21

java.awt.event Reference

21.1 ActionEvent ★

Description

Action events are fired off when the user performs an action on a component, such as pushing a button, double-clicking on a list item, or selecting a menu item. There is only one action event type, ACTION_PERFORMED.

Class Definition

```
public class java.awt.event.ActionEvent
   extends java.awt.AWTEvent {

  // Constants
  public final static int ACTION_FIRST;
  public final static int ACTION_LAST;
  public final static int ACTION_PERFORMED;
  public final static int ALT_MASK;
  public final static int CTRL_MASK;
  public final static int META_MASK;
  public final static int SHIFT_MASK;

  // Constructors
  public ActionEvent (Object source, int id, String command);
  public ActionEvent (Object source, int id, String command, int modifiers);

  // Instance Methods
  public String getActionCommand();
  public int getModifiers();
```

```
    public String paramString();
  }
```

Constants

ACTION_FIRST

`public final static int ACTION_FIRST`

Specifies the beginning range of action event ID values.

ACTION_LAST

`public final static int ACTION_LAST`

Specifies the ending range of action event ID values.

ACTION_PERFORMED

`public final static int ACTION_PERFORMED`

The only action event type; it indicates that the user has performed an action.

ALT_MASK

`public final static int ALT_MASK`

A constant representing the ALT key. ORed with other masks to form `modifiers` setting of an `AWTEvent`.

CTRL_MASK

`public final static int CTRL_MASK`

A constant representing the Control key. ORed with other masks to form `modifiers` setting of an `AWTEvent`.

META_MASK

`public final static int META_MASK`

A constant representing the META key. ORed with other masks to form `modifiers` setting of an `AWTEvent`.

SHIFT_MASK

```
public final static int SHIFT_MASK
```

A constant representing the Shift key. ORed with other masks to form `modifiers` setting of an `AWTEvent`.

Constructors

ActionEvent

```
public ActionEvent (Object source, int id, String command)
```

Parameters	*source*	The object that generated the event.
	id	The type ID of the event.
	command	The action command string.
Description	Constructs an `ActionEvent` with the given characteristics.	

```
public ActionEvent (Object source, int id, String command,
int modifiers)
```

Parameters	*source*	The object that generated the event.
	id	The type ID of the event.
	command	The action command string.
	modifiers	A combination of the key mask constants.
Description	Constructs an `ActionEvent` with the given characteristics.	

Instance Methods

getActionCommand

```
public String getActionCommand()
```

Returns	The action command string for this `ActionEvent`.
Description	Generally the action command string is the label of the component that generated the event. Also, when localization is necessary, the action command string can provide a setting that does not get localized.

getModifiers

```
public int getModifiers()
```

Returns	A combination of the key mask constants.
Description	Returns the modifier keys that were held down when this action was performed. This enables you to perform special processing if, for example, the user holds down Shift while pushing a button.

paramString

```
public String paramString()
```

Returns	String with current settings of `ActionEvent`.
Overrides	`AWTEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`ActionListener`, `AWTEvent`, `String`

21.2 ActionListener ★

Description

Objects that implement the `ActionListener` interface can receive `ActionEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.ActionListener
   extends java.util.EventListener {

  // Interface Methods
  public abstract void actionPerformed (ActionEvent e);
}
```

Interface Methods

actionPerformed

```
public abstract void actionPerformed (ActionEvent e)
```

Parameters	*e*	The action event that occurred.
Description	Notifies the `ActionListener` that an event occurred.	

See Also

`ActionEvent`, `AWTEventMulticaster`, `EventListener`

21.3 *AdjustmentEvent ★*

Description

AdjustmentEvents are generated by objects that implement the Adjustable interface. Scrollbar is one example of such an object.

Class Definition

```
public class java.awt.event.AdjustmentEvent
   extends java.awt.AWTEvent {

  // Constants
  public final static int ADJUSTMENT_FIRST;
  public final static int ADJUSTMENT_LAST;
  public final static int ADJUSTMENT_VALUE_CHANGED;
  public final static int BLOCK_DECREMENT;
  public final static int BLOCK_INCREMENT;
  public final static int TRACK;
  public final static int UNIT_DECREMENT;
  public final static int UNIT_INCREMENT;

  // Constructors
  public AdjustmentEvent (Adjustable source, int id, int type, int value);

  // Instance Methods
  public Adjustable getAdjustable();
  public int getAdjustmentType();
  public int getValue();
  public String paramString();
}
```

Constants

ADJUSTMENT_FIRST

