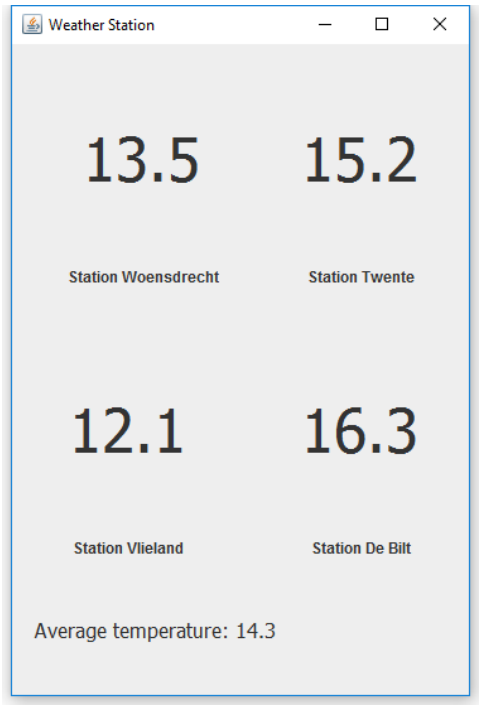


Assignment 3

Content of Application Development on Day 3
Lecture
The lecture explains object-oriented programming, methods, parameters, and the scope of variables.
Tutorial
In the tutorial you practise creating classes and methods, and applying global and local variables. You also learn how a program can use data from the Internet.

At the end of day 3 you will be able to
- Design and create classes and methods
- Use global and local variables
- Write expressions
- Write a program that uses data from the Internet

Reading between day 3 and day 4
Book:
Head First Java (K. Sierra & B. Bates): chapter 10, 13.
Aan de slag met Java *[Getting Started with Java]* (Gertjan Laan): chapter 3.9 - 4.
Online:
javatpoint.com:Swing introduction, JLabel, JPanel.

Creating a Weather Station

We are going to create an application that can display the temperature of several weather stations in the Netherlands, as well as the average temperature of those weather stations. You are given a *WeatherStation* class for reading the data. This class can read data using an XML file. XML files are used on the internet to provide data in a structured way (in a specific format). *Buienradar* is a site that provides weather data from several weather stations in XML form.

The example used in this assignment is a weather station with temperatures of four stations.

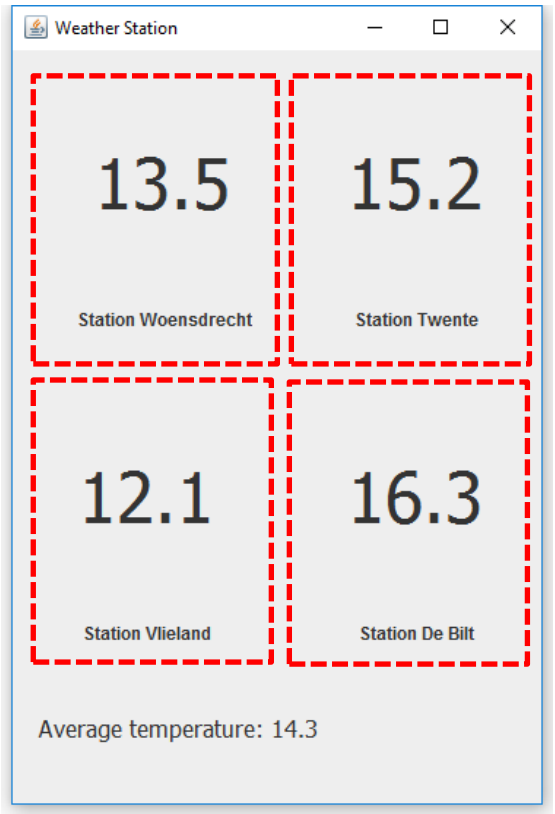
Tutorial Steps

1. Design and create the user interface
2. Add the WeatherStation class to the project
3. Create the display of the temperature
4. Add multiple instances of the class to the user interface
5. Calculate the average temperature and display it

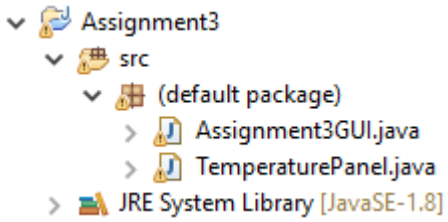
The steps are explained in detail below. You will have to apply many of the skills you have learned in the previous assignments. It may be useful, therefore, to check these assignments every now and then.

1. Design and create the user interface

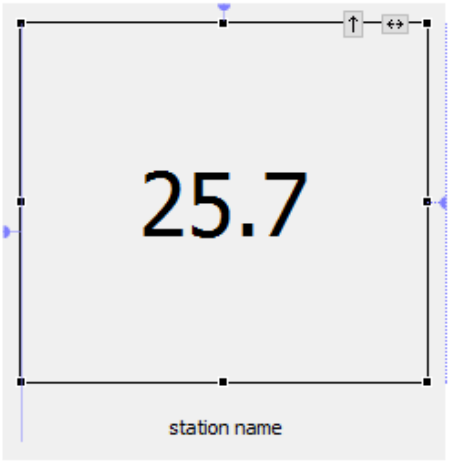
First, we create one of the four Panels for displaying the temperature with the name of the weather station. Then we use that Panel several times (copy) to display the data of several weather stations. Also, we create a separate part of the user interface in which the average temperature is displayed (at the bottom in the example).



