Swing from A to Z

Properties, Events, and Methods

By Richard G. Baldwin
baldwin.richard@iname.com

Java Programming, Lecture Notes # 1010

August 7, 2000

Preface

This series of lessons entitled *Swing from A to Z*, discusses the capabilities and features of Swing in quite a lot of detail. This series is intended for those persons who need to really understand what Swing is all about.

Recommended supplementary reading

It is recommended that in addition to studying this set of lessons, you also study my earlier lessons on Swing. A list of some of my Swing lessons can be found in an earlier lesson in this series. Links to the lessons themselves can be found at *Baldwin's Java Programming Tutorials*.

The earlier lessons will introduce you to the use of Swing while avoiding much of the detail included in this series.

Introduction

One of the most important things that we can do to understand all of Swing is to learn about the *properties*, *events*, and *methods* that Swing components inherit from the class named **JComponent** and its superclasses, **Container**, **Component**, and **Object**.

Apply to Swing components as a group

This is important because most Swing components extend **JComponent** either directly or indirectly. Thus, these properties, events, and methods apply to most of the components in Swing. We can learn about them as a group instead of having to learn about them on an individual component basis.

Each component can have other properties, events, and methods

In addition, individual Swing components can have other properties, events, and methods defined in subclasses of **JComponent**. We will deal with those properties, events, and methods on an individual component basis later in this series of lessons.

Lesson will identify properties, events, and methods

This lesson will identify the properties, events, and methods that Swing components inherit from **JComponent** and its superclasses. Subsequent lessons will discuss and illustrate many of them.

JavaBean Component Design Patterns

We can use JavaBean Component *design patterns* to identify properties, events, and methods of Swing components because Swing components are JavaBean Components.

Explained design patterns in earlier lessons

I'm not going to explain design patterns here, because my earlier lessons on JavaBeans provide a complete explanation of design patterns. Rather, I am simply going to use design patterns to identify and list the properties, events, and methods of **JComponent** and its superclasses.

If you are not familiar with Design Patterns in JavaBeans, see my earlier lessons on beans, which you will find at *Baldwin's Java Programming Tutorials*.

JComponent Properties

The properties defined in the **JComponent** class and its superclasses are important because they define the *default behavior and appearance* of most Swing components.

Figure 1 shows the accessor methods for the properties that are defined in the **JComponent** class.

Properties Defined in JComponent

- AccessibleContext **getAccessibleContext**()
- float **getAlignmentX**()
- void **setAlignmentX**(float alignmentX)
- float **getAlignmentY**()
- void **setAlignmentY**(float alignmentY)
- boolean **getAutoscrolls**()
- void **setAutoscrolls**(boolean autoscrolls)
- Color **getBackground**()
- void **setBackground**(Color bg)
- Border **getBorder**()
- void **setBorder**(Border border)
- int **getDebugGraphicsOptions**()
- void **setDebugGraphicsOptions**(int debugOptions)
- boolean **isDoubleBuffered**()
- void **setDoubleBuffered**(boolean aFlag)
- boolean **isEnabled**()
- void **setEnabled**(boolean enabled)
- boolean **isFocusCycleRoot**()
- boolean **isFocusTraversable**()
- Font **getFont**()
- void **setFont**(Font font)
- Color **getForeground**()
- void **setForeground**(Color fg)
- Graphics **getGraphics**()
- int **getHeight**()
- Insets **getInsets**()
- boolean **isManagingFocus**()
- Dimension **getMaximumSize**()
- void **setMaximumSize**(Dimension maximumSize)
- Dimension **getMinimumSize**()
- void **setMinimumSize**(Dimension minimumSize)
- Component **getNextFocusableComponent**()
- void **setNextFocusableComponent**(Component aComponent)
- boolean **isOpaque**()
- void **setOpaque**(boolean isOpaque)
- boolean **isOptimizedDrawingEnabled**()
- boolean **isPaintingTile**()
- Dimension **getPreferredSize**()
- void **setPreferredSize**(Dimension preferredSize)
- KeyStroke[] **getRegisteredKeyStrokes**()
- boolean **isRequestFocusEnabled**()
- void **setRequestFocusEnabled**(boolean aFlag)
- JRootPane **getRootPane**()
- String **getToolTipText**()
- void **setToolTipText**(String text)
- Container **getTopLevelAncestor**()
- String **getUIClassID**()
- boolean **isValidateRoot**()
- boolean **isVisible**()
- void **setVisible**(boolean aFlag)
- Rectangle **getVisibleRect**()

- int **getWidth**()
- int **getX**()
- int **getY**()

Figure 1

Inherited by most Swing components

The properties in Figure 1 are inherited by most Swing components.

