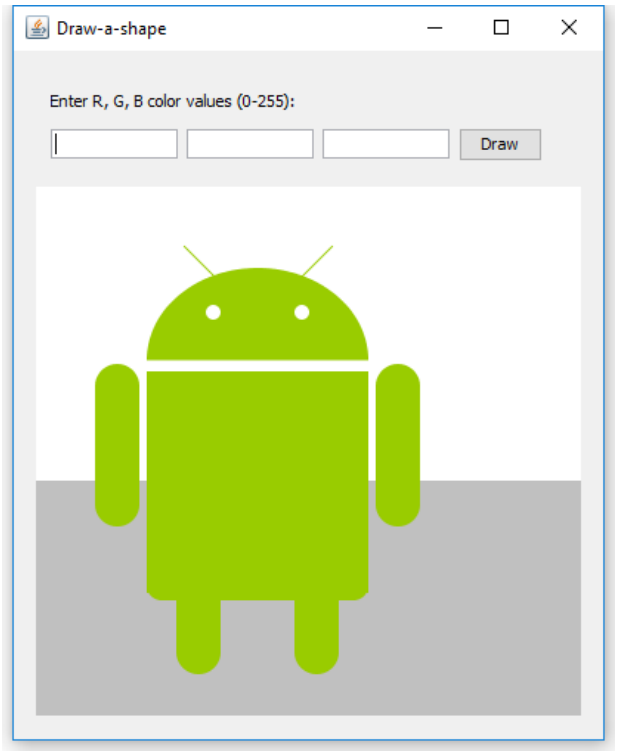


Assignment 2

Content Application Development Day 2
Lecture
The lecture covers the key language elements of the Java programming language. You are introduced to numerical data and variables and will learn to use these to create expressions.
Tutorial
In the tutorial you will use your knowledge of the language structure and language elements by writing simple programs, and you can write mathematical expressions.

At the end of day 2 you will be able to
- identify basic Java language components
- write a basic Java program
- describe the process of creating and running Java programs
- name and use different data types for numerical data
- write expressions
- use the standard mathematical library
- use methods

Reading between day 2 and day 3
Book:
Head First Java (K. Sierra & B. Bates): chapter 3, 12 (p 364-371).
Aan de slag met Java *[Getting Started with Java]* (Gertjan Laan): chapter 3 – 4.2.
Online:
javatpoint.com: Swing introduction, Graphics-in-swing, JPanel.

Extra exercise
Do the exercises and try the examples which come at the chapters indicated above, to increase your understanding of the subject matter. You can use this assignment as the basis for drawing graphics. How to practise examples and exercises from the book using Eclipse is explained in the appendix of Assignment 1.

Check-off assignments

You may complete assignments with 2 people. You don't have to, so you can also do them on your own. It is not allowed to complete assignments with 3 or more people and without any further checks this will automatically result in an unsatisfactory mark (‘failed’).

Have your assignments checked by the lecturer or assistant no later than at the next lecture (that is the deadline!). 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).

Drawing a simple shape

We are going to create an application that can draw a simple shape. This shape fills the screen, even when we resize the application. The color of the shape can be adjusted by entering the RGB values (three numbers) for the color components red, green and blue. These numbers can have values 0-255. Examples of how to draw shapes are given in the appendix.

Tutorial steps

1. Design and create the user interface
2. Write code for drawing
3. Set color
4. Read user input
5. Drawing with different colors
6. Draw the colored shape

These steps are explained in detail below.

1. Design and create the user interface

The user interface consists of two parts:

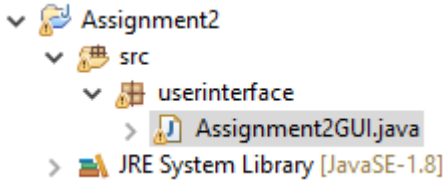
- A part for entering the RGB values
- A part for drawing

We can use a Panel for both parts, which is a component which holds other components, i.e. a container. You can also draw in a Panel.

Draw a sketch of the user interface that shows the division into Panels. A Panel contains three input fields, and an *Ok*-button (JButton) to execute the input of the values.

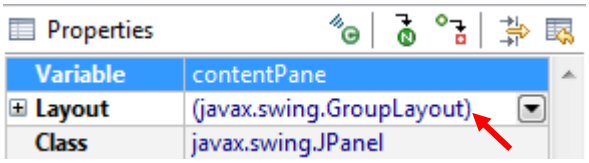
After you finish the design, you can start creating the user interface.

Start by creating a new project, via *File* > *New* > *Java Project*. Name the project "Assignment2".



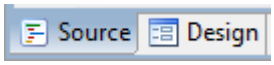
Add a JFrame and name it "Assignment2GUI" (shortcut: CTRL-N, you can find JFrame under *WindowBuilder* > *Swing Designer*).

Set the layout to "GroupLayout": in the WindowBuilder, click on contentPane, and select the Layout at Properties:



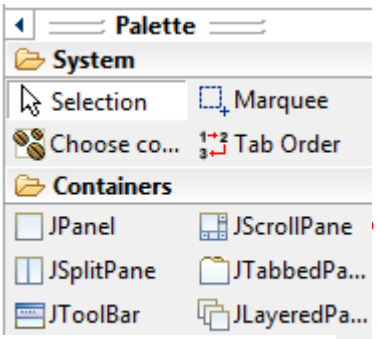
The steps to create a user interface are covered briefly. If you do not remember exactly how to do this, check Assignment 1.