Start with a new project "Assignment3". Add a JFrame and a JPanel (both via *File* > *New Other*, and then under *WindowBuilder*, *Swing Designer*). The JFrame will be the user interface of the entire application; name this "Assignment3GUI". The JPanel becomes a Panel for displaying the temperature of 1 station, which we will use several times. Call this "TemperaturePanel", for example:

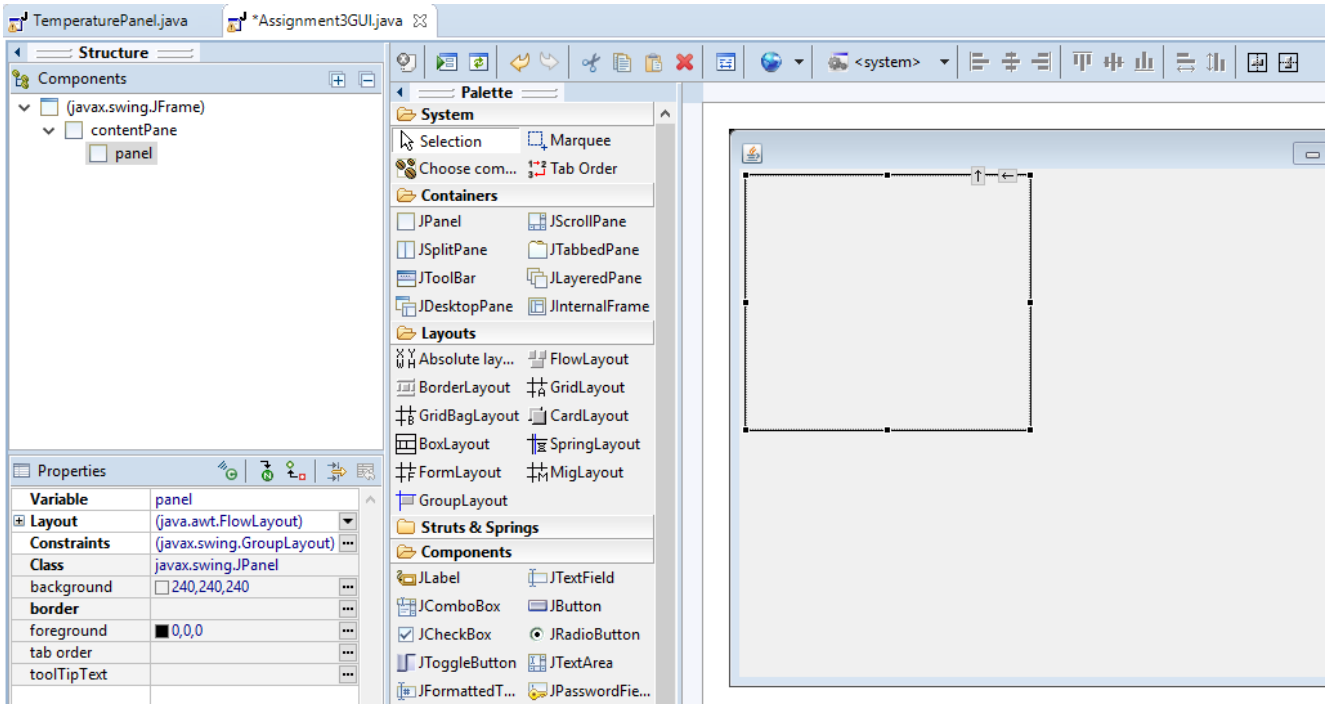


Using the Designer, add two labels (JLabel) to the *TemperaturePanel*, one of which with a large font (e.g. font size 48), which we will use to display the temperature. The smaller label is used to display the name of the weather station. Remember to choose a suitable layout (property of the Panel) and meaningful names for all the components.



Before continuing, save all the files via *File* > *Save All*.

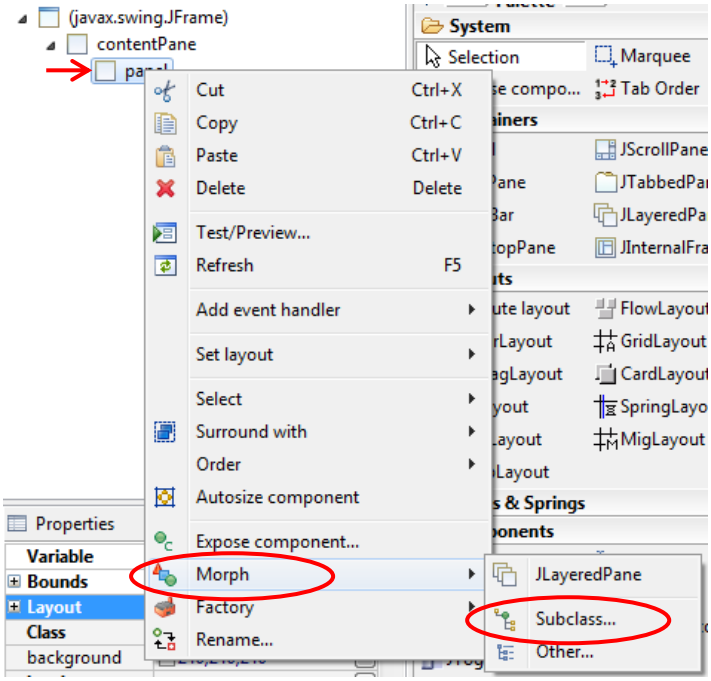
Now we are going to add the Panel to the main user interface. Switch to Assignment3GUI.java. Set the layout of the contentPane to GroupLayout (or any layout you prefer). Add a JPanel to the user interface:



We are now going to morph (change) this Panel into the TemperaturePanel we have just created. In the list of Components click with the right mouse button on the panel and select *Morph*, *Subclass*.