Some are shown in red

A few of the properties are shown in red in Figure 1. The properties shown in red are not defined in **JComponent**, but rather are inherited from the **Component** class. They are included in Figure 1 for completeness. In each case, they form the other half of a *setter-getter* method pair. (See my earlier lessons on JavaBeans if the *setter-getter* terminology is new to you.)

Container and Component Properties

Figure 2 shows the accessor methods for the properties defined in the **Container** and **Component** classes that are not overridden in the **JComponent** class. **Container** properties are shown in blue. **Component** properties are shown in red.

Container and Component Properties Inherited by JComponent

- Color **getBackground**()
- Rectangle **getBounds**()
- void **setBounds**(int x, int y, int width, int height)
- void **setBounds**(Rectangle r)
- ColorModel **getColorModel**()
- Component **getComponent**(int n)
- int **getComponentCount**()
- ComponentOrientation **getComponentOrientation**()
- void **setComponentOrientation** (ComponentOrientation orientation)
- Component[] **getComponents**()
- Cursor **getCursor**()
- void **setCursor**(Cursor cursor)
- boolean **isDisplayable**()
- DropTarget **getDropTarget**()
- boolean **isEnabled**()
- Font **getFont**()
- Color **getForeground**()
- boolean **isLightweight**()
- boolean **isShowing**()
- void **setDropTarget**(DropTarget dt)
- FontMetrics **getFontMetrics**(Font font)
- InputContext **getInputContext**()

- InputMethodRequests **getInputMethodRequests**()
- LayoutManager **getLayout**()
- void **setLayout**(LayoutManager mgr)
- Locale **getLocale**()
- void **setLocale**(Locale l)
- Point **getLocation**()
- void **setLocation**(int x, int y)
- void **setLocation**(Point p)
- Point **getLocationOnScreen**()
- String **getName**()
- void **setName**(String name)
- Container **getParent**()
- Dimension **getSize**()
- void **setSize**(int width, int height)
- void **setSize**(Dimension d)
- Toolkit **getToolkit**()
- Object **getTreeLock**()
- boolean **isValid**()
- boolean **isVisible**()

Figure 2

Also inherited by Swing components

Because **JComponent** extends **Container**, which extends **Component**, most of the Swing components inherit these properties as well. Hence, these properties also define the default behavior and appearance of most Swing components.

Property Names

Although this isn't too important for this context, it might be useful for you to know how JavaBean design patterns define the names of properties.

Name is based on name of method

The official name of a property is that portion of the name of the accessor method following *set*, *get*, or *is*, with the case of the first character changed to lower case. (There are also some special cases involving upper case and lower case that I won't discuss here.)

A property named *visible*

For example, the following two accessor methods refer to a property named **visible**.

- boolean **isVisible**()
- void **setVisible**(boolean aFlag)

You can call the first method to read the current value of the property, and you can call the second method to write a new value into the property.

JComponent Events

Another important aspect of the default behavior of most swing components is the set of standard event types that they can multicast.

Default event types identified in JComponent and its superclasses

The default event types are defined by the **JComponent**, **Container**, and **Component** classes. Other event types that are specific to individual Swing components may be defined in subclasses of **JComponent**.

Event types match registration methods

We can identify the default event types by identifying the event registration methods in the **JComponent**, **Container**, and **Component** classes. These registration methods are inherited by most Swing components.

Registration method names

According to JavaBean design patterns, the type of the event can be identified by the word(s) appearing between *add* and *Listener* in the name of the registration method.

A PropertyChange event

For example, the existence of a registration method named ***add*PropertyChange*Listener()*** indicates the ability to multicast an event of the **PropertyChangeEvent** class.

The method name also indicates the availability of an interface named **PropertyChangeListener**. This interface must be implemented by classes from which listener objects for this type of event are instantiated.

JComponent events

Figure 3 shows the event registration methods that are defined in the **JComponent** class.

Events Defined in the JComponent Class

- **addAncestorListener**(AncestorListener listener) Registers listener so that it will receive AncestorEvents when it or any of its ancestors move or are made visible or invisible.
- **addPropertyChangeListener** (PropertyChangeListener listener) Add a PropertyChangeListener to the listener list.
- **addPropertyChangeListener**(String propertyName, PropertyChangeListener listener) Add a PropertyChangeListener for a specific property.
- **addVetoableChangeListener** (VetoableChangeListener listener) Add a VetoableChangeListener to the listener list.

- **removeAncestorListener**(AncestorListener listener) Unregisters listener so that it will no longer receive AncestorEvents
- **removePropertyChangeListener** (PropertyChangeListener listener) Remove a PropertyChangeListener from the listener list.
- **removePropertyChangeListener**(String propertyName, PropertyChangeListener listener) Remove a PropertyChangeListener for a specific property.
- **removeVetoableChangeListener** (VetoableChangeListener listener) Remove a VetoableChangeListener from the listener list.