You can switch between the source code Editor and the WindowBuilder (Design) with the tabs at the bottom:



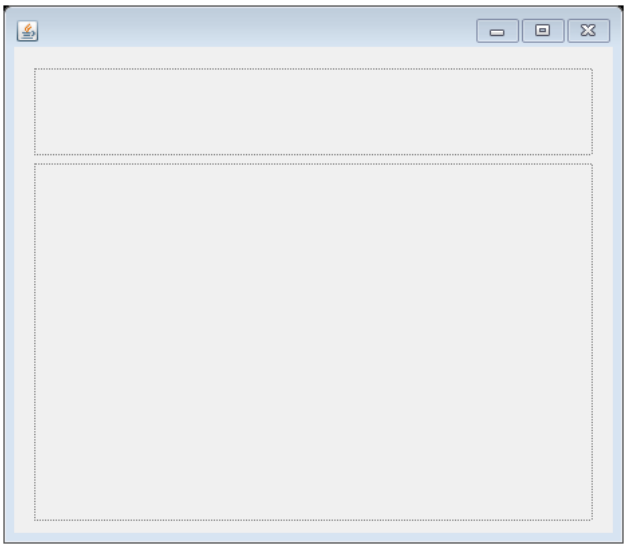
Tip: If the tabs do not appear, open the file by clicking it with the right mouse button: *Open With* > *WindowBuilder*…

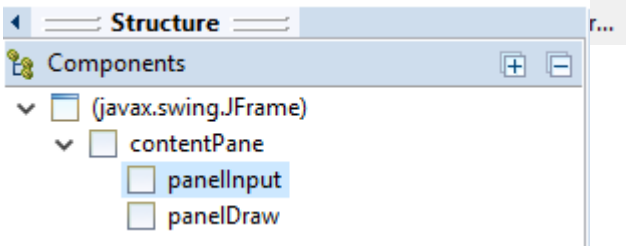
Now add two JPanels, one big Panel (the Panel used for drawing) and another smaller one, which will be used for input.
You can find JPanel under *Containers* in the *Palette*.



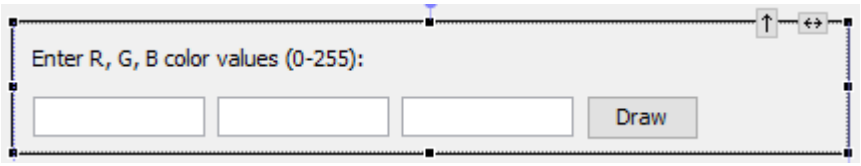
Give both Panels logical names (*Variable*field at *Properties*):

It will look this this, for example:

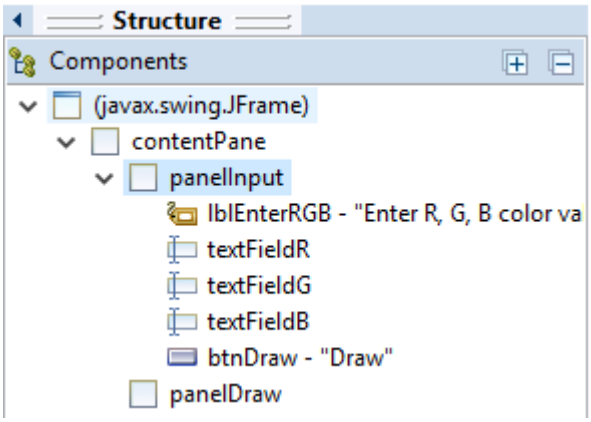




Set the layout of the smallest Panel to "GroupLayout". Now add a label (JLabel), three text fields (JTextField) and a button (JButton) to this Panel and make sure it looks similar to:



Give the components meaningful names. The structure should look similar as shown below. Also pay attention to the location of the components. The new components are in the Panel that we use for the input:



It is possible that your application does not properly fit in the Window. In that case, you can change the size of the window and various components to make it fit better. You can also look up the method call setBounds (...) in the code and change its parameters if necessary.

The user interface is now ready. Run your program using the Run button to check if everything works.

If you do not like the way the user interface elements are displayed, you can customise the style, using the Look-and-feel button at the top right of the WindowBuilder.

If you select *Windows* (located under *JRE*), it will look more like a Windows user interface.

Note: for this to work, you must first enable the "Apply chosen LookAndFeel …" option via *Window > Preferences*. This is located in the Preferences under *WindowBuilder, Swing, LookAndFeel*.

