Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

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

Post-Lab Report

Lab 8: Fun with Calculators 3

Group 1

Tuesday, 10.12.2019

Group members:

Muhammad Safarov	570690	muhammad.safarov@student.htw-berlin.de
Pavel Tsvyatkov	559632	pavel.tsvyatkov@student.htw-berlin.de

1. **Make another new copy of the Calculator (don't wreck your previous versions!) before you start. Or you can extend your Calculator to be able to do decimal, hex, and your chosen data type!**

We discussed together which calculator we are going to use, since at this point we had 4 different options to choose from. We decided to use the Calculator on which Pavel and Timo worked in Lab 5. We made another copy of it and before moving on to the exercises, we discussed further how we are going to approach the exercises and what we needed.

To be honest, we first wanted to work on sets. It seemed interesting and we thought that we will be able to deal with them. We had different ideas how we should solve the problem with sets, but we didn't come up with working solutions. We think that the problem was that we jumped straight to the code, trying out different ways to make the program work, instead of taking a step back and thinking about the exercise as a whole and not too much into the small details.

Then we decided to do the **dates calculator** and approached the problem in a different way. We thought about it step by step and explained exactly what we did in the next exercise.

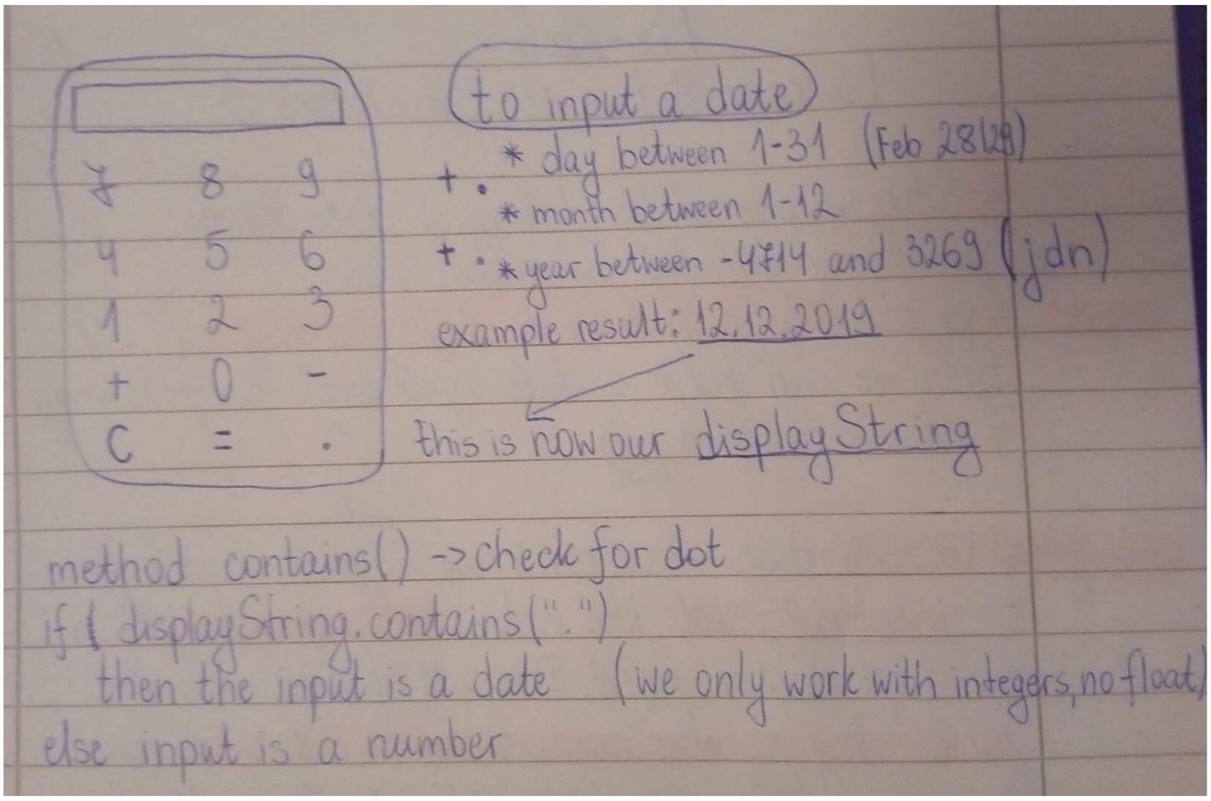
2. **If you are doing the *date calculator*, implement the following functions.**

    1. **input a date**
    2. **get the calculator to display a String for the weekday in the window (hint: you will need a button to push)**
    3. **add a number of days to a date, displaying the new date**
    4. **subtract a number of days from a date, displaying the new date**
    5. **subtract two dates, giving the number of days between the dates**

Before we started working on the exercises, we looked at our prelabs and then discussed some of the ideas that we had. Then we decided to take pen and paper and write down our thoughts to have a general idea of what we are doing. This would help us find out what we are potentially missing or what would need to be changed in this version of the calculator that we chose.

First, we thought about the input. We decided to add a new button to be able to write a dot in our calculator and input the date in the following format dd.mm.yyyy .

We sketched a calculator and a step by step example on the side how we would go about inputting a date. To input the date in that way we would need a displayString, which is similar to how we stored the infix expression in a String while working on Lab 6. When we need to know if our input is a date or a number, we use the contains() method on our displayString.


7 8 9
4 5 6
1 2 3
+ 0 -
C = .

to input a date

+ . * day between 1-31 (Feb 28|29)

* month between 1-12

+ . * year between -4714 and 3269 (jdn)

example result: 12.12.2019

this is now our displayString

method contains() -> check for dot

if (displayString.contains(".")

then the input is a date (we only work with integers, no float)

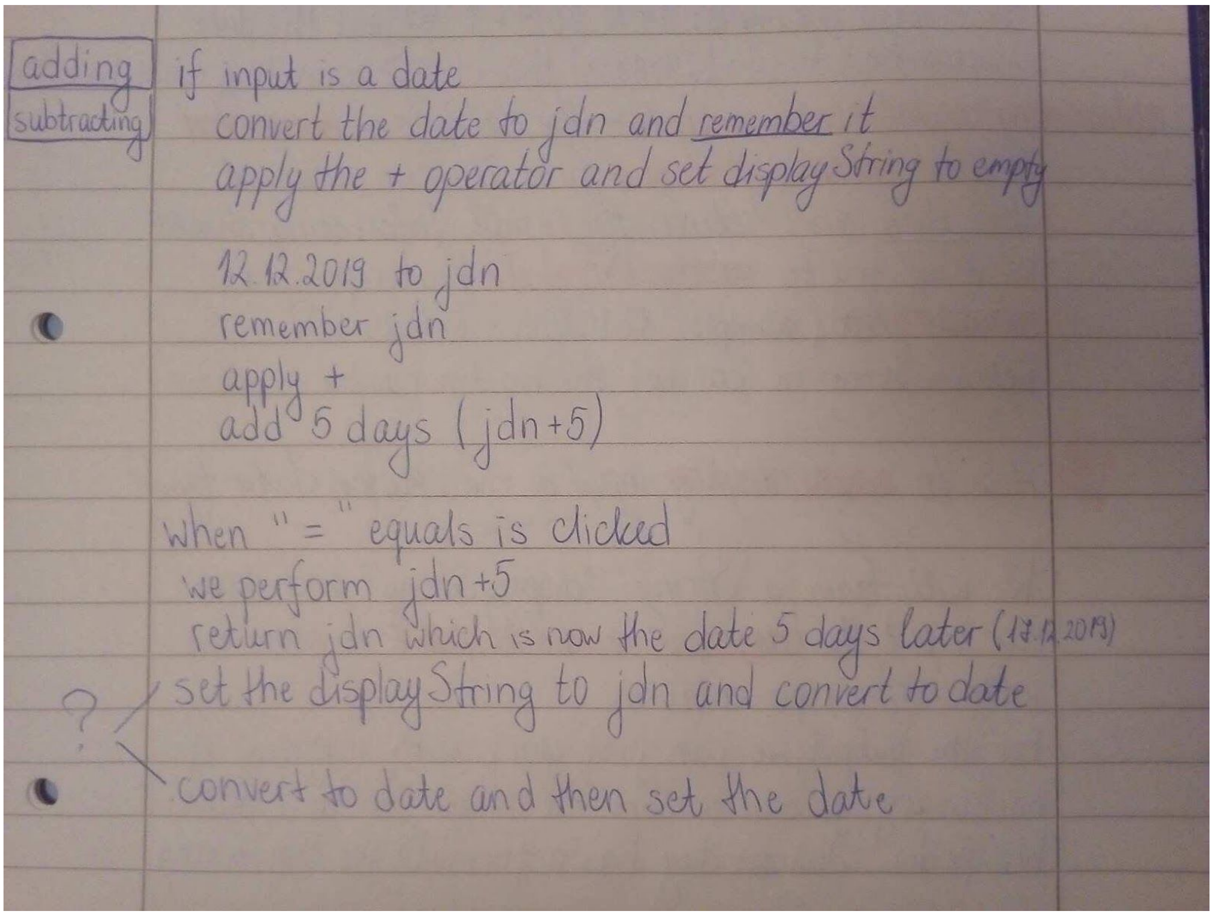
else input is a number

Now we knew we would need to add the displayString field and use the contains method.

Next thing we had to think about was adding and subtracting a number of days to a date.

Now that we know how to determine if the input is a date, we can use the methods in our JulianDate program to convert the date to a **julian date number(jdn)** and then we have to remember it. When " + " is clicked, we have the jdn stored and we can set the displayString to an empty string.

