CISC 3120
Design & Implementation of Software Applications I

Lecture #8 – Interactivity

M. Meyer
Course Page:
http://www.sci.brooklyn.cuny.edu/~meyer/CISC3120
Content

- The Java Event Delegation Model
- Events and Interfaces
- Checkboxes
- MouseEvents
- MouseMoveEvents
- A Simple Paint Utility
Java Event Delegation Model

- **Event Sources** are objects that produce (or fire) events.
  - Examples: Buttons, Scrollbars

- **Event Listeners** are objects that handle generated events. Listener objects must be registered with an event source(s).
  - `button.addActionListener(this);`
  - This is sometimes referred to as a call-back.

- **Events** are objects that encapsulate actions.
  - Each objects include a reference to the event source object, the time the event occurred, and other information.
Events & Interfaces

- For each Event there is a corresponding Listener interface.
  - The interface must be implemented by all listeners interested in receiving those types of event.
  - The interface specifies the handler methods that must be implemented by the listener.

  - **Event** $\rightarrow$ **Interface**
  - ActionEvent $\rightarrow$ ActionListener
  - AdjustmentEvent $\rightarrow$ AdjustmentListener

- There may be more than one method per event/listener because each event may have several types. (MouseEvent)
Handling Multiple Events

- A single user action may generate many events.
  - In some cases several different listener methods will be invoked with the same event object.
  - For example clicking the mouse with the cursor on a Button will generate an ActionEvent and three MouseEvents!
- In most cases the code will specify that all but one of the events are ignored.
  - In the case of Buttons your code will typically ignore any MouseEvents and only listen for the ActionEvent.
- Java makes no guarantee about the ordering of events (generate or processed).
Types of Events

- Applets can support over a dozen unique events, by:
  - Implementing the appropriate interface,
  - Registering an event listener with the object that will be creating the event, using the appropriate type -> `scrollbar.addAdjustmentListener(this);`
  - Adding the required event handling method to the applet (or other object) code.

- For now will can do most of what we want to do with the following:
<table>
<thead>
<tr>
<th>Event, Interface</th>
<th>addmethod()</th>
<th>Components supporting events of this type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>addActionListener()</td>
<td>Button, TextField, CheckBoxMenuItem, CheckBox, ComboBox, Text Area and Text Field</td>
</tr>
<tr>
<td>AdjustmentEvent</td>
<td>addAdjustmentListener()</td>
<td>Scrollbar</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>addMouseListener()</td>
<td>Component and derivatives*</td>
</tr>
<tr>
<td>MouseMotionEvent</td>
<td>addMouseMotionListener()</td>
<td>Component and derivatives*</td>
</tr>
<tr>
<td>TextEvent</td>
<td>addTextListener()</td>
<td>TextArea and TextField</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>addItemListener()</td>
<td>CheckBox, CheckBoxMenuItem, ComboBox,</td>
</tr>
</tbody>
</table>
The Checkbox Component

- The Checkbox component allows the user to check off one or more choices from a fixed set of choices.
- Checkbox objects may either exist in isolation from each other or they may be created within a CheckboxGroup object.
- Checkboxes that are in a CheckboxGroup are mutually exclusive (only one box in the group may be selected, if you select one the others will unselect) -- in this case they are often called radio buttons.
- ItemListeners can be added to the checkboxes to handle events generated by the checkboxes.
Applet_11
Mouse Events

- The event-handling idea should be familiar by now:
  - register as listener, handle the event when it occurs, using information obtained from the corresponding Event object
- In the past, we placed a component on the applet and registered another object (the applet) with that component as a listener for its events.
- With the mouse, can register the applet itself as the object generating the event:
  - `addMouseListener(this);
  - // same as this.addMouseListener(this);
- This is equivalent to the applet invoking one of its own methods.
Mouse Events

- The MouseEvent/Listener only handles mouse button events (pressing and releasing the button), entering and exiting the component's real estate.
- For purposes of performance, mouse motion events (which occur with much more frequency) are handled via a separate MouseMotionEvent/Listener.
- The MouseListener first interface contains more than one method that needs to be implemented.
- We are only interested in the click event but all the other methods of the MouseListener interface (mouseEntered, mousePressed, ...) must be included as well (although they can be left empty).
Applet_12
MouseMotionListener

- MouseMotionListener events are generated whenever the mouse is moved within a Components real estate.
- The MouseMotionListener events specifically identify both dragging and moving behaviors.
- MouseMotion events are handy for enabling actions when a mouse moves over a “hotspot” on the screen.
- You can also MouseMotion events to allow users to drag items, although that will require MouseListener and MouseMotionListener at the same time.
Applet_12b
Putting it together

- Before we can move on and look at animations and complex interfaces, it will be worthwhile to examine a simple, minimally-featured, paint applet.

- You already have all of the skills you need to create the following applet:
  1. The applet should allow the user to control the length and thickness of a displayed line.
  2. Buttons and ScrollBars should be used for the input to the applet to limit the complexity of the controls.
  3. Textfields to be used to give the user feedback about what they have changed.
Considerations

- Where are we going to put the Buttons, Scrollbars and TextFields?
  - How many are we going to need/create?
  - How are we going to arrange them?
  - How are we going to compensate for the possibility of the user changing the applet size.

- The line will be, “drawn” using x and y coordinates.
  - Where are we going to draw it?
  - How will we keep it from overlapping with the components that have been added to the applet?

- What extra variables are we going to need to track in order to draw the line correctly?
Classes

- **Applet_14**
  - As with our previous examples our main class will be an Applet subclass.
  - It will be responsible for creating the interface elements and adding them to the applet container.
  - It will be responsible for performing the layout (using Panel and Layout objects).
  - It will also be responsible for things like the background color... **BUT**
  - **Question:** Should it be responsible for handling events generated by the Buttons and the Scrollbar?
Classes

- **PaintCanvas**
  - It will be better to have our “drawing area” as a separate class.
  - `PaintCanvas` will extend the `Canvas` class which will require us to have a `paint()` method.
  - `PaintCanvas` will be responsible for listening and reacting to the components (provide event handling methods) for the components that were created and added to the applet object.
  - We will also need functions to handle the “draw line” logic.
Applet_14
You can do this!