javax.swing

Class JComponent

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

Direct Known Subclasses:

AbstractButton, BasicInternalFrameTitlePane, Box, Box.Filler, JColorChooser, JComboBox, JFileChooser, JInternalFrame, JInternalFrame.JDesktopIcon, JLabel, JLayer, JLayeredPane, JList, JMenuBar, JOptionPane, JPanel, JPopupMenu, JProgressBar, JRootPane, JScrollBar, JScrollPane, JSeparator, JSlider, JSpinner, JSplitPane, JTabbedPane, JTable, JTableHeader, JTextComponent, JToolBar, JToolTip, JTree, JViewport

---

```
public abstract class JComponent
extends Container
implements Serializable
```

The base class for all Swing components except top-level containers. To use a component that inherits from `JComponent`, you must place the component in a containment hierarchy whose root is a top-level Swing container. Top-level Swing containers -- such as `JFrame`, `JDialog`, and `JApplet` -- are specialized components that provide a place for other Swing components to paint themselves. For an explanation of containment hierarchies, see Swing Components and the Containment Hierarchy, a section in *The Java Tutorial*.

The `JComponent` class provides:

- The base class for both standard and custom components that use the Swing architecture.
- A "pluggable look and feel" (L&F) that can be specified by the programmer or (optionally) selected by the user at runtime. The look and feel for each component is provided by a *UI delegate* -- an object that descends from `ComponentUI`. See How to Set the Look and Feel in *The Java Tutorial* for more information.
- Comprehensive keystroke handling. See the document Keyboard Bindings in Swing, an article in *The Swing Connection*, for more information.
- Support for tool tips -- short descriptions that pop up when the cursor lingers over a component. See How to Use Tool Tips in *The Java Tutorial* for more information.
- Support for accessibility. `JComponent` contains all of the methods in the `Accessible` interface, but it doesn't actually implement the interface. That is the responsibility of the individual classes that extend `JComponent`.
- Support for component-specific properties. With the `putClientProperty(java.lang.Object, java.lang.Object)` and `getClientProperty(java.lang.Object)` methods, you can associate name-object pairs with any object that descends from `JComponent`.
- An infrastructure for painting that includes double buffering and support for borders. For more information see Painting and How to Use Borders, both of which are sections in *The Java Tutorial*.

For more information on these subjects, see the Swing package description and *The Java Tutorial* section The JComponent Class.

`JComponent` and its subclasses document default values for certain properties. For example, `JTable` documents the default row height as 16. Each `JComponent` subclass that has a `ComponentUI` will create the `ComponentUI` as part of its constructor. In order to provide a particular look and feel each `ComponentUI` may set properties back on the `JComponent` that created it. For example, a custom look and feel may require `JTable`s to have a row height of 24. The documented defaults are the value of a property BEFORE the `ComponentUI` has been installed. If you need a specific value for a particular property you should explicitly set it.

In release 1.4, the focus subsystem was rearchitected. For more information, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Warning: Swing is not thread safe. For more information see Swing's Threading Policy.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the `java.beans` package. Please see `XMLEncoder`.

See Also:

`KeyStroke`, `Action`, `setBorder(javax.swing.border.Border)`, `registerKeyboardAction(java.awt.event.ActionListener, java.lang.String, javax.swing.KeyStroke, int)`, `JOptionPane`, `setDebugGraphicsOptions(int)`, `setToolTipText(java.lang.String)`, `setAutoscrolls(boolean)`

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
class	**JComponent.AccessibleJComponent** Inner class of JComponent used to provide default support for accessibility.

Nested classes/interfaces inherited from class java.awt.Container

`Container.AccessibleAWTContainer`

Nested classes/interfaces inherited from class java.awt.Component

`Component.AccessibleAWTComponent`, `Component.BaselineResizeBehavior`, `Component.BltBufferStrategy`, `Component.FlipBufferStrategy`

Field Summary

Fields

Modifier and Type	Field and Description
protected AccessibleContext	**accessibleContext** The `AccessibleContext` associated with this `JComponent`.
protected EventListenerList	**listenerList** A list of event listeners for this component.
static String	**TOOL_TIP_TEXT_KEY** The comment to display when the cursor is over the component, also known as a "value tip", "flyover help", or "flyover label".
protected ComponentUI	**ui** The look and feel delegate for this component.
static int	**UNDEFINED_CONDITION** Constant used by some of the APIs to mean that no condition is defined.
static int	**WHEN_ANCESTOR_OF_FOCUSED_COMPONENT** Constant used for `registerKeyboardAction` that means that the command should be invoked when the receiving component is an ancestor of the focused component or is itself the focused component.
static int	**WHEN_FOCUSED** Constant used for `registerKeyboardAction` that means that the command should be invoked when the component has the focus.
static int	**WHEN_IN_FOCUSED_WINDOW** Constant used for `registerKeyboardAction` that means that the command should be invoked when the receiving component is in the window that has the focus or is itself the focused component.

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
JComponent() Default JComponent constructor.

Method Summary

Methods

Modifier and Type	Method and Description
void	**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.
void	**addNotify**() Notifies this component that it now has a parent component.
void	**addVetoableChangeListener**(VetoableChangeListener listener) Adds a VetoableChangeListener to the listener list.
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) Gives 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	**disable**() **Deprecated.** *As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).*
void	**enable**() **Deprecated.** *As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).*
void	**firePropertyChange**(String propertyName, boolean oldValue, boolean newValue) Support for reporting bound property changes for boolean properties.
void	**firePropertyChange**(String propertyName, char oldValue, char newValue) Reports a bound property change.

void	**firePropertyChange**(**String** propertyName, int oldValue, int newValue) Support for reporting bound property changes for integer properties.
protected void	**fireVetoableChange**(**String** propertyName, **Object** oldValue, **Object** newValue) Supports reporting constrained property changes.
AccessibleContext	**getAccessibleContext**() Returns the AccessibleContext associated with this JComponent.
ActionListener	**getActionForKeyStroke**(**KeyStroke** aKeyStroke) Returns the object that will perform the action registered for a given keystroke.
ActionMap	**getActionMap**() Returns the ActionMap used to determine what Action to fire for particular KeyStroke binding.
float	**getAlignmentX**() Overrides Container.getAlignmentX to return the vertical alignment.
float	**getAlignmentY**() Overrides Container.getAlignmentY to return the horizontal alignment.
AncestorListener[]	**getAncestorListeners**() Returns an array of all the ancestor listeners registered on this component.
boolean	**getAutoscrolls**() Gets the autoscrolls property.
int	**getBaseline**(int width, int height) Returns the baseline.
Component.BaselineResizeBehavior	**getBaselineResizeBehavior**() Returns an enum indicating how the baseline of the component changes as the size changes.
Border	**getBorder**() Returns the border of this component or null if no border is currently set.
Rectangle	**getBounds**(**Rectangle** rv) Stores the bounds of this component into "return value" rv and returns rv.
Object	**getClientProperty**(**Object** key) Returns the value of the property with the specified key.
protected **Graphics**	**getComponentGraphics**(**Graphics** g) Returns the graphics object used to paint this component.
JPopupMenu	**getComponentPopupMenu**() Returns JPopupMenu that assigned for this component.
int	**getConditionForKeyStroke**(**KeyStroke** aKeyStroke) Returns the condition that determines whether a registered action occurs in response to the specified keystroke.
int	**getDebugGraphicsOptions**() Returns the state of graphics debugging.
static **Locale**	**getDefaultLocale**() Returns the default locale used to initialize each JComponent's locale property upon creation.
FontMetrics	**getFontMetrics**(**Font** font) Gets the FontMetrics for the specified Font.
Graphics	**getGraphics**() Returns this component's graphics context, which lets you draw on a component.
int	**getHeight**() Returns the current height of this component.
boolean	**getInheritsPopupMenu**() Returns true if the JPopupMenu should be inherited from the parent.
InputMap	**getInputMap**() Returns the InputMap that is used when the component has focus.
InputMap	**getInputMap**(int condition)

	Returns the `InputMap` that is used during `condition`.
`InputVerifier`	`getInputVerifier()` Returns the input verifier for this component.
`Insets`	`getInsets()` If a border has been set on this component, returns the border's insets; otherwise calls `super.getInsets`.
`Insets`	`getInsets(Insets insets)` Returns an `Insets` object containing this component's inset values.
`<T extends EventListener> T[]`	`getListeners(Class<T> listenerType)` Returns an array of all the objects currently registered as *Foo*`Listeners` upon this `JComponent`.
`Point`	`getLocation(Point rv)` Stores the x,y origin of this component into "return value" `rv` and returns `rv`.
`Dimension`	`getMaximumSize()` If the maximum size has been set to a non-`null` value just returns it.
`Dimension`	`getMinimumSize()` If the minimum size has been set to a non-`null` value just returns it.
`Component`	`getNextFocusableComponent()` **Deprecated.** *As of 1.4, replaced by `FocusTraversalPolicy`.*
`Point`	`getPopupLocation(MouseEvent event)` Returns the preferred location to display the popup menu in this component's coordinate system.
`Dimension`	`getPreferredSize()` If the `preferredSize` has been set to a non-`null` value just returns it.
`KeyStroke[]`	`getRegisteredKeyStrokes()` Returns the `KeyStrokes` that will initiate registered actions.
`JRootPane`	`getRootPane()` Returns the `JRootPane` ancestor for this component.
`Dimension`	`getSize(Dimension rv)` Stores the width/height of this component into "return value" `rv` and returns `rv`.
`Point`	`getToolTipLocation(MouseEvent event)` Returns the tooltip location in this component's coordinate system.
`String`	`getToolTipText()` Returns the tooltip string that has been set with `setToolTipText`.
`String`	`getToolTipText(MouseEvent event)` Returns the string to be used as the tooltip for *event*.
`Container`	`getTopLevelAncestor()` Returns the top-level ancestor of this component (either the containing `Window` or `Applet`), or `null` if this component has not been added to any container.
`TransferHandler`	`getTransferHandler()` Gets the `transferHandler` property.
`String`	`getUIClassID()` Returns the `UIDefaults` key used to look up the name of the `swing.plaf.ComponentUI` class that defines the look and feel for this component.
`boolean`	`getVerifyInputWhenFocusTarget()` Returns the value that indicates whether the input verifier for the current focus owner will be called before this component requests focus.
`VetoableChangeListener[]`	`getVetoableChangeListeners()` Returns an array of all the vetoable change listeners registered on this component.
`Rectangle`	`getVisibleRect()` Returns the `Component`'s "visible rectangle" - the intersection of this component's visible rectangle, `new Rectangle(0, 0, getWidth(), getHeight())`, and all of