After we click " = " , we can internally calculate **jdn + 5** and return the result. Then we had to think about how we are going to convert the number to a date and display it in the calculator.



adding
subtracting

if input is a date
convert the date to jdn and remember it
apply the + operator and set displayString to empty

12.12.2019 to jdn
remember jdn
apply +
add 5 days (jdn+5)

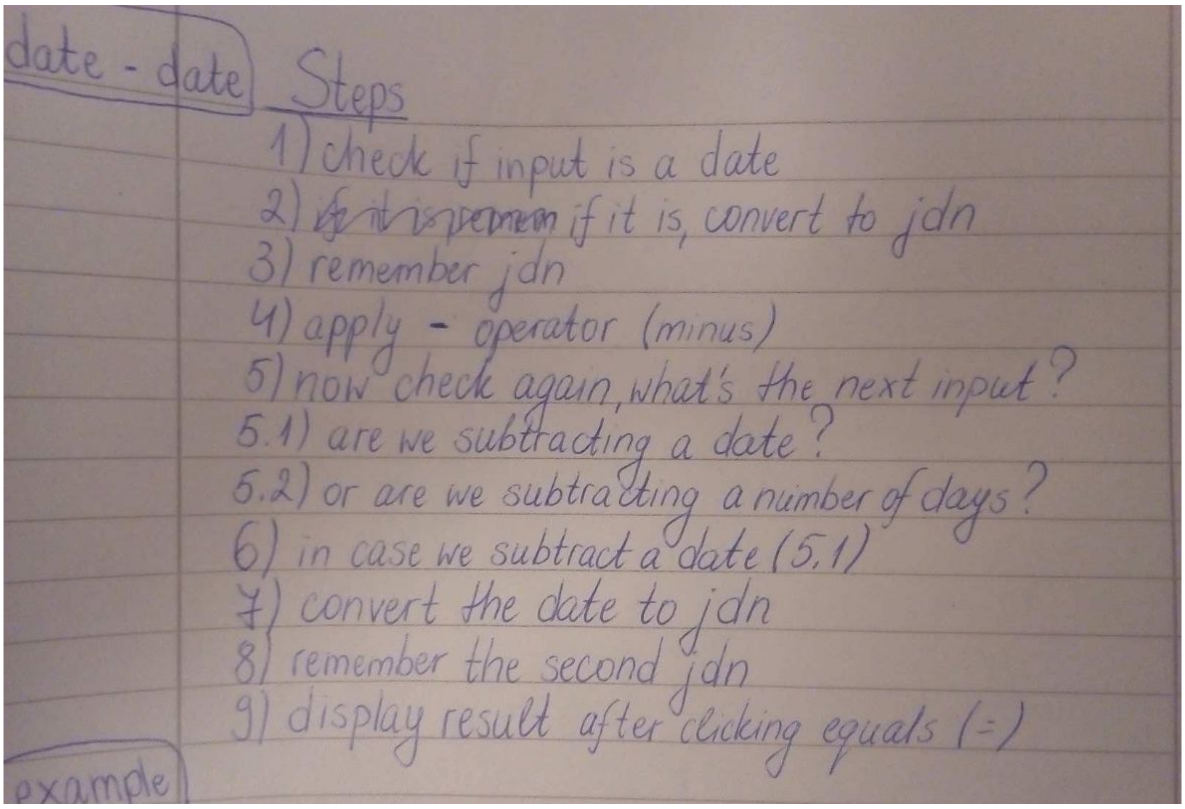
when "=" equals is clicked
we perform jdn+5
return jdn which is now the date 5 days later (17.12.2019)

? set the displayString to jdn and convert to date
convert to date and then set the date

After writing this down, we now know we will need a method to remember the display value internally and that we will have to deal with conversion from julian date number to date and setting the display to the date. The same process should also be working when we subtract a number of days from a date.

To get the weekday we are going to add a new button "getWeekday". We will also need the method from our JulianDate project, when we had to determine the day of the week we were born on. When we press getWeekday, we convert the date(our displayString) to julian day number and store it in a variable. We can then pass that to our method that determines the day, return the result and display it on the calculator.

We thought about the process when we are **subtracting a date from a date**. We wrote down the following steps.



date - date

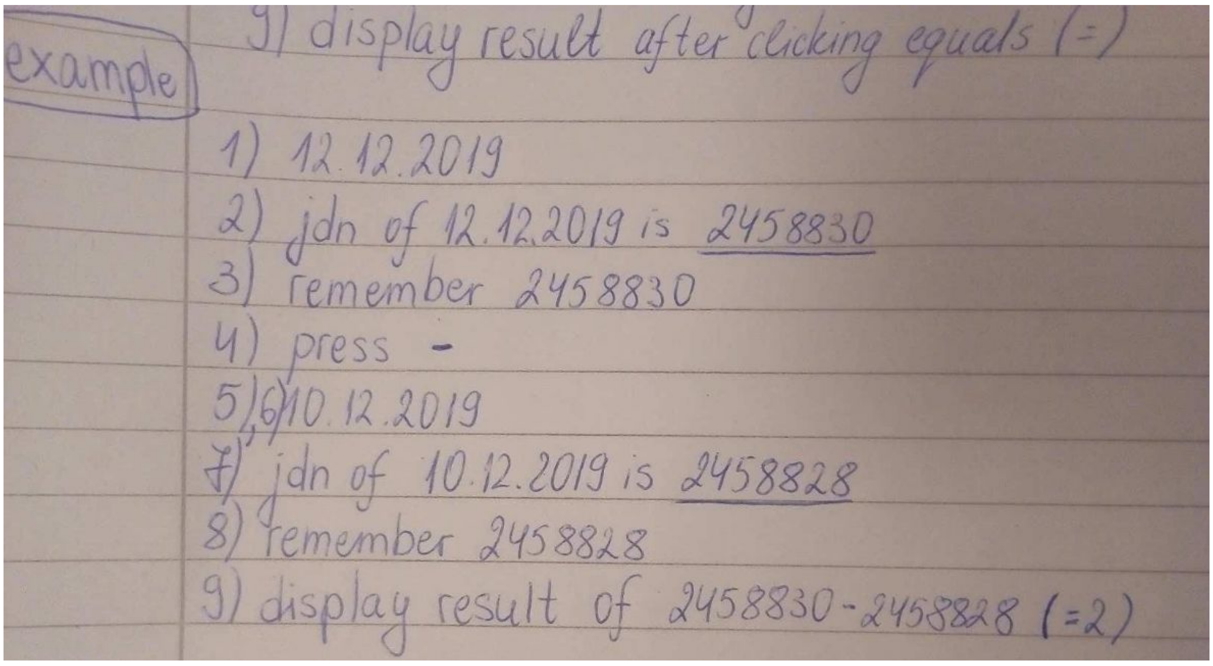
Steps

1) check if input is a date
2) ~~if it is remem~~ if it is, convert to jdn
3) remember jdn
4) apply - operator (minus)
5) now check again, what's the next input?
5.1) are we subtracting a date?
5.2) or are we subtracting a number of days?
6) in case we subtract a date (5.1)
7) convert the date to jdn
8) remember the second jdn
9) display result after clicking equals (=)

example

Writing the steps down was both interesting and helpful, because we were able to break down the whole process into smaller steps and see how we should solve the problem.

We also wrote down the following example to see the process with actual values.



9) display result after clicking equals (=)

example

1) 12.12.2019
2) jdn of 12.12.2019 is 2458830
3) remember 2458830
4) press -
5),6) 10.12.2019
7) jdn of 10.12.2019 is 2458828
8) remember 2458828
9) display result of 2458830 - 2458828 (=2)

Now that we knew what our approach will be, we were ready to move on to the exercises.

2.1 Input a date

We decided our specific date format that we are going to use to be DD.MM.YYYY

In the UserInterface class we added the following field.

```
    protected String displayString = "";
```

In the makeFrame method we also added the dot to our button panel.

```
        addButton(buttonPanel, ".");
```

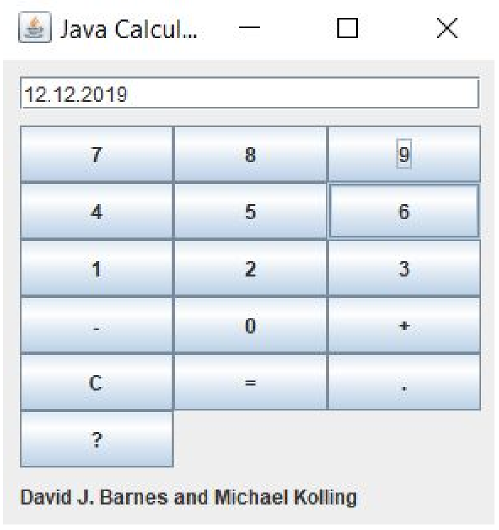
And then in the actionPerformed we added the following, so we can input a dot.

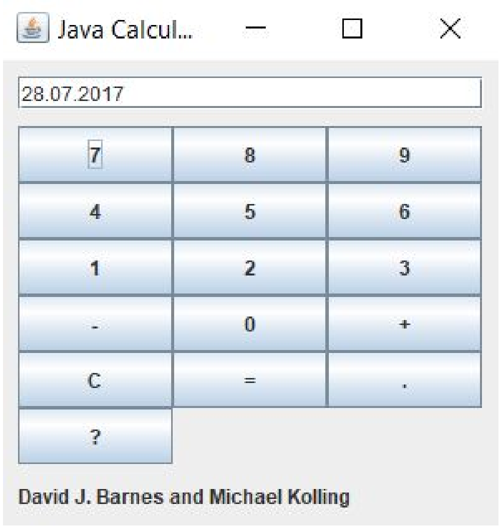
```
        else if(command.equals(".")) {
            displayString += command;
        }
```

We also changed the redisplay method in the following way.

```
    protected void redisplay()
    {
        display.setText("" + displayString);

    }
```

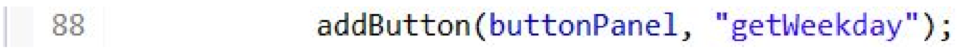
Now we were able to input a date in our calculator, so we decided to test it.


12.12.2019


28.07.2017

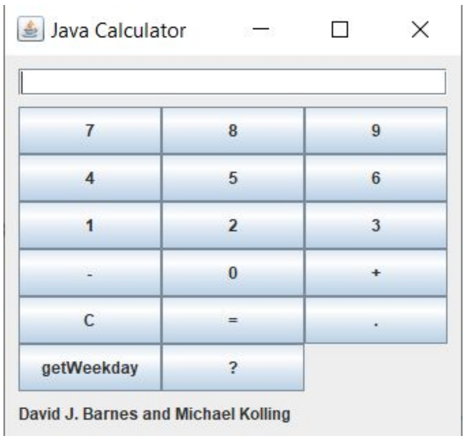
2.2 get the calculator to display a String for the weekday in the window (hint: you will need a button to push)

In the UserInterface class we added a new button to the button panel called "getWeekday".