```
public final static int ADJUSTMENT_FIRST
```

Specifies the beginning range of adjustment event ID values.

ADJUSTMENT_LAST

```
public final static int ADJUSTMENT_LAST
```

Specifies the ending range of adjustment event ID values.

ADJUSTMENT_VALUE_CHANGED

```
public final static int ADJUSTMENT_VALUE_CHANGED
```

Event type ID for value changed.

BLOCK_DECREMENT

```
public final static int BLOCK_DECREMENT
```

Adjustment type for block decrement.

BLOCK_INCREMENT

```
public final static int BLOCK_INCREMENT
```

Adjustment type for block increment.

TRACK

```
public final static int TRACK
```

Adjustment type for tracking.

UNIT_DECREMENT

```
public final static int UNIT_DECREMENT
```

Adjustment type for unit decrement.

UNIT_INCREMENT

```
public final static int UNIT_INCREMENT
```

Adjustment type for unit increment.

Constructors

AdjustmentEvent

```
public AdjustmentEvent (Adjustable source, int id, int
type, int value)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
	type	The type of adjustment event.
	value	The value of the `Adjustable` object.
Description	Constructs an `AdjustmentEvent` with the given characteristics.	

Instance Methods

getAdjustable

`public Adjustable getAdjustable()`

Returns	The source of this event.

getAdjustmentType

`public int getAdjustmentType()`

Returns	One of the adjustment type constants.
Description	The type will be `BLOCK_DECREMENT`, `BLOCK_INCREMENT`, `TRACK`, `UNIT_DECREMENT`, or `UNIT_INCREMENT`.

getValue

`public int getValue()`

Returns	The new value of the `Adjustable` object.

paramString

`public String paramString()`

Returns	String with current settings of the `AdjustmentEvent`.
Overrides	`AWTEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`Adjustable`, `AdjustmentListener`, `AWTEvent`, `Scrollbar`

21.4 AdjustmentListener ★

Description

Objects that implement the `AdjustmentListener` interface can receive `AdjustmentEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.AdjustmentListener
   extends java.util.Eventlistener {

  // Interface Methods
  public abstract void adjustmentValueChanged (AdjustmentEvent e);
}
```

Interface Methods

adjustmentPerformed

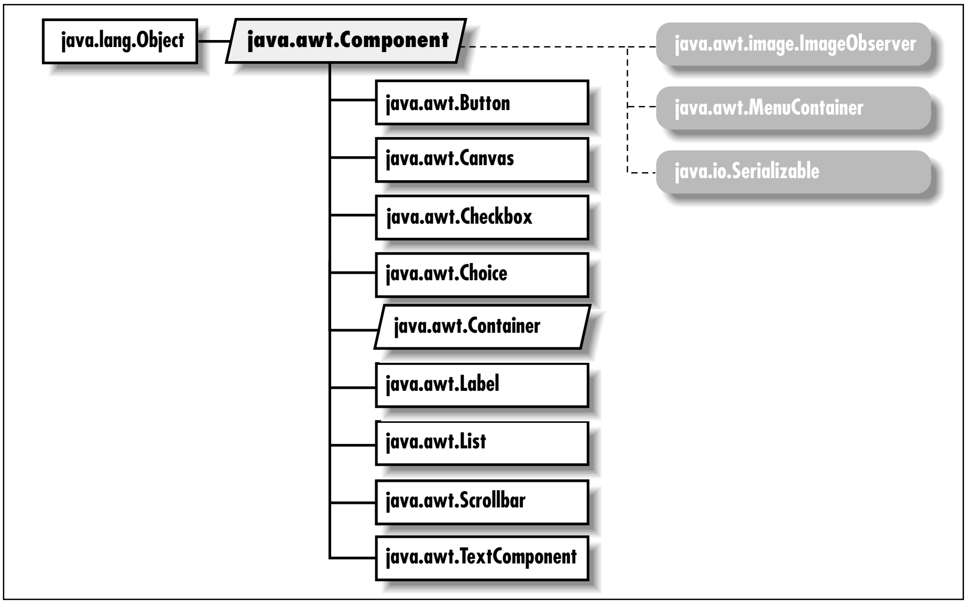
```
public abstract void adjustmentValueChanged
(AdjustmentEvent e)
```

Parameters	*e*	The adjustment event that occurred.
Description	Notifies the `AdjustmentListener` that an event occurred.	

See Also

`AdjustmentEvent`, `AWTEventMulticaster`, `EventListener`

21.5 ComponentAdapter ★



Description

`ComponentAdapter` is a class that implements the methods of `ComponentListener` with empty functions. It may be easier for you to extend `ComponentAdapter`, overriding only those methods you are interested in, than to implement `ComponentListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.ComponentAdapter
   extends java.lang.Object
   implements java.awt.event.ComponentListener {

  // Instance Methods
  public void componentHidden (ComponentEvent e);
  public void componentMoved (ComponentEvent e);
  public void componentResized (ComponentEvent e);
  public void componentShown (ComponentEvent e);
}
```

