Victoria University of Wellington
School of Mathematics, Statistics and Computer Science

SWEN221: Software Development

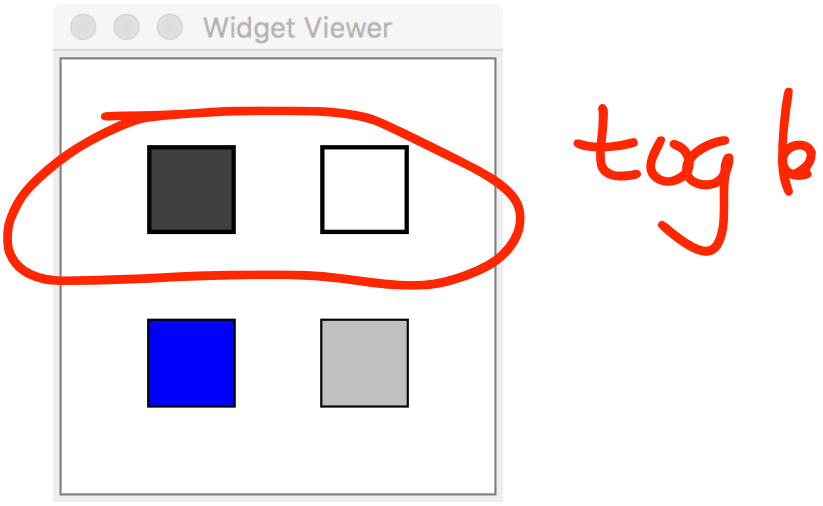
Lab Handout

The purpose of this Lab is to give you hands on practice using Java reflection. Before the end of the lab, you should submit your solutions via the *online submission system* which will automatically mark it. You may submit as many times as you like in order to improve your mark and the final deadline will be Friday @ 23:59.

Useful information about Java reflection can be found in the tutorial from
https://docs.oracle.com/javase/tutorial/reflect/.

1 A Widget System

The program provided with this lab implements a very primitive *widget system* which is inspired by modern windowing systems. In this system *widgets* occupy areas on the screen and can respond to mouse clicks. Widgets have internal state which they can use to control how they look. The following illustrates a simple widget system:



Here we see a very simple widget systems constructed from *five* widgets. The four visible squares each constitute one widget, whilst the *background* itself constitutes another widget. The two widgets on the top row are *toggles* — when clicked on, they will change to either the *on* or *off* state. The two widgets on the bottom row are *counters* — when clicked on, they will increment an internal counter and set their counter according to this.

2 What to do

The key challenge in this lab lies in interacting with widgets through reflection. In particular, *you will be unable to see the source code for one of the widgets*. This means that reflection provides the only way in which you can interact with this widget. The *Inspector* provides the skeleton for a suite of methods for creating widgets and reading/writing their attributes. Some notes:

- `Inspector.newWidget(String name, Rectangle dimensions)`. This instantiates a new instance of the class given by `name`. To do this, you will need to first find the appropriate constructor in the widget's class, and then instantiate a new object using `Constructor.newInstance()`.
- `Inspector.getAttribute(Widget widget, String name)`. This reads the value of field `name` in the given `widget`. To do this, it must call the corresponding "getter" method. For example, for a field called `color` the corresponding getter would be `getColor()`.
- `Inspector.setAttribute(Widget widget, String name, Object value)`. This writes a given `value` into field `name` in the given `widget`. To do this, it must call the corresponding "setter" method. For example, for a field named `color` of type `Color` the corresponding setter would be `setColor(Color)`.

Your goal in this lab is to complete the above methods in the Inspector class.

Submission

Your lab solution should be submitted electronically via the *online submission system*, linked from the course homepage. The required files are:

```
swen221/lab8/core/Canvas.java
swen221/lab8/core/Inspector.java
swen221/lab8/core/Widget.java
swen221/lab8/Main.java
swen221/lab8/tests/WidgetTests.java
swen221/lab8/util/AbstractWidget.java
swen221/lab8/util/Point.java
swen221/lab8/util/Rectangle.java
swen221/lab8/views/WidgetViewer.java
swen221/lab8/widgets/Background.java
swen221/lab8/widgets/Counter.java
swen221/lab8/widgets/Toggle.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All JUnit test files supplied for the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. This does not prohibit you from adding new tests, as you can still create additional JUnit test files. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.