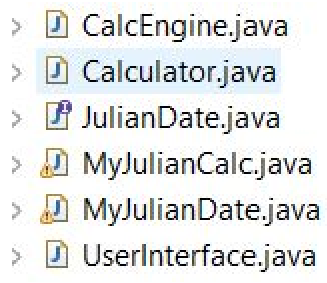
```
addButton(buttonPanel, "getWeekday");
```

We created a new calculator just to see how it looks like.



To get the weekday, we need to convert the date with our specific input dd.mm.yyyy to julian day number. We are going to reuse the classes that we already have from Lab 4.

We needed the JulianDate interface and the class MyJulianDate that implements the interface. We used Pavel's version of those two classes. We also created a new class MyJulianCalc. At this point we had 6 classes.

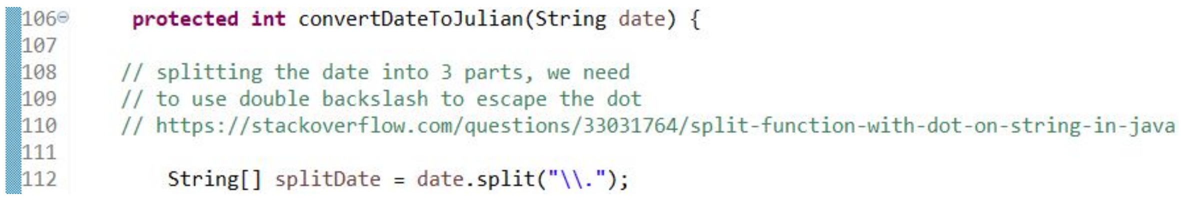


- CalcEngine.java
- Calculator.java
- JulianDate.java
- MyJulianCalc.java
- MyJulianDate.java
- UserInterface.java

In the MyJulianCalc class we created a new method and called it convertDateToJulian which takes a String as a parameter and returns an int. The String parameter we are going to pass in this method is the date that we have as a displayString in the calculator.

We remembered about the split method that we also used last semester, when we worked on the Tech Support project and we had to split on the whitespaces.

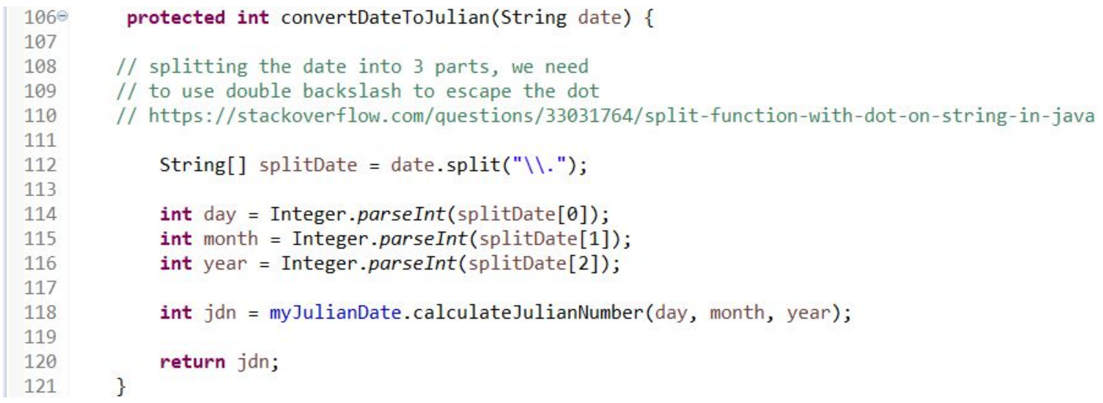
We decided to use the split method here as well so we can split the date we have as a String on the dot. We looked at an example here:
(Source: https://stackoverflow.com/questions/33031764/split-function-with-dot-on-string-in-java).
We noticed that we had to use double backslash to escape the dot. We used the split() method on date and stored the result in a new String array which we called splitDate. We made sure to include comments about everything so we can keep track of what we were doing. Our new method looked like that so far.



```
    protected int convertDateToJulian(String date) {

   // splitting the date into 3 parts, we need
   // to use double backslash to escape the dot
   // https://stackoverflow.com/questions/33031764/split-function-with-dot-on-string-in-java

       String[] splitDate = date.split("\\.");
```

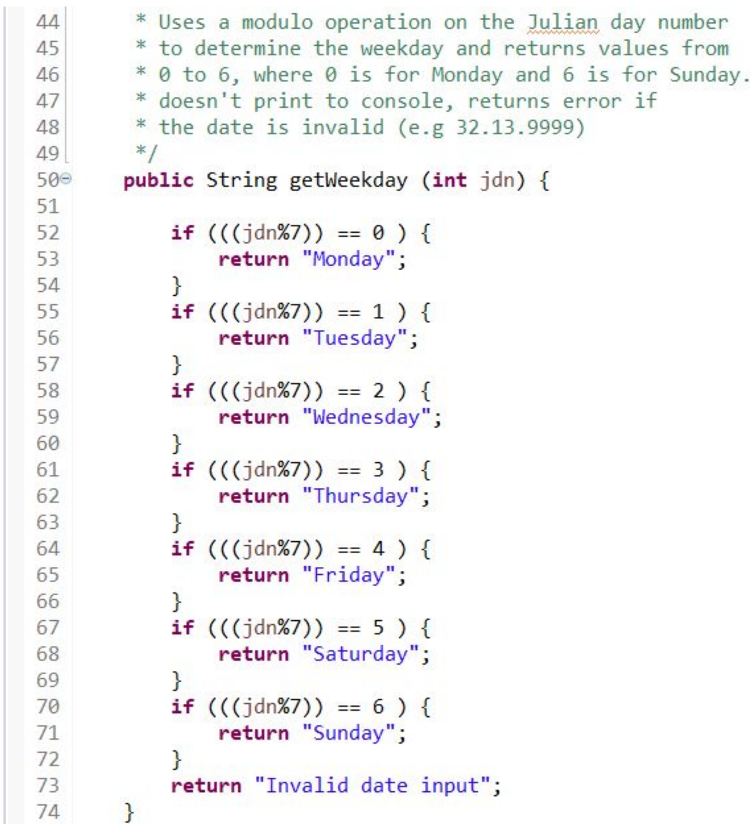
For the next step to work, we need to be sure the date is in the specific format dd.mm.yyyy In our string array splitDate we now have three elements. The first element is the number for the day and we can access it using splitDate[0]. Since we have it as a String, we need to use Integer.parseInt and pass splitDate[0] as a parameter and then store the result in a new int variable that we called day. We repeat the same step to store the month and the year, using splitDate[1] and splitDate[2] respectively.

Then we created a new int variable which we called jdn. We call the method calculateJulianNumber, pass the day, month and year that we got from splitDate as parameters and store the result in jdn. Then we return jdn, which now holds the value of the julian day number of the date we passed to this method. Our whole method looked like that.



```
    protected int convertDateToJulian(String date) {

   // splitting the date into 3 parts, we need
   // to use double backslash to escape the dot
   // https://stackoverflow.com/questions/33031764/split-function-with-dot-on-string-in-java

       String[] splitDate = date.split("\\.");

       int day = Integer.parseInt(splitDate[0]);
       int month = Integer.parseInt(splitDate[1]);
       int year = Integer.parseInt(splitDate[2]);

       int jdn = myJulianDate.calculateJulianNumber(day, month, year);

       return jdn;
   }
```

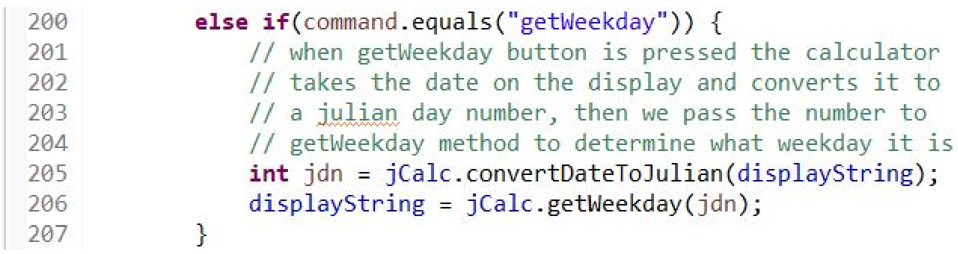
To determine the weekday we will use the following method that we called getWeekday.



```java
     * Uses a modulo operation on the Julian day number
     * to determine the weekday and returns values from
     * 0 to 6, where 0 is for Monday and 6 is for Sunday.
     * doesn't print to console, returns error if
     * the date is invalid (e.g 32.13.9999)
     */
    public String getWeekday (int jdn) {

        if (((jdn%7)) == 0 ) {
            return "Monday";
        }
        if (((jdn%7)) == 1 ) {
            return "Tuesday";
        }
        if (((jdn%7)) == 2 ) {
            return "Wednesday";
        }
        if (((jdn%7)) == 3 ) {
            return "Thursday";
        }
        if (((jdn%7)) == 4 ) {
            return "Friday";
        }
        if (((jdn%7)) == 5 ) {
            return "Saturday";
        }
        if (((jdn%7)) == 6 ) {
            return "Sunday";
        }
        return "Invalid date input";
    }
```

We already had that method, back from when we worked on Lab 4 about Julian dates. We changed it to return the corresponding day of the week instead of printing to the console. In case the date input was invalid we return a proper message.