Instance Methods

componentHidden

`public void componentHidden (ComponentEvent e)`

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is hidden.	

componentMoved

`public void componentMoved (ComponentEvent e)`

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is moved.	

componentResized

`public void componentResized (ComponentEvent e)`

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is resized.	

componentShown

`public void componentShown (ComponentEvent e)`

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is shown.	

See Also

`Component`, `ComponentEvent`, `ComponentListener`

21.6 ComponentEvent ★

Description

Component events are generated when a component is shown, hidden, moved, or resized. AWT automatically deals with component moves and resizing; these events are provided only for notification. Subclasses of `ComponentEvent` deal with other specific component-level events.

Class Definition

```
public class java.awt.event.ComponentEvent
   extends java.awt.AWTEvent {

  // Constants
  public final static int COMPONENT_FIRST;
  public final static int COMPONENT_HIDDEN;
  public final static int COMPONENT_LAST;
  public final static int COMPONENT_MOVED;
  public final static int COMPONENT_RESIZED;
  public final static int COMPONENT_SHOWN;

  // Constructors
  public ComponentEvent (Component source, int id);

  // Instance Methods
  public Component getComponent();
  public String paramString();
}
```

Constants

COMPONENT_FIRST

```
public final static int COMPONENT_FIRST
```

Specifies the beginning range of component event ID values.

COMPONENT_HIDDEN

```
public final static int COMPONENT_HIDDEN
```

Event type ID indicating that the component was hidden.

COMPONENT_LAST

```
public final static int COMPONENT_LAST
```

Specifies the ending range of component event ID values.

COMPONENT_MOVED

```
public final static int COMPONENT_MOVED
```

Event type ID indicating that the component was moved.

COMPONENT_RESIZED

```
public final static int COMPONENT_RESIZED
```

Event type ID indicating that the component was resized.

COMPONENT_SHOWN

```
public final static int COMPONENT_SHOWN
```

Event type ID indicating that the component was shown.

Constructors

ComponentEvent

```
public ComponentEvent (Component source, int id)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
Description	Constructs a `ComponentEvent` with the given characteristics.	

Instance Methods

getComponent

```
public Component getComponent()
```

Returns	The source of this event.

paramString

```
public String paramString()
```

Returns	String with current settings of the `ComponentEvent`.
Overrides	`AWTEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

AWTEvent, Component, ComponentAdapter, ComponentListener, ContainerEvent, FocusEvent, InputEvent, PaintEvent, WindowEvent

21.7 ComponentListener ★

Description

Objects that implement the ComponentListener interface can receive ComponentEvent objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.ComponentListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void componentHidden (ComponentEvent e);
  public abstract void componentMoved (ComponentEvent e);
  public abstract void componentResized (ComponentEvent e);
  public abstract void componentShown (ComponentEvent e);
}
```

Interface Methods

componentHidden

`public abstract void componentHidden (ComponentEvent e)`

Parameters	*e*	The component event that occurred.
Description	Notifies the ComponentListener that a component was hidden.	

componentMoved

`public abstract void componentMoved (ComponentEvent e)`

Parameters	*e*	The component event that occurred.
Description	Notifies the ComponentListener that a component was moved.	

componentResized

```
public abstract void componentResized (ComponentEvent e)
```

Parameters	*e*	The component event that occurred.
Description	Notifies the `ComponentListener` that a component was resized.	

componentShown

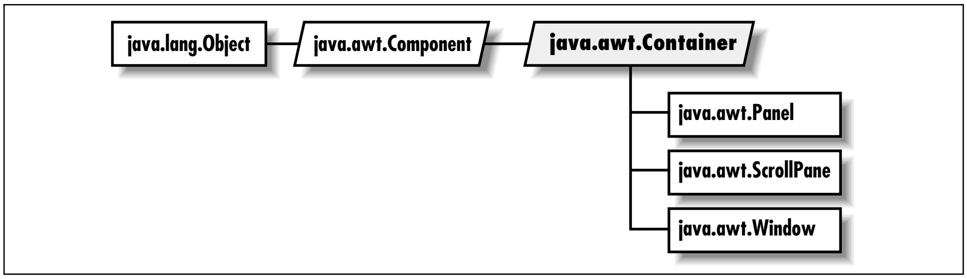
```
public abstract void componentShown (ComponentEvent e)
```

Parameters	*e*	The component event that occurred.
Description	Notifies the `ComponentListener` that a component was shown.	

