
169      */
170     public void equals() {
171         // This should completes the building of a second operand,
172         // so ensure that we really have a left operand, an operator
173         // and a right operand.
174         if (haveLeftOperand && lastOperator != '?' && buildingDisplayValue) {
175             calculateResult();
176             lastOperator = '?';
177             buildingDisplayValue = false;
178         }
179         else {
180             keySequenceError();
181         }
182     }
183
184     /**
185      * The 'C' (clear) button was pressed. Reset everything to a starting state.
186      */
187     public void clear() {
188         lastOperator = '?';
189         haveLeftOperand = false;
190         buildingDisplayValue = false;
191         displayValue = 0;
192         isNegative = false;
193     }
194
195     /**
196      * @return The title of this calculation engine.
197      */
198     public String getTitle() {
199         return "Java Calculator";
200     }
201
202     /**
203      * @return The author of this engine.
204      */
205     public String getAuthor() {
206         return "Andy Lindemann & Sebastian Denzer";
207     }
208
209     /**
210      * @return The version number of this engine.
211      */
212     public String getVersion() {
213         return "Version 1.1 alpha";
214     }
215
216     /**
217      * Combine leftOperand, lastOperator, and the current display value. The result becomes both the leftOpe
218      */
219     private void calculateResult() {
220         // Use long calculation to check if is out of int range.
221         long result;
222
223         switch (lastOperator) {
224         case '+':
225             result = (long) leftOperand + displayValue;
226             break;
227         case '-':
228             result = (long) leftOperand - displayValue;
229             break;
230         case '*':
231             result = (long) leftOperand * displayValue;
232             break;
233         case '/':
234             if (displayValue == 0) {
235                 divisionByZeroError();
236                 clear();
237                 return;
238             }
239             result = (long) leftOperand / displayValue;
240             break;
241         case '^':
242             result = (long) Math.pow(leftOperand, displayValue);
243             break;
244         default:
245             keySequenceError();
246             return;
```

```
247         }
248
249         // If long result is in int range, do accept calculation.
250         if (result <= Integer.MAX_VALUE && result >= Integer.MIN_VALUE) {
251             displayValue = (int) result;
252             haveLeftOperand = true;
253             leftOperand = displayValue;
254         }
255         // Else out-of-range error.
256         else {
257             outOfRangeError();
258         }
259
260     }
261
262     /**
263      * Apply an operator.
264      *
265      * @param operator
266      *            The operator to apply.
267      */
268     private void applyOperator(char operator) {
269         // If we are not in the process of building a new operand
270         // then it is an error, unless we have just calculated a
271         // result using '='.
272         if (!buildingDisplayValue && !(haveLeftOperand && lastOperator == '?')) {
273             keySequenceError();
274             return;
275         }
276
277         if (lastOperator != '?') {
278             // First apply the previous operator.
279             calculateResult();
280         }
281         else {
282             // The displayValue now becomes the left operand of this
283             // new operator.
284             haveLeftOperand = true;
285             leftOperand = displayValue;
286         }
287         lastOperator = operator;
288         buildingDisplayValue = false;
289     }
290
291     /**
292      * Report an error in the sequence of keys that was pressed.
293      */
294     private void keySequenceError() {
295         System.out.println("A key sequence error has occurred.");
296         // Reset everything.
297         clear();
298     }
299
300     /**
301      * Report an out of int range error.
302      */
303     private void outOfRangeError() {
304         System.out.println("The value was out of int range.");
305     }
306
307     private void divisionByZeroError() {
308         System.out.println("Division by Zero.");
309     }
310
311 }
312
```

```java
package calculator_full_solution;

/**
 * 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 static CalcEngine engine;
    private static UserInterface gui;

    public static void main(String[] args) {

        System.out.println(Integer.MAX_VALUE);
        System.out.println(Integer.MIN_VALUE);
        engine = new CalcEngine();
        gui = new UserInterface(engine);
        gui.setVisible(true);
    }
}

```