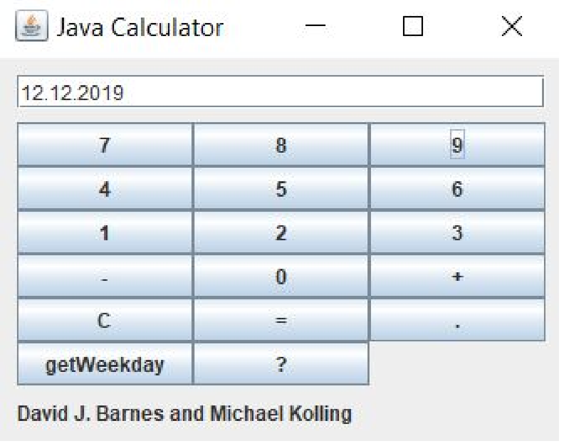
Now we went back to our UserInterface class and added the following in our actionPerformed method. When getWeekday is pressed, we convert the displayString to julian day number, use the number to determine the weekday and set it as a displayString.



```java
        else if(command.equals("getWeekday")) {
            // when getWeekday button is pressed the calculator
            // takes the date on the display and converts it to
            // a julian day number, then we pass the number to
            // getWeekday method to determine what weekday it is
            int jdn = jCalc.convertDateToJulian(displayString);
            displayString = jCalc.getWeekday(jdn);
        }
```

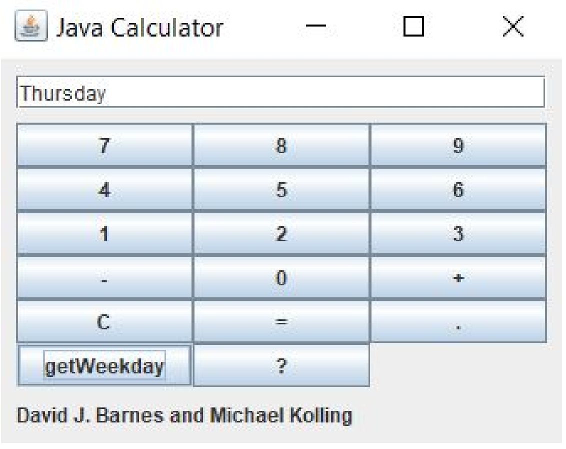
We decided to test our method and see the result.

First we input the date 12.12.2019 in the calculator and after pressing getWeekday, we got Thursday and it was the correct answer.



12.12.2019

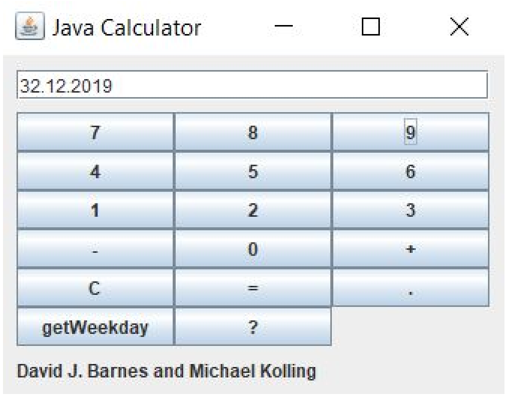
David J. Barnes and Michael Kolling



Thursday

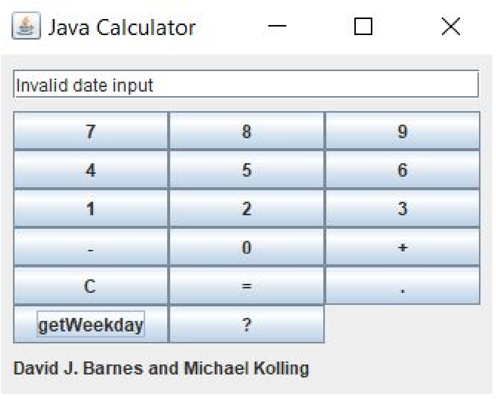
David J. Barnes and Michael Kolling

We also tested with an invalid date and it correctly displayed the Invalid date input message.



32.12.2019

David J. Barnes and Michael Kolling

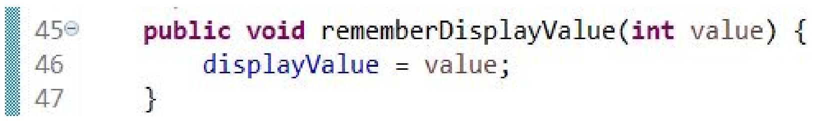


Invalid date input

David J. Barnes and Michael Kolling

2.3 add a number of days to a date, displaying the new date

To solve this problem we will need to be able to remember the displayValue internally, so we went on and created a new method in our CalcEngine called rememberDisplayValue. The method looks like that.

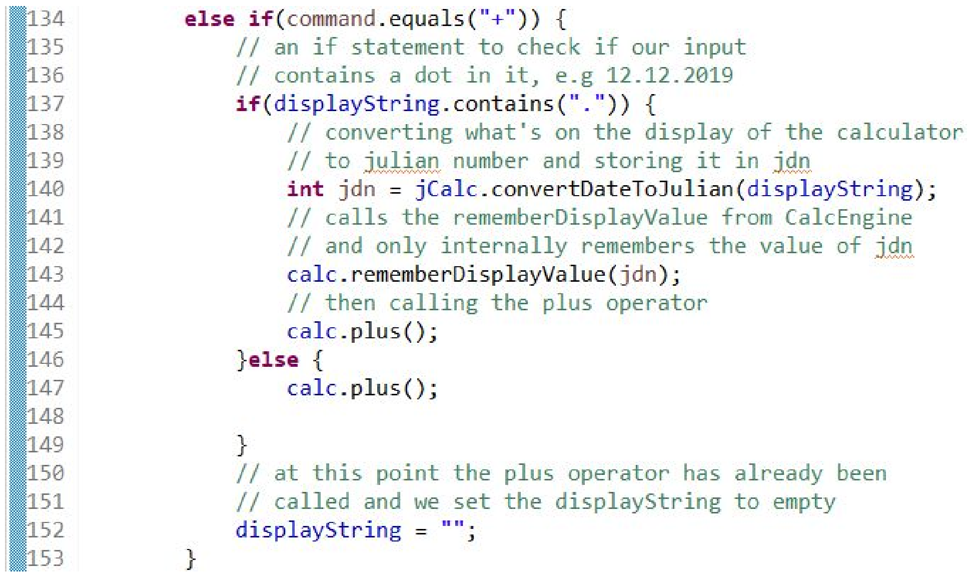


```java
public void rememberDisplayValue(int value) {
    displayValue = value;
}
```

After that we have to change the functionality when + gets pressed.

We made sure to write lots of comments so we know what we were doing. When “+” is pressed in the calculator, it checks if the displayString contains a dot. If it does, we know that the input(displayString) is a date. We use our convertDateToJulian method and pass the displayString as parameter. The result gets stored in a new int variable jdn. After that we remember jdn, using the method we created before that, and finally call the plus method.

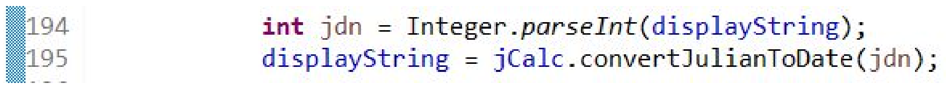
In the end we also have to set the displayString to an empty string, so we can now input the number of days we want to add.



```java
else if(command.equals("+")) {
    // an if statement to check if our input
    // contains a dot in it, e.g 12.12.2019
    if(displayString.contains(".")) {
        // converting what's on the display of the calculator
        // to julian number and storing it in jdn
        int jdn = jCalc.convertDateToJulian(displayString);
        // calls the rememberDisplayValue from CalcEngine
        // and only internally remembers the value of jdn
        calc.rememberDisplayValue(jdn);
        // then calling the plus operator
        calc.plus();
    }else {
        calc.plus();

    }
    // at this point the plus operator has already been
    // called and we set the displayString to empty
    displayString = "";
}
```

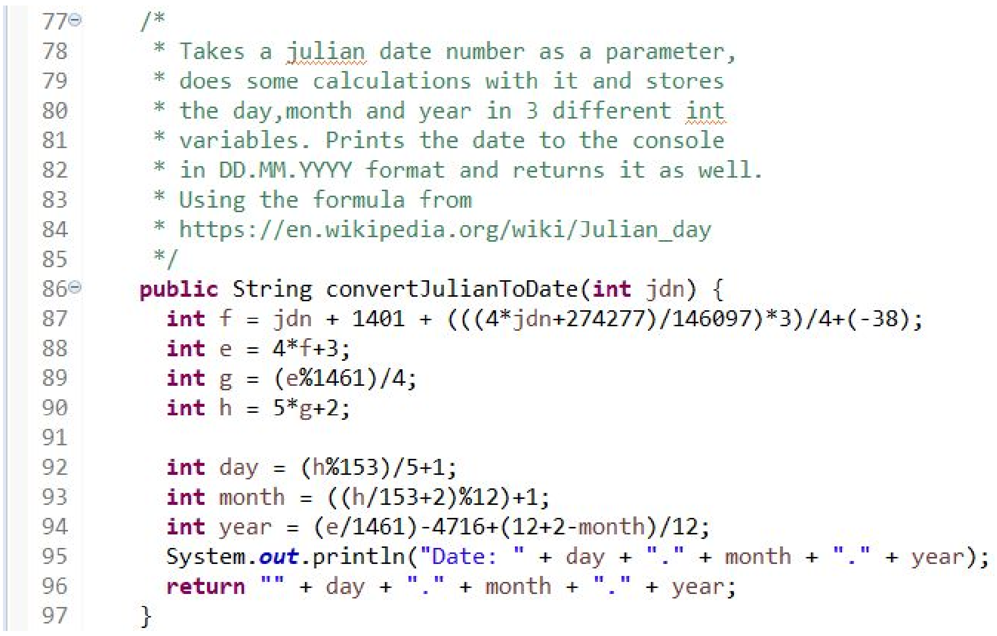
Then we also had to change the functionality when “=” equals gets pressed in the calculator.

We added the following lines to our code, when equals gets pressed.



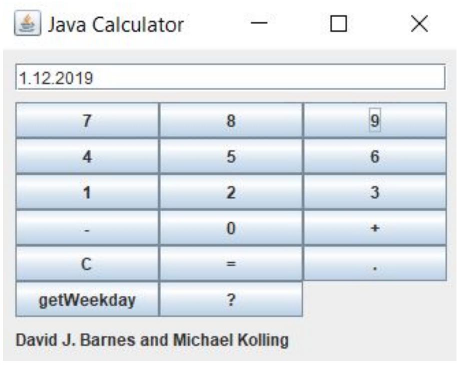
```java
        int jdn = Integer.parseInt(displayString);
        displayString = jCalc.convertJulianToDate(jdn);
```

We created a new int variable jdn. Again we use the Integer.parseInt() method and pass the displayString as parameter, then we store the result in jdn. To be able to display the date correctly on our calculator, we now need a method to convert from julian day number to Gregorian date. We found the formula for the conversion in the Wikipedia page about Julian day (Source: https://en.wikipedia.org/wiki/Julian_day). We created a new method called convertJulianToDate and made the needed calculations. In the end we return the Gregorian date in the specific format DD.MM.YYYY .