Figure 3

Because most Swing components extend this class, they are able to multicast events of these types.

Container and Component Events

Figure 4 shows the event types for which registration methods are defined in the **Container** and **Component** classes.

Events Defined in the Container and Component Classes

- **addContainerListener**(ContainerListener l) Adds the specified container listener to receive container events from this container.
- **removeContainerListener** (ContainerListener l) Removes the specified container listener so it no longer receives container events from this container.
- **addComponentListener**(ComponentListener l) Adds the specified component listener to receive component events from this component.
- **addFocusListener**(FocusListener l) Adds the specified focus listener to receive focus events from this component when this component gains input focus.
- **addInputMethodListener** (InputMethodListener l) Adds the specified input method listener to receive input method events from this component.
- **addKeyListener**(KeyListener l) Adds the specified key listener to receive key events from this component.
- **addMouseListener**(MouseListener l) Adds the specified mouse listener to receive mouse events from this component.
- **addMouseMotionListener** (MouseMotionListener l) Adds the specified mouse motion listener to receive mouse motion events from this component.
- **removeComponentListener** (ComponentListener l) Removes the specified component listener so that it no longer receives component events from this component.
- **removeFocusListener**(FocusListener l) Removes the specified focus listener so that it no longer receives focus events from this component.
- **removeInputMethodListener** (InputMethodListener l) Removes the specified input method listener so that it no longer receives input method events from this component.
- **removeKeyListener**(KeyListener l) Removes the specified key listener so that it no longer receives key events from this component.
- **removeMouseListener**(MouseListener l) Removes the specified mouse listener so that it no longer receives mouse events from this component.
- **removeMouseMotionListener** (MouseMotionListener l) Removes the specified mouse motion listener so that it no longer receives mouse motion events from this component.

Figure 4

Because most Swing components extend these classes indirectly, they are able to multicast events of these types also.

Events defined in the **Container** class are shown in blue. Events defined in the **Component** class are shown in red.

Events familiar to AWT programmers

If you are already familiar with the use of event-driven programming using the Delegation Event Model and the AWT, you should already be familiar with all but one of these event types. (If not, see my tutorial lessons on the *Delegation Event Model*.)

One new event type

The one event type that may not be familiar to you is the **InputMethod** event.

According to one of my favorite authors, David Flanagan, *"Application-level code should never have to use this class."*

He also states *"Application-level code should never have to use or implement this interface."*

If you would like to know more about his reasoning, see his book entitled *Java Foundation Classes in a Nutshell*, published by O'Reilly.

Exposed Methods of JComponent

In addition to the default behavior provided by properties and events, default behavior for many swing components is also established by the public methods of the **JComponent** class and its superclasses.

Swing components have this behavior unless they override

All Swing components that extend **JComponent** either directly or indirectly will exhibit this default behavior unless they override the methods to provide behavior that is more appropriate for those components.

Figure 5 lists the **public** methods of the class **JComponent** that are not included in one of the previous lists. These are the public methods of the **JComponent** class that are not accessor methods for properties, and are not registration methods for events.

Public Methods of the JComponent Class

- void **addNotify**() Notification to this component that it now has a parent component.