	its ancestors' visible rectangles.
int	**getWidth**() Returns the current width of this component.
int	**getX**() Returns the current x coordinate of the component's origin.
int	**getY**() Returns the current y coordinate of the component's origin.
void	**grabFocus**() Requests that this Component get the input focus, and that this Component's top-level ancestor become the focused Window.
void	**hide**() **Deprecated.**
boolean	**isDoubleBuffered**() Returns whether this component should use a buffer to paint.
static boolean	**isLightweightComponent**(**Component** c) Returns true if this component is lightweight, that is, if it doesn't have a native window system peer.
boolean	**isManagingFocus**() **Deprecated.** *As of 1.4, replaced by `Component.setFocusTraversalKeys(int, Set)` and `Container.setFocusCycleRoot(boolean)`.*
boolean	**isOpaque**() Returns true if this component is completely opaque.
boolean	**isOptimizedDrawingEnabled**() Returns true if this component tiles its children -- that is, if it can guarantee that the children will not overlap.
boolean	**isPaintingForPrint**() Returns `true` if the current painting operation on this component is part of a `print` operation.
protected boolean	**isPaintingOrigin**() Returns `true` if a paint triggered on a child component should cause painting to originate from this Component, or one of its ancestors.
boolean	**isPaintingTile**() Returns true if the component is currently painting a tile.
boolean	**isRequestFocusEnabled**() Returns `true` if this `JComponent` should get focus; otherwise returns `false`.
boolean	**isValidateRoot**() If this method returns true, `revalidate` calls by descendants of this component will cause the entire tree beginning with this root to be validated.
void	**paint**(**Graphics** g) Invoked by Swing to draw components.
protected void	**paintBorder**(**Graphics** g) Paints the component's border.
protected void	**paintChildren**(**Graphics** g) Paints this component's children.
protected void	**paintComponent**(**Graphics** g) Calls the UI delegate's paint method, if the UI delegate is non-`null`.
void	**paintImmediately**(int x, int y, int w, int h) Paints the specified region in this component and all of its descendants that overlap the region, immediately.
void	**paintImmediately**(**Rectangle** r) Paints the specified region now.
protected **String**	**paramString**() Returns a string representation of this `JComponent`.
void	**print**(**Graphics** g)

	Invoke this method to print the component to the specified `Graphics`.
`void`	**`printAll`**`(`**`Graphics`** `g)` Invoke this method to print the component.
`protected void`	**`printBorder`**`(`**`Graphics`** `g)` Prints the component's border.
`protected void`	**`printChildren`**`(`**`Graphics`** `g)` Prints this component's children.
`protected void`	**`printComponent`**`(`**`Graphics`** `g)` This is invoked during a printing operation.
`protected void`	**`processComponentKeyEvent`**`(`**`KeyEvent`** `e)` Processes any key events that the component itself recognizes.
`protected boolean`	**`processKeyBinding`**`(`**`KeyStroke`** `ks, `**`KeyEvent`** `e, int condition, boolean pressed)` Invoked to process the key bindings for `ks` as the result of the `KeyEvent` `e`.
`protected void`	**`processKeyEvent`**`(`**`KeyEvent`** `e)` Overrides `processKeyEvent` to process events.
`protected void`	**`processMouseEvent`**`(`**`MouseEvent`** `e)` Processes mouse events occurring on this component by dispatching them to any registered `MouseListener` objects, refer to **`Component.processMouseEvent(MouseEvent)`** for a complete description of this method.
`protected void`	**`processMouseMotionEvent`**`(`**`MouseEvent`** `e)` Processes mouse motion events, such as MouseEvent.MOUSE_DRAGGED.
`void`	**`putClientProperty`**`(`**`Object`** `key, `**`Object`** `value)` Adds an arbitrary key/value "client property" to this component.
`void`	**`registerKeyboardAction`**`(`**`ActionListener`** `anAction, `**`KeyStroke`** `aKeyStroke, int aCondition)` This method is now obsolete, please use a combination of `getActionMap()` and `getInputMap()` for similiar behavior.
`void`	**`registerKeyboardAction`**`(`**`ActionListener`** `anAction, `**`String`** `aCommand, `**`KeyStroke`** `aKeyStroke, int aCondition)` This method is now obsolete, please use a combination of `getActionMap()` and `getInputMap()` for similiar behavior.
`void`	**`removeAncestorListener`**`(`**`AncestorListener`** `listener)` Unregisters `listener` so that it will no longer receive `AncestorEvents`.
`void`	**`removeNotify`**`()` Notifies this component that it no longer has a parent component.
`void`	**`removeVetoableChangeListener`**`(`**`VetoableChangeListener`** `listener)` Removes a `VetoableChangeListener` from the listener list.
`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`**`()` **Deprecated.** *As of 1.4, replaced by `FocusTraversalPolicy.getDefaultComponent(Container).requestFocus()`*
`void`	**`requestFocus`**`()` Requests that this `Component` gets the input focus.
`boolean`	**`requestFocus`**`(boolean temporary)` Requests that this `Component` gets the input focus.
`boolean`	**`requestFocusInWindow`**`()` Requests that this `Component` gets the input focus.
`protected boolean`	**`requestFocusInWindow`**`(boolean temporary)` Requests that this `Component` gets the input focus.

void	**resetKeyboardActions**() Unregisters all the bindings in the first tier InputMaps and ActionMap.
void	**reshape**(int x, int y, int w, int h) **Deprecated.** *As of JDK 5, replaced by Component.setBounds(int, int, int, int).* *Moves and resizes this component.*
void	**revalidate**() Supports deferred automatic layout.
void	**scrollRectToVisible**(Rectangle aRect) Forwards the scrollRectToVisible() message to the JComponent's parent.
void	**setActionMap**(ActionMap am) Sets the ActionMap to am.
void	**setAlignmentX**(float alignmentX) Sets the the vertical alignment.
void	**setAlignmentY**(float alignmentY) Sets the the horizontal alignment.
void	**setAutoscrolls**(boolean autoscrolls) Sets the autoscrolls property.
void	**setBackground**(Color bg) Sets the background color of this component.
void	**setBorder**(Border border) Sets the border of this component.
void	**setComponentPopupMenu**(JPopupMenu popup) Sets the JPopupMenu for this JComponent.
void	**setDebugGraphicsOptions**(int debugOptions) Enables or disables diagnostic information about every graphics operation performed within the component or one of its children.
static void	**setDefaultLocale**(Locale l) Sets the default locale used to initialize each JComponent's locale property upon creation.
void	**setDoubleBuffered**(boolean aFlag) Sets whether this component should use a buffer to paint.
void	**setEnabled**(boolean enabled) Sets whether or not this component is enabled.
void	**setFocusTraversalKeys**(int id, Set<? extends AWTKeyStroke> keystrokes) Sets the focus traversal keys for a given traversal operation for this Component.
void	**setFont**(Font font) Sets the font for this component.
void	**setForeground**(Color fg) Sets the foreground color of this component.
void	**setInheritsPopupMenu**(boolean value) Sets whether or not getComponentPopupMenu should delegate to the parent if this component does not have a JPopupMenu assigned to it.
void	**setInputMap**(int condition, InputMap map) Sets the InputMap to use under the condition condition to map.
void	**setInputVerifier**(InputVerifier inputVerifier) Sets the input verifier for this component.
void	**setMaximumSize**(Dimension maximumSize) Sets the maximum size of this component to a constant value.
void	**setMinimumSize**(Dimension minimumSize) Sets the minimum size of this component to a constant value.
void	**setNextFocusableComponent**(Component aComponent)

	Deprecated. *As of 1.4, replaced by* `FocusTraversalPolicy`
void	`setOpaque(boolean isOpaque)` If true the component paints every pixel within its bounds.
void	`setPreferredSize(Dimension preferredSize)` Sets the preferred size of this component.
void	`setRequestFocusEnabled(boolean requestFocusEnabled)` Provides a hint as to whether or not this `JComponent` should get focus.
void	`setToolTipText(String text)` Registers the text to display in a tool tip.
void	`setTransferHandler(TransferHandler newHandler)` Sets the `TransferHandler`, which provides support for transfer of data into and out of this component via cut/copy/paste and drag and drop.
protected void	`setUI(ComponentUI newUI)` Sets the look and feel delegate for this component.
void	`setVerifyInputWhenFocusTarget(boolean verifyInputWhenFocusTarget)` Sets the value to indicate whether input verifier for the current focus owner will be called before this component requests focus.
void	`setVisible(boolean aFlag)` Makes the component visible or invisible.
void	`unregisterKeyboardAction(KeyStroke aKeyStroke)` This method is now obsolete.
void	`update(Graphics g)` Calls `paint`.
void	`updateUI()` Resets the UI property to a value from the current look and feel.

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, createImage, createImage, createVolatileImage, createVolatileImage, disableEvents, dispatchEvent, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, processComponentEvent,

processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processMouseWheelEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

ui

```
protected transient ComponentUI ui
```

The look and feel delegate for this component.

listenerList

```
protected EventListenerList listenerList
```

A list of event listeners for this component.

WHEN_FOCUSED

```
public static final int WHEN_FOCUSED
```

Constant used for `registerKeyboardAction` that means that the command should be invoked when the component has the focus.

See Also:

Constant Field Values

WHEN_ANCESTOR_OF_FOCUSED_COMPONENT

```
public static final int WHEN_ANCESTOR_OF_FOCUSED_COMPONENT
```

Constant used for `registerKeyboardAction` that means that the command should be invoked when the receiving component is an ancestor of the focused component or is itself the focused component.

See Also:

Constant Field Values

WHEN_IN_FOCUSED_WINDOW

```
public static final int WHEN_IN_FOCUSED_WINDOW
```

Constant used for `registerKeyboardAction` that means that the command should be invoked when the receiving component is in the window that has the focus or is itself the focused component.

See Also:

Constant Field Values

UNDEFINED_CONDITION

```
public static final int UNDEFINED_CONDITION
```

Constant used by some of the APIs to mean that no condition is defined.

See Also:

Constant Field Values

TOOL_TIP_TEXT_KEY

```
public static final String TOOL_TIP_TEXT_KEY
```

The comment to display when the cursor is over the component, also known as a "value tip", "flyover help", or "flyover label".

See Also:

Constant Field Values

accessibleContext

```
protected AccessibleContext accessibleContext
```

The `AccessibleContext` associated with this `JComponent`.