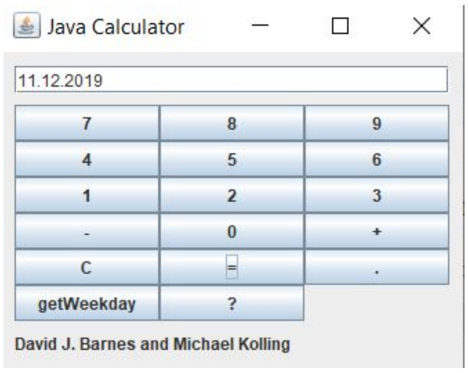


```java
    /*
     * Takes a julian date number as a parameter,
     * does some calculations with it and stores
     * the day,month and year in 3 different int
     * variables. Prints the date to the console
     * in DD.MM.YYYY format and returns it as well.
     * Using the formula from
     * https://en.wikipedia.org/wiki/Julian_day
     */
    public String convertJulianToDate(int jdn) {
      int f = jdn + 1401 + (((4*jdn+274277)/146097)*3)/4+(-38);
      int e = 4*f+3;
      int g = (e%1461)/4;
      int h = 5*g+2;

      int day = (h%153)/5+1;
      int month = ((h/153+2)%12)+1;
      int year = (e/1461)-4716+(12+2-month)/12;
      System.out.println("Date: " + day + "." + month + "." + year);
      return "" + day + "." + month + "." + year;
    }
```

We went on and tested if the calculator works correctly. We set the date to 1.12.2019 and added 10 days. The calculator worked correctly.



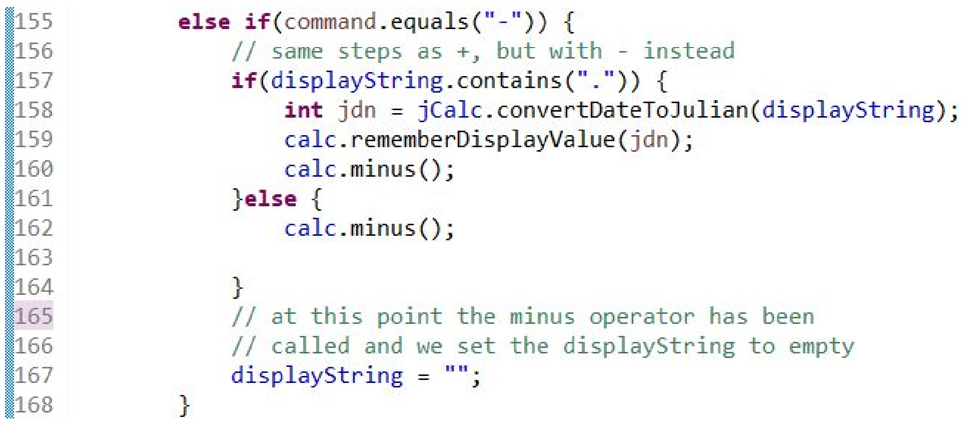
1.12.2019



11.12.2019

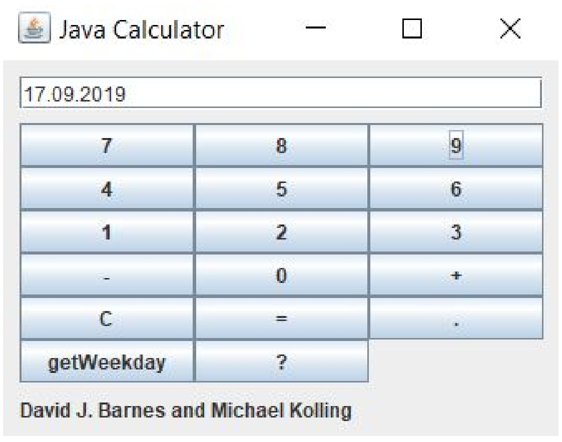
2.4 Subtract a number of days from a date, displaying the new date

The idea is the same as in previous task. The only thing we had to do is to change the functionality of the minus operator. We decided to add a screenshot as we have described the whole idea behind it in the previous task.

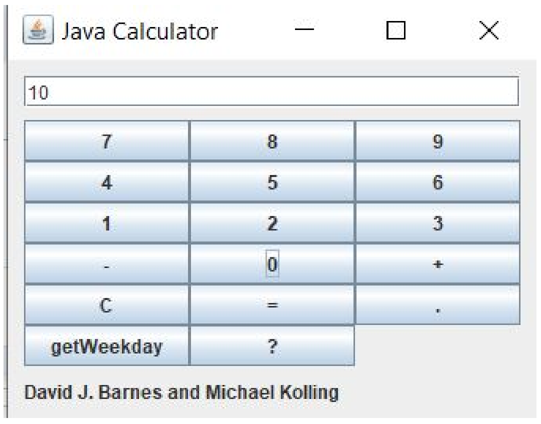


```
        else if(command.equals("-")) {
            // same steps as +, but with - instead
            if(displayString.contains(".")) {
                int jdn = jCalc.convertDateToJulian(displayString);
                calc.rememberDisplayValue(jdn);
                calc.minus();
            }else {
                calc.minus();

            }
            // at this point the minus operator has been
            // called and we set the displayString to empty
            displayString = "";
        }
```

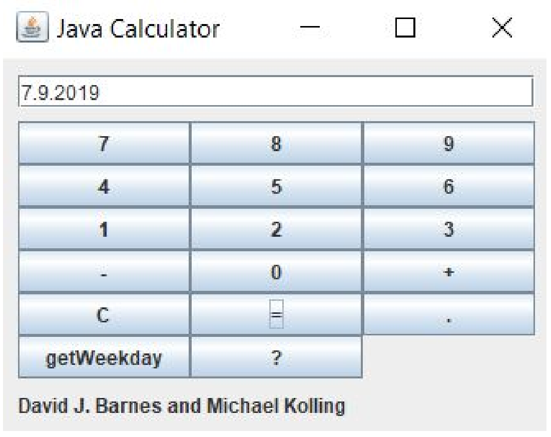
We tried to subtract 10 days from 17.09.2019



```
17.09.2019
```



```
10
```



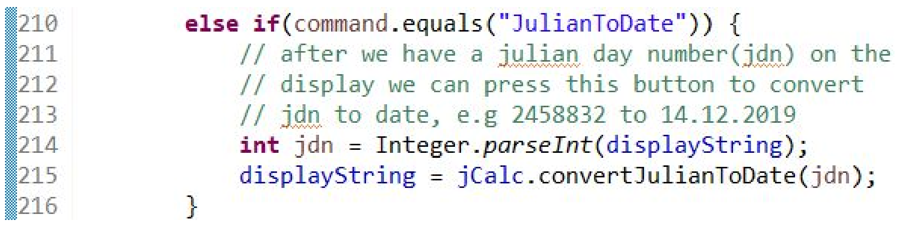
```
7.9.2019
```

We can see on the screenshot that the amount of days was successfully subtracted from the date. Our method works properly.

2.5 Subtract two dates, giving the number of days between the dates

To solve this exercise, we had to rework our equals method. To be able to add and subtract a number of days, while also being able to subtract two dates, we had to create a new button "JulianToDate". We collaborated with Martin from the other group, who gave us a hint to ad a new button. We also wrote another else if statement in our actionPerformed method.

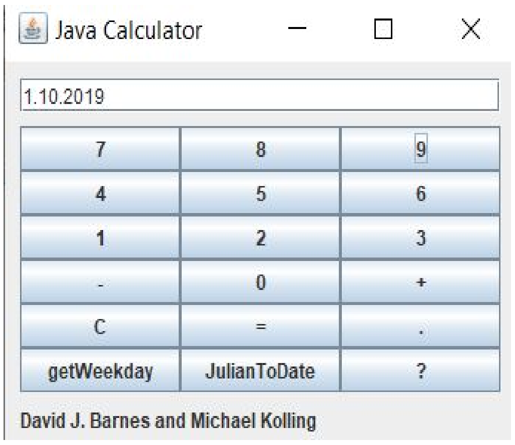
The two lines of code which we added earlier in our equals method in exercise 2.3, had to be moved here. In that way, after we add a number of days to a date, we get the julian day number on the calculator's display. Then we need to press the JulianToDate button and the date gets transformed to Gregorian date.



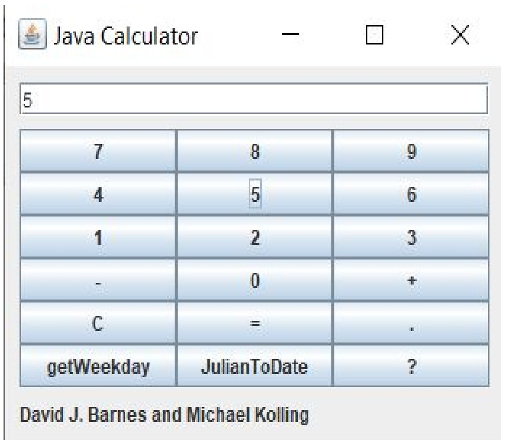
```
        else if(command.equals("JulianToDate")) {
            // after we have a julian day number(jdn) on the
            // display we can press this button to convert
            // jdn to date, e.g 2458832 to 14.12.2019
            int jdn = Integer.parseInt(displayString);
            displayString = jCalc.convertJulianToDate(jdn);
        }
```

We decided to test how this works and included an example with screenshots here.

We input the date 1.10.2019 and then we add 5.

