

Chapter 22 Containers, Layout Managers and Borders

1. Yes, although it is not practical.
2. To set an image icon in a JFrame's title bar, use the `setIconImage(ImageIcon)` method in `JFrame`. You cannot set an image icon in JApplet's title bar.
3. All of the following methods (contentPane, iconImage, jMenuBar, resizable, title) are in JFrame, but only contentPane and jMenuBar are in JApplet.
4. Java uses a variety of layout managers to layout components in the container. In VB and Delphi, the components are placed in absolute positions and sizes.
5. The size of a component in a container is determined by many factors, such as:
 - [BL]The type of layout manager used by the container.
 - [BL]The layout constraints associated with each component.
 - [BL]The size of the container.
 - [BX]Certain properties common to all components (such as preferredSize, minimumSize, maximumSize, alignmentX, and alignmentY).
6. The preferredSize property indicates the ideal size at which the component looks best. Depending on the rules of the particular layout manager, this property may or may not be considered. For example, the preferred size of a component is used in a container with a FlowLayout manager, but ignored if it is placed in a container with a GridLayout manager.

The minimumSize property specifies the minimum size at which the component is useful. For most GUI components, minimumSize is the same as preferredSize. Layout managers generally respect minimumSize more than preferredSize.

The maximumSize property specifies the maximum size needed by a component, so that the layout manager won't wastefully give space to a component that does not need it. For instance, BorderLayout limits the center component's size to its maximum size, and gives the space to edge components.
7. The alignmentX (alignmentY) property specifies how the component would like to be aligned relative to other components along the x-axis (y-axis). This value should be a number between 0 and 1, where 0 represents alignment along the origin, 1 is aligned the farthest away from the origin, 0.5 is centered, etc. These two properties are used in the BoxLayout and OverlayLayout.

8. CardLayout places components in the container as cards. Only one card is visible at a time, and the container acts as a stack of cards. The ordering of cards is determined by the container's own internal ordering of its component objects. CardLayout defines a set of methods that allow an application to flip through the cards sequentially or to show a specified card directly.

To construct a CardLayout, use the following constructors:

```
[BL]      public CardLayout(int hGap, int vGap)
```

```
[BL]      public GridLayout()
```

9. The GridBagLayout manager is the most flexible and the most complex. It is similar to the GridLayout manager in the sense that both layout managers arrange components in a grid. The components of GridBagLayout can vary in size, however, and can be added in any order. To use GridBagLayout effectively, you must customize the GridBagConstraints of one or more of its components. You customize a GridBagConstraints object by setting one or more of its public instance variables. These variables specify the component location, size, growth factor, anchor, inset, filling, and padding.

10. Yes. For convenience, Java also supports absolute layout that enables you to place components at a fixed location. In this case, the component must be placed using the component's instance method setBounds() (defined in java.awt.Component).

Absolute positions and sizes are fine if the application is developed and deployed on the same platform, but what looks fine on a development system may not look right on a deployment system on a different platform.

11. You can use BoxLayout in any container, but it is simpler to use the Box class, which is a container of BoxLayout. To create a Box container, use one of the following two static methods:

```
Box box1 = Box.createHorizontalBox();  
Box box2 = Box.createVerticalBox();
```

The former creates a box that contains components horizontally, and the latter creates a box that contains components vertically.

A filler is an invisible component. There are three kinds of fillers: struts, rigid areas, and glues.

A *strut* simply adds some space between components. The static method createHorizontalStrut(int) in the Box class is used to create a horizontal strut, and the static method createVerticalStrut(int) to create a vertical strut. For example,

the following code adds a vertical strut of 8 pixels between two buttons in a vertical box.

```
box2.add(new JButton("Button 1"));  
box2.add(Box.createVerticalStrut(8));  
box2.add(new JButton("Button 2"));
```

A *rigid area* is a two-dimensional space that can be created using the static method createRigidArea(dimension) in the Box class. For example, the following code adds a rigid area 10 pixels wide and 20 pixels high into a box.

```
box2.add(Box.createRigidArea(new Dimension(10, 20));
```

A *glue* separates components as much as possible. For example, by adding a glue between two components in a horizontal box, you place one component at the left end and the other at the right end. A glue can be created using the Box.createGlue() method.

12. OverlayLayout is a Swing layout manager that arranges components on top of each other.

To create an OverlayLayout, use the following constructor:

```
public OverlayLayout(Container target)
```

13. SpringLayout is a new Swing layout manager introduced in JDK 1.4. The idea of SpringLayout is put a flexible spring around a component. The spring may compress or expand to place the components in desired locations.

To create a SpringLayout, use the default constructor:

```
public SpringLayout()
```

A spring is an instance of the Spring class, which has a preferred value, minimum value, maximum value, and actual value. The getPreferredValue(), getMinimumValue(), getMaximumValue(), and getValue() methods retrieve these values. The setValue(int value) method can be used to set an actual value.

14. You can create a custom layout manager by implementing LayoutManager or one of the existing classes.

15. JScrollPane is a component that supports *automatic* scrolling without coding. A JScrollPane can be viewed as a specialized container with a view port for displaying the contained component. In addition to horizontal and vertical scrollbars, a JScrollPane can have a column header, a row header, and corners.

Normally, you have the component and you want to place the component in a scroll pane. A convenient way to create a scroll pane for a component is to use the JScrollPane(component) constructor.

16. JTabbedPane is a useful Swing container that provides a set of mutually exclusive tabs for accessing multiple components.

Usually you place the panels inside a JTabbedPane and associate a tab with each panel. JTabbedPane is easy to use, since the selection of the panel is handled automatically by clicking the corresponding tab. You can switch between a group of panels by clicking on a tab with a given title and/or icon.

17. JSplitPane is a convenient Swing container that contains two components with a separate divider. The bar can divide the container horizontally or vertically, and can be dragged to change the amount of space occupied by each component. To use JSplitPane, add a component to its left(top) and another one to its right (bottom).

18. All of these containers have their own fixed layout managers.

19. You can create these concrete borders using the constructors of these classes or use the static methods in `javax.swing.BorderFactory`.

20. Yes. You can set a border for every Swing GUI component and a border object can be shared.

21. All border classes and interfaces are in `javax.swing.border` package except that `BorderFactory` is in `javax.swing` package.