# Crash Course Java
# Colorclicker

louisV, mareinK, koenD

2015

## 1 Introduction

### 1.1 Swing

During this course, we will be using Swing to create our graphical user interface (GUI). There many good tutorials for Swing on the web. Just Google for `swing tutorial`. The ones provided by `Oracle` are excellent for further reading.
As the Swing package is very popular, you should be able to complete all courses in the crash course on your own, with a little help of Google. Part of the course is learning how to find information on the internet when you are stuck, so don't hesitate to search for examples or additional information!

## 2 Our first Swing application

### 2.1 Colorclicker

During this tutorial you will create a simple Java application. It will contain a button that you can use to change the color of a rectangle. The tutorial will teach you how to write Java classes, how to create a program window, and how to create interaction between the user and the program, as well as interaction between classes within the program.

### 2.2 Main class

Just like in C++, the first thing that is started by Java is the `main` function. In Java, all code is part of a class. So, we must create a class to contain the function. Create a new Java Project and then a new Class (`File` → `New...`). Name the class `Main`. You should see an empty Main class.

```
public class Main {

}
```

Now type the following 'method' (a function inside a class) in the Main class:

```
public static void main(String... args) {

}
```

- `public` means the method can be accessed from outside the class; `private` would mean the method could only be called from within the class.
- `static` means the class does not need to be instantiated (made into an object) in order to call the method: it can be called directly on the class. From a static method, you can only access other static methods and variables.

- `String... args` is a variable amount of arguments that is passed to the `main` method. We won't be using these arguments in our program, but we need to accept them in our `main` method.

For now, we will leave our `main` method empty and work on our program window.

## 2.3 Window class

Create another class and name it `Window`. To use this class as a Swing window frame, we need to give it certain properties that Swing provides. We can do this by 'extending' Swing's `JFrame` class. When one class extends another class, the extending class becomes a subclass, inheriting the methods and variables of the superclass. All of this functionality can then be used within the subclass.
To have `Window` extend `JFrame`, we change the class declaration to the following:

```
public class Window extends JFrame {

}
```

At this point, you may notice that Eclipse displays a bulb with a red cross at the start of the line, as well as a wavy red line under `JFrame`. This is because `JFrame` exists in a library or 'package' that we must tell Java to search. We do this by typing the following at the top of the file:

```
import javax.swing.JFrame;
```

In the body of the `Window` class, write the following constructor. This method will be called when the class is instantiated.

```
public Window() {
   // 'super' calls a function inherited from the parent class (JFrame)
   super();
   super.setTitle("Callbacks");
   super.setSize(new Dimension(420, 350));
   // Make sure the window appears in the middle of your screen
   super.setLocationRelativeTo(null);
   // Determines what should happen when the frame is closed
   super.setDefaultCloseOperation(EXIT\_ON\_CLOSE);
   // Chooses a certain layout type for the elements in this frame
   super.getContentPane().setLayout(new BorderLayout());

   // TODO: add elements to the content pane

   // Set the window to visible! Yup... This is necessary
   super.setVisible(true);
}
```

After adding this, you will need to import more library classes. You can do this easily by pressing Ctrl+Shift+O. This should automatically add import statements for `java.awt.BorderLayout` and `java.awt.Dimension`.
Since we want our program to show the window when it starts, we will tell the `main` method to do so. Write the following code in the `main` method.

```
SwingUtilities.invokeLater(new Runnable() {
   public void run() {
      new Window();
   }
});
```

The workings of this code will not be completely explained at this moment. Notice in particular the instantiation of our `Window` class on the third line of this code snippet. Make sure to add the necessary imports.
We can now run our program by pressing the green arrow button, or right clicking the `Main` class and choosing `Run As...` → `Java Application`. You should see an empty grey window titled 'Callbacks'.

### 2.4 ButtonPanel class

In order for there to be interaction with the user, we will create a button. We will be using this button to change the color of a rectangle. In Swing, all elements (including buttons) live within panels. A panel is a component that can contain other Swing components such as buttons, text fields and other panels. To create a panel for our button, we will be extending another Swing class: `JPanel`. Create a new class called `ButtonPanel` and make it extend `JPanel`. Write the constructor for `ButtonPanel` and add the body as follows:

```
public class ButtonPanel extends JPanel {
   public ButtonPanel() {
      super();
      // Add a button to the panel. The argument to the JButton constructor
      // will become the text on the button.
      JButton b = new JButton("Change color!");
      this.add(b);
   }
}
```

As you can see, we create a `JButton`, setting its face text using its constructor. We then add the button to the `ButtonPanel`: the button is now part of the panel. However, the button will not show up in our window, as we have not added the panel to our window yet. We will do this later.