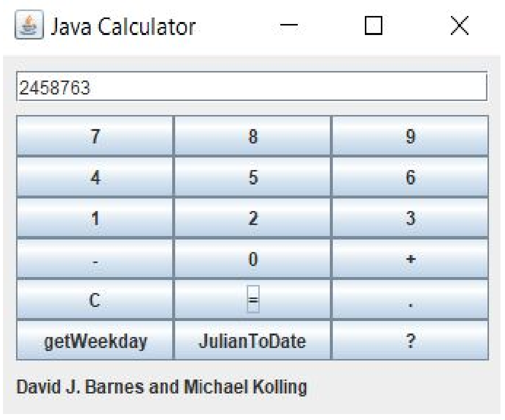


1.10.2019

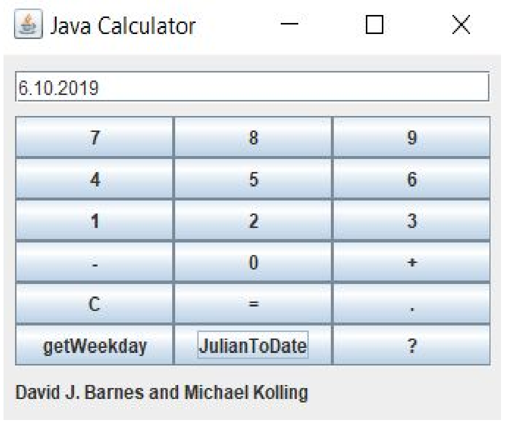


5

Our calculator shows the julian day number 2458763. Then we have to click the JulianToDate button and the date gets transformed to Gregorian date.

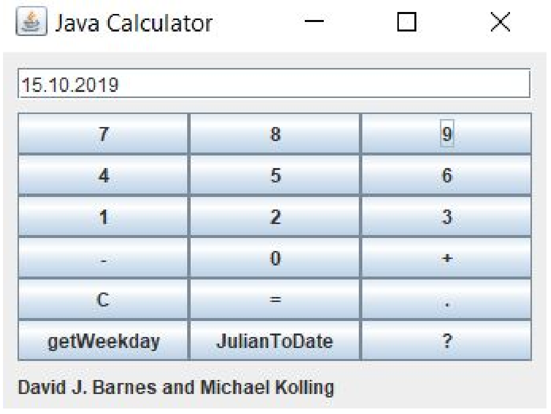


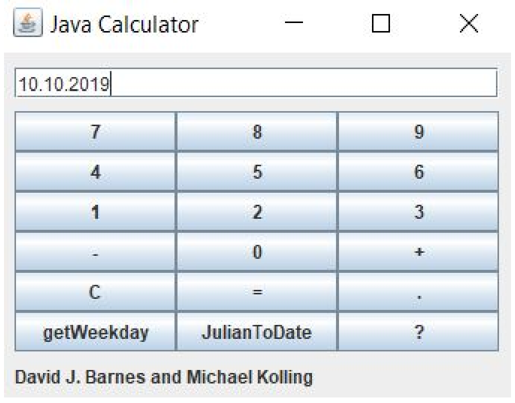
2458763



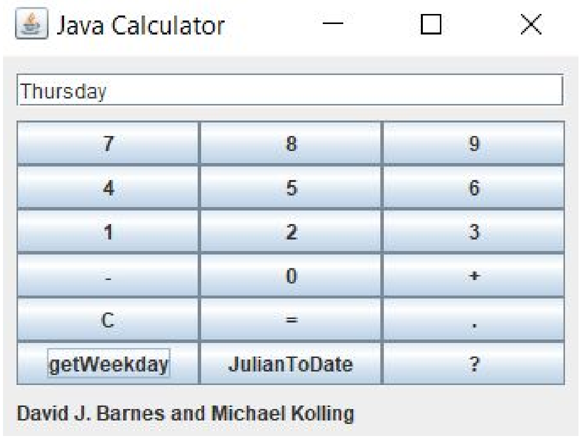
6.10.2019

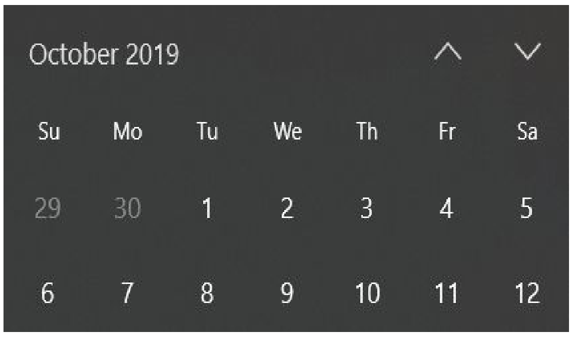
We also tested and subtracted a number of days and it worked correctly. We subtracted 5 days from 15.10.2019 and got 2458762 as a result. After pressing JulianToDate the correct date was displayed.





We made sure our getWeekday button still works fine as well. When pressed, our calculator showed Thursday. We made sure to check it in the Windows calendar and it was correct.





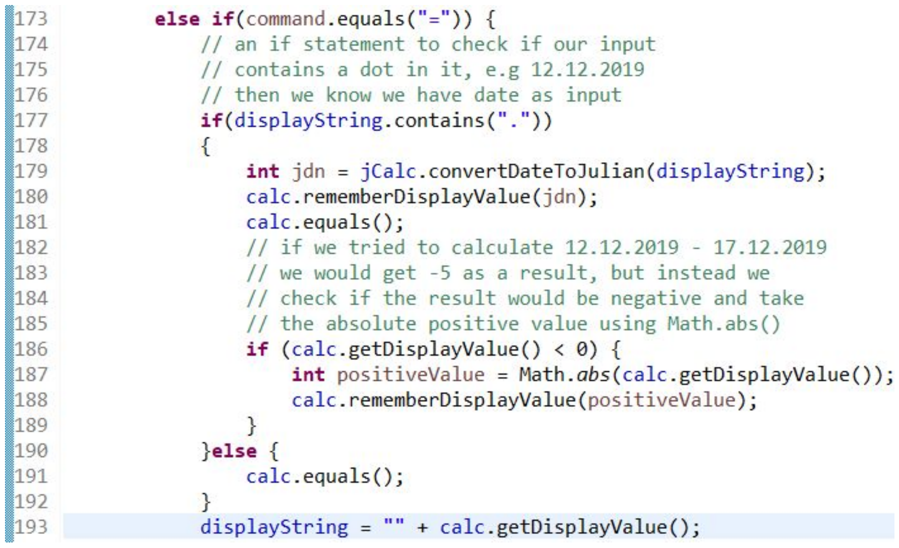
Then we went back to reworking our equals method.

When equals is pressed, we check if the current displayString is a date. If it is, we convert the date to julian day number and store it in a new int variable jdn. We remember the value of jdn, using our rememberDisplayValue method and then call the equals method.

We also added another if statement, in case we tried something like “ 12.12.2019 - 17.12.2019 “ in the calculator. Again, we made sure to comment on everything to remember what we were thinking when we were writing the code. Since the dates get converted to julian day number, the second date has a bigger value. We used the method Math.abs() which returns the absolute value.

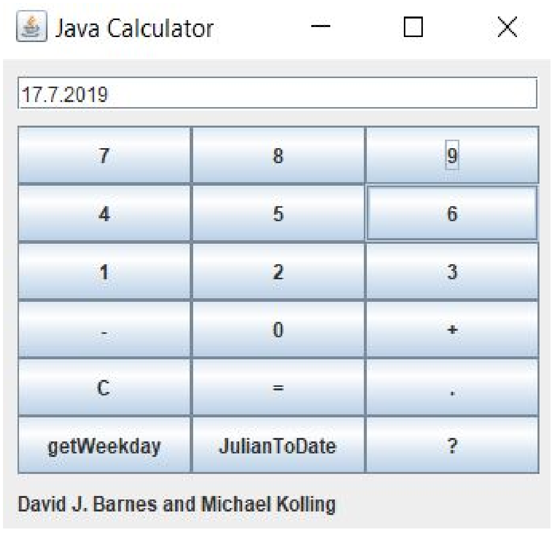
We were not sure if that would be necessary, but it was something we decided to add.

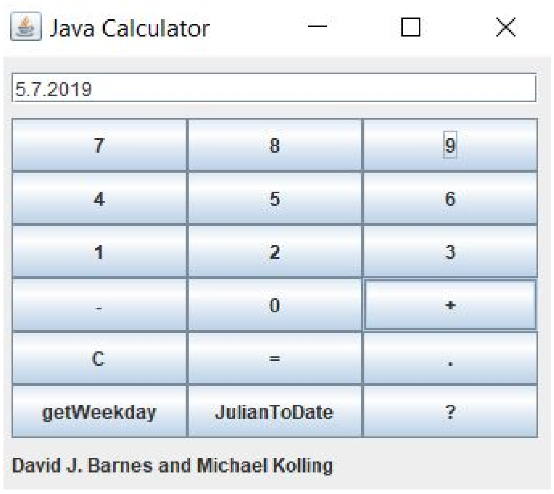
In the actionPerformed method, the functionality when equals is pressed is the following.



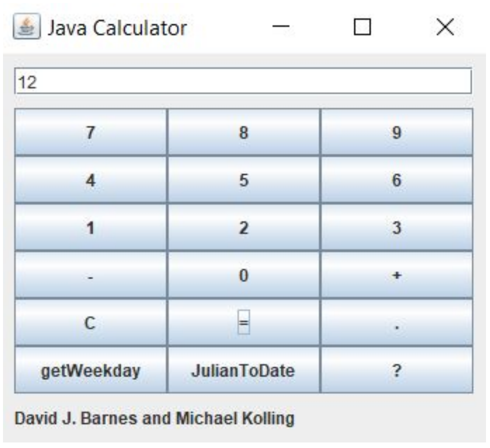
```
        else if(command.equals("=")) {
            // an if statement to check if our input
            // contains a dot in it, e.g 12.12.2019
            // then we know we have date as input
            if(displayString.contains("."))
            {
                int jdn = jCalc.convertDateToJulian(displayString);
                calc.rememberDisplayValue(jdn);
                calc.equals();
                // if we tried to calculate 12.12.2019 - 17.12.2019
                // we would get -5 as a result, but instead we
                // check if the result would be negative and take
                // the absolute positive value using Math.abs()
                if (calc.getDisplayValue() < 0) {
                    int positiveValue = Math.abs(calc.getDisplayValue());
                    calc.rememberDisplayValue(positiveValue);
                }
            }else {
                calc.equals();
            }
            displayString = "" + calc.getDisplayValue();
```

We went on and tested how our calculator works. We subtracted 5.7.2019 from 17.7.2019.