```
 1  package calculator_full_solution;
 2
 3  import java.awt.*;
 4  import java.awt.event.*;
 5  import javax.swing.*;
 6  import javax.swing.border.*;
 7
 8  /**
 9   * A graphical user interface for the calculator. No calculation is being done here. This class is responsil
10   * then refers to the "CalcEngine" to do all the real work.
11   *
12   * @author David J. Barnes and Michael Kolling
13   * @version 2008.03.30
14   */
15  public class UserInterface implements ActionListener {
16      private CalcEngine calc;
17      private boolean showingAuthor;
18
19      private JFrame frame;
20      private JTextField display;
21      private JLabel status;
22
23      /**
24       * Create a user interface.
25       *
26       * @param engine
27       *             The calculator engine.
28       */
29      public UserInterface(CalcEngine engine) {
30          calc = engine;
31          showingAuthor = true;
32          calc.setToDecimal();
33          makeFrame();
34          frame.setVisible(true);
35      }
36
37      /**
38       * Set the visibility of the interface.
39       *
40       * @param visible
41       *             true if the interface is to be made visible, false otherwise.
42       */
43      public void setVisible(boolean visible) {
44          frame.setVisible(visible);
45      }
46
47      /**
48       * Make the frame for the user interface.
49       */
50      private void makeFrame() {
51          if (frame != null) {
52              frame.getContentPane().removeAll();
53          }
54          else {
55              frame = new JFrame(calc.getTitle());
56              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
57          }
58
59          JPanel contentPane = (JPanel) frame.getContentPane();
60          contentPane.setLayout(new BorderLayout(8, 8));
61          contentPane.setBorder(new EmptyBorder(10, 10, 10, 10));
62
63          display = new JTextField();
64          contentPane.add(display, BorderLayout.NORTH);
65
66          JPanel buttonPanel;
67
68          if (calc.isHexadecimal()) {
69
70              buttonPanel = new JPanel(new GridLayout(6, 6));
71
72              addButton(buttonPanel, "D");
73              addButton(buttonPanel, "E");
74              addButton(buttonPanel, "F");
75              buttonPanel.add(new JLabel(" "));
76              addButton(buttonPanel, "clr");
77              addButton(buttonPanel, "?");
78
79              addButton(buttonPanel, "A");
80              addButton(buttonPanel, "B");
81              addButton(buttonPanel, "C");
82              buttonPanel.add(new JLabel(" "));
```

```
 83              addButton(buttonPanel, "*");
 84              addButton(buttonPanel, "/");
 85
 86              addButton(buttonPanel, "7");
 87              addButton(buttonPanel, "8");
 88              addButton(buttonPanel, "9");
 89              buttonPanel.add(new JLabel(" "));
 90              addButton(buttonPanel, "+");
 91              addButton(buttonPanel, "-");
 92
 93              addButton(buttonPanel, "4");
 94              addButton(buttonPanel, "5");
 95              addButton(buttonPanel, "6");
 96              buttonPanel.add(new JLabel(" "));
 97              addButton(buttonPanel, "dec");
 98              addButton(buttonPanel, "=");
 99
100              addButton(buttonPanel, "1");
101              addButton(buttonPanel, "2");
102              addButton(buttonPanel, "3");
103              buttonPanel.add(new JLabel(" "));
104              buttonPanel.add(new JLabel(" "));
105              buttonPanel.add(new JLabel(" "));
106
107              addButton(buttonPanel, "0");
108              buttonPanel.add(new JLabel(" "));
109              buttonPanel.add(new JLabel(" "));
110              buttonPanel.add(new JLabel(" "));
111              buttonPanel.add(new JLabel(" "));
112          }
113          else {
114              buttonPanel = new JPanel(new GridLayout(4, 6));
115              addButton(buttonPanel, "7");
116              addButton(buttonPanel, "8");
117              addButton(buttonPanel, "9");
118              buttonPanel.add(new JLabel(" "));
119              addButton(buttonPanel, "clr");
120              addButton(buttonPanel, "?");
121
122              addButton(buttonPanel, "4");
123              addButton(buttonPanel, "5");
124              addButton(buttonPanel, "6");
125              buttonPanel.add(new JLabel(" "));
126              addButton(buttonPanel, "*");
127              addButton(buttonPanel, "/");
128
129              addButton(buttonPanel, "1");
130              addButton(buttonPanel, "2");
131              addButton(buttonPanel, "3");
132              buttonPanel.add(new JLabel(" "));
133              addButton(buttonPanel, "+");
134              addButton(buttonPanel, "-");
135
136              addButton(buttonPanel, "0");
137              buttonPanel.add(new JLabel(" "));
138              buttonPanel.add(new JLabel(" "));
139              buttonPanel.add(new JLabel(" "));
140              addButton(buttonPanel, "hex");
141              addButton(buttonPanel, "=");
142
143          }
144
145          contentPane.add(buttonPanel, BorderLayout.CENTER);
146
147          status = new JLabel(calc.getAuthor());
148          contentPane.add(status, BorderLayout.SOUTH);
149
150          frame.pack();
151      }
152
153      /**
154       * Add a button to the button panel.
155       *
156       * @param panel
157       *            The panel to receive the button.
158       * @param buttonText
159       *            The text for the button.
160       */
161      private void addButton(Container panel, String buttonText) {
162          JButton button = new JButton(buttonText);
163          button.addActionListener(this);
164          panel.add(button);
```