### 2.5 RectPanel class

The final class we will need is a panel to contain our rectangle. Again, this class will extend `JPanel`. Since this class will control our rectangle, it should know the color of the rectangle. For this purpose, we will create a member variable in the class, as follows:

```
public class RectPanel extends JPanel {
   private Color color;
}
```

We have made a private member variable. Remember, this means it can only be accessed from within this class. But we want to change the color when the button is pressed, and the button is not inside this class. **Therefore, you must think of a way to allow changing the variable from the outside, while keeping the variable private.**
We will want to set a default starting color for our rectangle. We can do this in the constructor: the method that is called when `RectPanel` is created. Create a constructor and add the following lines:

```
super();
this.color = Color.BLACK;
```

Now to create our rectangle. We will tell Swing to draw a rectangle each time it shows the panel. Swing asks every `JPanel` (including subclasses such as our `RectPanel`) how it should be drawn using a method called `paintComponent` that exists in `JPanel`. To change how our own panel answers this call, we will 'override' the `paintComponent`. To override a method means to redefine its functionality. Add the following under the `RectPanel` constructor:

```
@Override
public void paintComponent(Graphics g) {
   super.paintComponent(g);
   g.setColor(this.color);
   g.fillRect(100, 30, 200, 200);
}
```

This uses the color from our class s`this.color` to set the brush color of the graphics object, which will then paint our rectangle.

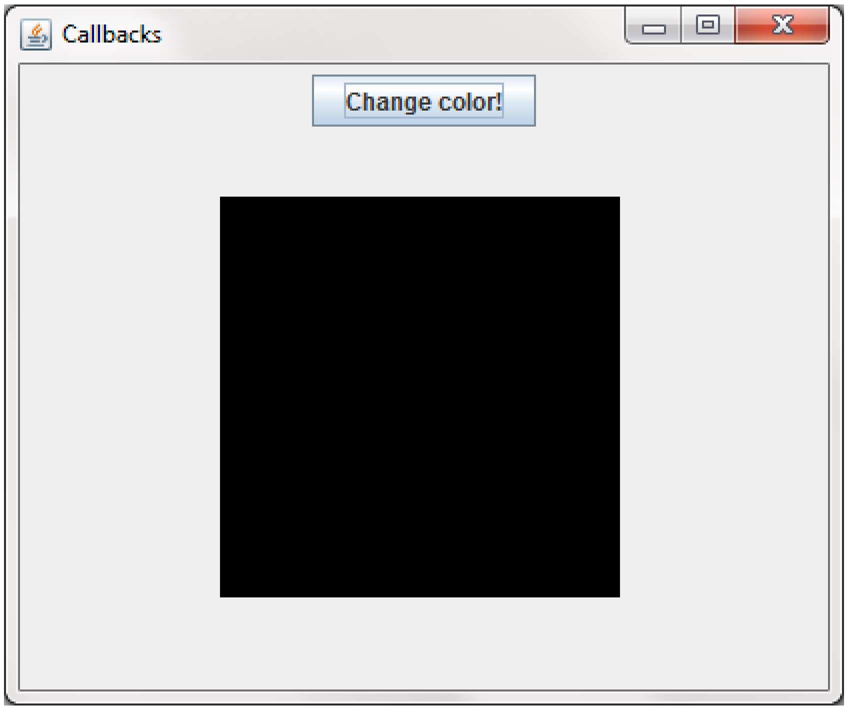
## 2.6 Bringing it all together

We will now add our homemade button and rectangle panels to the window. We will start by instantiating the panels in our `Window` constructor, and then add them to the window's content pane. Type the following at the 'TODO' comment:

```
        RectPanel rp = new RectPanel();
        ButtonPanel bp = new ButtonPanel();

        // Places the RectPanel in the center of the frame
        super.getContentPane().add(rp, BorderLayout.CENTER);
        // Places the ButtonPanel in the top of the frame
        super.getContentPane().add(bp, BorderLayout.NORTH);
```

If you run the program now, you should see the following:



### 2.7 ActionListener

We're now going to implement the interaction with the user; i.e. actually making the rectangle change color when the user presses the button. To achieve this, we need to work with 'events'. Every time the button is clicked, it 'fires' an event which calls all objects that are 'listening' for this event. The objects can then execute some code specific for this event. This is called a 'callback'. This works by adding an implementation of the `ActionListener` interface to the button. The `ActionListener` interface defines one function: `void actionPerformed(ActionEvent e)`. Implementing a class works much like extending: the subclass inherits the functionality of the superclass.
Create a new class `ButtonHandler` and make it implement the `ActionListener` interface:

```
public class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // TODO: add code here that will
        // be ran when the button is clicked
    }
}
```

Now add a new instance of this actionlistener to the button you created in the ButtonPanel:

```
b.addActionListener(new ButtonHandler());
```

You now need to add more code to the `actionPerformed` method (and maybe to the classes?) to achieve the needed functionality. You're free to add methods and variables as you see fit!

## 3 Exercise

Add the needed functionality such that the color of the rectangle is changed to a random color each time that the button is clicked.

## 3.1 Tip

It might not be immediately obvious how to solve this problem, but there is a lot of Java documentation out there. As always, Google is your friend. Search phrases such as ”java rectangle color”, ”java list” or ”java random element from list” can get you a long way.

## 3.2 Extra credit 1

If you're up for a challenge, try implementing another actionlistener that changes the text of the button to the current rectangle color each time you click on it!

## 3.3 Extra credit 2

If you're still eager to keep going, try adding more buttons to the window that, when clicked, change other things in the window (size of the rectangle, text on the button, add other shapes...). Be creative!