Swing Worksheet 2012-2013

Matthew Chalmers

This worksheet introduces the Swing GUI toolkit in a way that requires you to work independently. This introduction is a sequence of selected 'official' online lessons and descriptions from Oracle's web site. They often call such lessons 'trails'. In them you will often see example code. Instead of you just reading the code on the web page, I recommend that you download the code, read through it in NetBeans or Eclipse, and then run it there. See how the code's input and output relate to Swing structures and their description in the lesson.

1. Introduction to Swing

First, go through Oracle's own introduction to the Swing toolkit: *Getting Started With Swing*. This gives an overview of Swing, and gives basic instructions on how to set it up and then run a basic Swing 'hello world' application from the command line.

You may prefer to use Eclipse, but Oracle uses NetBeans for its examples. So, step on to *Learning Swing with the Netbeans IDE,* and work through its CelsiusConverter example.

2. Swing Components

Take a brief look at *Using Swing Components*. This is a large section of the Java tutorials, but this course will only look at selected parts of it.

Skim through the list of components on the *How To Use Various Components* page. Don't go into detail of every one right now, unless you want to, but note just how many different tools and widgets are available for you to use in this toolkit. This page is a useful reference for later.

Note the top-level containers: at least one of these components must be present in any Swing application. Go in detail through *Using Top–Level Containers*.

Go in detail through the *How To Use Models* trail, which sets out the general principles of Swing models. Note that the word 'model' is not used in quite the same way as in the Model-View-Controller UIMS pattern, although it does relate to a data structure that underlies a visual component, and gives a degree of separation between storage and graphical representation. (MVC goes further in this regard.) Swing models are important to know about because "most Swing components have models. A button (JButton), for example, has a model (a ButtonModel object) that stores the button's state — what its keyboard mnemonic is, whether it's enabled, selected, or pressed, and so on. Some components have multiple models. A list (JList), for example, uses a ListModel to hold the list's contents, and a ListSelectionModel to track the list's current selection. [Models] give you flexibility in determining how data is stored and retrieved. For example, if you're designing a spreadsheet application that displays data in a sparsely populated table, you can create your own table model that is optimized for such use."

Within the model trail, there is a reference to the *How To Use Sliders* page. Go through that too in detail, so that the JSlider is your first example of a basic component. Then, go through a lesson based on a basic and commonly used component, JTextField: *How To Use The Text Field*.

Lastly, go through the lesson on a more complex but very powerful component, JTable: *How To Use Tables*. This material also includes some discussion of events and event listeners, which are not covered in detail until later but you should be able to get through it now. You may find it useful to look back it again after you have worked on events and event listeners, as presented in the next section.

3. Event listeners in Swing

Models, as introduced above, form one piece of common infrastructure in Swing components. We now move on to another piece of Swing infrastructure that you may have already seen many

references to in the previous material: event listeners, which let components receive and respond to input. This section will focus on a third piece of infrastructure, directed towards visual output: layout managers.

Underlying Swing components and their events are threads. If you are not familiar with these from other courses then have a look at the first two sections of the *Concurrency* lesson: *Processes and Threads,* and *Thread Objects*.

Work through the *How To Use Progress Bars* lesson to see a concrete example of threads in Swing. Then, go through two selected parts of the longer lesson on *Writing Event Listeners*: the *Introduction* and then the *General Information about Writing Event Listeners*.

Note also the *Implementing Listeners for Commonly Handled Events* page. It has lots of sections that give details about implementing the most common kinds of event listeners. Go through the first of these sections, *How To Write an Action Listener*. Action listeners are probably the easiest and most common event handlers to implement. You implement an action listener to define what should be done when an user performs a certain operation. Examples: when the user clicks a button, chooses a menu item, and presses Enter in a text field. The result is that an actionPerformed message is sent to all action listeners that are registered on the relevant component.

Go through the *Mouse Listener lesson* too — another simple but common listener.

Lastly, look at an example that will take you back to the table lesson again, and so perhaps make that clearer now: *How to Write a Table Model Listener*.

4. Swing Layout Managers

You will be looking at selected parts of a *longer lesson* on layout managers—part of Swing infrastructure used to control the spatial positions and relationships of components in a container.

First, have a look through the set of layout managers available in *A Visual Guide To Layout Managers*. Then go through the lessons *Using Layout Managers* and *How Layout Management Works* in detail. For future reference, you might note that there are also pages within the longer lesson on how to use the different layout managers, how to create a custom layout manager and how to solve common layout manager problems, but you don't have to go through these in detail now.

Go through the short lesson on doing the management of layout yourself: *Doing Without A Layout Manager (Absolute Positioning)*.

5. Painting in Swing

Go through the lesson on *Performing Custom Painting*. Many programs will get by just fine without writing their own painting code; they will simply use the standard GUI components that are already available in the Swing API. However, sometimes you need specific control over how your graphics are drawn. This lesson explores custom painting by creating a simple GUI application that draws a shape in response to the user's mouse activity, focusing on the underlying painting concepts.

Within the custom painting lesson is a useful page that you do not have to study now, but should note for future reference: *Solving Common Painting Problems*.