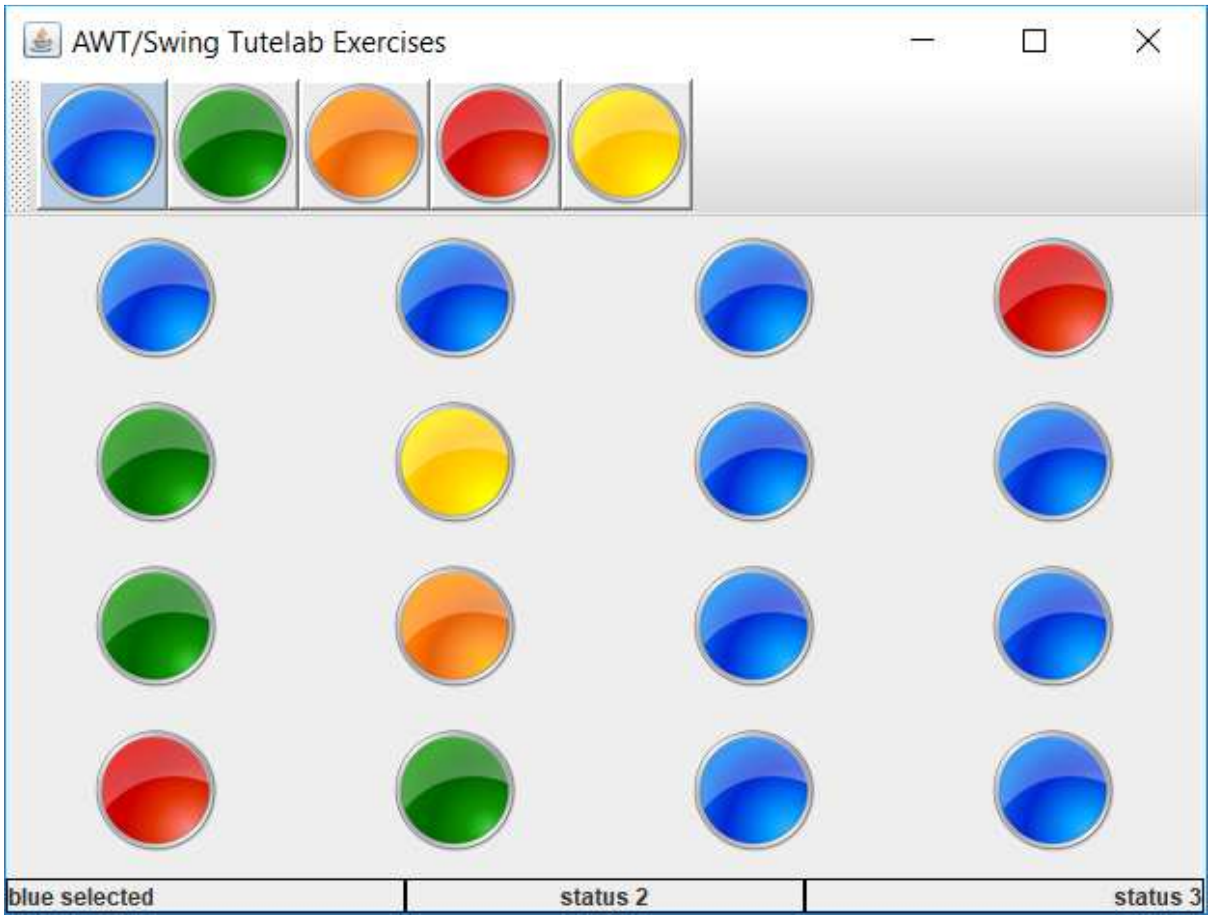
Software Architecture: Design and Implementation

TuteLab 8 – AWT/Swing and Patterns (MVC and Observer)



Last week you were asked to implement the application behavior shown in the screenshot above, whereby mousing over the coloured circles changes them to the currently selected colour.

This week you are going to continue working on this example but with an emphasis on code structure and the application of basic design patterns to achieve cohesion with known and controlled coupling.

The main goal is to implement the *MVC* pattern and *Observer* patterns in your code.

At the basic level you should have three packages *model*, *view* and *controller*.

The *view* package will contain all of the UI functionality (AWT/Swing classes which as previously specified should comprise at a minimum separate classes for the frame, toolbar, grid of circles and status bar). Ideally, you already have these or something similar from last week.

The *controller* package should contain any listeners and at a minimum must have at least one mouse listener to implement the colour change functionality.

The *model* package is an addition from last week and will contain at least one class with a supporting interface that is responsible for maintaining the state of the coloured circles. In other words, it is not enough to just graphically change the circle colour and forget about it, rather you need to store this state in the model. In effect, this is what we would call a *view-model* since it actually contains view state but the principles of implementation are the same for any other MVC implementation with a more business rule or data oriented model.

IMPLEMENTATION HINTS

One of the first challenges you will have is how do you get references to the various classes from each other. i.e. how do the views access the model so that they can draw the appropriate colour state, or how does the controller access the model so that it can change the state in response to a mouse event. A simple approach is to pass parameters via constructors.

Next you will consider how you represent the various colour states and map them to the implementing icons. Enums are one possibility here and a separate “factory” class for retrieving icons would aid cohesion.

Finally, you are to implement the observer pattern using the `java.util.Observable` class and `java.util.Observer` interface. The individual labels that represent the coloured circles in the grid should implement the `Observer` interface whereby changing the colour of the circle involves first changing

the state in the model (which is `Observable`) which automatically notifies the relevant label (view) so that it can redraw itself automatically based on changed model state.

You will also need some form of notification for the status bar message and can also use observer/observable here but may wish to investigate `PropertyChangeListeners` instead (see methods `addPropertyChangeListener(…)` in `java.awt.Container`)