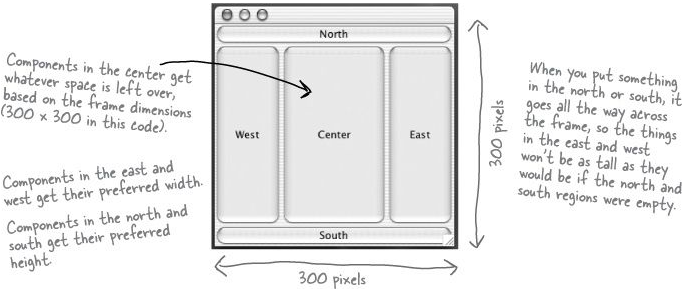
**B”H**

**Chapter 13**

**Swing**

* Layout Manager objects control the size and location of the widgets in a Java GUI.
* They do a ton of work on your behalf.
* Component is the more correct term for what we’ve been calling a widget. The things a user sees and interacts with. Text fields, buttons, scrollable lists, radio buttons, etc. are all components. They all extend javax.swing.JComponent.
* You can stick just about anything into anything else. But most of the time, you’ll add user interactive components such as buttons and lists into background components such as frames and panels.
* The layout manager controls the components contained within the component the layout manager is associated with.
  1. In other words, if a frame holds a panel, and the panel holds a button, the panel’s layout manager controls the size and placement of the button, while the frame’s layout manager controls the size and placement of the panel. The button, on the other hand, doesn’t need a layout manager because the button isn’t holding other components.
* Different layout managers have different policies for arranging components (like, arrange in a grid, make them all the same size, stack them vertically, etc.) but the components being laid out do get at least some small say in the matter.
* The Big Three layout managers:

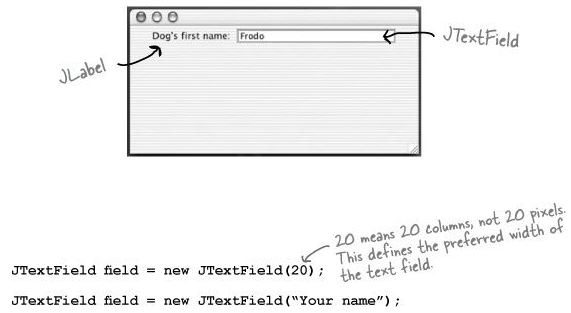
1. BorderLayout manager:
   * Divides a background component into five regions.
   * You can add only one component per region to a background.
   * Components laid out usually don’t get to have their preferred size.
   * BorderLayout is the default layout manager for a frame.
   * Components in the north and south get their preferred height, but not width. Components in the east and west get their preferred width, but not height. The component in the center gets whatever is left over (unless you use pack()).
   * The pack() method is like shrink-wrap for the components; it uses the full preferred size of the center component, then determines the size of the frame using the center as a starting point, building the rest based on what’s in the other regions.



1. FlowLayout manager:
   * Each component is the size it wants to be, and they’re laid out left to right in the order that they’re added. So when a component won’t fit horizontally, it drops to the next “line” in the layout.
   * FlowLayout is the default layout manager for a panel.
   * See Chapter13Sample01.java
2. BoxLayout manager:
   * It’s like FlowLayout in that each component gets to have its own size, and the components are placed in the order in which they’re added.
   * But, unlike FlowLayout, a BoxLayout manager can stack the components vertically (or horizontally, but usually we’re just concerned with vertically). Unlike FlowLayout, BoxLayout can force a ‘new line’ to make the components wrap to the next line, even if there’s room for them to fit horizontally.
   * See Chapter13Sample02.java

* There is a setSize(), but the layout managers will ignore it. There’s a distinction between the preferred size of the component and the size you want it to be. The preferred size is based on the size the component actually needs (the component makes that decision for itself). The layout manager calls the component’s getPreferredSize() method, and that method doesn’t care if you’ve previously called setSize() on the component.
  + A layout manager asks components for their preferred size, before making a decision about the layout. Depending on the layout manager’s policies, it might respect all, some, or none of the component’s wishes.
* On a component by component basis, you can call setLayout( null) and then it’s up to you to hard-code the exact screen locations and dimensions. In the long run, though, it’s almost always easier to use layout managers.