After pressing equals, we got the correct result.



It was nice to see that our idea worked. When subtracting a date from a date, we convert them both to julian day number, subtract the second date from the first and present the result on the calculator.

What did we learn?

Pavel Tsvyatkov

In this exercise I learned how to correctly split a date on the dot by using double backslash to escape it. Then I saw how we can store the parts we got after splitting into different variables and then use them for something else. It was very useful again to write down the process of solving the exercises step by step with pen and paper, before writing any code. When I see the process broken down into steps like this, it's a lot easier to find solutions.

Muhammad Safarov

As you have mentioned in the beginning of the lab, it's very useful to understand that we can create something that we can wrap something very complicated into it and make it work. As we were dealing with julian date before, i came to the conclusion that its pretty easy to solve another problem or maybe create something really cool like this calculator if you really understand the whole idea of solutions of the previous problems. I enjoyed working with Pavel. Great cooperator and team worker.

Appendix

Code

CalcEngine Class

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

   // Are we already building a value in the display, or will the
   // next digit be the first of a new one?
   private boolean buildingDisplayValue;
```

```
// Has a left operand already been entered (or calculated)?
private boolean haveLeftOperand;
// The most recent operator that was entered.
private char lastOperator;

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

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

/**
 * @return The value that should currently be displayed
 * on the calculator display.
 */
public int getDisplayValue()
{
   return displayValue;
}
/*
 * Remembering the display value in an integer
 * internally, so we can add and subtract after
 */
public void rememberDisplayValue(int value) {
                displayValue = value;
        }

/**
 * 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 '=' 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 1.0";
}

/**
 * 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;
        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();
    }
}
```

Calculator Class

```
/**
 * 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 {
	public static void main(String[] args) {
		Calculator calc = new Calculator();
		calc.show();
	}

	private CalcEngine engine;
	private UserInterface gui;

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

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

JulianDate interface

```
public interface JulianDate {
        public void setDay(int i);
        public int getDay();
        public void setMonth(int i);
        public int getMonth();
        public void setYear(int i);
        public int getYear();
        public boolean checkYear(int year);
        public boolean checkMonth(int month);
        public boolean checkDay(int day, int month);
        public int calculateJulianNumber(int day, int month, int year);
}
```

MyJulianCalc Class

```
import java.util.Calendar;
/*
 * Similar to what we did in Lab4, we are using the
 * getTodaysDate method to get the current date.
 * Introduced the new methods convertDateToJulian
 * and convertJulianToDate. Also changed the method
 * getWeekday to return the day.
 */
public class MyJulianCalc {

        private int dayToday, monthToday, yearToday;
        MyJulianDate myJulianDate;


        /*
         * Constructor for this class
         * creates a new MyJulianDate
         * calls getTodaysDate
         */
        public MyJulianCalc() {
                myJulianDate = new MyJulianDate();
```

```
		getTodaysDate();
	}




	/*
	 * creates a new calendar instance, gets
	 * today's day, month and year and stores
	 * them in the respective variable
	 */
	public void getTodaysDate() {

	Calendar calendar = Calendar.getInstance();
	dayToday = calendar.get(Calendar.DAY_OF_MONTH);
	monthToday = calendar.get(Calendar.MONTH)+1;
	yearToday = calendar.get(Calendar.YEAR);

	}



	/*
	 * Reworked birthday method from Lab 4
	 * Uses a modulo operation on the Julian day number
	 * to determine the weekday and returns values from
	 * 0 to 6, where 0 is for Monday and 6 is for Sunday.
	 * doesn't print to console, returns error if
	 * the date is invalid (e.g 32.13.9999)
	 */
	public String getWeekday (int jdn) {

		if (((jdn%7)) == 0 ) {
			return "Monday";
		}
		if (((jdn%7)) == 1 ) {
			return "Tuesday";
		}
		if (((jdn%7)) == 2 ) {
			return "Wednesday";
		}
		if (((jdn%7)) == 3 ) {
			return "Thursday";
		}
		if (((jdn%7)) == 4 ) {
			return "Friday";
		}
		if (((jdn%7)) == 5 ) {
```

```
                return "Saturday";
            }
            if (((jdn%7)) == 6 ) {
                return "Sunday";
            }
            return "Invalid date input";
        }


        /*
         * Takes a julian date number as a parameter,
         * does some calculations with it and stores
         * the day,month and year in 3 different int
         * variables. Prints the date to the console
         * in DD.MM.YYYY format and returns it as well.
         * Using the formula from
         * https://en.wikipedia.org/wiki/Julian_day
         */
        public String convertJulianToDate(int jdn) {
          int f = jdn + 1401 + (((4*jdn+274277)/146097)*3)/4+(-38);
          int e = 4*f+3;
          int g = (e%1461)/4;
          int h = 5*g+2;

          int day = (h%153)/5+1;
    int month = ((h/153+2)%12)+1;
          int year = (e/1461)-4716+(12+2-month)/12;
          System.out.println("Date: " + day + "." + month + "." + year);
          return "" + day + "." + month + "." + year;
        }

        /*
         * Takes a date as parameter, stores day,month and
         * year in a String array using split() on the
         * parameter. It's important that the date is in
         * DD.MM.YYYY format. Parsing all three parts, then
         * using them to calculate the julian date number
         */
         protected int convertDateToJulian(String date) {

        // splitting the date into 3 parts, we need
        // to use double backslash to escape the dot
        //
https://stackoverflow.com/questions/33031764/split-function-with-dot-on-string-in-java

            String[] splitDate = date.split("\\.");
            int day = Integer.parseInt(splitDate[0]);
            int month = Integer.parseInt(splitDate[1]);
            int year = Integer.parseInt(splitDate[2]);

            int jdn = myJulianDate.calculateJulianNumber(day, month, year);

            return jdn;
        }
}
```

MyJulianDate Class

```
/*
 * This class implements the JulianDate interface.
 */
public class MyJulianDate implements JulianDate{

        private int day, month, year, julianDayNumber;
        private double julianDate;


        public MyJulianDate() {

        }
        public MyJulianDate(int day, int month, int year) {
                this.day = day;
                this.month = month;
                this.year = year;
        }

        //sets the day to a valid day
        public void setDay(int day) {
                this.day = day;
        }

        //returns the day field
        public int getDay() {
                return day;
        }

        //sets the month to a valid month
        public void setMonth(int month) {
```

```
		this.month = month;
	}

	//returns the month field
	public int getMonth() {
		return month;
	}

	//sets the year to a valid year
	public void setYear(int year) {
		this.year = year;
	}
	//returns the year field
	public int getYear() {
		return year;
	}


	/*
	 * checking if the year is in the allowed range
	 */
	public boolean checkYear(int year) {
		if (year > -4714 && year < 3269) {
			return true;
		} else {
			System.out.println("The entered year doesn't fit to the range of the
Julian Calendar.");
			return false;
		}
	}

	/*
	 * checking if the month is in the allowed range
	 *
	 */
	public boolean checkMonth(int month) {
		if (month>0 && month<13) {
			return true;
		} else
			return false;

	}

	/*
	 * checks if the entered day and month are valid
	 */
	public boolean checkDay(int day, int month) {
		//February
		if (month == 2 && day > 0 && day < 29) {
			return true;
		}
		//April,June
		if (month<8 && month != 2 && month%2==0 && day>0 && day < 31) {
			return true;
		}
		//January,March,May,July
		else if (month<8 && month%2 !=0 && day >0 && day<32) {
			return true;
		}
		//August,October,December
		else if (month > 7 && month%2==0 && day>0 && day<32) {
			return true;
		}
		//September,October
		else if (month > 7 && month%2!=0 && day>0 && day<31) {
			return true;
		}
		else {
			return false;
		}
	}

	/*
	 * performs the necessary checks, if the entered day,month and
	 * year are correct then calculates them and returns the
	 * julian date number, else if input is incorrect it returns -1
	 */
	public int calculateJulianNumber(int day, int month, int year) {

		if (checkDay(day, month) && checkMonth(month) && checkYear(year)) {
			julianDayNumber = (1461 * (year + 4800 + (month - 14)/12))/4 +
					(367 * (month - 2 - 12 * ((month - 14)/12)))/12 -
					(3 * ((year + 4900 + (month - 14)/12)/100))/4 + day -
32075;
			return julianDayNumber;
		}
		else
			return -1;

	}

}
```