You previously practiced 'Morphing' in Assignment 2 with the DrawingPanel class.

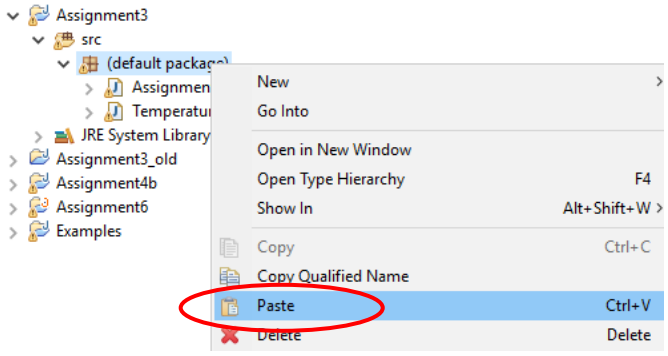
If you have problems, Save All files and retry.

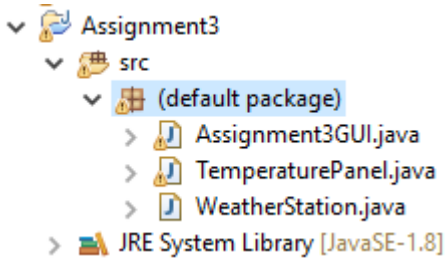


Type "TemperaturePanel" in the screen and click *Ok*.

2. Add the WeatherStation class to the project

The zip-file of the assignment contains a file called WeatherStation.java. Make sure you have extracted the zip file. Copy the Weatherstation.java (using the Windows Explorer) file and paste this in Eclipse by clicking with the right mouse button on the *default package* and selecting *Paste*:





The project will now look like the image on the right.

3. Create the display of the temperature

We can now add an object of type WeatherStation to the TemperaturePanel-class, so we can use methods of this object to set up the station and retrieve data (like the temperature and name) from that station by calling the method.

In the pieces of sample code below, the declaration of the variable (step 1) for the object is shown in red and the assignment using the new statement in blue (step 2).

First, declare a class variable *w* of type *WeatherStation* and make it *private*:

Creating an object consists of two steps:

1. Creating a (class) variable for the object
2. Creating the object in memory, with a **new** statement

More info @ slide 17 of the presentation

```
public class TemperaturePanel extends JPanel {

    private WeatherStation w;

    /**
     * Create the panel.
     */
    public TemperaturePanel() {
```

Next, use a *new* statement to create the object for *w*. We do this at the end of the constructor:

```
public class TemperaturePanel extends JPanel {

    private WeatherStation w;

    /**
     * Create the panel.
     */
    public TemperaturePanel() {

        JLabel labelTemp = new JLabel("25.7");
        labelTemp.setFont(new Font("Tahoma", Font.PLAIN, 48));
        labelTemp.setHorizontalAlignment(SwingConstants.CENTER);

        JLabel lblStationName = new JLabel("station name");
        lblStationName.setHorizontalAlignment(SwingConstants.CENTE
        GroupLayout groupLayout = new GroupLayout(this);
        setLayout(groupLayout);

        w = new WeatherStation();
    }
}
```

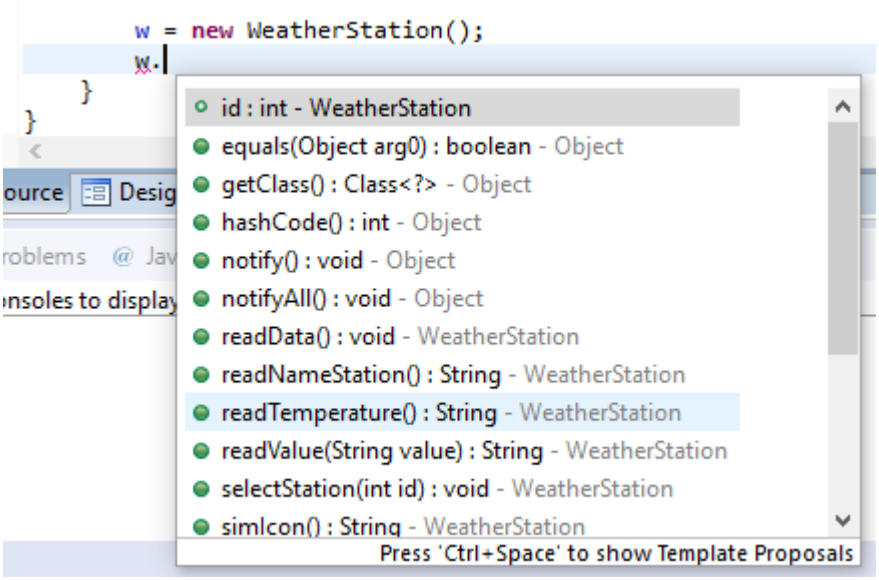
This is the constructor of this class. This is a special method that is used for, among others, creating objects within the class.