```
165     }
166
167     /**
168      * An interface action has been performed. Find out what it was and handle it.
169      *
170      * @param event
171      *            The event that has occured.
172      */
173     public void actionPerformed(ActionEvent event) {
174         String command = event.getActionCommand();
175
176         if (command.equals("0") || command.equals("1") || command.equals("2") || command.equals("3") || comm
177                 || command.equals("6") || command.equals("7") || command.equals("8") || command.equals("9")
178                 || command.equals("C") || command.equals("D") || command.equals("E") || command.equals("F"))
179             int number = Integer.parseInt(command, calc.getDivisor());
180             calc.numberPressed(number);
181         }
182         else if (command.equals("+")) {
183             calc.plus();
184         }
185         else if (command.equals("-")) {
186             calc.minus();
187         }
188         else if (command.equals("=")) {
189             calc.equals();
190         }
191         else if (command.equals("*")) {
192             calc.multiply();
193         }
194         else if (command.equals("/")) {
195             calc.divide();
196         }
197         else if (command.equals("^")) {
198             calc.power();
199         }
200         else if (command.equals("clr")) {
201             calc.clear();
202         }
203         else if (command.equals("?")) {
204             showInfo();
205         }
206         else if (command.equals("hex")) {
207             calc.setToHexadecimal();
208             makeFrame();
209         }
210         else if (command.equals("dec")) {
211             calc.setToDecimal();
212             makeFrame();
213         }
214         // else unknown command.
215
216         redisplay();
217     }
218
219     /**
220      * Update the interface display to show the current value of the calculator.
221      */
222     private void redisplay() {
223
224         display.setText(calc.getDisplayValueString());
225
226     }
227
228     /**
229      * Toggle the info display in the calculator's status area between the author and version information.
230      */
231     private void showInfo() {
232         if (showingAuthor)
233             status.setText(calc.getVersion());
234         else
235             status.setText(calc.getAuthor());
236
237         showingAuthor = !showingAuthor;
238     }
239 }
240
```

```java
package calculator_full_solution;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class CalcEngineTest {
    private CalcEngine calc;
    @Before
    public void setUp() throws Exception {
        calc = new CalcEngine();
    }

    @Test
    public void testPlus() {
        calc.setToDecimal();
        calc.numberPressed(2);
        calc.numberPressed(5);
        calc.plus();
        calc.numberPressed(7);
        calc.equals();
        assertEquals(32, calc.getDisplayValue());
        assertEquals("32", calc.getDisplayValueString());
        calc.setToHexadecimal();
        assertEquals("20", calc.getDisplayValueString());

        calc.setToDecimal();
        calc.numberPressed(2);
        calc.numberPressed(5);
        calc.plus();
        calc.numberPressed(0);
        calc.equals();
        assertEquals(25, calc.getDisplayValue());
        assertEquals("25", calc.getDisplayValueString());
        calc.setToHexadecimal();
        assertEquals("19", calc.getDisplayValueString());

        calc.setToDecimal();
        calc.minus();
        calc.numberPressed(7);
        calc.numberPressed(5);
        calc.numberPressed(1);
        calc.plus();
        calc.minus();
        calc.numberPressed(7);
        calc.equals();
        assertEquals(-758, calc.getDisplayValue());
        assertEquals("-758", calc.getDisplayValueString());
        calc.setToHexadecimal();
        assertEquals("0", calc.getDisplayValueString());

        calc.setToDecimal();
        calc.minus();
        calc.numberPressed(1);
        calc.numberPressed(7);
        calc.plus();
        calc.numberPressed(1);
        calc.numberPressed(9);
        calc.equals();
        assertEquals(2, calc.getDisplayValue());
        assertEquals("2", calc.getDisplayValueString());
        calc.setToHexadecimal();
        assertEquals("2", calc.getDisplayValueString());

        calc.setToDecimal();
        calc.numberPressed(2);
        calc.numberPressed(1);
        calc.numberPressed(4);
        calc.numberPressed(7);
        calc.numberPressed(4);
        calc.numberPressed(8);
        calc.numberPressed(3);
        calc.numberPressed(6);
        calc.numberPressed(4);
        calc.numberPressed(7);
        calc.plus();
        calc.numberPressed(1);
        calc.equals();
        assertEquals(1, calc.getDisplayValue());
    }

```