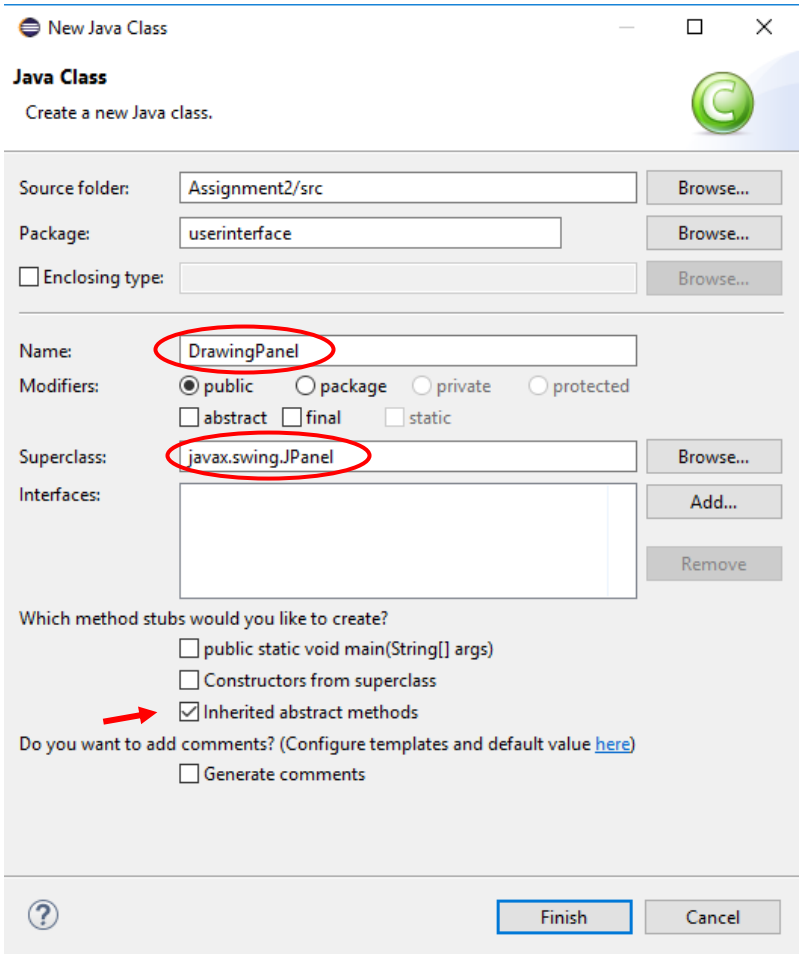
Make the draw-button the default button
To make the draw-button the default button which will be triggered when the ENTER key is pressed, add the following line of code at the end of the constructor *Assignment2GUI()*:

```
// make btnDraw the default button when ENTER is pressed:
getRootPane().setDefaultButton(btnDraw);
```

2. Write code for drawing

To be able to execute drawing commands we must create an extended version of the class JPanel.

Add a new class to the project (*File* > *New* > *Class*). Give this the name "DrawingPanel" and type "javax.swing.JPanel" at the field Superclass:



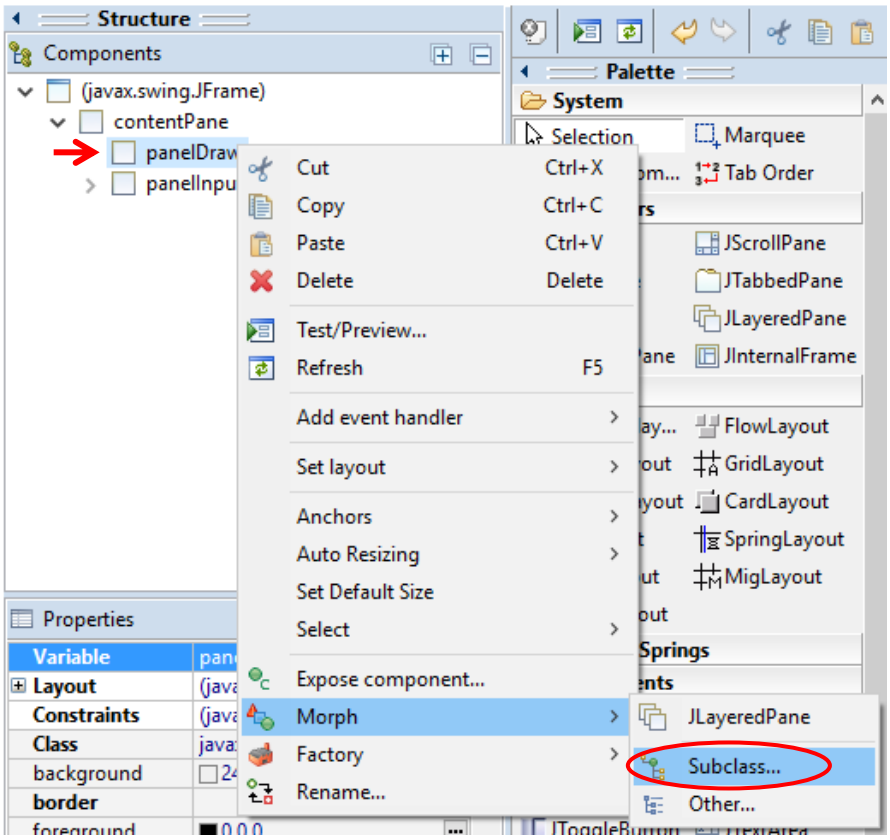
Pay attention to upper case and lower case characters when copying the values for *Name* and *Superclass* here.

For example, in javax.swing.JPanel you must write JPanel with a **capital** J and P!

At *Superclass*, you can use the Browse button as help, to avoid mistakes.

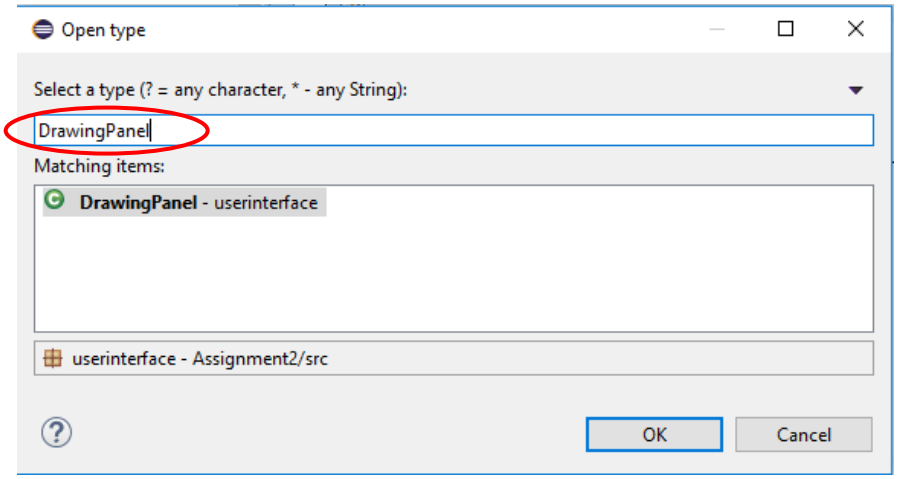
Before you Finish, make sure the option "Inherited abstract methods" is enabled and other options are disabled.