A constructor has the same name as the class.

More info @ slide 7 of the presentation

As you can see, the components of the user interface are defined in the constructor. These are the two labels that we have created using the WindowBuilder. The names of the labels (in this example *labelTemp* and *labelStationName*) may be different for you.

After the object for the weather station has been created, we can request things from the object by means of the method calls. For instance, the Weather Station has a *readTemperature()* method to retrieve the temperature. You can get a list of all methods of object ***w*** when you type a 'w' followed by a dot:

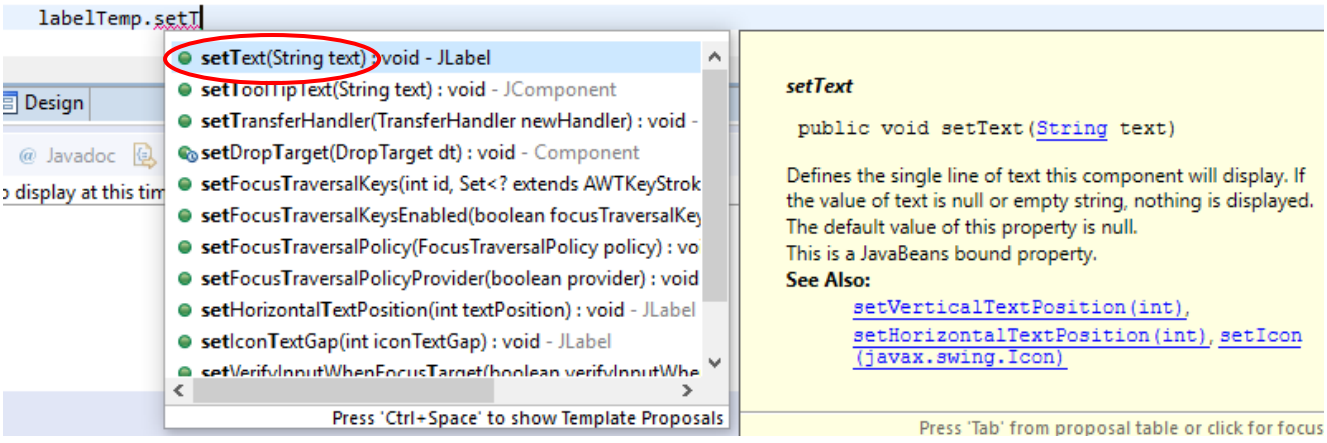


We can request the temperature from object ***w*** by calling the *readTemperature()* method. This method results in a String. So, we create a new variable of type String and give it the value that results from the method call:

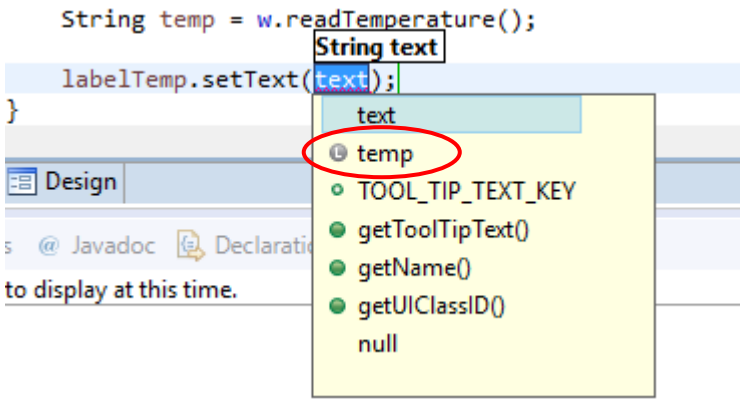
```
String temp = w.readTemperature();
```

Now we are going to display the value of the temperature in the label for the temperature. To do this, the label has a *setText()* method.

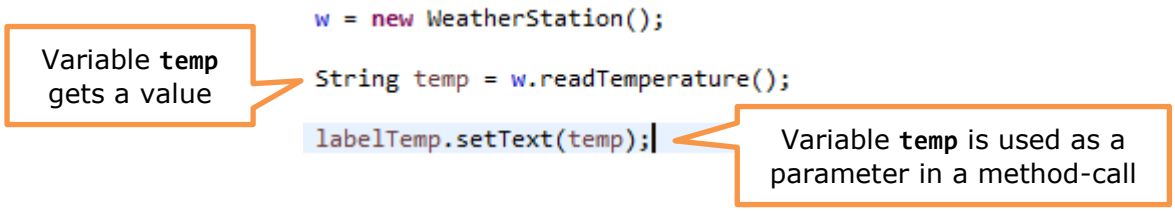
Type the variable name of the label of the temperature (in this example *labelTemp*), followed by a dot. Start typing the method name (setText) and select that method from the list by double-clicking it:



Next, a selection screen appears where you can select a parameter from a number of suggestions:



Eclipse has detected that we have created a new variable *temp* of type String on the previous line, as a result of which it is also included in the suggestions, because it "fits" as a parameter (it is of the same type). We select the suggested *temp* variable by double-clicking it. We close the line with a semicolon. The three lines of code we created is finished, and looks like:



You can now check if the temperature is actually displayed: run the program using the Run button .