Constructor Detail

JComponent

```
public JComponent()
```

Default `JComponent` constructor. This constructor does very little initialization beyond calling the `Container` constructor. For example, the initial layout manager is `null`. It does, however, set the component's locale property to the value returned by `JComponent.getDefaultLocale`.

See Also:

`getDefaultLocale()`

Method Detail

setInheritsPopupMenu

```
public void setInheritsPopupMenu(boolean value)
```

Sets whether or not `getComponentPopupMenu` should delegate to the parent if this component does not have a `JPopupMenu` assigned to it.

The default value for this is false, but some `JComponent` subclasses that are implemented as a number of `JComponents` may set this to true.

This is a bound property.

Parameters:

`value` - whether or not the JPopupMenu is inherited

Since:

1.5

See Also:

`setComponentPopupMenu(javax.swing.JPopupMenu)`

getInheritsPopupMenu

```
public boolean getInheritsPopupMenu()
```

Returns true if the JPopupMenu should be inherited from the parent.

Since:

1.5

See Also:

`setComponentPopupMenu(javax.swing.JPopupMenu)`

setComponentPopupMenu

```
public void setComponentPopupMenu(JPopupMenu popup)
```

Sets the `JPopupMenu` for this `JComponent`. The UI is responsible for registering bindings and adding the necessary listeners such that the `JPopupMenu` will be shown at the appropriate time. When the `JPopupMenu` is shown depends upon the look and feel: some may show it on a mouse event, some may enable a key binding.

If `popup` is null, and `getInheritsPopupMenu` returns true, then `getComponentPopupMenu` will be delegated to the parent. This provides for a way to make all child components inherit the popupmenu of the parent.

This is a bound property.

Parameters:

`popup` - - the popup that will be assigned to this component may be null

Since:

1.5

See Also:

`getComponentPopupMenu()`

getComponentPopupMenu

```
public JPopupMenu getComponentPopupMenu()
```

Returns JPopupMenu that assigned for this component. If this component does not have a JPopupMenu assigned to it and getInheritsPopupMenu is true, this will return getParent().getComponentPopupMenu() (assuming the parent is valid.)

Returns:

JPopupMenu assigned for this component or null if no popup assigned

Since:

1.5

See Also:

setComponentPopupMenu(javax.swing.JPopupMenu)

updateUI

```
public void updateUI()
```

Resets the UI property to a value from the current look and feel. JComponent subclasses must override this method like this:

```
public void updateUI() {
   setUI((SliderUI)UIManager.getUI(this);
}
```

See Also:

setUI(javax.swing.plaf.ComponentUI), UIManager.getLookAndFeel(), UIManager.getUI(javax.swing.JComponent)

setUI

```
protected void setUI(ComponentUI newUI)
```

Sets the look and feel delegate for this component. JComponent subclasses generally override this method to narrow the argument type. For example, in JSlider:

```
public void setUI(SliderUI newUI) {
    super.setUI(newUI);
}
```

Additionally JComponent subclasses must provide a getUI method that returns the correct type. For example:

```
public SliderUI getUI() {
    return (SliderUI)ui;
}
```

Parameters:

newUI - the new UI delegate

See Also:

updateUI(), UIManager.getLookAndFeel(), UIManager.getUI(javax.swing.JComponent)

getUIClassID

```
public String getUIClassID()
```

Returns the `UIDefaults` key used to look up the name of the `swing.plaf.ComponentUI` class that defines the look and feel for this component. Most applications will never need to call this method. Subclasses of `JComponent` that support pluggable look and feel should override this method to return a `UIDefaults` key that maps to the `ComponentUI` subclass that defines their look and feel.

Returns:

the `UIDefaults` key for a `ComponentUI` subclass

See Also:

`UIDefaults.getUI(javax.swing.JComponent)`

getComponentGraphics

```
protected Graphics getComponentGraphics(Graphics g)
```

Returns the graphics object used to paint this component. If `DebugGraphics` is turned on we create a new `DebugGraphics` object if necessary. Otherwise we just configure the specified graphics object's foreground and font.

Parameters:

g - the original `Graphics` object

Returns:

a `Graphics` object configured for this component

paintComponent

```
protected void paintComponent(Graphics g)
```

Calls the UI delegate's paint method, if the UI delegate is non-`null`. We pass the delegate a copy of the `Graphics` object to protect the rest of the paint code from irrevocable changes (for example, `Graphics.translate`).

If you override this in a subclass you should not make permanent changes to the passed in `Graphics`. For example, you should not alter the clip `Rectangle` or modify the transform. If you need to do these operations you may find it easier to create a new `Graphics` from the passed in `Graphics` and manipulate it. Further, if you do not invoker super's implementation you must honor the opaque property, that is if this component is opaque, you must completely fill in the background in a non-opaque color. If you do not honor the opaque property you will likely see visual artifacts.

The passed in `Graphics` object might have a transform other than the identify transform installed on it. In this case, you might get unexpected results if you cumulatively apply another transform.

Parameters:

g - the `Graphics` object to protect

See Also:

`paint(java.awt.Graphics)`, `ComponentUI`

paintChildren

```
protected void paintChildren(Graphics g)
```

Paints this component's children. If `shouldUseBuffer` is true, no component ancestor has a buffer and the component children can use a buffer if they have one. Otherwise, one ancestor has a buffer currently in use and children should not use a buffer to paint.

Parameters:

g - the `Graphics` context in which to paint

See Also:

paint(java.awt.Graphics), Container.paint(java.awt.Graphics)

paintBorder

```
protected void paintBorder(Graphics g)
```

Paints the component's border.

If you override this in a subclass you should not make permanent changes to the passed in Graphics. For example, you should not alter the clip Rectangle or modify the transform. If you need to do these operations you may find it easier to create a new Graphics from the passed in Graphics and manipulate it.

Parameters:

g - the Graphics context in which to paint

See Also:

paint(java.awt.Graphics), setBorder(javax.swing.border.Border)

update

```
public void update(Graphics g)
```

Calls paint. Doesn't clear the background but see ComponentUI.update, which is called by paintComponent.

Overrides:

update in class Container

Parameters:

g - the Graphics context in which to paint

See Also:

paint(java.awt.Graphics), paintComponent(java.awt.Graphics), ComponentUI

paint

```
public void paint(Graphics g)
```

Invoked by Swing to draw components. Applications should not invoke paint directly, but should instead use the repaint method to schedule the component for redrawing.

This method actually delegates the work of painting to three protected methods: paintComponent, paintBorder, and paintChildren. They're called in the order listed to ensure that children appear on top of component itself. Generally speaking, the component and its children should not paint in the insets area allocated to the border. Subclasses can just override this method, as always. A subclass that just wants to specialize the UI (look and feel) delegate's paint method should just override paintComponent.

Overrides:

paint in class Container

Parameters:

g - the Graphics context in which to paint

See Also:

paintComponent(java.awt.Graphics), paintBorder(java.awt.Graphics), paintChildren(java.awt.Graphics), getComponentGraphics(java.awt.Graphics), repaint(long, int, int, int, int)

printAll

```
public void printAll(Graphics g)
```

Invoke this method to print the component. This method invokes `print` on the component.

Overrides:

`printAll` in class `Component`

Parameters:

`g` - the `Graphics` context in which to paint

See Also:

`print(java.awt.Graphics)`, `printComponent(java.awt.Graphics)`, `printBorder(java.awt.Graphics)`, `printChildren(java.awt.Graphics)`

print

```
public void print(Graphics g)
```

Invoke this method to print the component to the specified `Graphics`. This method will result in invocations of `printComponent`, `printBorder` and `printChildren`. It is recommended that you override one of the previously mentioned methods rather than this one if your intention is to customize the way printing looks. However, it can be useful to override this method should you want to prepare state before invoking the superclass behavior. As an example, if you wanted to change the component's background color before printing, you could do the following:

```
public void print(Graphics g) {
    Color orig = getBackground();
    setBackground(Color.WHITE);

    // wrap in try/finally so that we always restore the state
    try {
        super.print(g);
    } finally {
        setBackground(orig);
    }
}
```

Alternatively, or for components that delegate painting to other objects, you can query during painting whether or not the component is in the midst of a print operation. The `isPaintingForPrint` method provides this ability and its return value will be changed by this method: to `true` immediately before rendering and to `false` immediately after. With each change a property change event is fired on this component with the name "`paintingForPrint`".

This method sets the component's state such that the double buffer will not be used: painting will be done directly on the passed in `Graphics`.

Overrides:

`print` in class `Container`

Parameters:

`g` - the `Graphics` context in which to paint

See Also:

`printComponent(java.awt.Graphics)`, `printBorder(java.awt.Graphics)`, `printChildren(java.awt.Graphics)`, `isPaintingForPrint()`

printComponent

```
protected void printComponent(Graphics g)
```

This is invoked during a printing operation. This is implemented to invoke `paintComponent` on the component. Override this if you wish to add special painting behavior when printing.

Parameters:

g - the `Graphics` context in which to paint

Since:

1.3

See Also:

`print(java.awt.Graphics)`

printChildren

```
protected void printChildren(Graphics g)
```

Prints this component's children. This is implemented to invoke `paintChildren` on the component. Override this if you wish to print the children differently than painting.

Parameters:

g - the `Graphics` context in which to paint

Since:

1.3

See Also:

`print(java.awt.Graphics)`

printBorder

```
protected void printBorder(Graphics g)
```

Prints the component's border. This is implemented to invoke `paintBorder` on the component. Override this if you wish to print the border differently that it is painted.

Parameters:

g - the `Graphics` context in which to paint

Since:

1.3

See Also:

`print(java.awt.Graphics)`

isPaintingTile

```
public boolean isPaintingTile()
```

Returns true if the component is currently painting a tile. If this method returns true, paint will be called again for another tile. This method returns false if you are not painting a tile or if the last tile is painted. Use this method to keep some state you might need between tiles.

Returns:

true if the component is currently painting a tile, false otherwise

isPaintingForPrint

```
public final boolean isPaintingForPrint()
```

Returns `true` if the current painting operation on this component is part of a `print` operation. This method is useful when you want to customize what you print versus what you show on the screen.

You can detect changes in the value of this property by listening for property change events on this component with name "`paintingForPrint`".

Note: This method provides complimentary functionality to that provided by other high level Swing printing APIs. However, it deals strictly with painting and should not be confused as providing information on higher level print processes. For example, a `JTable.print()` operation doesn't necessarily result in a continuous rendering of the full component, and the return value of this method can change multiple times during that operation. It is even possible for the component to be painted to the screen while the printing process is ongoing. In such a case, the return value of this method is `true` when, and only when, the table is being painted as part of the printing process.