See Also

`AWTEventMulticaster`, `ComponentAdapter`, `ComponentEvent`, `EventListener`

21.8 ContainerAdapter ★



Description

The `ContainerAdapter` class implements the methods of `ContainerListener` with empty functions. It may be easier for you to extend `ContainerAdapter`, overriding only those methods you are interested in, than to implement `ContainerListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.ContainerAdapter
    extends java.lang.Object
    implements java.awt.event.ContainerListener {

  // Instance Methods
```

```
    public void componentAdded (ContainerEvent e);
    public void componentRemoved (ContainerEvent e);
  }
```

Instance Methods

componentAdded

```
public void componentAdded (ComponentEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is added to a container.	

componentRemoved

```
public void componentRemoved (ComponentEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component is removed from a container.	

See Also

ContainerEvent, ContainerListener

21.9 ContainerEvent ★

Description

Container events are fired off when a component is added to or removed from a container. The AWT automatically deals with adding components to containers; these events are provided only for notification.

Class Definition

```
public class java.awt.event.ContainerEvent
   extends java.awt.event.ComponentEvent {

  // Constants
  public final static int COMPONENT_ADDED;
  public final static int COMPONENT_REMOVED;
  public final static int CONTAINER_FIRST;
  public final static int CONTAINER_LAST;

  // Constructors
  public ContainerEvent (Component source, int id, Component child);

  // Instance Methods
```

```
    public Component getChild();
    public Container getContainer();
    public String paramString();
  }
```

Constants

COMPONENT_ADDED

`public final static int COMPONENT_ADDED`

Event type ID indicating that a component was added to a container.

CONTAINER_FIRST

`public final static int CONTAINER_FIRST`

Specifies the beginning range of container event ID values.

CONTAINER_LAST

`public final static int CONTAINER_LAST`

Specifies the ending range of container event ID values.

COMPONENT_REMOVED

`public final static int COMPONENT_REMOVED`

Event type ID indicating that a component was removed from a container.

Constructors

ContainerEvent

```
public ContainerEvent (Component source, int id, Component
child)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
	child	The component that was added or removed.
Description	Constructs a `ContainerEvent` with the given characteristics.	

Instance Methods

getChild

`public Component getChild()`

Returns	The component that is being added or removed.

getContainer

```
public Container getContainer()
```

Returns	The container for this event.

paramString

```
public String paramString()
```

Returns	String with current settings of the `ContainerEvent`.
Overrides	`ComponentEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`Component`, `ComponentEvent`, `Container`, `ContainerAdapter`, `ContainerListener`

21.10 ContainerListener ★

Description

Objects that implement the `ContainerListener` interface can receive `ContainerEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.ContainerListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void componentAdded (ContainerEvent e);
  public abstract void componentRemoved (ContainerEvent e);
}
```

Interface Methods

componentAdded

```
public abstract void componentAdded (ContainerEvent e)
```

Parameters	*e*	The event that occurred.
Description	Notifies the `ContainerListener` that a component has been added to the container.	

componentRemoved

```
public abstract void componentRemoved (ContainerEvent e)
```

Parameters	*e*	The event that occurred.
Description	Notifies the `ContainerListener` that a component has been removed from the container.	

See Also

`ContainerAdapter`, `ContainerEvent`, `EventListener`

21.11 FocusAdapter ★

Description

The `FocusAdapter` class implements the methods of `FocusListener` with empty functions. It may be easier for you to extend `FocusAdapter`, overriding only those methods you are interested in, than to implement `FocusListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.FocusAdapter
    extends java.lang.Object
    implements java.awt.event.FocusListener {

   // Instance Methods
   public void focusGained (FocusEvent e);
   public void focusLost (FocusEvent e);
}
```

Instance Methods

focusGained

```
public void focusGained (FocusEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component gains focus.	

focusLost

```
public void focusLost (FocusEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a component loses focus.	

See Also

`FocusEvent`, `FocusListener`

21.12 FocusEvent ★

Description

Focus events are generated when a component gets or loses input focus. Focus events come in two flavors, permanent and temporary. Permanent focus events occur with explicit focus changes. For example, when the user tabs through components, this causes permanent focus events. An example of a temporary focus event is when a component loses focus as its containing window is deactivated.