Include a comment for the lines you have just written.

Similarly, add more code which displays the name of the weather station in the other label.

More info on parameters of methods @ slides 9+10 of the presentation

4. Add multiple instances of the class to the user interface

The WeatherStation class can retrieve data from a number of weather stations. When creating the object, you can specify the ID of a weather station using a parameter. If this parameter is missing, the default weather station from the XML file is used. To use multiple weather stations, we must know the station ID. We can find this in the XML file at the following address xml.buienradar.nl (this is unfortunately in Dutch). Open this address in a web browser and view the XML file that is displayed. Data of a weather station always starts with:

```
<weerstation id="6340">
```

Followed by the data of that station.

If we want to use data from a different station, we need to specify the `id` as parameter when creating the object:

```
w = new WeatherStation(6340);
```

Because we want to create multiple panels later, each with different IDs, it is useful to make the ID configurable by making it a parameter for the constructor. Therefore, we adjust the constructor of TemperaturePanel, so that this is assigned a parameter ID. Because this ID is an integer, the type will therefore be **int**. Change the head of the constructor to add the parameter ID:

```
public class TemperaturePanel extends JPanel {

    private WeatherStation w;

    /**
     * Create the panel.
     */
    public TemperaturePanel(int id) {
```

And add the *id* variable as a parameter when creating the object of the Weather Station:

```
w = new WeatherStation(id);
```

Now save the TemperaturePanel.java file (CTRL+S). Once you have done this, an error message will appear in Assignment3GUI.java at the call of the constructor:

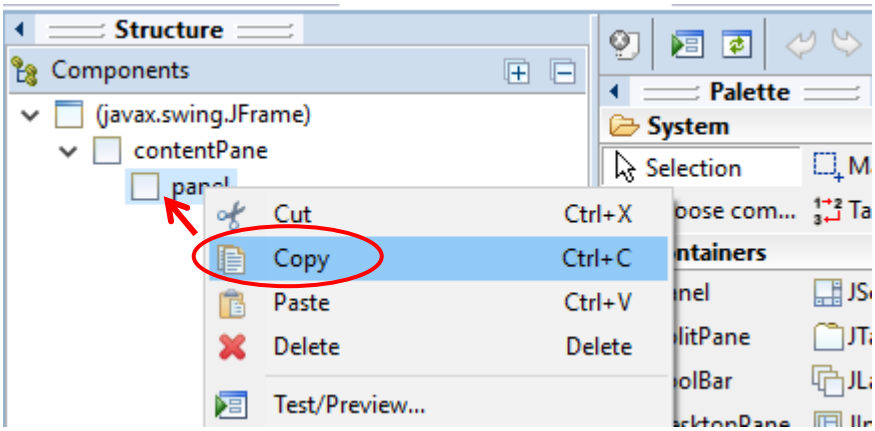
```
40          TemperaturePanel panel = new TemperaturePanel();
```

To correct this, we need to add a station ID as a parameter:

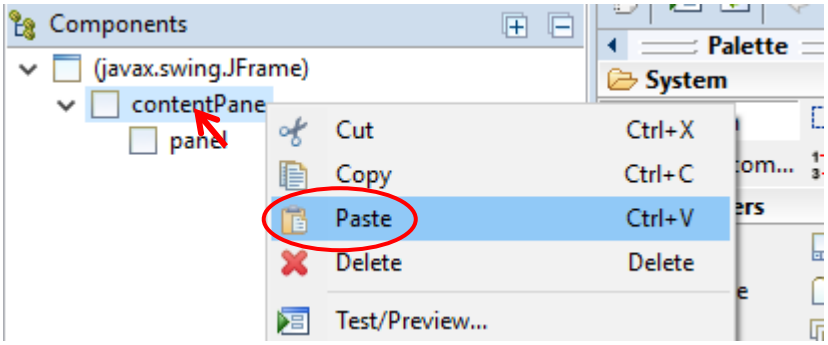
```
TemperaturePanel panel = new TemperaturePanel(6340);
```


The TemperaturePanel is now ready to display data of multiple stations. To make sure that all changes work, you must save all files in your project (*File > Save All)*.