Returns:

true if the current painting operation on this component is part of a print operation

Since:

1.6

See Also:

`print(java.awt.Graphics)`

isManagingFocus

```
@Deprecated
public boolean isManagingFocus()
```

Deprecated. *As of 1.4, replaced by `Component.setFocusTraversalKeys(int, Set)` and `Container.setFocusCycleRoot(boolean)`.*
In release 1.4, the focus subsystem was rearchitected. For more information, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Changes this `JComponent`'s focus traversal keys to CTRL+TAB and CTRL+SHIFT+TAB. Also prevents `SortingFocusTraversalPolicy` from considering descendants of this JComponent when computing a focus traversal cycle.

See Also:

`Component.setFocusTraversalKeys(int, java.util.Set<? extends java.awt.AWTKeyStroke>)`, `SortingFocusTraversalPolicy`

setNextFocusableComponent

```
@Deprecated
public void setNextFocusableComponent(Component aComponent)
```

Deprecated. *As of 1.4, replaced by `FocusTraversalPolicy`*
In release 1.4, the focus subsystem was rearchitected. For more information, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Overrides the default `FocusTraversalPolicy` for this `JComponent`'s focus traversal cycle by unconditionally setting the specified `Component` as the next `Component` in the cycle, and this `JComponent` as the specified `Component`'s previous `Component` in the cycle.

Parameters:

`aComponent` - the `Component` that should follow this `JComponent` in the focus traversal cycle

See Also:

`getNextFocusableComponent()`, `FocusTraversalPolicy`

getNextFocusableComponent

```
@Deprecated
public Component getNextFocusableComponent()
```

Deprecated. *As of 1.4, replaced by* `FocusTraversalPolicy`.
In release 1.4, the focus subsystem was rearchitected. For more information, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Returns the `Component` set by a prior call to `setNextFocusableComponent(Component)` on this `JComponent`.

Returns:

the `Component` that will follow this `JComponent` in the focus traversal cycle, or `null` if none has been explicitly specified

See Also:

`setNextFocusableComponent(java.awt.Component)`

setRequestFocusEnabled

```
public void setRequestFocusEnabled(boolean requestFocusEnabled)
```

Provides a hint as to whether or not this `JComponent` should get focus. This is only a hint, and it is up to consumers that are requesting focus to honor this property. This is typically honored for mouse operations, but not keyboard operations. For example, look and feels could verify this property is true before requesting focus during a mouse operation. This would often times be used if you did not want a mouse press on a `JComponent` to steal focus, but did want the `JComponent` to be traversable via the keyboard. If you do not want this `JComponent` focusable at all, use the `setFocusable` method instead.

Please see How to Use the Focus Subsystem, a section in *The Java Tutorial*, for more information.

Parameters:

`requestFocusEnabled` - indicates whether you want this `JComponent` to be focusable or not

See Also:

Focus Specification, `Component.setFocusable(boolean)`

isRequestFocusEnabled

```
public boolean isRequestFocusEnabled()
```

Returns `true` if this `JComponent` should get focus; otherwise returns `false`.

Please see How to Use the Focus Subsystem, a section in *The Java Tutorial*, for more information.

Returns:

`true` if this component should get focus, otherwise returns `false`

See Also:

`setRequestFocusEnabled(boolean)`, Focus Specification, `Component.isFocusable()`

requestFocus

```
public void requestFocus()
```

Requests that this `Component` gets the input focus. Refer to `Component.requestFocus()` for a complete description of this method.

Note that the use of this method is discouraged because its behavior is platform dependent. Instead we recommend the use of `requestFocusInWindow()`. If you would like more information on focus, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Overrides:

`requestFocus` in class `Component`

Since:

1.4

See Also:

`Component.requestFocusInWindow()`, `Component.requestFocusInWindow(boolean)`

requestFocus

```
public boolean requestFocus(boolean temporary)
```

Requests that this `Component` gets the input focus. Refer to `Component.requestFocus(boolean)` for a complete description of this method.

Note that the use of this method is discouraged because its behavior is platform dependent. Instead we recommend the use of `requestFocusInWindow(boolean)`. If you would like more information on focus, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Overrides:

`requestFocus` in class `Component`

Parameters:

`temporary` - boolean indicating if the focus change is temporary

Returns:

`false` if the focus change request is guaranteed to fail; `true` if it is likely to succeed

Since:

1.4

See Also:

`Component.requestFocusInWindow()`, `Component.requestFocusInWindow(boolean)`

requestFocusInWindow

```
public boolean requestFocusInWindow()
```

Requests that this `Component` gets the input focus. Refer to `Component.requestFocusInWindow()` for a complete description of this method.

If you would like more information on focus, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Overrides:

`requestFocusInWindow` in class `Component`

Returns:

`false` if the focus change request is guaranteed to fail; `true` if it is likely to succeed

Since:

1.4

See Also:

`Component.requestFocusInWindow()`, `Component.requestFocusInWindow(boolean)`

requestFocusInWindow

```
protected boolean requestFocusInWindow(boolean temporary)
```

Requests that this `Component` gets the input focus. Refer to `Component.requestFocusInWindow(boolean)` for a complete description of this method.

If you would like more information on focus, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Overrides:

`requestFocusInWindow` in class `Component`

Parameters:

`temporary` - boolean indicating if the focus change is temporary

Returns:

`false` if the focus change request is guaranteed to fail; `true` if it is likely to succeed

Since:

1.4

See Also:

`Component.requestFocusInWindow()`, `Component.requestFocusInWindow(boolean)`

grabFocus

```
public void grabFocus()
```

Requests that this Component get the input focus, and that this Component's top-level ancestor become the focused Window. This component must be displayable, visible, and focusable for the request to be granted.

This method is intended for use by focus implementations. Client code should not use this method; instead, it should use `requestFocusInWindow()`.

See Also:

`requestFocusInWindow()`

setVerifyInputWhenFocusTarget

```
public void setVerifyInputWhenFocusTarget(boolean verifyInputWhenFocusTarget)
```

Sets the value to indicate whether input verifier for the current focus owner will be called before this component requests focus. The default is true. Set to false on components such as a Cancel button or a scrollbar, which should activate even if the input in the current focus owner is not "passed" by the input verifier for that component.

Parameters:

`verifyInputWhenFocusTarget` - value for the `verifyInputWhenFocusTarget` property

Since:

1.3

See Also:

InputVerifier, setInputVerifier(javax.swing.InputVerifier), getInputVerifier(), getVerifyInputWhenFocusTarget()

getVerifyInputWhenFocusTarget

```
public boolean getVerifyInputWhenFocusTarget()
```

Returns the value that indicates whether the input verifier for the current focus owner will be called before this component requests focus.

Returns:

value of the verifyInputWhenFocusTarget property

Since:

1.3

See Also:

InputVerifier, setInputVerifier(javax.swing.InputVerifier), getInputVerifier(), setVerifyInputWhenFocusTarget(boolean)

getFontMetrics

```
public FontMetrics getFontMetrics(Font font)
```

Gets the FontMetrics for the specified Font.

Overrides:

getFontMetrics in class Component

Parameters:

font - the font for which font metrics is to be obtained

Returns:

the font metrics for font

Throws:

NullPointerException - if font is null

Since:

1.5

See Also:

Component.getFont(), Component.getPeer(), ComponentPeer.getFontMetrics(Font), Toolkit.getFontMetrics(Font)

setPreferredSize

```
public void setPreferredSize(Dimension preferredSize)
```

Sets the preferred size of this component. If preferredSize is null, the UI will be asked for the preferred size.

Overrides:

setPreferredSize in class Component

Parameters:

`preferredSize` - The new preferred size, or null

See Also:

`Component.getPreferredSize()`, `Component.isPreferredSizeSet()`

getPreferredSize

```
public Dimension getPreferredSize()
```

If the `preferredSize` has been set to a non-`null` value just returns it. If the UI delegate's `getPreferredSize` method returns a non `null` value then return that; otherwise defer to the component's layout manager.

Overrides:

`getPreferredSize` in class `Container`

Returns:

the value of the `preferredSize` property

See Also:

`setPreferredSize(java.awt.Dimension)`, `ComponentUI`

setMaximumSize

```
public void setMaximumSize(Dimension maximumSize)
```

Sets the maximum size of this component to a constant value. Subsequent calls to `getMaximumSize` will always return this value; the component's UI will not be asked to compute it. Setting the maximum size to `null` restores the default behavior.

Overrides:

`setMaximumSize` in class `Component`

Parameters:

`maximumSize` - a `Dimension` containing the desired maximum allowable size

See Also:

`getMaximumSize()`

getMaximumSize

```
public Dimension getMaximumSize()
```

If the maximum size has been set to a non-`null` value just returns it. If the UI delegate's `getMaximumSize` method returns a non-`null` value then return that; otherwise defer to the component's layout manager.

Overrides:

`getMaximumSize` in class `Container`

Returns:

the value of the `maximumSize` property

See Also:

`setMaximumSize(java.awt.Dimension)`, `ComponentUI`

setMinimumSize

```
public void setMinimumSize(Dimension minimumSize)
```

Sets the minimum size of this component to a constant value. Subsequent calls to `getMinimumSize` will always return this value; the component's UI will not be asked to compute it. Setting the minimum size to `null` restores the default behavior.

Overrides:

`setMinimumSize` in class `Component`

Parameters:

`minimumSize` - the new minimum size of this component

See Also:

`getMinimumSize()`

getMinimumSize

```
public Dimension getMinimumSize()
```

If the minimum size has been set to a non-`null` value just returns it. If the UI delegate's `getMinimumSize` method returns a non-`null` value then return that; otherwise defer to the component's layout manager.

Overrides:

`getMinimumSize` in class `Container`

Returns:

the value of the `minimumSize` property

See Also:

`setMinimumSize(java.awt.Dimension)`, `ComponentUI`

contains

```
public boolean contains(int x,
                        int y)
```

Gives the UI delegate an opportunity to define the precise shape of this component for the sake of mouse processing.

Overrides:

`contains` in class `Component`

Parameters:

x - the *x* coordinate of the point

y - the *y* coordinate of the point

Returns:

true if this component logically contains x,y

See Also:

`Component.contains(int, int)`, `ComponentUI`

setBorder

`public void setBorder(Border border)`

Sets the border of this component. The `Border` object is responsible for defining the insets for the component (overriding any insets set directly on the component) and for optionally rendering any border decorations within the bounds of those insets. Borders should be used (rather than insets) for creating both decorative and non-decorative (such as margins and padding) regions for a swing component. Compound borders can be used to nest multiple borders within a single component.