Class Definition

```
public class java.awt.event.FocusEvent
   extends java.awt.event.ComponentEvent {

  // Constants
  public final static int FOCUS_FIRST;
  public final static int FOCUS_GAINED;
  public final static int FOCUS_LAST;
  public final static int FOCUS_LOST;

  // Constructors
  public FocusEvent (Component source, int id);
  public FocusEvent (Component source, int id, boolean temporary);

  // Instance Methods
  public boolean isTemporary();
  public String paramString();
}
```

Constants

`FOCUS_FIRST`

`public final static int FOCUS_FIRST`

Specifies the beginning range of focus event ID values.

`FOCUS_GAINED`

`public final static int FOCUS_GAINED`

Event type ID indicating that the component gained the input focus.

`FOCUS_LAST`

`public final static int FOCUS_LAST`

Specifies the ending range of focus event ID values.

`FOCUS_LOST`

`public final static int FOCUS_LOST`

Event type ID indicating that the component lost the input focus.

Constructors

`FocusEvent`

`public FocusEvent (Component source, int id)`

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
Description	Constructs a non-temporary `FocusEvent` with the given characteristics.	

```
public FocusEvent (Component source, int id, boolean
temporary)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
	temporary	A flag indicating whether this is a temporary focus event.
Description	Constructs a `FocusEvent` with the given characteristics.	

Instance Methods

`isTemporary`

```
public boolean isTemporary()
```

Returns	`true` if this is a temporary focus event; `false` otherwise.

`paramString`

```
public String paramString()
```

Returns	String with current settings of the `FocusEvent`.
Overrides	`ComponentEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`Component`, `ComponentEvent`, `FocusAdapter`, `FocusListener`

21.13 FocusListener ★

Description

Objects that implement the `FocusListener` interface can receive `FocusEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.FocusListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void focusGained (FocusEvent e);
  public abstract void focusLost (FocusEvent e);
}
```

Interface Methods

`focusGained`

```
public abstract void focusGained (FocusEvent e)
```

Parameters	*e*	The component event that occurred.
Description	Notifies the `FocusListener` that a component gained the input focus.	

focusLost

```
public abstract void focusLost (FocusEvent e)
```

Parameters	*e*	The component event that occurred.
Description	Notifies the `FocusListener` that a component lost the input focus.	

See Also

`AWTEventMulticaster`, `EventListener`, `FocusAdapter`, `FocusEvent`

21.14 InputEvent ★

Description

`InputEvent` is the root class for representing user input events. Input events are passed to listeners before the event source processes them. If one of the listeners consumes an event by using `consume()`, the event will not be processed by the event source peer.

Class Definition

```
public abstract class java.awt.event.InputEvent
   extends java.awt.event.ComponentEvent {

  // Constants
  public final static int ALT_MASK;
  public final static int BUTTON1_MASK;
  public final static int BUTTON2_MASK;
  public final static int BUTTON3_MASK;
  public final static int CTRL_MASK;
  public final static int META_MASK;
  public final static int SHIFT_MASK;

  // Instance Methods
  public void consume();
  public int getModifiers();
  public long getWhen();
  public boolean isAltDown();
  public boolean isConsumed();
  public boolean isControlDown();
  public boolean isMetaDown();
  public boolean isShiftDown();
}
```

Constants

ALT_MASK

```
public final static int ALT_MASK
```

The ALT key mask. ORed with other masks to form modifiers setting of event.

BUTTON1_MASK

```
public final static int BUTTON1_MASK
```

The mouse button 1 key mask. ORed with other masks to form modifiers setting of event.

BUTTON2_MASK

```
public final static int BUTTON2_MASK
```

The mouse button 2 key mask. ORed with other masks to form modifiers setting of event. This constant is identical to `ALT_MASK`.

BUTTON3_MASK

```
public final static int BUTTON3_MASK
```

The mouse button 3 key mask. ORed with other masks to form modifiers setting of event. This constant is identical to `ALT_MASK`.

CTRL_MASK

```
public final static int CTRL_MASK
```

The Control key mask. ORed with other masks to form modifiers setting of event.

META_MASK

```
public final static int META_MASK
```

The Meta key mask. ORed with other masks to form modifiers setting of event.

SHIFT_MASK

```
public final static int SHIFT_MASK
```

The Shift key mask. ORed with other masks to form modifiers setting of event.