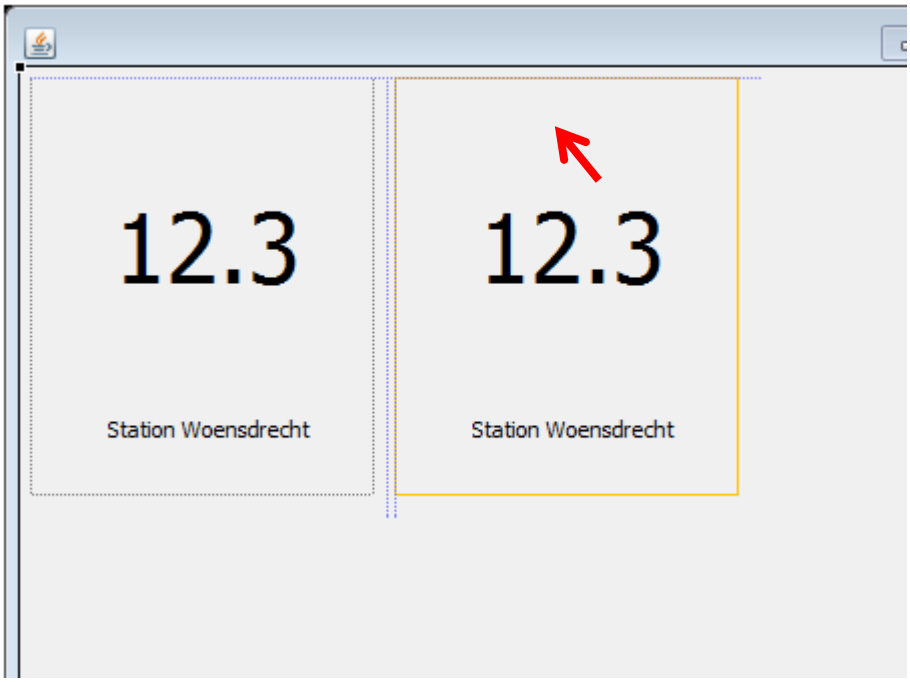
We can now make a copy of the panel by clicking it with the right mouse button and selecting *Copy*:



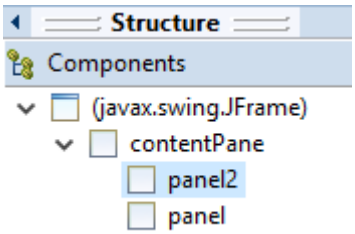
We paste a copy in the *contentPane*:



And then move the mouse to the location in the WindowBuilder, where we click on the location where we want to copy:



We change the name of the fresh copy to "panel2":



We have now created a *new object* of the TemperaturePanel class, also called a new *instance*.

Read more on creating new objects @ slides 7+8 of the presentation

Now change the id for *panel2* in the source code:

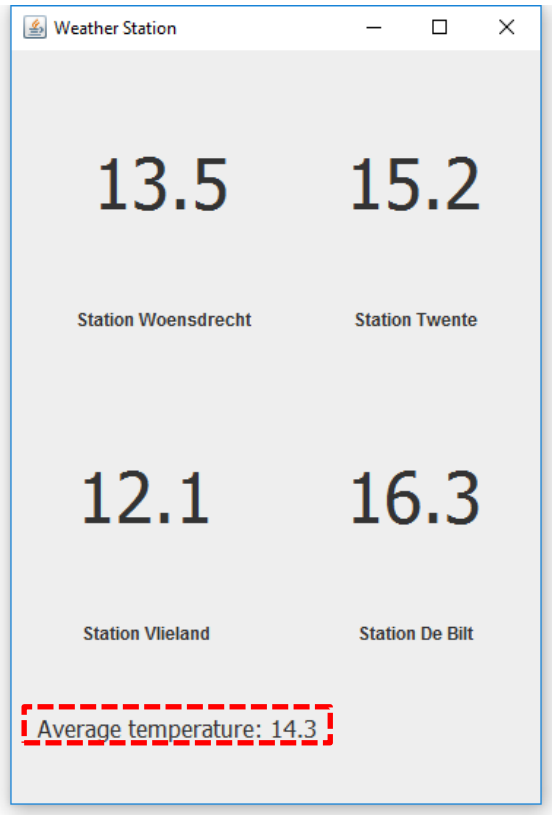
```
TemperaturePanel panel2 = new TemperaturePanel(6290);
```

The result:

You may now add 1 or 2 panels of other weather stations.

5. Calculate the average temperature and display this

To display the average temperature of all the stations in our application, add a label to the user interface. See the example on the right:



To calculate the average, we need the temperature as a number (of type double). The temperature as we use it now is text (a String). So we have to *convert* a String to a double. An example:

```
String text = "21.45";
double d = Double.parseDouble(text);
```

Add a method to TemperaturePanel
We can request the temperature as a number from each panel, if we add a new method to the TemperaturePanel class, which: **a)** requests the temperature from the weather station, and **b)** converts this into a number (double). You can design (think of) and add (write code) this method and add it to the TemperaturePanel class.

You have previously used a method that gets the temperature from a weather station (in step 3 of this assignment).

Once you have created the method, you can get the temperature from each panel in the Assignment3GUI class by calling this method for each panel. A draft for these method calls and calculation will look like this:

```
// get temperature of panel:
double t1 =

// get temperature of other panels:


// calculate average:
double average =
```

The name of the method you have created and the calculation have been shaded grey here. You can add this code at the end of the constructor, all the way at the bottom of the class.

After calculating the average, the value can be displayed in the label you have created for that purpose, by calling the *setText()* method of that label.
The outcome of the calculation is a number. The *setText()* method expects a String as parameter. You can place a number in the label by attaching the number to a String using the **+** operator:

```
// show calculated average in the label:
lblAverageTemperature.setText( "Average temperature: " + average );
```

The result of the calculation can be a number with many decimal places. Make sure you have an accurate presentation of the output, i.e. 1 or 2 decimal places. Use what you have learned about formatting output in the lecture.