Although technically you can set the border on any object that inherits from `JComponent`, the look and feel implementation of many standard Swing components doesn't work well with user-set borders. In general, when you want to set a border on a standard Swing component other than `JPanel` or `JLabel`, we recommend that you put the component in a `JPanel` and set the border on the `JPanel`.

This is a bound property.

Parameters:

`border` - the border to be rendered for this component

See Also:

`Border`, `CompoundBorder`

getBorder

`public Border getBorder()`

Returns the border of this component or `null` if no border is currently set.

Returns:

the border object for this component

See Also:

`setBorder(javax.swing.border.Border)`

getInsets

`public Insets getInsets()`

If a border has been set on this component, returns the border's insets; otherwise calls `super.getInsets`.

Overrides:

`getInsets` in class `Container`

Returns:

the value of the insets property

See Also:

`setBorder(javax.swing.border.Border)`

getInsets

`public Insets getInsets(Insets insets)`

Returns an `Insets` object containing this component's inset values. The passed-in `Insets` object will be reused if possible. Calling methods cannot assume that the same object will be returned, however. All existing values within this object are overwritten. If `insets` is null, this will allocate a new one.

Parameters:

`insets` - the `Insets` object, which can be reused

Returns:

the `Insets` object

See Also:

`getInsets()`

getAlignmentY

```
public float getAlignmentY()
```

Overrides `Container.getAlignmentY` to return the horizontal alignment.

Overrides:

`getAlignmentY` in class `Container`

Returns:

the value of the `alignmentY` property

See Also:

`setAlignmentY(float)`, `Component.getAlignmentY()`

setAlignmentY

```
public void setAlignmentY(float alignmentY)
```

Sets the the horizontal alignment.

Parameters:

`alignmentY` - the new horizontal alignment

See Also:

`getAlignmentY()`

getAlignmentX

```
public float getAlignmentX()
```

Overrides `Container.getAlignmentX` to return the vertical alignment.

Overrides:

`getAlignmentX` in class `Container`

Returns:

the value of the `alignmentX` property

See Also:

`setAlignmentX(float)`, `Component.getAlignmentX()`

setAlignmentX

```
public void setAlignmentX(float alignmentX)
```

Sets the the vertical alignment.

Parameters:

`alignmentX` - the new vertical alignment

See Also:

`getAlignmentX()`

setInputVerifier

```
public void setInputVerifier(InputVerifier inputVerifier)
```

Sets the input verifier for this component.

Parameters:

`inputVerifier` - the new input verifier

Since:

1.3

See Also:

`InputVerifier`

getInputVerifier

```
public InputVerifier getInputVerifier()
```

Returns the input verifier for this component.

Returns:

the `inputVerifier` property

Since:

1.3

See Also:

`InputVerifier`

getGraphics

```
public Graphics getGraphics()
```

Returns this component's graphics context, which lets you draw on a component. Use this method to get a `Graphics` object and then invoke operations on that object to draw on the component.

Overrides:

`getGraphics` in class `Component`

Returns:

this components graphics context

See Also:

`Component.paint(java.awt.Graphics)`

setDebugGraphicsOptions

```
public void setDebugGraphicsOptions(int debugOptions)
```

Enables or disables diagnostic information about every graphics operation performed within the component or one of its children.

Parameters:

`debugOptions` - determines how the component should display the information; one of the following options:
- DebugGraphics.LOG_OPTION - causes a text message to be printed.
- DebugGraphics.FLASH_OPTION - causes the drawing to flash several times.
- DebugGraphics.BUFFERED_OPTION - creates an `ExternalWindow` that displays the operations performed on the View's offscreen buffer.
- DebugGraphics.NONE_OPTION disables debugging.
- A value of 0 causes no changes to the debugging options.

`debugOptions` is bitwise OR'd into the current value

getDebugGraphicsOptions

```
public int getDebugGraphicsOptions()
```

Returns the state of graphics debugging.

Returns:

a bitwise OR'd flag of zero or more of the following options:
- DebugGraphics.LOG_OPTION - causes a text message to be printed.
- DebugGraphics.FLASH_OPTION - causes the drawing to flash several times.
- DebugGraphics.BUFFERED_OPTION - creates an `ExternalWindow` that displays the operations performed on the View's offscreen buffer.
- DebugGraphics.NONE_OPTION disables debugging.
- A value of 0 causes no changes to the debugging options.

See Also:

`setDebugGraphicsOptions(int)`

registerKeyboardAction

```
public void registerKeyboardAction(ActionListener anAction,
                        String aCommand,
                        KeyStroke aKeyStroke,
                        int aCondition)
```

This method is now obsolete, please use a combination of `getActionMap()` and `getInputMap()` for similiar behavior. For example, to bind the `KeyStroke aKeyStroke` to the `Action anAction` now use:

```
component.getInputMap().put(aKeyStroke, aCommand);
component.getActionMap().put(aCommmand, anAction);
```

The above assumes you want the binding to be applicable for `WHEN_FOCUSED`. To register bindings for other focus states use the `getInputMap` method that takes an integer.

Register a new keyboard action. `anAction` will be invoked if a key event matching `aKeyStroke` occurs and `aCondition` is verified. The `KeyStroke` object defines a particular combination of a keyboard key and one or more modifiers (alt, shift, ctrl, meta).

The `aCommand` will be set in the delivered event if specified.

The `aCondition` can be one of:

WHEN_FOCUSED

The action will be invoked only when the keystroke occurs while the component has the focus.

WHEN_IN_FOCUSED_WINDOW

The action will be invoked when the keystroke occurs while the component has the focus or if the component is in the window that has the focus. Note that the component need not be an immediate descendent of the window -- it can be anywhere in the window's containment hierarchy. In other words, whenever *any* component in the window has the focus, the action registered with this component is invoked.

WHEN_ANCESTOR_OF_FOCUSED_COMPONENT

The action will be invoked when the keystroke occurs while the component has the focus or if the component is an ancestor of the component that has the focus.

The combination of keystrokes and conditions lets you define high level (semantic) action events for a specified keystroke+modifier combination (using the KeyStroke class) and direct to a parent or child of a component that has the focus, or to the component itself. In other words, in any hierarchical structure of components, an arbitrary key-combination can be immediately directed to the appropriate component in the hierarchy, and cause a specific method to be invoked (usually by way of adapter objects).

If an action has already been registered for the receiving container, with the same charCode and the same modifiers, `anAction` will replace the action.

Parameters:

`anAction` - the `Action` to be registered

`aCommand` - the command to be set in the delivered event

`aKeyStroke` - the `KeyStroke` to bind to the action

`aCondition` - the condition that needs to be met, see above

See Also:

`KeyStroke`

registerKeyboardAction

```
public void registerKeyboardAction(ActionListener anAction,
                        KeyStroke aKeyStroke,
                        int aCondition)
```

This method is now obsolete, please use a combination of `getActionMap()` and `getInputMap()` for similiar behavior.

unregisterKeyboardAction

```
public void unregisterKeyboardAction(KeyStroke aKeyStroke)
```

This method is now obsolete. To unregister an existing binding you can either remove the binding from the `ActionMap/InputMap`, or place a dummy binding the `InputMap`. Removing the binding from the `InputMap` allows bindings in parent `InputMaps` to be active, whereas putting a dummy binding in the `InputMap` effectively disables the binding from ever happening.

Unregisters a keyboard action. This will remove the binding from the `ActionMap` (if it exists) as well as the `InputMaps`.

getRegisteredKeyStrokes

```
public KeyStroke[] getRegisteredKeyStrokes()
```

Returns the `KeyStrokes` that will initiate registered actions.

Returns:

an array of `KeyStroke` objects

See Also:

`registerKeyboardAction(java.awt.event.ActionListener, java.lang.String, javax.swing.KeyStroke, int)`

getConditionForKeyStroke

```
public int getConditionForKeyStroke(KeyStroke aKeyStroke)
```

Returns the condition that determines whether a registered action occurs in response to the specified keystroke.

For Java 2 platform v1.3, a `KeyStroke` can be associated with more than one condition. For example, 'a' could be bound for the two conditions `WHEN_FOCUSED` and `WHEN_IN_FOCUSED_WINDOW` condition.

Returns:

the action-keystroke condition

getActionForKeyStroke

```
public ActionListener getActionForKeyStroke(KeyStroke aKeyStroke)
```

Returns the object that will perform the action registered for a given keystroke.

Returns:

the `ActionListener` object invoked when the keystroke occurs

resetKeyboardActions

```
public void resetKeyboardActions()
```

Unregisters all the bindings in the first tier `InputMaps` and `ActionMap`. This has the effect of removing any local bindings, and allowing the bindings defined in parent `InputMap/ActionMaps` (the UI is usually defined in the second tier) to persist.

setInputMap

```
public final void setInputMap(int condition,
               InputMap map)
```

Sets the `InputMap` to use under the condition `condition` to `map`. A `null` value implies you do not want any bindings to be used, even from the UI. This will not reinstall the UI `InputMap` (if there was one). `condition` has one of the following values:

- `WHEN_IN_FOCUSED_WINDOW`
- `WHEN_FOCUSED`
- `WHEN_ANCESTOR_OF_FOCUSED_COMPONENT`

If `condition` is `WHEN_IN_FOCUSED_WINDOW` and `map` is not a `ComponentInputMap`, an `IllegalArgumentException` will be thrown. Similarly, if `condition` is not one of the values listed, an `IllegalArgumentException` will be thrown.

Parameters:

`condition` - one of the values listed above

`map` - the `InputMap` to use for the given condition

Throws:

`IllegalArgumentException` - if `condition` is `WHEN_IN_FOCUSED_WINDOW` and `map` is not an instance of `ComponentInputMap`; or if `condition` is not one of the legal values specified above

Since:

1.3

getInputMap

```
public final InputMap getInputMap(int condition)
```

Returns the `InputMap` that is used during `condition`.

Parameters:

`condition` - one of WHEN_IN_FOCUSED_WINDOW, WHEN_FOCUSED, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT

Returns:

the `InputMap` for the specified `condition`

Since:

1.3

getInputMap

```
public final InputMap getInputMap()
```

Returns the `InputMap` that is used when the component has focus. This is convenience method for `getInputMap(WHEN_FOCUSED)`.