```
 83      @Test
 84      public void testMinus() {
 85
 86          calc.setToDecimal();
 87          calc.numberPressed(2);
 88          calc.numberPressed(5);
 89          calc.minus();
 90          calc.numberPressed(7);
 91          calc.equals();
 92          assertEquals(18, calc.getDisplayValue());
 93          assertEquals("18", calc.getDisplayValueString());
 94          calc.setToHexadecimal();
 95          assertEquals("12", calc.getDisplayValueString());
 96
 97          calc.setToDecimal();
 98          calc.numberPressed(2);
 99          calc.numberPressed(5);
100          calc.minus();
101          calc.numberPressed(0);
102          calc.equals();
103          assertEquals(25, calc.getDisplayValue());
104          assertEquals("25", calc.getDisplayValueString());
105          calc.setToHexadecimal();
106          assertEquals("19", calc.getDisplayValueString());
107
108          calc.setToDecimal();
109          calc.minus();
110          calc.numberPressed(7);
111          calc.numberPressed(5);
112          calc.numberPressed(1);
113          calc.minus();
114          calc.minus();
115          calc.numberPressed(7);
116          calc.equals();
117          assertEquals(-744, calc.getDisplayValue());
118          assertEquals("-744", calc.getDisplayValueString());
119          calc.setToHexadecimal();
120          assertEquals("0", calc.getDisplayValueString());
121
122          calc.setToDecimal();
123          calc.minus();
124          calc.numberPressed(1);
125          calc.numberPressed(7);
126          calc.minus();
127          calc.numberPressed(1);
128          calc.numberPressed(9);
129          calc.equals();
130          assertEquals(-36, calc.getDisplayValue());
131          assertEquals("-36", calc.getDisplayValueString());
132          calc.setToHexadecimal();
133          assertEquals("0", calc.getDisplayValueString());
134
135          calc.setToDecimal();
136          calc.minus();
137          calc.numberPressed(2);
138          calc.numberPressed(1);
139          calc.numberPressed(4);
140          calc.numberPressed(7);
141          calc.numberPressed(4);
142          calc.numberPressed(8);
143          calc.numberPressed(3);
144          calc.numberPressed(6);
145          calc.numberPressed(4);
146          calc.numberPressed(8);
147          calc.minus();
148          calc.numberPressed(1);
149          calc.equals();
150          assertEquals(1, calc.getDisplayValue());
151      }
152
153      @Test
154      public void testMultiply() {
155          calc.setToDecimal();calc.numberPressed(2);
156          calc.numberPressed(5);
157          calc.multiply();
158          calc.numberPressed(7);
159          calc.equals();
160          assertEquals(175, calc.getDisplayValue());
161          assertEquals("175", calc.getDisplayValueString());
162          calc.setToHexadecimal();
163          assertEquals("af", calc.getDisplayValueString());
164
```