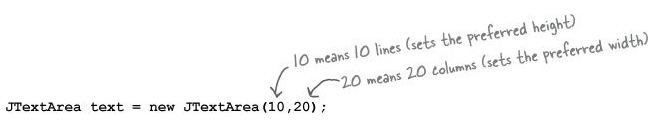
**JTextField**

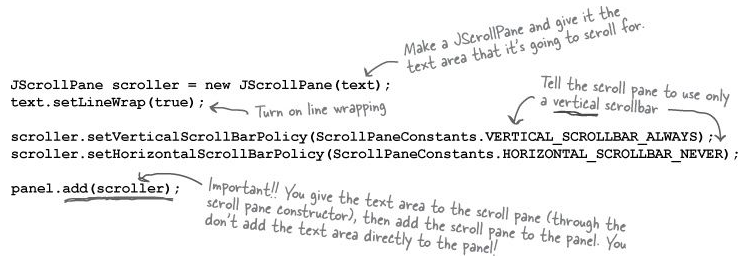


* Some sample code:
  + System.out.println( field.getText());
  + field.setText(" whatever");
  + field.setText("");
  + Register for key events if you really want to hear about it every time the user presses a key: field.addActionListener( myActionListener);
  + Select/ Highlight the text in the field: field.selectAll();
  + Put the cursor back in the field (so the user can just start typing): field.requestFocus();

**JTextArea**

* It doesn’t come out of the box with scroll bars or line wrapping. To make a JTextArea scroll, you have to stick it in a ScrollPane.
* Some sample code:





text.setText(" Not all who are lost are wandering");

text.append(" button clicked");

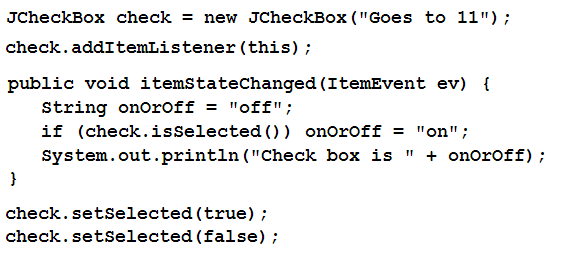
text.selectAll();

text.requestFocus();

* See Chapter13Sample03.java

**JCheckBox**

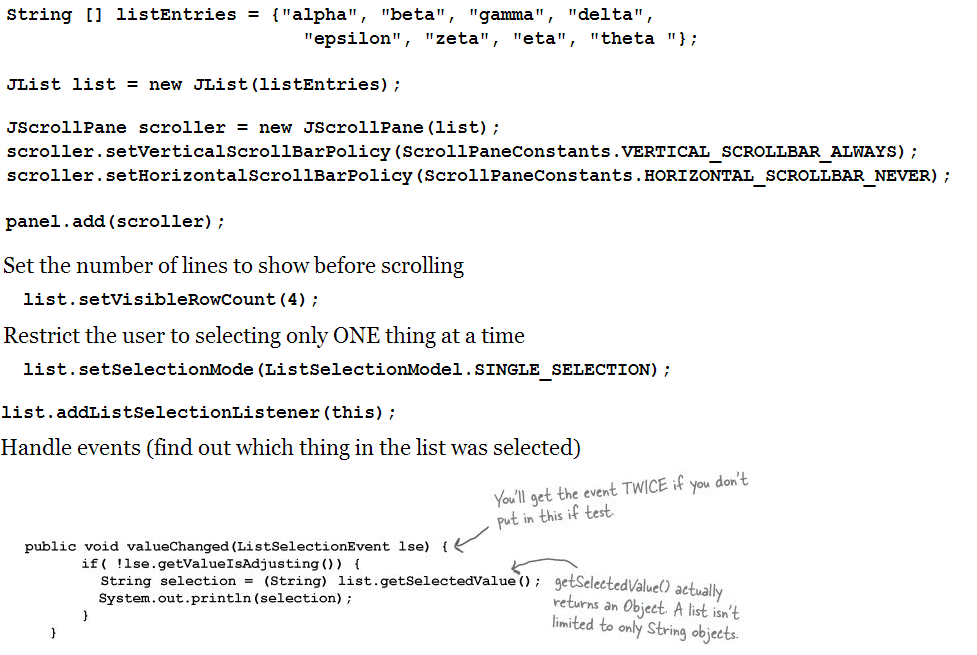
* Sample snippets:



* See Chapter13Sample03.java

**JList**

* JList constructor takes an array of any object type. They don’t have to be Strings, but a String representation will appear in the list.
* Sample snippets:



* See Chapter13Sample04.java