Returns:

the `InputMap` used when the component has focus

Since:

1.3

setActionMap

```
public final void setActionMap(ActionMap am)
```

Sets the `ActionMap` to `am`. This does not set the parent of the `am` to be the `ActionMap` from the UI (if there was one), it is up to the caller to have done this.

Parameters:

`am` - the new `ActionMap`

Since:

1.3

getActionMap

```
public final ActionMap getActionMap()
```

Returns the `ActionMap` used to determine what `Action` to fire for particular `KeyStroke` binding. The returned `ActionMap`, unless otherwise set, will have the `ActionMap` from the UI set as the parent.

Returns:

the `ActionMap` containing the key/action bindings

Since:

1.3

getBaseline

```
public int getBaseline(int width,
                       int height)
```

Returns the baseline. The baseline is measured from the top of the component. This method is primarily meant for `LayoutManager`s to align components along their baseline. A return value less than 0 indicates this component does not have a reasonable baseline and that `LayoutManager`s should not align this component on its baseline.

This method calls into the `ComponentUI` method of the same name. If this component does not have a `ComponentUI` -1 will be returned. If a value >= 0 is returned, then the component has a valid baseline for any size >= the minimum size and `getBaselineResizeBehavior` can be used to determine how the baseline changes with size.

Overrides:

`getBaseline` in class `Component`

Parameters:

`width` - the width to get the baseline for

`height` - the height to get the baseline for

Returns:

the baseline or < 0 indicating there is no reasonable baseline

Throws:

`IllegalArgumentException` - if width or height is < 0

Since:

1.6

See Also:

`getBaselineResizeBehavior()`, `FontMetrics`

getBaselineResizeBehavior

```
public Component.BaselineResizeBehavior getBaselineResizeBehavior()
```

Returns an enum indicating how the baseline of the component changes as the size changes. This method is primarily meant for layout managers and GUI builders.

This method calls into the `ComponentUI` method of the same name. If this component does not have a `ComponentUI` `BaselineResizeBehavior.OTHER` will be returned. Subclasses should never return `null`; if the baseline can not be calculated return `BaselineResizeBehavior.OTHER`. Callers should first ask for the baseline using `getBaseline` and if a value >= 0 is returned use this method. It is acceptable for this method to return a value other than `BaselineResizeBehavior.OTHER` even if `getBaseline` returns a value less than 0.

Overrides:

`getBaselineResizeBehavior` in class `Component`

Returns:

an enum indicating how the baseline changes as the component size changes

Since:

1.6

See Also:

`getBaseline(int, int)`

requestDefaultFocus

```
@Deprecated
public boolean requestDefaultFocus()
```

Deprecated. *As of 1.4, replaced by* `FocusTraversalPolicy.getDefaultComponent(Container).requestFocus()`
In release 1.4, the focus subsystem was rearchitected. For more information, see How to Use the Focus Subsystem, a section in *The Java Tutorial*.

Requests focus on this `JComponent`'s `FocusTraversalPolicy`'s default `Component`. If this `JComponent` is a focus cycle root, then its `FocusTraversalPolicy` is used. Otherwise, the `FocusTraversalPolicy` of this `JComponent`'s focus-cycle-root ancestor is used.

See Also:

`FocusTraversalPolicy.getDefaultComponent(java.awt.Container)`

setVisible

```
public void setVisible(boolean aFlag)
```

Makes the component visible or invisible. Overrides `Component.setVisible`.

Overrides:

`setVisible` in class `Component`

Parameters:

`aFlag` - true to make the component visible; false to make it invisible

See Also:

`Component.isVisible()`, `Component.invalidate()`

setEnabled

```
public void setEnabled(boolean enabled)
```

Sets whether or not this component is enabled. A component that is enabled may respond to user input, while a component that is not enabled cannot respond to user input. Some components may alter their visual representation when they are disabled in order to provide feedback to the user that they cannot take input.

Note: Disabling a component does not disable its children.

Note: Disabling a lightweight component does not prevent it from receiving MouseEvents.

Overrides:

`setEnabled` in class `Component`

Parameters:

`enabled` - true if this component should be enabled, false otherwise

See Also:

`Component.isEnabled()`, `Component.isLightweight()`

setForeground

```
public void setForeground(Color fg)
```

Sets the foreground color of this component. It is up to the look and feel to honor this property, some may choose to ignore it.

Overrides:

`setForeground` in class `Component`

Parameters:

`fg` - the desired foreground `Color`

See Also:

`Component.getForeground()`

setBackground

```
public void setBackground(Color bg)
```

Sets the background color of this component. The background color is used only if the component is opaque, and only by subclasses of `JComponent` or `ComponentUI` implementations. Direct subclasses of `JComponent` must override `paintComponent` to honor this property.

It is up to the look and feel to honor this property, some may choose to ignore it.

Overrides:

`setBackground` in class `Component`

Parameters:

`bg` - the desired background `Color`

See Also:

`Component.getBackground()`, `setOpaque(boolean)`

setFont

```
public void setFont(Font font)
```

Sets the font for this component.

Overrides:

`setFont` in class `Container`

Parameters:

`font` - the desired `Font` for this component

See Also:

`Component.getFont()`

getDefaultLocale

```
public static Locale getDefaultLocale()
```

Returns the default locale used to initialize each JComponent's locale property upon creation. The default locale has "AppContext" scope so that applets (and potentially multiple lightweight applications running in a single VM) can have their own setting. An applet can safely alter its default locale because it will have no affect on other applets (or the browser).

Returns:

the default `Locale`.

Since:

1.4

See Also:

`setDefaultLocale(java.util.Locale)`, `Component.getLocale()`, `Component.setLocale(java.util.Locale)`

setDefaultLocale

```
public static void setDefaultLocale(Locale l)
```

Sets the default locale used to initialize each JComponent's locale property upon creation. The initial value is the VM's default locale. The default locale has "AppContext" scope so that applets (and potentially multiple lightweight applications running in a single VM) can have their own setting. An applet can safely alter its default locale because it will have no affect on other applets (or the browser).

Parameters:

`l` - the desired default `Locale` for new components.

Since:

1.4

See Also:

`getDefaultLocale()`, `Component.getLocale()`, `Component.setLocale(java.util.Locale)`

processComponentKeyEvent

```
protected void processComponentKeyEvent(KeyEvent e)
```

Processes any key events that the component itself recognizes. This is called after the focus manager and any interested listeners have been given a chance to steal away the event. This method is called only if the event has not yet been consumed. This method is called prior to the keyboard UI logic.

This method is implemented to do nothing. Subclasses would normally override this method if they process some key events themselves. If the event is processed, it should be consumed.

processKeyEvent

```
protected void processKeyEvent(KeyEvent e)
```

Overrides `processKeyEvent` to process events.

Overrides:

`processKeyEvent` in class `Component`

Parameters:

`e` - the key event

See Also:

`KeyEvent`, `KeyListener`, `KeyboardFocusManager`, `DefaultKeyboardFocusManager`, `Component.processEvent(java.awt.AWTEvent)`, `Component.dispatchEvent(java.awt.AWTEvent)`, `Component.addKeyListener(java.awt.event.KeyListener)`, `Component.enableEvents(long)`, `Component.isShowing()`

processKeyBinding

```
protected boolean processKeyBinding(KeyStroke ks,
                        KeyEvent e,
                        int condition,
                        boolean pressed)
```

Invoked to process the key bindings for `ks` as the result of the `KeyEvent e`. This obtains the appropriate `InputMap`, gets the binding, gets the action from the `ActionMap`, and then (if the action is found and the component is enabled) invokes `notifyAction` to notify the action.

Parameters:

`ks` - the `KeyStroke` queried

`e` - the `KeyEvent`

`condition` - one of the following values:
- JComponent.WHEN_FOCUSED
- JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT
- JComponent.WHEN_IN_FOCUSED_WINDOW

`pressed` - true if the key is pressed

Returns:

true if there was a binding to an action, and the action was enabled

Since:

1.3

setToolTipText

```
public void setToolTipText(String text)
```

Registers the text to display in a tool tip. The text displays when the cursor lingers over the component.

See How to Use Tool Tips in *The Java Tutorial* for further documentation.

Parameters:

`text` - the string to display; if the text is `null`, the tool tip is turned off for this component

See Also:

`TOOL_TIP_TEXT_KEY`

getToolTipText

```
public String getToolTipText()
```

Returns the tooltip string that has been set with `setToolTipText`.

Returns:

the text of the tool tip

See Also:

`TOOL_TIP_TEXT_KEY`

getToolTipText

```
public String getToolTipText(MouseEvent event)
```

Returns the string to be used as the tooltip for *event*. By default this returns any string set using `setToolTipText`. If a component provides more extensive API to support differing tooltips at different locations, this method should be overridden.

getToolTipLocation

```
public Point getToolTipLocation(MouseEvent event)
```

Returns the tooltip location in this component's coordinate system. If `null` is returned, Swing will choose a location. The default implementation returns `null`.

Parameters:

`event` - the `MouseEvent` that caused the `ToolTipManager` to show the tooltip

Returns:

always returns `null`

getPopupLocation

```
public Point getPopupLocation(MouseEvent event)
```

Returns the preferred location to display the popup menu in this component's coordinate system. It is up to the look and feel to honor this property, some may choose to ignore it. If `null`, the look and feel will choose a suitable location.

Parameters:

`event` - the `MouseEvent` that triggered the popup to be shown, or `null` if the popup is not being shown as the result of a mouse event

Returns:

location to display the `JPopupMenu`, or `null`

Since:

1.5

createToolTip

```
public JToolTip createToolTip()
```

Returns the instance of `JToolTip` that should be used to display the tooltip. Components typically would not override this method, but it can be used to cause different tooltips to be displayed differently.

Returns:

the `JToolTip` used to display this toolTip

scrollRectToVisible

```
public void scrollRectToVisible(Rectangle aRect)
```

Forwards the `scrollRectToVisible()` message to the `JComponent`'s parent. Components that can service the request, such as `JViewport`, override this method and perform the scrolling.