Next, we change panelDraw in the user interface into a *DrawingPanel*. Right-click the panel in the list of Components and select *Morph* > *Subclass*:



Before you start Morphing save DrawingPanel.java. Also check if it does not contain errors.

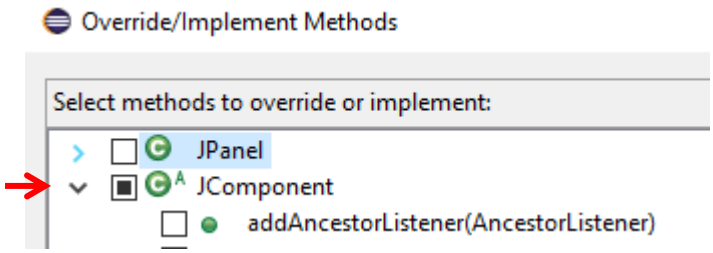
Type "DrawingPanel" in the text field and click on *Ok*:



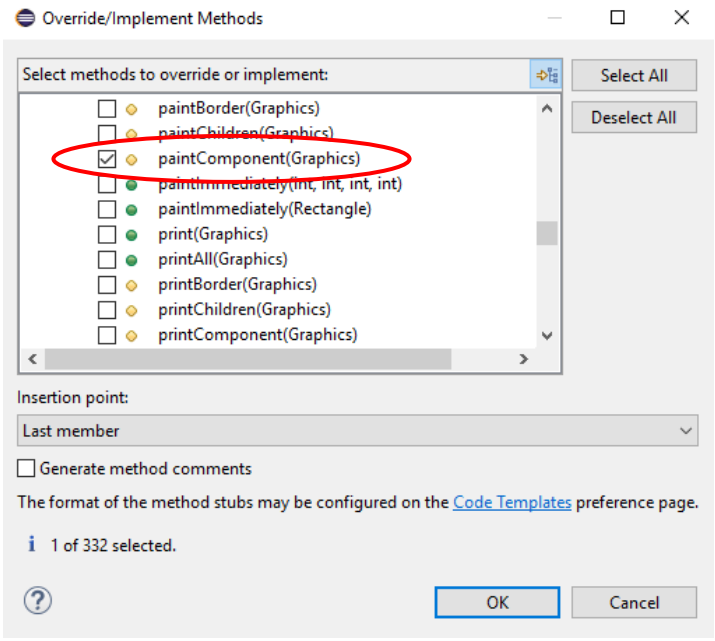
Now a drawing method can be added to the DrawingPanel class.

Select *DrawingPanel.java* in the Package Explorer. Select *Source* > *Override/Implement Methods…* from the menu.

In the list that appears, find JComponent and open that (click on the triangle in front JComponent):



Find the paintComponent() method of JComponent and select it. Then click on *OK*.



The code now looks like:

```
public class DrawingPanel extends JPanel {

    @Override
    protected void paintComponent(Graphics arg0) {
        // TODO Auto-generated method stub
        super.paintComponent(arg0);
    }

}
```

Change to ***g***

Change to ***g***

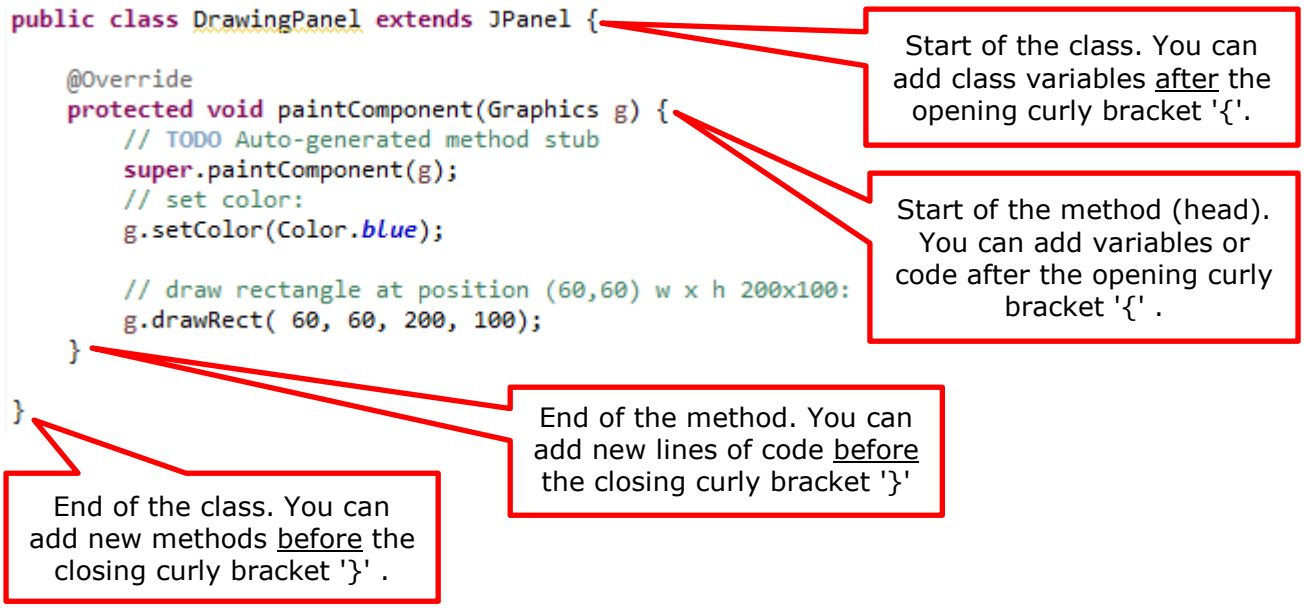
Examples of drawing with Java and the related techniques are explained in appendix 1 of this assignment. Read that first.