More info @ slide 16 of the presentation

Finish

Add a title and comments to the application (as you learned in Assignment 1). Do this for both Java files. Don't forget to enter both your names and student numbers in the comment at the @author field.
The `WeatherStation.java` file is not yours, so you do not need to change the comments or author information there.

Before you have the assignment reviewed please check:

- Proper comments, names and student numbers in headers of Java files
- The average temperature is calculated and properly formatted

Check-off assignments

Have your assignments checked by the lecturer or assistant no later than at the next lecture.
That is the deadline, which is applicable for all assignments (this will not be repeated further).
The purpose of the check is that you demonstrate that you understand the subject matter. The assignment is graded as a pass (1) or a fail (0).
Assignments 2 through 7 count towards the grade (details have been provided in the lecture).

Extra challenge

Display the humidity in the TemperaturePanel. Add a label for that, and use method **`readValue()`** of the WeatherStation class to get the humidity like this:

```
String humidity = w.readValue("luchtvochtigheid");
```

The appendix shows how to setup a serial connection to a temperature & humidity sensor build with an Arduino and display the values from these in this application. If you have time left, you could start adding that.

Summary

On this third day you have written Java code and learned how to apply variables in expressions to run calculations.
In addition, you have learned the following.

- Using expressions to perform calculations.
- Calling and determining the right methods from existing classes.
- Using variables as parameters for methods.
- Coming up with and creating a method.

Appendix 1: display local weather data from a connected temperature sensor

Step 1: setup communication

Install the jSerialComm library
To be able to communicate via a serial connection we will add the jSerialComm library to the project. Download it as a .jar file here. Copy the .jar file in the Windows Explorer. In Eclipse, right-click the main folder ("Assignment3") and paste it into that folder:

Next, add the library to the build path: right click it, and select *Build Path* > *Add to Build Path*. Now you can use the library in your project.

Add code for the serial connection
First import the library by adding the following line at the top of **Assignment3GUI.java**:

```
import com.fazecast.jSerialComm.*;
```

Next, add the example code from this text file to the class **Assignment3GUI**: copy the code and paste it just before the very last closing bracket: **}**.

Call the **`initializeSerialPort()`** method at the end of the constructor. This is right after the spot where you show the calculated average in the label.

Next, find the lines with the class-variables used for communication:

```
// used for serial communication:
String readline;
SerialPort comPort;
String commPort = "COM14";
int baudrate = 115200;
```

Change the variable **`commPort`** to reflect the COM-port your Arduino uses (the number will be different). You can find the COM-port in the Arduino IDE: *Tools* > *Port*.

On a Mac, comports are looking different. It might be something like `/dev/tty.usbmodem*` or `/dev/tty.usbserial*`

There is a piece of code in comments (`"// get a list of available ports"`) which you might un-comment to see a list of ports.

Test
You can now run a sketch on your Arduino and should see the output of the sketch in the Console of Eclipse if you run this Application. Make sure:

- The Serial Monitor of the Arduino IDE is closed
- The Arduino is connected with the USB cable
- The proper COM port is used in the Java Application

Step 2: adjust TemperaturePanel class to display local temperature

To be able to display the temperature coming from the Arduino, we need to adjust the class TemperaturePanel.

First we add a new constructor to the class TemperaturePanel:

```
public TemperaturePanel() {
}
```

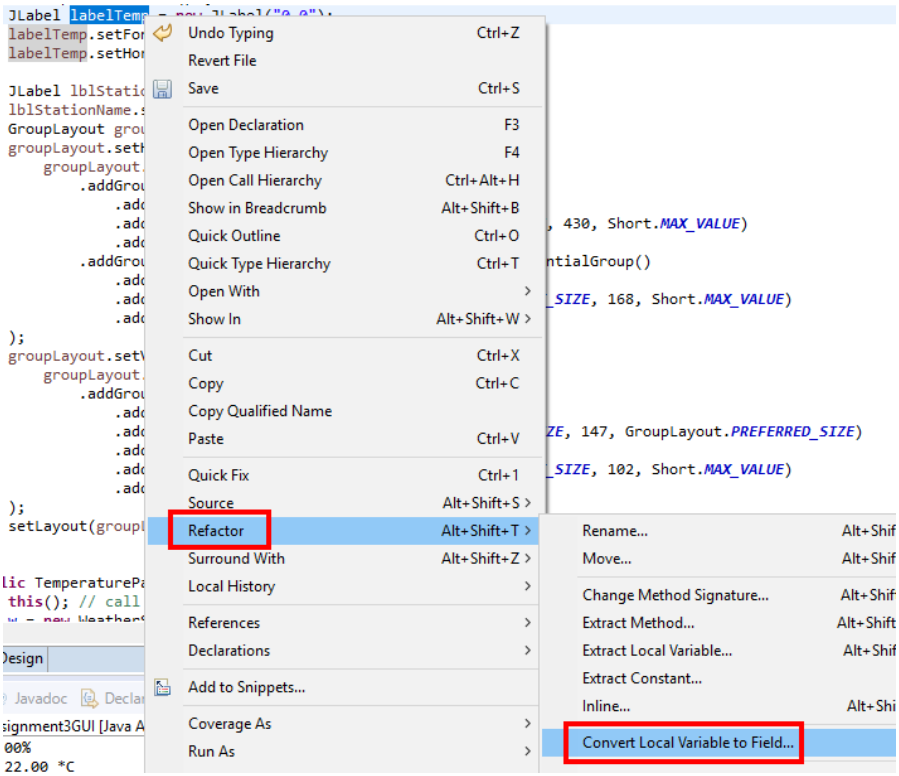
And we move all userinterface-related code to this constructor.