Parameters:

`aRect` - the visible `Rectangle`

See Also:

JViewport

setAutoscrolls

```
public void setAutoscrolls(boolean autoscrolls)
```

Sets the autoscrolls property. If true mouse dragged events will be synthetically generated when the mouse is dragged outside of the component's bounds and mouse motion has paused (while the button continues to be held down). The synthetic events make it appear that the drag gesture has resumed in the direction established when the component's boundary was crossed. Components that support autoscrolling must handle mouseDragged events by calling scrollRectToVisible with a rectangle that contains the mouse event's location. All of the Swing components that support item selection and are typically displayed in a JScrollPane (JTable, JList, JTree, JTextArea, and JEditorPane) already handle mouse dragged events in this way. To enable autoscrolling in any other component, add a mouse motion listener that calls scrollRectToVisible. For example, given a JPanel, myPanel:

```
MouseMotionListener doScrollRectToVisible = new MouseMotionAdapter() {
    public void mouseDragged(MouseEvent e) {
       Rectangle r = new Rectangle(e.getX(), e.getY(), 1, 1);
       ((JPanel)e.getSource()).scrollRectToVisible(r);
   }
};
myPanel.addMouseMotionListener(doScrollRectToVisible);
```

The default value of the autoScrolls property is false.

Parameters:

autoscrolls - if true, synthetic mouse dragged events are generated when the mouse is dragged outside of a component's bounds and the mouse button continues to be held down; otherwise false

See Also:

getAutoscrolls(), JViewport, JScrollPane

getAutoscrolls

```
public boolean getAutoscrolls()
```

Gets the autoscrolls property.

Returns:

the value of the autoscrolls property

See Also:

JViewport, setAutoscrolls(boolean)

setTransferHandler

```
public void setTransferHandler(TransferHandler newHandler)
```

Sets the TransferHandler, which provides support for transfer of data into and out of this component via cut/copy/paste and drag and drop. This may be null if the component does not support data transfer operations.

If the new TransferHandler is not null, this method also installs a **new** DropTarget on the component to activate drop handling through the TransferHandler and activate any built-in support (such as calculating and displaying potential drop locations). If you do not wish for this component to respond in any way to drops, you can disable drop support entirely either by removing the drop target (setDropTarget(null)) or by de-activating it (getDropTaget().setActive(false)).

If the new TransferHandler is null, this method removes the drop target.

Under two circumstances, this method does not modify the drop target: First, if the existing drop target on this component was explicitly set by the developer to a `non-null` value. Second, if the system property `suppressSwingDropSupport` is `true`. The default value for the system property is `false`.

Please see How to Use Drag and Drop and Data Transfer, a section in *The Java Tutorial*, for more information.

Parameters:

`newHandler` - the new `TransferHandler`

Since:

1.4

See Also:

`TransferHandler`, `getTransferHandler()`

getTransferHandler

`public TransferHandler getTransferHandler()`

Gets the `transferHandler` property.

Returns:

the value of the `transferHandler` property

Since:

1.4

See Also:

`TransferHandler`, `setTransferHandler(javax.swing.TransferHandler)`

processMouseEvent

`protected void processMouseEvent(MouseEvent e)`

Processes mouse events occurring on this component by dispatching them to any registered `MouseListener` objects, refer to `Component.processMouseEvent(MouseEvent)` for a complete description of this method.

Overrides:

`processMouseEvent` in class `Component`

Parameters:

`e` - the mouse event

Since:

1.5

See Also:

`Component.processMouseEvent(java.awt.event.MouseEvent)`

processMouseMotionEvent

`protected void processMouseMotionEvent(MouseEvent e)`

Processes mouse motion events, such as MouseEvent.MOUSE_DRAGGED.

Overrides:

processMouseMotionEvent in class Component

Parameters:

e - the MouseEvent

See Also:

MouseEvent

enable

```
@Deprecated
public void enable()
```

Deprecated. *As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).*

Overrides:

enable in class Component

disable

```
@Deprecated
public void disable()
```

Deprecated. *As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).*

Overrides:

disable in class Component

getAccessibleContext

```
public AccessibleContext getAccessibleContext()
```

Returns the AccessibleContext associated with this JComponent. The method implemented by this base class returns null. Classes that extend JComponent should implement this method to return the AccessibleContext associated with the subclass.

Overrides:

getAccessibleContext in class Component

Returns:

the AccessibleContext of this JComponent

getClientProperty

```
public final Object getClientProperty(Object key)
```

Returns the value of the property with the specified key. Only properties added with putClientProperty will return a non-null value.

Parameters:

key - the being queried

Returns:

the value of this property or null

See Also:

putClientProperty(java.lang.Object, java.lang.Object)

putClientProperty

```
public final void putClientProperty(Object key,
                    Object value)
```

Adds an arbitrary key/value "client property" to this component.

The get/putClientProperty methods provide access to a small per-instance hashtable. Callers can use get/putClientProperty to annotate components that were created by another module. For example, a layout manager might store per child constraints this way. For example:

```
 componentA.putClientProperty("to the left of", componentB);
```

If value is null this method will remove the property. Changes to client properties are reported with PropertyChange events. The name of the property (for the sake of PropertyChange events) is key.toString().

The clientProperty dictionary is not intended to support large scale extensions to JComponent nor should be it considered an alternative to subclassing when designing a new component.

Parameters:

key - the new client property key

value - the new client property value; if null this method will remove the property

See Also:

getClientProperty(java.lang.Object),
Container.addPropertyChangeListener(java.beans.PropertyChangeListener)

setFocusTraversalKeys

```
public void setFocusTraversalKeys(int id,
                        Set<? extends AWTKeyStroke> keystrokes)
```

Sets the focus traversal keys for a given traversal operation for this Component. Refer to Component.setFocusTraversalKeys(int, java.util.Set<? extends java.awt.AWTKeyStroke>) for a complete description of this method.

Overrides:

setFocusTraversalKeys in class Container

Parameters:

id - one of KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS, KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS, or KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS

keystrokes - the Set of AWTKeyStroke for the specified operation

Throws:

IllegalArgumentException - if id is not one of KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS, KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS, or KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS, or if keystrokes contains null, or if any Object in keystrokes is not an AWTKeyStroke, or if any keystroke represents a KEY_TYPED event, or if any keystroke already maps to another focus traversal operation for this Component

Since:

1.5

See Also:

KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS, KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS, KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS

isLightweightComponent

```
public static boolean isLightweightComponent(Component c)
```

Returns true if this component is lightweight, that is, if it doesn't have a native window system peer.

Returns:

true if this component is lightweight

reshape

```
@Deprecated
public void reshape(int x,
                    int y,
                    int w,
                    int h)
```

Deprecated. *As of JDK 5, replaced by `Component.setBounds(int, int, int, int)`.*

Moves and resizes this component.

Overrides:

reshape in class Component

Parameters:

x - the new horizontal location

y - the new vertical location

w - the new width

h - the new height

See Also:

Component.setBounds(int, int, int, int)

getBounds

```
public Rectangle getBounds(Rectangle rv)
```

Stores the bounds of this component into "return value" rv and returns rv. If rv is null a new Rectangle is allocated. This version of getBounds is useful if the caller wants to avoid allocating a new Rectangle object on the heap.

Overrides:

getBounds in class Component

Parameters:

rv - the return value, modified to the component's bounds

Returns:

rv; if rv is null return a newly created Rectangle with this component's bounds

getSize

```
public Dimension getSize(Dimension rv)
```

Stores the width/height of this component into "return value" `rv` and returns `rv`. If `rv` is `null` a new `Dimension` object is allocated. This version of `getSize` is useful if the caller wants to avoid allocating a new `Dimension` object on the heap.

Overrides:

`getSize` in class `Component`

Parameters:

`rv` - the return value, modified to the component's size

Returns:

`rv`

getLocation

```
public Point getLocation(Point rv)
```

Stores the x,y origin of this component into "return value" `rv` and returns `rv`. If `rv` is `null` a new `Point` is allocated. This version of `getLocation` is useful if the caller wants to avoid allocating a new `Point` object on the heap.

Overrides:

`getLocation` in class `Component`

Parameters:

`rv` - the return value, modified to the component's location

Returns:

`rv`

getX

```
public int getX()
```

Returns the current x coordinate of the component's origin. This method is preferable to writing `component.getBounds().x`, or `component.getLocation().x` because it doesn't cause any heap allocations.

Overrides:

`getX` in class `Component`

Returns:

the current x coordinate of the component's origin

getY

```
public int getY()
```

Returns the current y coordinate of the component's origin. This method is preferable to writing `component.getBounds().y`, or `component.getLocation().y` because it doesn't cause any heap allocations.

Overrides:

`getY` in class `Component`

Returns:

the current y coordinate of the component's origin

getWidth

```
public int getWidth()
```

Returns the current width of this component. This method is preferable to writing `component.getBounds().width`, or `component.getSize().width` because it doesn't cause any heap allocations.

Overrides:

`getWidth` in class `Component`

Returns:

the current width of this component

getHeight

```
public int getHeight()
```

Returns the current height of this component. This method is preferable to writing `component.getBounds().height`, or `component.getSize().height` because it doesn't cause any heap allocations.

Overrides:

`getHeight` in class `Component`

Returns:

the current height of this component

isOpaque

```
public boolean isOpaque()
```

Returns true if this component is completely opaque.

An opaque component paints every pixel within its rectangular bounds. A non-opaque component paints only a subset of its pixels or none at all, allowing the pixels underneath it to "show through". Therefore, a component that does not fully paint its pixels provides a degree of transparency.

Subclasses that guarantee to always completely paint their contents should override this method and return true.

Overrides:

`isOpaque` in class `Component`

Returns:

true if this component is completely opaque

See Also:

`setOpaque(boolean)`

setOpaque

```
public void setOpaque(boolean isOpaque)
```

If true the component paints every pixel within its bounds. Otherwise, the component may not paint some or all of its pixels, allowing the underlying pixels to show through.

The default value of this property is false for `JComponent`. However, the default value for this property on most standard `JComponent` subclasses (such as `JButton` and `JTree`) is look-and-feel dependent.