You can now add drawing methods (of the Graphics class) to the method just created. Use the examples in the appendix to draw a few sample shapes. In the examples, the parameter of the paintComponent() method has the name 'g'. It is useful to also change the name of your variable arg0 into 'g'. In that case, your code of the paintComponent() method is the same as that of the examples.

Run your program using the Run button to check if the shapes are drawn correctly.

Add variables and methods to a class

In the next section, methods and variables are added to a class. Variables are usually added at the beginning of the class (class variables) or at the beginning of a method. Methods are usually added at the end of the class. Where that is exactly, is explained below.



Curly *brackets* '{' '}' are sometimes called curly *braces* also… In Dutch, they are called 'accolades'.

3. Set color

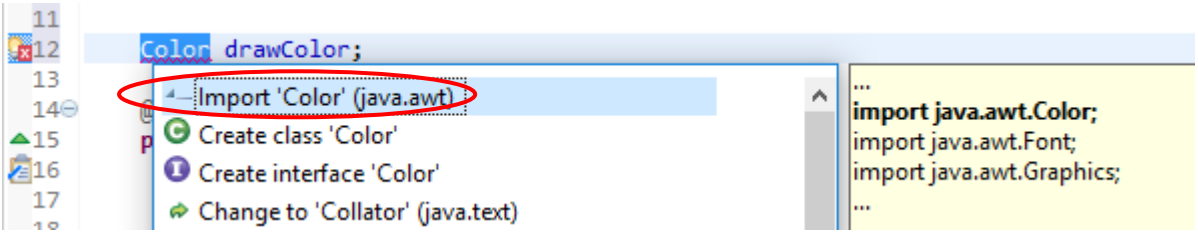
Drawing with Java and the related techniques are explained in appendix 1.

To draw with a certain color, the DrawingPanel class must be able to remember the font color.

First add a variable for the color to the DrawingPanel class. Do this at the top of the class, on a blank line after the opening bracket '{':

```
Color drawColor;
```

This variable is of the type `Color`. This is a type from a Java library. If this is the first time you are using it, an light bulb appears in the margin and red lines will appear underneath the unknow word. Click on the light bulb and double-click on "Import Color" to use `Color` from the `java.awt` library (we will *import* that library):



We are going to make the default color black by assigning a value to the *drawColor* variable. To do this, we change the line with the variable we just created into:

```
Color drawColor = Color.black; // default drawing color black
```

We now need to add a method which can set the color (assign a value to the variable drawColor). This method is given below. Add the code to the DrawingPanel class. **Note the location:** Add the code at the bottom of the class, on a blank line before the last closing bracket '}'. This method gets three variables *r*, *g* and *b* as parameters and uses those to create a new color (`new Color`) which is assigned to the class variable *drawColor* that we have just created.

```
public void setColor(int r, int g, int b) {
      drawColor = new Color(r % 256, g % 256, b % 256);
      repaint(); // draw again because the color has been changed.
}
```


4. Read user input

More on input of integers @ slide 20 of the presentation

Double-click on the Draw-button in the user interface to create an Event Handler. You also did this in the previous assignment.

To read the values entered from the text fields, you must call the *getText()* method of the text fields. The output (the result) of this method is a String. Therefore, we first declare a variable of type String and assign the outcome of the call of the *getText()* method to it. This is done as follows:

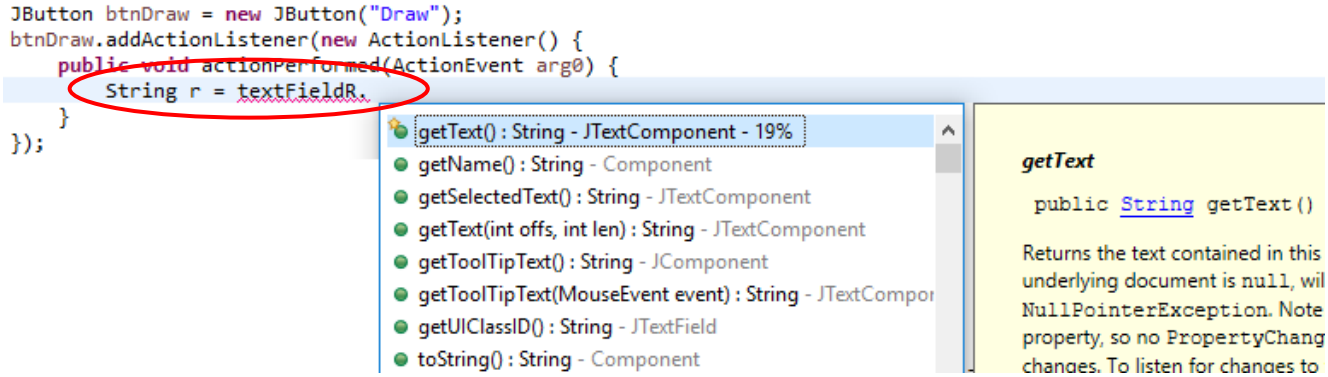
First type the declaration of the variable followed by an = sign to assign the value:

```
String r =
```