Instance Methods

consume

`public void consume()`

Description	A consumed event will not be delivered to its source for default processing.

getModifiers

`public int getModifiers()`

Returns	The modifier flags, a combination of the `_MASK` constants.
Description	Use this method to find out what modifier keys were pressed when an input event occurred.

getWhen

`public long getWhen()`

Returns	The time at which this event occurred.
Description	The time of the event is returned as the number of milliseconds since the epoch (00:00:00 UTC, January 1, 1970). Conveniently, `java.util.Date` has a constructor that accepts such values.

isAltDown

`public boolean isAltDown()`

Returns	`true` if the Alt key was pressed; `false` otherwise.

isConsumed

`public boolean isConsumed()`

Returns	`true` if the event has been consumed; `false` otherwise.

isControlDown

`public boolean isControlDown()`

Returns	`true` if the Control key was pressed; `false` otherwise.

isMetaDown

`public boolean isMetaDown()`

Returns	`true` if the Meta key was pressed; `false` otherwise.

isShiftDown

```
public boolean isShiftDown()
```

Returns `true` if the Shift key was pressed; `false` otherwise.

See Also

`ComponentEvent`, `KeyEvent`, `MouseEvent`

21.15 ItemEvent ★

Description

`ItemEvents` are generated by objects that implement the `ItemSelectable` interface. `Choice` is one example of such an object.

Class Definition

```
public class java.awt.event.ItemEvent
   extends java.awt.AWTEvent {

  // Constants
  public final static int DESELECTED;
  public final static int ITEM_FIRST;
  public final static int ITEM_LAST;
  public final static int ITEM_STATE_CHANGED;
  public final static int SELECTED;

  // Constructors
  public ItemEvent (ItemSelectable source, int id, Object item, int stateChange);

// Instance Methods
  public Object getItem();
  public ItemSelectable getItemSelectable();
  public int getStateChange();
  public String paramString();
}
```

Constants

DESELECTED

```
public final static int DESELECTED
```

Indicates that an item was deselected.

ITEM_FIRST

```
public final static int ITEM_FIRST
```

Specifies the beginning range of item event ID values.

ITEM_LAST

```
public final static int ITEM_LAST
```

Specifies the ending range of item event ID values.

ITEM_STATE_CHANGED

```
public final static int ITEM_STATE_CHANGED
```

An event type indicating that an item was selected or deselected.

SELECTED

```
public final static int SELECTED
```

Indicates that an item was selected.

Constructors

ItemEvent

```
public ItemEvent (ItemSelectable source, int id, Object
item, int stateChange)
```

Parameters	*source*	The object that generated the event.
	id	The type ID of the event.
	item	The item whose state is changing.
	stateChange	Either `SELECTED` or `DESELECTED`.
Description	Constructs an `ItemEvent` with the given characteristics.	

Instance Methods

getItem

```
public Object getItem()
```

Returns	The item pertaining to this event.
Description	Returns the item whose changed state triggered this event.

getItemSelectable

```
public ItemSelectable getItemSelectable()
```

Returns	The source of this event.

Description	Returns an object that implements the `ItemSelectable` interface.

`getStateChange`

```
public int getStateChange()
```

Returns	The change in state that triggered this event. The new state is returned.
Description	This method will return `SELECTED` or `DESELECTED`.

`paramString`

```
public String paramString()
```

Returns	String with current settings of `ItemEvent`.
Overrides	`AWTEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`AWTEvent`, `ItemSelectable`, `ItemListener`

21.16 ItemListener ★

Description

Objects that implement the `ItemListener` interface can receive `ItemEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.ItemListener
   extends java.util.EventListener {

  // Interface Methods
  public abstract void itemStateChanged (ItemEvent e);
}
```

Interface Methods

itemStateChanged

```
public abstract void itemStateChanged (ItemEvent e)
```

Parameters	*e*	The item event that occurred.
Description	Notifies the `ItemListener` that an event occurred.	

See Also

`AWTEventMulticaster`, `EventListener`, `ItemEvent`

21.17 KeyAdapter ★

Description

The `KeyAdapter` class implements the methods of `KeyListener` with empty functions. It may be easier for you to extend `KeyAdapter`, overriding only those methods you are interested in, than to implement `KeyListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.KeyAdapter
   extends java.lang.Object
   implements java.awt.event.KeyListener {

  // Instance Methods
  public void keyPressed (KeyEvent e);
  public void keyReleased (KeyEvent e);
  public void keyTyped (KeyEvent e);
}
```

Instance Methods

keyPressed

```
public void keyPressed (KeyEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a key is pressed.	

keyReleased

```
public void keyReleased (KeyEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a pressed key is released.	