Parameters:

`isOpaque` - true if this component should be opaque

See Also:

`isOpaque()`

computeVisibleRect

```
public 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. The return value is stored in `visibleRect`.

Parameters:

`visibleRect` - a `Rectangle` computed as the intersection of all visible rectangles for this component and all of its ancestors -- this is the return value for this method

See Also:

`getVisibleRect()`

getVisibleRect

```
public Rectangle getVisibleRect()
```

Returns the `Component`'s "visible rectangle" - the intersection of this component's visible rectangle, `new Rectangle(0, 0, getWidth(), getHeight())`, and all of its ancestors' visible rectangles.

Returns:

the visible rectangle

firePropertyChange

```
public void firePropertyChange(String propertyName,
                               boolean oldValue,
                               boolean newValue)
```

Support for reporting bound property changes for boolean properties. This method can be called when a bound property has changed and it will send the appropriate PropertyChangeEvent to any registered PropertyChangeListeners.

Overrides:

`firePropertyChange` in class `Component`

Parameters:

`propertyName` - the property whose value has changed

`oldValue` - the property's previous value

`newValue` - the property's new value

firePropertyChange

```
public void firePropertyChange(String propertyName,
                       int oldValue,
                       int newValue)
```

Support for reporting bound property changes for integer properties. This method can be called when a bound property has changed and it will send the appropriate PropertyChangeEvent to any registered PropertyChangeListeners.

Overrides:

`firePropertyChange` in class `Component`

Parameters:

`propertyName` - the property whose value has changed

`oldValue` - the property's previous value

`newValue` - the property's new value

firePropertyChange

```
public void firePropertyChange(String propertyName,
                       char oldValue,
                       char newValue)
```

Description copied from class: `Component`
Reports a bound property change.

Overrides:

`firePropertyChange` in class `Component`

Parameters:

`propertyName` - the programmatic name of the property that was changed

`oldValue` - the old value of the property (as a char)

`newValue` - the new value of the property (as a char)

See Also:

`Component.firePropertyChange(java.lang.String, java.lang.Object, java.lang.Object)`

fireVetoableChange

```
protected void fireVetoableChange(String propertyName,
                       Object oldValue,
                       Object newValue)
                            throws PropertyVetoException
```

Supports reporting constrained property changes. This method can be called when a constrained property has changed and it will send the appropriate `PropertyChangeEvent` to any registered `VetoableChangeListeners`.

Parameters:

`propertyName` - the name of the property that was listened on

`oldValue` - the old value of the property

`newValue` - the new value of the property

Throws:

`PropertyVetoException` - when the attempt to set the property is vetoed by the component

addVetoableChangeListener

```
public void addVetoableChangeListener(VetoableChangeListener listener)
```

Adds a `VetoableChangeListener` to the listener list. The listener is registered for all properties.

Parameters:

`listener` - the `VetoableChangeListener` to be added

removeVetoableChangeListener

```
public void removeVetoableChangeListener(VetoableChangeListener listener)
```

Removes a `VetoableChangeListener` from the listener list. This removes a `VetoableChangeListener` that was registered for all properties.

Parameters:

`listener` - the `VetoableChangeListener` to be removed

getVetoableChangeListeners

```
public VetoableChangeListener[] getVetoableChangeListeners()
```

Returns an array of all the vetoable change listeners registered on this component.

Returns:

all of the component's `VetoableChangeListener`s or an empty array if no vetoable change listeners are currently registered

Since:

1.4

See Also:

`addVetoableChangeListener(java.beans.VetoableChangeListener)`, `removeVetoableChangeListener(java.beans.VetoableChangeListener)`

getTopLevelAncestor

```
public Container getTopLevelAncestor()
```

Returns the top-level ancestor of this component (either the containing `Window` or `Applet`), or `null` if this component has not been added to any container.

Returns:

the top-level `Container` that this component is in, or `null` if not in any container

addAncestorListener

```
public void 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. Events are also sent when the component or its ancestors are added or removed from the containment hierarchy.

Parameters:

`listener` - the `AncestorListener` to register

See Also:

`AncestorEvent`

removeAncestorListener

```
public void removeAncestorListener(AncestorListener listener)
```

Unregisters `listener` so that it will no longer receive `AncestorEvents`.

Parameters:

`listener` - the `AncestorListener` to be removed

See Also:

`addAncestorListener(javax.swing.event.AncestorListener)`

getAncestorListeners

```
public AncestorListener[] getAncestorListeners()
```

Returns an array of all the ancestor listeners registered on this component.

Returns:

all of the component's `AncestorListeners` or an empty array if no ancestor listeners are currently registered

Since:

1.4

See Also:

`addAncestorListener(javax.swing.event.AncestorListener)`, `removeAncestorListener(javax.swing.event.AncestorListener)`

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

Returns an array of all the objects currently registered as *Foo*`Listeners` upon this `JComponent`. *Foo*`Listeners` are registered using the `add`*Foo*`Listener` method.

You can specify the `listenerType` argument with a class literal, such as *Foo*`Listener.class`. For example, you can query a `JComponent c` for its mouse listeners with the following code:

```
MouseListener[] mls = (MouseListener[])(c.getListeners(MouseListener.class));
```

If no such listeners exist, this method returns an empty array.

Overrides:

`getListeners` in class `Container`

Parameters:

`listenerType` - the type of listeners requested; this parameter should specify an interface that descends from `java.util.EventListener`

Returns:

an array of all objects registered as *Foo*`Listeners` on this component, or an empty array if no such listeners have been added

Throws:

`ClassCastException` - if `listenerType` doesn't specify a class or interface that implements `java.util.EventListener`

Since:

1.3

See Also:

`getVetoableChangeListeners()`, `getAncestorListeners()`

addNotify

```
public void addNotify()
```

Notifies this component that it now has a parent component. When this method is invoked, the chain of parent components is set up with `KeyboardAction` event listeners. This method is called by the toolkit internally and should not be called directly by programs.

Overrides:

`addNotify` in class `Container`

See Also:

`registerKeyboardAction(java.awt.event.ActionListener, java.lang.String, javax.swing.KeyStroke, int)`

removeNotify

```
public void removeNotify()
```

Notifies this component that it no longer has a parent component. When this method is invoked, any `KeyboardActions` set up in the the chain of parent components are removed. This method is called by the toolkit internally and should not be called directly by programs.

Overrides:

`removeNotify` in class `Container`

See Also:

`registerKeyboardAction(java.awt.event.ActionListener, java.lang.String, javax.swing.KeyStroke, int)`

repaint

```
public 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. The component will be repainted after all of the currently pending events have been dispatched.

Overrides:

`repaint` in class `Component`

Parameters:

`tm` - this parameter is not used

`x` - the x value of the dirty region

`y` - the y value of the dirty region

`width` - the width of the dirty region

`height` - the height of the dirty region

See Also:

`isPaintingOrigin()`, `Component.isShowing()`, `RepaintManager.addDirtyRegion(javax.swing.JComponent, int, int, int, int)`

repaint

```
public void repaint(Rectangle r)
```

Adds the specified region to the dirty region list if the component is showing. The component will be repainted after all of the currently pending events have been dispatched.

Parameters:

`r` - a `Rectangle` containing the dirty region

See Also:

`isPaintingOrigin()`, `Component.isShowing()`, `RepaintManager.addDirtyRegion(javax.swing.JComponent, int, int, int, int)`

revalidate

```
public void revalidate()
```

Supports deferred automatic layout.

Calls `invalidate` and then adds this component's `validateRoot` to a list of components that need to be validated. Validation will occur after all currently pending events have been dispatched. In other words after this method is called, the first validateRoot (if any) found when walking up the containment hierarchy of this component will be validated. By default, `JRootPane`, `JScrollPane`, and `JTextField` return true from `isValidateRoot`.

This method will automatically be called on this component when a property value changes such that size, location, or internal layout of this component has been affected. This automatic updating differs from the AWT because programs generally no longer need to invoke `validate` to get the contents of the GUI to update.

Overrides:

`revalidate` in class `Component`

See Also:

`Component.invalidate()`, `Container.validate()`, `isValidateRoot()`, `RepaintManager.addInvalidComponent(javax.swing.JComponent)`

isValidateRoot

```
public boolean isValidateRoot()
```

If this method returns true, `revalidate` calls by descendants of this component will cause the entire tree beginning with this root to be validated. Returns false by default. `JScrollPane` overrides this method and returns true.

Overrides:

`isValidateRoot` in class `Container`

Returns:

always returns false

See Also:

`revalidate()`, `Component.invalidate()`, `Container.validate()`, `Container.isValidateRoot()`

isOptimizedDrawingEnabled

```
public boolean isOptimizedDrawingEnabled()
```

Returns true if this component tiles its children -- that is, if it can guarantee that the children will not overlap. The repainting system is substantially more efficient in this common case. `JComponent` subclasses that can't make this guarantee, such as `JLayeredPane`, should override this method to return false.

Returns:

always returns true

isPaintingOrigin

```
protected boolean isPaintingOrigin()
```

Returns `true` if a paint triggered on a child component should cause painting to originate from this Component, or one of its ancestors.

Calling `repaint(long, int, int, int, int)` or `paintImmediately(int, int, int, int)` on a Swing component will result in calling the `paintImmediately(int, int, int, int)` method of the first ancestor which `isPaintingOrigin()` returns `true`, if there are any.

`JComponent` subclasses that need to be painted when any of their children are repainted should override this method to return `true`.

Returns:

always returns `false`

See Also:

`paintImmediately(int, int, int, int)`

paintImmediately

```
public void paintImmediately(int x,
                             int y,
                             int w,
                             int h)
```

Paints the specified region in this component and all of its descendants that overlap the region, immediately.

It's rarely necessary to call this method. In most cases it's more efficient to call repaint, which defers the actual painting and can collapse redundant requests into a single paint call. This method is useful if one needs to update the display while the current event is being dispatched.

This method is to be overridden when the dirty region needs to be changed for components that are painting origins.

Parameters:

`x` - the x value of the region to be painted

`y` - the y value of the region to be painted

`w` - the width of the region to be painted

`h` - the height of the region to be painted

See Also:

repaint(long, int, int, int, int), isPaintingOrigin()

paintImmediately

public void paintImmediately(Rectangle r)

Paints the specified region now.

Parameters:

r - a Rectangle containing the region to be painted

setDoubleBuffered

public void setDoubleBuffered(boolean aFlag)

Sets whether this component should use a buffer to paint. If set to true, all the drawing from this component will be done in an offscreen painting buffer. The offscreen painting buffer will the be copied onto the screen. If a Component is buffered and one of its ancestor is also buffered, the ancestor buffer will be used.

Parameters:

aFlag - if true, set this component to be double buffered

isDoubleBuffered

public boolean isDoubleBuffered()

Returns whether this component should use a buffer to paint.

Overrides:

isDoubleBuffered in class Component

Returns:

true if this component is double buffered, otherwise false

getRootPane

public JRootPane getRootPane()

Returns the JRootPane ancestor for this component.

Returns:

the JRootPane that contains this component, or null if no JRootPane is found

paramString

protected String paramString()

Returns a string representation of this JComponent. This method is intended to be used only for debugging purposes, and the content and format of the returned string may vary between implementations. The returned string may be empty but may not be null.

Overrides:

`paramString` in class `Container`

Returns:

a string representation of this `JComponent`

hide

```
@Deprecated
public void hide()
```

Deprecated.

Overrides:

`hide` in class `Component`