The original constructor now looks like this (or something similar):

```
public TemperaturePanel(int id) {
	this(); // call base constructor
	w = new WeatherStation(id);
	String temp = w.readTemperature();
	labelTemp.setText(temp);
	lblStationName.setText(w.readNameStation());
}
```

The first line of code, the call to `this()` will not be there, so add that to that constructor. It calls the other constructor (which now contains only the userinterface code).

It is likely that you will have some errors now, because the variables `labelTemp` and `lblStationName` are inaccessible, because they are in the other constructor. We will make class-variables of those. Find their definition in the other constructor, double click them to select them and right-click:



Choose *Refactor > Convert Local Variable to Field…* which will turn it into a class-variable. Do this for both variables.

The constructor with the userinterface-code should now looks like (there might be more code):

```
labelTemp = new JLabel("0.0");
labelTemp.setFont(new Font("Tahoma", Font.PLAIN, 48));
labelTemp.setHorizontalAlignment(SwingConstants.CENTER);

lblStationName = new JLabel("local");
```

You may change the text of the label for the StationName to "local".

Now add a setter (a method which can set a value) for the variable `labelTemp`:

```
public void setLabelTemp(String temperature) {
    labelTemp.setText(temperature);
}
```

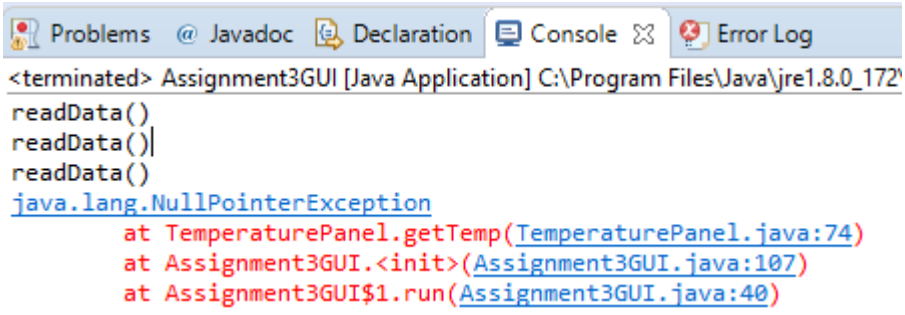
We will use this method to change the temperature displayed.
Save the file TemperaturePanel.java (CTRL+S).

Switch to the userinterface class (Assignment3GUI.java) and add the following code to the receive() method:

```
if (readline.startsWith("Temperature:")) {
  String[] words = readline.split(" "); // split the line into words
  double temp = Double.parseDouble(words[1]); // get the second word
  String formattedTemp = String.format(Locale.getDefault(), "%.1f", temp); // format it
  panel4.setLabelTemp(formattedTemp); // set the temperature-label in the panel
}
```

This might result in an error because `panel4` is not known (here). We assume the name of one of your temperature-panels is `panel4`, but you might have different names. Find one of the panels in the userinterface code in the constructor (somewhere above) and change that variable into a class-variable (*Refactor > Convert Local Variable to Field…).* Make sure the variable name matches with the one use in the method receive().

Running the application now might result in a crash:



This is because the `getTemp()` method tries to get the temperature from the WeatherStation w, but this has not been initialized for the TemperaturePanel which displays the local temperature (that panel is not connected to an online WeatherStation):

```
public double getTemp() {
    String temp = w.readTemperature();
    return Double.parseDouble(temp);
}
```

This will go wrong, because **w** is null

A fix for this can be:

```
public double getTemp() {
    String temp;
    if (w==null)
        temp = labelTemp.getText();
    else
        temp = w.readTemperature();
    return Double.parseDouble(temp);
}
```

Try if everything works as expected now.

If you have added a label to display the humidity in the TemperaturePanel (which was part of the extra challenge of the regular assignment), you can display the humidity you get via the serial connection also.
Start by adding a setter which can set the label for the humidity in the class TemperaturePanel, just like we did for the temperature label.

Also add a new if-statement to the receive() method of the Assignment3GUI class. Copy the original if-statement, and change the lines to match for humidity (you will have to understand the code to accomplish this).

Add a temperature-warning
If you would like to add a warning if the temperature is (too) high, you could add something like:

```
protected void receive(String line) {
    System.out.println(line);
    if (readline.startsWith("Temperature:")) {
        String[] words = readline.split(" ");
        double temp = Double.parseDouble(words[1]);
        if (temp>20) panel4.setBackground(Color.RED);
        String formattedTemp = String.format(Locale.getDefault(), "%.1f", temp);
        panel4.setLabelTemp(formattedTemp);
    }
}
```

Fix problem of average temperature
The average temperature shown is calculated only when the app starts, and at that moment, the temperature is not yet received from the Arduino. Therefore the average is not correct. To fix this, move the code that calculates the average to a separate method. This means the objects for the panels will have to become class variables. Call the new method in the receive() method.