keyTyped

```
public void keyTyped (KeyEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a key has been pressed and released.	

See Also

KeyEvent, KeyListener

21.18 KeyEvent ★

Description

Key events are generated when the user types on the keyboard.

Class Definition

```
public class java.awt.event.KeyEvent
   extends java.awt.event.InputEvent {

  // Constants
  public final static int CHAR_UNDEFINED;
  public final static int KEY_FIRST;
  public final static int KEY_LAST;
  public final static int KEY_PRESSED;
  public final static int KEY_RELEASED;
  public final static int KEY_TYPED;
  public final static int VK_0;
  public final static int VK_1;
  public final static int VK_2;
  public final static int VK_3;
  public final static int VK_4;
  public final static int VK_5;
  public final static int VK_6;
  public final static int VK_7;
  public final static int VK_8;
  public final static int VK_9;
  public final static int VK_A;
  public final static int VK_ACCEPT;
  public final static int VK_ADD;
```

```
    public final static int VK_ALT;
    public final static int VK_B;
    public final static int VK_BACK_QUOTE;
    public final static int VK_BACK_SLASH;
    public final static int VK_BACK_SPACE;
    public final static int VK_C;
    public final static int VK_CANCEL;
    public final static int VK_CAPS_LOCK;
    public final static int VK_CLEAR;
    public final static int VK_CLOSE_BRACKET;
    public final static int VK_COMMA;
    public final static int VK_CONTROL;
    public final static int VK_CONVERT;
    public final static int VK_D;
    public final static int VK_DECIMAL;
    public final static int VK_DELETE;
    public final static int VK_DIVIDE;
    public final static int VK_DOWN;
    public final static int VK_E;
    public final static int VK_END;
    public final static int VK_ENTER;
    public final static int VK_EQUALS;
    public final static int VK_ESCAPE;
    public final static int VK_F;
    public final static int VK_F1;
    public final static int VK_F2;
    public final static int VK_F3;
    public final static int VK_F4;
    public final static int VK_F5;
    public final static int VK_F6;
    public final static int VK_F7;
    public final static int VK_F8;
    public final static int VK_F9;
    public final static int VK_F10;
    public final static int VK_F11;
    public final static int VK_F12;
    public final static int VK_FINAL;
    public final static int VK_G;
    public final static int VK_H;
    public final static int VK_HELP;
    public final static int VK_HOME;
    public final static int VK_I;
    public final static int VK_INSERT;
    public final static int VK_J;
    public final static int VK_K;
    public final static int VK_KANA;
    public final static int VK_KANJI;
    public final static int VK_L;
    public final static int VK_LEFT;
    public final static int VK_M;
    public final static int VK_META;
    public final static int VK_MODECHANGE;
    public final static int VK_MULTIPLY;
    public final static int VK_N;
    public final static int VK_NONCONVERT;
    public final static int VK_NUM_LOCK;
    public final static int VK_NUMPAD0;
    public final static int VK_NUMPAD1;
    public final static int VK_NUMPAD2;
    public final static int VK_NUMPAD3;
    public final static int VK_NUMPAD4;
    public final static int VK_NUMPAD5;
    public final static int VK_NUMPAD6;
    public final static int VK_NUMPAD7;
    public final static int VK_NUMPAD8;
    public final static int VK_NUMPAD9;
    public final static int VK_O;
    public final static int VK_OPEN_BRACKET;
    public final static int VK_P;
    public final static int VK_PAGE_DOWN;
    public final static int VK_PAGE_UP;
    public final static int VK_PAUSE;
    public final static int VK_PERIOD;
    public final static int VK_PRINTSCREEN;
    public final static int VK_Q;
    public final static int VK_QUOTE;
    public final static int VK_R;
    public final static int VK_RIGHT;
    public final static int VK_S;
    public final static int VK_SCROLL_LOCK;
    public final static int VK_SEMICOLON;
    public final static int VK_SEPARATER;
    public final static int VK_SHIFT;
    public final static int VK_SLASH;
    public final static int VK_SPACE;
    public final static int VK_SUBTRACT;
    public final static int VK_T;
    public final static int VK_TAB;
    public final static int VK_U;
    public final static int VK_UNDEFINED;
    public final static int VK_UP;
    public final static int VK_V;
    public final static int VK_W;
    public final static int VK_X;
    public final static int VK_Y;
    public final static int VK_Z;

    // Constructors
```

```
  public KeyEvent (Component source, int id, long when, int modifiers,
    int keyCode, char keyChar);