Behind this, type the variable name of the text field:

```
String r = textFieldR
```

If we type a dot immediately after this, we get a list of methods of the text field. Find the *getText()* method in that list and click on it to read the documentation of that method:



After double-clicking on the method, it is added to the code:

```
String r = textFieldR.getText()
```

Add a semi-colon ';' at the end to finish the line of code:

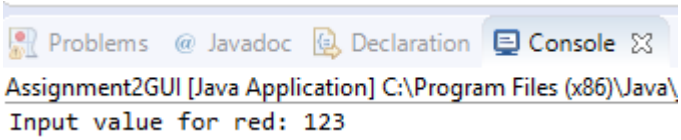
```
String r = textFieldR.getText();
```

On the next line, you can display the value entered in the Console, by calling the *println* method:

```
System.out.println("Input value for red: "+r);
```

When the program is running you can use this to check if entering numbers works. Run your program to check if entering a number in the field for red works (press the Draw-button). You should see this in the Console Window:

If the Console is not visible, you can show it via: *Window* > *Show View* > *Console*.



You must now repeat these steps for reading and displaying the values entered for the green (g) and blue (b) text fields and make sure that all three values are displayed in the Console.

5. Drawing with different colors

To draw using the color values entered we need to call the setColor() method of the DrawingPanel class in the Event Handler we have just created. This method expects three integers (r, g and b) as parameters. However, in the Event Handler the three values from the text field are read as String (text). Therefore, we must convert the String to an integer. To do this, add this code (at the end of the Event Handler):

More on input of integers @ slide 20 of the presentation

Tip: if you do not remember exactly where the Event Handler (of a button) is, you can always double-click on the button in the Window Builder.

```
// get integer-value from String r:
int rValue = Integer.parseInt(r);
```

Repeat this for the g and b values.

Now add the call of the *setColor()* method to the Event Handler:

```
panelDraw.setColor(rValue, gValue, bValue);
```

We assume that you have named the Panel for drawing "panelDraw" (at step 1 of this assignment).

You might notice that an error occurs here. Also if you do not experience an error, please read on:

Scope: region in code where a variable (or object) is valid.

The problem might arise, when building the user interface, the order of creation of components might differ. If button *btnDraw* is created before panel *panelDraw*, the variable *panelDraw* will not be in the same scope.
To solve this, we are going to define the variable at the right location, as a class variable. A class variable is also called attribute or property of the class. A class variable is always defined at the top of the class and accessible in all methods of the class.

Find the declaration of *panelDraw*. You can also use CTRL-F to search. The declaration looks like this:

```
DrawingPanel panelDraw = new DrawingPanel();
```

This is a declaration and an assignment in one line. The assignment is the part after the = sign (assignment operator). We are going to split the declaration and the assignment. The declaration is placed at the top of the class, where it becomes a property (or class-variable) of the class. Below, the declaration is shown in red and the assignment in blue.

```
DrawingPanel panelDraw = new DrawingPanel();
```

Copy the declaration to the top of the class:

```
public class Assignment2GUI extends JFrame {

    DrawingPanel panelDraw;
```

At the original location, delete the first word so only the assignment (the blue part) remains:

```
panelDraw = new DrawingPanel();
```

If you did get an error message at the call of the *setColor()* method, that should have disappeared now, because the panelDraw variable is now valid in the entire class, instead of only in (part of) the method: it has become a class-variable.

6. Draw the colored shape

To make sure the selected color will actually be used for drawing, we need to set the drawing color in the *paintComponent()* method in **DrawingPanel.java**. If you have used an example from the appendix, you have already set the color by calling the setColor() method. This call must now use the color variable *drawColor* as a parameter. Find the call for the setColor() method and adjust this as follows:

```
// set color:
g.setColor(drawColor);
```

Check if previously added test shapes are now drawn using the specified color (enter values for r, g and b and click on the button).

You now have to draw a more complex shape. Some possibilities: a mushroom, a traffic light or a pie. The shape must consist of at least two different basic shapes.

Use your own creativity to draw the shape you have chosen. Remember that you can get help to find out what kind of drawing methods there are. You can get help in different ways:

1. In Eclipse: in the *paintComponent()* method, enter a '*g*' (the Graphics-object), followed by a dot, after which all methods of the Graphics class are listed; click on one to see what it is. Drawing methods start with "draw…" or "fill…"
2. Google: search for terms like "java draw figure" or "java draw nice icon" and so on. You might like for instance the Andoid-icon.
3. The API description of the Graphics class

Delete the test shapes you made earlier using the drawing methods (or place them in comments).

If you want the dimensions of the shape to be dependent on the size of the Panel in which is drawn, you can do that as follows: You can retrieve the width and the height using methods getWidth()and getHeight().