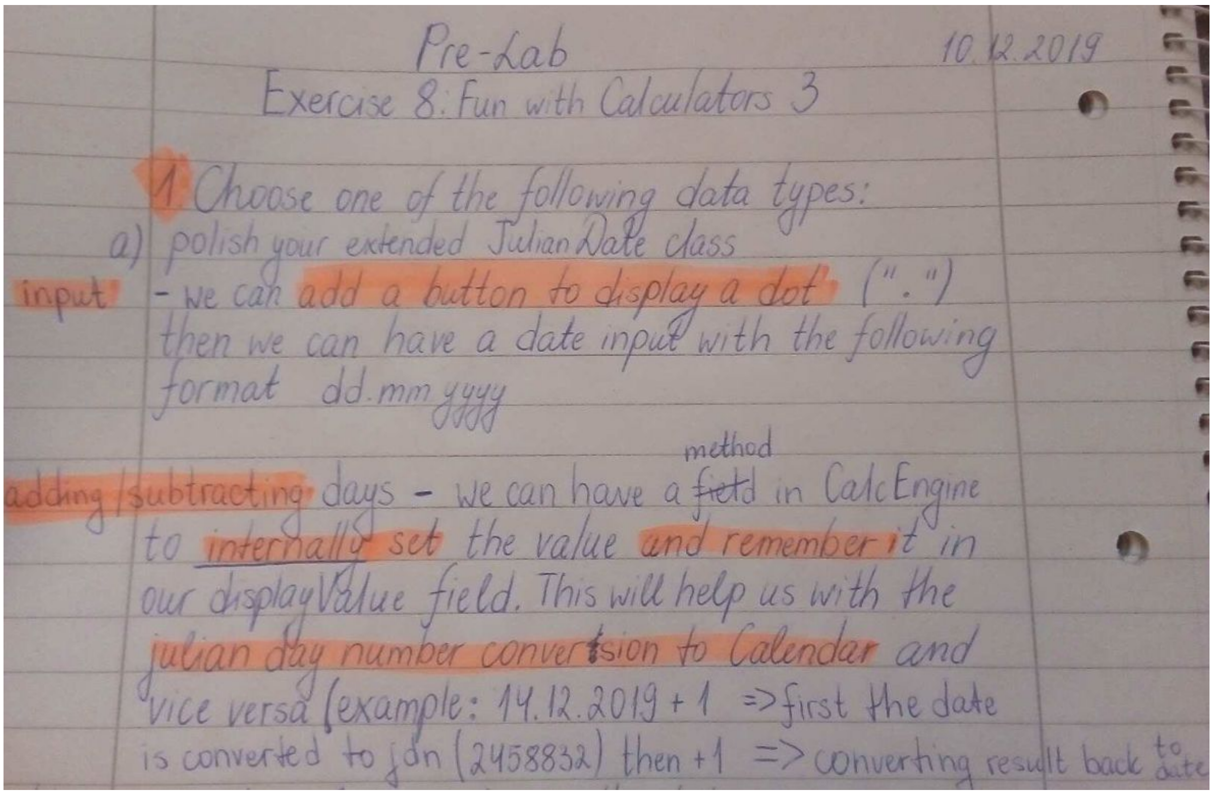
UserInterface Class

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

/**
 * A graphical user interface for the calculator. No calculation is being
 * done here. This class is responsible just for putting up the display on
 * screen. It then refers to the "CalcEngine" to do all the real work.
 *
 * @author David J. Barnes and Michael Kolling
 * @version 2008.03.30
 */
public class UserInterface
    implements ActionListener
{
    protected CalcEngine calc;
    protected boolean showingAuthor;
    protected JPanel buttonPanel = new JPanel();
    protected JFrame frame;
    protected JTextField display;
    protected JLabel status;
    protected String displayString = "";
    protected MyJulianCalc jCalc;
    protected MyJulianDate myJulianDate;

    /**
     * Create a user interface.
     * @param engine The calculator engine.
     */
    public UserInterface(CalcEngine engine)
    {
        calc = engine;
        jCalc = new MyJulianCalc();
        myJulianDate = new MyJulianDate();
        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);

        buttonPanel.setLayout(new GridLayout(6, 4));

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

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


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

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

        // added dot, getWeekday, JulianToDate for Lab 8
        // collaborated with Martin from the other group
        addButton(buttonPanel, ".");
        addButton(buttonPanel, "getWeekday");
        addButton(buttonPanel, "JulianToDate");
        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")) {
    displayString += command;
    int number = Integer.parseInt(command);
    calc.numberPressed(number);
}
else if(command.equals("+")) {
    // an if statement to check if our input
    // contains a dot in it, e.g 12.12.2019
    if(displayString.contains(".")) {
            // converting what's on the display of the calculator
            // to julian number and storing it in jdn
            int jdn = jCalc.convertDateToJulian(displayString);
            // calls the rememberDisplayValue from CalcEngine
            // and only internally remembers the value of jdn
            calc.rememberDisplayValue(jdn);
            // then calling the plus operator
            calc.plus();
    }else {
                        calc.plus();

                }
    // at this point the plus operator has already been
    // called and we set the displayString to empty
    displayString = "";
}

else if(command.equals("-")) {
    // same steps as +, but with - instead
    if(displayString.contains(".")) {
            int jdn = jCalc.convertDateToJulian(displayString);
            calc.rememberDisplayValue(jdn);
            calc.minus();
    }else {
                        calc.minus();

                }
    // at this point the minus operator has been
    // called and we set the displayString to empty
    displayString = "";
}

else if(command.equals(".")) {
    displayString += command;
}
else if(command.equals("=")) {
    // an if statement to check if our input
    // contains a dot in it, e.g 12.12.2019
    // then we know we have date as input
    if(displayString.contains("."))
    {
            int jdn = jCalc.convertDateToJulian(displayString);
            calc.rememberDisplayValue(jdn);
            calc.equals();
            // if we tried to calculate 12.12.2019 - 17.12.2019
            // we would get -5 as a result, but instead we
            // check if the result would be negative and take
            // the absolute positive value using Math.abs()
            if (calc.getDisplayValue() < 0) {
                            int positiveValue = Math.abs(calc.getDisplayValue());
                            calc.rememberDisplayValue(positiveValue);
                    }
    }else {
            calc.equals();
                }
    displayString = "" + calc.getDisplayValue();

// following makes adding/subtracting display the date
// correctly transformed, but breaks the date - date
//  int jdn = Integer.parseInt(displayString);
//  displayString = julianCalc.jdnToCalendar(jdn);
}

else if(command.equals("getWeekday")) {
    // when getWeekday button is pressed the calculator
    // takes the date on the display and converts it to
    // a julian day number, then we pass the number to
    // getWeekday method to determine what weekday it is
    int jdn = jCalc.convertDateToJulian(displayString);
    displayString = jCalc.getWeekday(jdn);
}

else if(command.equals("JulianToDate")) {
    // after we have a julian day number(jdn) on the
    // display we can press this button to convert
    // jdn to date, e.g 2458832 to 14.12.2019
    int jdn = Integer.parseInt(displayString);
    displayString = jCalc.convertJulianToDate(jdn);
}
// clears the display ( sets to an empty string)
else if(command.equals("C")) {
   displayString = "";
}
```

```
        else if(command.equals("?")) {
            showInfo();
        }

        // else unknown command.

        redisplay();
    }



    /**
     * Update the interface display to show the current value of the
     * calculator.
     * Same redisplay method as the one we used in Lab 7, where we
     * had to deal with an infix string input.
     */
    protected void redisplay()
    {
        display.setText("" + displayString);

    }

    protected String getDisplayString() {
            return displayString;
    }
    /**
     * 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;
    }
}
```

Pre-Lab

- **Pavel**



Pre-Lab 10.12.2019

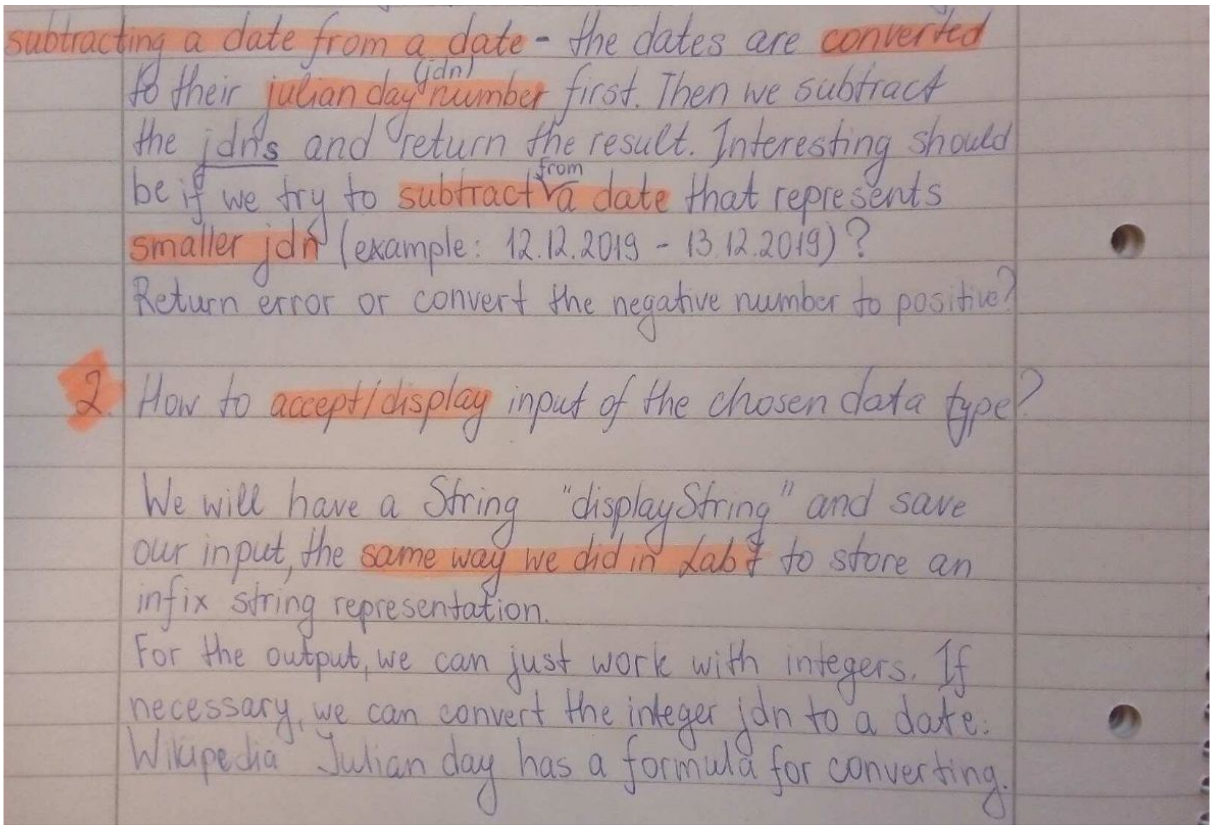
Exercise 8: Fun with Calculators 3

1 Choose one of the following data types:

a) polish your extended JulianDate class

input - we can add a button to display a dot (".") then we can have a date input with the following format dd.mm.yyyy

adding/subtracting days - we can have a ~~field~~ method in CalcEngine to internally set the value and remember it in our displayValue field. This will help us with the julian day number conver~~t~~sion to Calendar and vice versa (example: 14.12.2019 + 1 => first the date is converted to jdn (2458832) then +1 => converting result back to date



subtracting a date from a date - the dates are converted to their julian day (jdn) number first. Then we subtract the jdn's and return the result. Interesting should be if we try to subtract from a date that represents smaller jdn (example: 12.12.2019 - 13.12.2019)?
Return error or convert the negative number to positive?

2 How to accept/display input of the chosen data type?

We will have a String "displayString" and save our input, the same way we did in Lab 7 to store an infix string representation.
For the output, we can just work with integers. If necessary, we can convert the integer jdn to a date: Wikipedia "Julian day" has a formula for converting.