```
165         calc.setToDecimal();
166         calc.numberPressed(2);
167         calc.numberPressed(5);
168         calc.multiply();
169         calc.numberPressed(0);
170         calc.equals();
171         assertEquals(0, calc.getDisplayValue());
172         assertEquals("0", calc.getDisplayValueString());
173         calc.setToHexadecimal();
174         assertEquals("0", calc.getDisplayValueString());
175
176         calc.setToDecimal();
177         calc.minus();
178         calc.numberPressed(7);
179         calc.numberPressed(5);
180         calc.numberPressed(1);
181         calc.multiply();
182         calc.minus();
183         calc.numberPressed(7);
184         calc.equals();
185         assertEquals(5257, calc.getDisplayValue());
186         assertEquals("5257", calc.getDisplayValueString());
187         calc.setToHexadecimal();
188         assertEquals("1489", calc.getDisplayValueString());
189
190         calc.setToDecimal();
191         calc.minus();
192         calc.numberPressed(1);
193         calc.numberPressed(7);
194         calc.multiply();
195         calc.numberPressed(1);
196         calc.numberPressed(9);
197         calc.equals();
198         assertEquals(-323, calc.getDisplayValue());
199         assertEquals("-323", calc.getDisplayValueString());
200         calc.setToHexadecimal();
201         assertEquals("0", calc.getDisplayValueString());
202
203         calc.setToDecimal();
204         calc.numberPressed(2);
205         calc.numberPressed(1);
206         calc.numberPressed(4);
207         calc.numberPressed(7);
208         calc.numberPressed(4);
209         calc.numberPressed(8);
210         calc.numberPressed(3);
211         calc.numberPressed(6);
212         calc.numberPressed(4);
213         calc.numberPressed(7);
214         calc.plus();
215         calc.numberPressed(1);
216         calc.equals();
217         assertEquals(1, calc.getDisplayValue());
218
219     }
220
221     @Test
222     public void testDivide() {
223         calc.setToDecimal();
224         calc.numberPressed(2);
225         calc.numberPressed(5);
226         calc.divide();
227         calc.numberPressed(7);
228         calc.equals();
229         assertEquals(3, calc.getDisplayValue());
230         assertEquals("3", calc.getDisplayValueString());
231         calc.setToHexadecimal();
232         assertEquals("3", calc.getDisplayValueString());
233
234         calc.setToDecimal();
235         calc.numberPressed(2);
236         calc.numberPressed(5);
237         calc.divide();
238         calc.numberPressed(0);
239         calc.equals();
240         assertEquals(0, calc.getDisplayValue());
241         assertEquals("0", calc.getDisplayValueString());
242         calc.setToHexadecimal();
243         assertEquals("0", calc.getDisplayValueString());
244
245         calc.setToDecimal();
246         calc.minus();
```

```
247         calc.numberPressed(7);
248         calc.numberPressed(5);
249         calc.numberPressed(1);
250         calc.divide();
251         calc.minus();
252         calc.numberPressed(7);
253         calc.equals();
254         assertEquals(107, calc.getDisplayValue());
255         assertEquals("107", calc.getDisplayValueString());
256         calc.setToHexadecimal();
257         assertEquals("6b", calc.getDisplayValueString());
258
259         calc.setToDecimal();
260         calc.minus();
261         calc.numberPressed(1);
262         calc.numberPressed(7);
263         calc.multiply();
264         calc.numberPressed(1);
265         calc.numberPressed(9);
266         calc.equals();
267         assertEquals(-323, calc.getDisplayValue());
268         assertEquals("-323", calc.getDisplayValueString());
269         calc.setToHexadecimal();
270         assertEquals("0", calc.getDisplayValueString());
271 }
272
273     public void testHex () {
274
275         calc.setToHexadecimal();
276
277         calc.numberPressed(15);
278         calc.numberPressed(14);
279         calc.numberPressed(4);
280         calc.numberPressed(7);
281         calc.numberPressed(12);
282
283         assertEquals(1041532, calc.getDisplayValue());
284         assertEquals("fe47c", calc.getDisplayValueString());
285
286
287         calc.numberPressed(15);
288         calc.numberPressed(15);
289         calc.numberPressed(15);
290         calc.numberPressed(15);
291         calc.numberPressed(15);
292         calc.numberPressed(15);
293         calc.numberPressed(15);
294
295         assertEquals(268435455, calc.getDisplayValue());
296         assertEquals("fffffff", calc.getDisplayValueString());
297
298         calc.numberPressed(15);
299         calc.numberPressed(0);
300         calc.numberPressed(10);
301         calc.numberPressed(14);
302         calc.numberPressed(0);
303         calc.numberPressed(13);
304         calc.numberPressed(10);
305
306         assertEquals(15773194, calc.getDisplayValue());
307         assertEquals("f0ae0a", calc.getDisplayValueString());
308
309         calc.numberPressed(10);
310         calc.numberPressed(11);
311         calc.numberPressed(12);
312         calc.numberPressed(13);
313         calc.numberPressed(14);
314         calc.numberPressed(15);
315
316         assertEquals(11259375, calc.getDisplayValue());
317         assertEquals("abcdef", calc.getDisplayValueString());
318     }
319
320 }
321
```