To draw a rectangle that creates a horizon effect in the drawing, for example (see screenshot on the first page of this assignment), you can use this code:

```
int h = getHeight(), w = getWidth();
int horizon = h/2-20;
g.setColor(Color.lightGray);
g.fillRect(0, h-horizon, w, horizon);
```

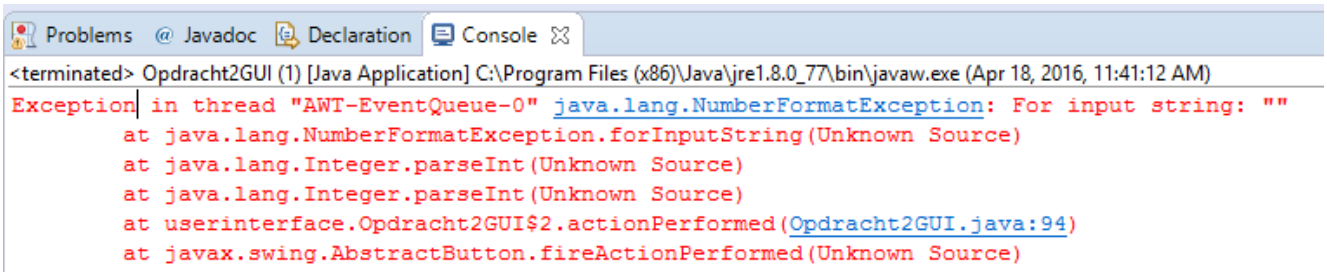
Finish

Give your application a title (property of the JFrame) and add comments (as you learned in assignment 1). Do this for both Java files. Don't forget to enter your name(s) in the comment at the @author field.

Extra challenge

This is not part of the assignment, but you may add this if you like.

The application is not yet entirely fool proof. What happens when we start the application and immediately click on the Draw button? Check the output in the Console. A red error message probably appears here. If you scroll up, you will see that this is an "Exception": something Java is unable to run.



In one of the lines at the top you will also see a reference to a line number in a Java file. That's where the error occurs. It is probably one of the lines with a call of the parseInt() method. This method cannot handle a parameter that is not an integer. In this case, it is an empty String.

To solve this problem, we could check if there is an integer in the relevant field. And if not, we place a zero (0) in the field.

We can use r.matches("\\d+") to check if the String read in variable r only contains digits (integers). This expression is *true* if the String consists of integers. Using the Boolean not-operator, an **!**, we can then build this if-statement:

```
if (!r.matches("\\d+")) { r="0"; textFieldR.setText(r); }
```

Insert this line at the location after the call to getText() (after reading the String). Repeat for the g and b values.

Test the application.

Summary

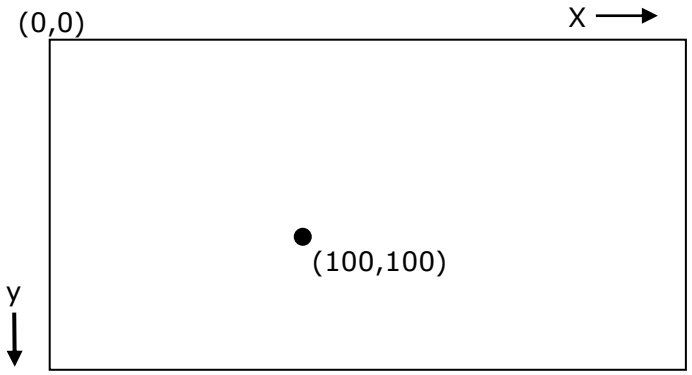
On this second day you have written Java code and learned how to apply variables in expressions to perform calculations.

You have also learned the following:

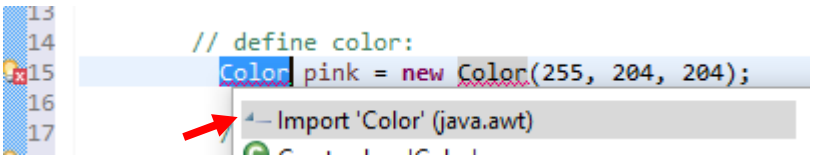
- Create a user interface.
- Read and use values in a user interface.
- Use drawing methods.
- Use of variables in expressions.
- Apply mathematical functions.
- Find information using the online help function.

Appendix 1: drawing with Java graphics

To start drawing inside a panel, first setup a DrawingPanel as described in step 2 of this assignment. Next, add a paintComponent method and add drawing commands to it. The origin in a panel is the upper left corner:



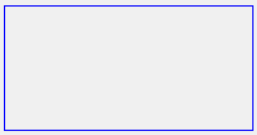
The first time you use a new class, like **Color**, you might get a warning bulb in the margin:



To solve, just click the bulb and double-click the first option to Import that class from the library (in this case **java.awt**).

Draw a rectangle

```
public void paintComponent(Graphics g) {

  // set color:
  g.setColor(Color.blue);

  // draw rectangle at position (60,60) w x h 200x100:
  g.drawRect( 60, 60, 200, 100);

}
```