- void **computeVisibleRect**(Rectangle visibleRect) Returns the Component's "visible rect rectangle" - the intersection of the visible rectangles for this component and all of its ancestors.
- boolean **contains**(int x, int y) Give the UI delegate an opportunity to define the precise shape of this component for the sake of mouse processing.
- JToolTip **createToolTip**() Returns the instance of JToolTip that should be used to display the tooltip.
- void **firePropertyChange**(String propertyName, boolean oldValue, boolean newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, byte oldValue, byte newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, char oldValue, char newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, double oldValue, double newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, float oldValue, float newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, int oldValue, int newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, long oldValue, long newValue) Reports a bound property change.
- void **firePropertyChange**(String propertyName, short oldValue, short newValue) Reports a bound property change.
- ActionListener **getActionForKeyStroke**(KeyStroke aKeyStroke) Return the object that will perform the action registered for a given keystroke.
- Rectangle **getBounds**(Rectangle rv) Store the bounds of this component into "return value" rv and return rv.
- Object **getClientProperty**(Object key) Returns the value of the property with the specified key.
- int **getConditionForKeyStroke**(KeyStroke aKeyStroke) Return the condition that determines whether a registered action occurs in response to the specified keystroke.
- Insets **getInsets**(Insets insets) Returns an Insets object containing this component's inset values.
- Point **getLocation**(Point rv) Store the x,y origin of this component into "return value" rv and return rv.
- Dimension **getSize**(Dimension rv) Store the width/height of this component into "return value" rv and return rv.
- Point **getToolTipLocation**(MouseEvent event) Return the tooltip location in the receiving component coordinate system If null is returned, Swing will choose a location.
- String **getToolTipText**(MouseEvent event) Returns the string to be used as the tooltip for event.
- void **grabFocus**() Set the focus on the receiving component.
- boolean **hasFocus**() Returns true if this Component has the keyboard focus.
- static boolean **isLightweightComponent**(Component c) Returns true if this component is a lightweight, i.e.
- void **paintImmediately**(int x, int y, int w, int h) Paint the specified region in this component and all of its descendants that overlap the region, immediately.
- void **paintImmediately**(Rectangle r) Paint the specified region now.
- void **putClientProperty**(Object key, Object value) Add an arbitrary key/value "client property" to this component.
- void **registerKeyboardAction**(ActionListener anAction, KeyStroke aKeyStroke, int aCondition) Calls registerKeyboardAction(ActionListener,String, KeyStroke, condition) with a null command.
- void **registerKeyboardAction**(ActionListener anAction, String aCommand, KeyStroke aKeyStroke, int aCondition) Register a new keyboard action.
- void **removeNotify**() Notification to this component that it no longer has a parent component.
- void **repaint**(long tm, int x, int y, int width, int height) Adds the specified region to the dirty region list if the component is showing.
- void **repaint**(Rectangle r) Adds the specified region to the dirty region list if the component is showing.
- boolean **requestDefaultFocus**() Request the focus for the component that should have the focus by default.
- void **requestFocus**() Set focus on the receiving component if isRequestFocusEnabled returns true
- void **resetKeyboardActions**() Unregister all keyboard actions
- void **reshape**(int x, int y, int w, int h) Moves and resizes this component.
- void **revalidate**() Support for deferred automatic layout.
- void **scrollRectToVisible**(Rectangle aRect) Forwards the scrollRectToVisible() message to the JComponent's parent.
- void **unregisterKeyboardAction**(KeyStroke aKeyStroke) Unregister a keyboard action.
- void **update**(Graphics g) Calls paint(g).
- void **updateUI**() Resets the UI property to a value from the current look and feel.

Figure 5

Exposed methods

Many Swing components expose the methods in Figure 5 to the outside world as a result of extending the **JComponent** class.

JavaBean introspection

For example, these are the methods that a JavaBeans *introspection* process (based on *design patterns*) would consider available for invocation by other beans.

Some are overridden methods

Some of the methods in Figure 5 are overridden versions of methods originally defined in **Container**, **Component**, or **Object**. They have been overridden to cause their behavior to be more appropriate for Swing components.

Others are new methods

Other methods in Figure 5 are new methods designed to provide new behavior for Swing components.

Some will be overridden later

Some of the methods in Figure 5 are overridden further down the inheritance hierarchy in the classes from which specific Swing components are instantiated. In those cases, they are overridden to make the behavior more appropriate for those specific components.

Those not overridden define default behavior

However, many of them are not overridden, and those that are not overridden define default behavior for many Swing components.

Numerous other methods are inherited also

There are numerous other methods that Swing components inherit from **Container**, **Component**, and **Object** and expose to the outside world. However, since they are not specific to Swing, I haven't listed them here.

Summary

The primary purpose of this lesson has been to provide reference information in summary form on Swing *properties*, *events*, and *methods*.

Will use in subsequent lessons

This information will be used in subsequent lessons that discuss appearance and behavior that is common to many Swing components.

Useful to understand common behavior

It is useful to understand the appearance and behavior that is common to many Swing components before getting into the details of appearance and behavior that are specific to individual components.

Learning the common behaviors first can greatly accelerate the learning process.

Sun documentation is the final authority

Please note that all of this information was extracted from the documentation for JDK 1.2.2 from Sun.

Extracting the information was a tedious process, and it is possible that I may have made some mistakes.

In the event of any conflict between what I have presented here and the Sun documentation, you should consider the Sun documentation to be the final authority.

Where To From Here?

The next several lessons will discuss and illustrate some of the common Swing properties, events, and methods listed above.

About the author

Richard Baldwin *is a college professor and private consultant whose primary focus is a combination of Java and XML. In addition to the many platform-independent benefits of Java applications, he believes that a combination of Java and XML will become the primary driving force in the delivery of structured information on the Web.*

Richard has participated in numerous consulting projects involving Java, XML, or a combination of the two. He frequently provides onsite Java and/or XML training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Java Programming Tutorials, which has gained a worldwide following among experienced and aspiring Java programmers. He has also published articles on Java Programming in Java Pro magazine.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

baldwin.richard@iname.com

-end-