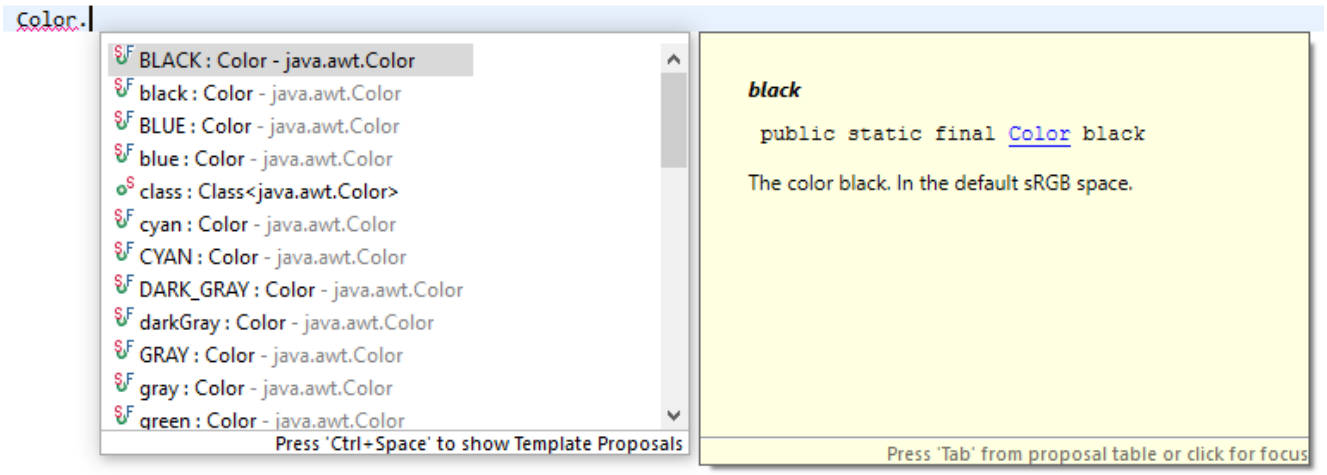
Draw text

```
public void paintComponent(Graphics g) {

  // set the font:
  g.setFont(new Font("TimesRoman", Font.PLAIN, 20));

  // set color:
  g.setColor(Color.red);

  // draw a String at position (10,20):
  g.drawString("paint some text", 10, 20);

}
```

Can you:
Change the size of the font?
Change the position of the text?

Use Color attributes: Help-while-you-type

To find out what colors are available, just type Color followed by a dot (.):



Mix your own RGB-color

Create a new color with:

```
new Color(255, 204, 204)
```

In the next example, we draw an oval using this color.

Draw an oval

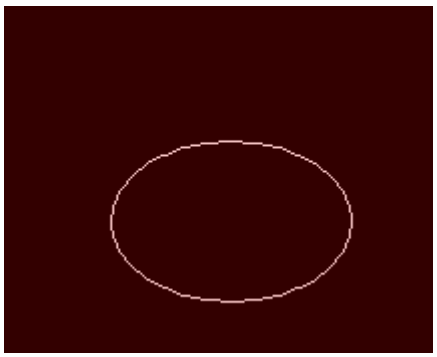
```
public void paintComponent(Graphics g) {

  // define color:
  Color pink = new Color(255, 204, 204);

  // set color:
  g.setColor( pink );

  // draw oval at position (100,100) width 120, height 80:
  g.drawOval(100, 100, 120, 80);
}
```

Can you:
Change the oval into a circle?

Change the background

To change the background color of a JPanel, simply call the *setBackground* method:

```
setBackground( new Color(51,0,0) );
```



Improve quality of drawing

You might notice that sometimes the quality of drawings is poor:

This can be improved by instructing the graphics renderer to improve the rendering quality:

```
// improve rendering by turning on Antialiasing:
Graphics2D g2 = (Graphics2D)g;
g2.setRenderingHints(new RenderingHints(
RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON ));
```

Resulting in:

Add this code before the drawing commands.
More information.

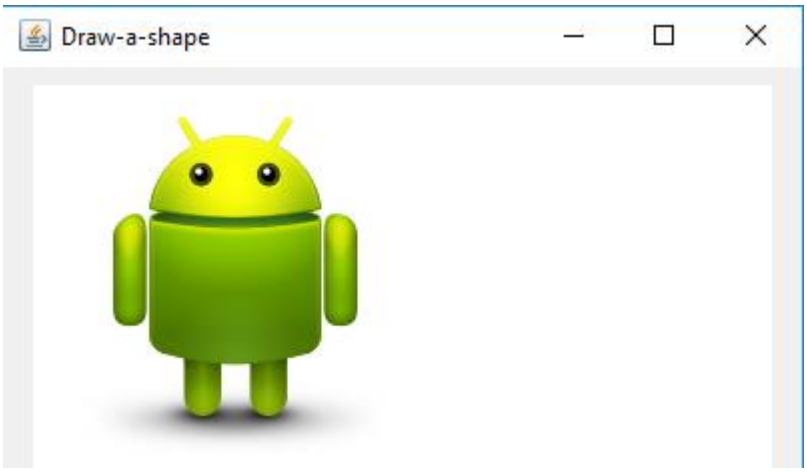
Draw a filled shape

The previous shapes drawn by draw… methods each have a counterpart to draw the shape filled: those methods start with fill…
So to draw a filled rectangle we use the method *fillRectangle*:

```
// draw filled rectangle at position (60,60) w x h 200x100:
g.fillRect( 60, 60, 200, 100);
```

Draw an image

The example on the next page draws an image (in a panel). The image is first read in the constructor of the Panel, which is called only once. This prevents the image from being read from the file each time it is (re)drawn.



```
BufferedImage image; // class variable

public DrawingPanel() { // constructor
  super();
  File file = new File("image.png"); // read image from project-folder
  try {
       image = ImageIO.read(file);
  } catch (Exception e) {
       System.err.println("Unable to read file");
       return;
  }
}

@Override
protected void paintComponent(Graphics g) {
  super.paintComponent(g);

  setBackground(Color.white);

  if (image != null) // if image was loaded
       g.drawImage(image, 0, 0, this);
}
```