  // Class Methods
  public static String getKeyModifiersText(int modifiers);
  public static String getKeyText(int keyCode);

  // Instance Methods
  public char getKeyChar();
  public int getKeyCode();
  public boolean isActionKey();
  public String paramString();
  public void setKeyChar (char keyChar);
  public void setKeyCode (int keyCode);
  public void setModifiers (int modifiers);
}
```

Constants

CHAR_UNDEFINED

`public final static int CHAR_UNDEFINED`

This constant is used for key presses have that no associated character.

KEY_FIRST

`public final static int KEY_FIRST`

Specifies the beginning range of key event ID values.

KEY_LAST

`public final static int KEY_LAST`

Specifies the ending range of key event ID values.

KEY_PRESSED

`public final static int KEY_PRESSED`

An event ID type for a key press.

KEY_RELEASED

`public final static int KEY_RELEASED`

An event ID type for a key release.

KEY_TYPED

```
public final static int KEY_TYPED
```

An event ID type for a typed key (a press and a release).

`VK_0`

```
public final static int VK_0
```

The 0 key.

`VK_1`

```
public final static int VK_1
```

The 1 key.

`VK_2`

```
public final static int VK_2
```

The 2 key.

`VK_3`

```
public final static int VK_3
```

The 3 key.

`VK_4`

```
public final static int VK_4
```

The 4 key.

`VK_5`

```
public final static int VK_5
```

The 5 key.

`VK_6`

```
public final static int VK_6
```

The 6 key.

`VK_7`

```
public final static int VK_7
```

The 7 key.

VK_8

`public final static int VK_8`

The 8 key.

VK_9

`public final static int VK_9`

The 9 key.

VK_A

`public final static int VK_A`

The ‘a’ key.

VK_ACCEPT

`public final static int VK_ACCEPT`

This constant is used for Asian keyboards.

VK_ADD

`public final static int VK_ADD`

The plus (+) key on the numeric keypad.

VK_ALT

`public final static int VK_ALT`

The Alt key.

VK_B

`public final static int VK_B`

The ‘b’ key.

VK_BACK_QUOTE

`public final static int VK_BACK_QUOTE`

The backquote (‘) key.

VK_BACK_SLASH

`public final static int VK_BACK_SLASH`

The backslash key.

VK_BACK_SPACE

```
public final static int VK_BACK_SPACE
```

The Backspace key.

VK_C

```
public final static int VK_C
```

The ‘c’ key.

VK_CANCEL

```
public final static int VK_CANCEL
```

The Cancel key.

VK_CAPS_LOCK

```
public final static int VK_CAPS_LOCK
```

The Caps Lock key.

VK_CLEAR

```
public final static int VK_CLEAR
```

The Clear key.

VK_CLOSE_BRACKET

```
public final static int VK_CLOSE_BRACKET
```

The close bracket ‘]’ key.

VK_COMMA

```
public final static int VK_COMMA
```

The comma (,) key.

VK_CONTROL

```
public final static int VK_CONTROL
```

The Control key.

VK_CONVERT

```
public final static int VK_CONVERT
```

This constant is used for Asian keyboards.

VK_D

`public final static int VK_D`

The ‘d’ key.

VK_DECIMAL

`public final static int VK_DECIMAL`

The decimal (.) key on the numeric keypad.

VK_DELETE

`public final static int VK_DELETE`

The Delete key.

VK_DIVIDE

`public final static int VK_DIVIDE`

The divide (/) key on the numeric keypad.

VK_DOWN

`public final static int VK_DOWN`

The Down arrow key.

VK_E

`public final static int VK_E`

The ‘e’ key.

VK_END

`public final static int VK_END`

The End key.

VK_ENTER

`public final static int VK_ENTER`

The Enter key.

VK_EQUALS

`public final static int VK_ EQUALS`

The equals (=) key.

VK_ESCAPE

```
public final static int VK_ESCAPE
```

The Escape key.

VK_F

```
public final static int VK_F
```

The 'f' key.

VK_F1

```
public final static int VK_F1
```

The F1 key.

VK_F2

```
public final static int VK_F2
```

The F2 key.

VK_F3

```
public final static int VK_F3
```

The F3 key.

VK_F4

```
public final static int VK_F4
```

The F4 key.

VK_F5

```
public final static int VK_F5
```

The F5 key.

VK_F6

```
public final static int VK_F6
```

The F6 key.

VK_F7

```
public final static int VK_F7
```

The F7 key.

VK_F8

```
public final static int VK_F8
```

The F8 key.

VK_F9

```
public final static int VK_F9
```

The F9 key.

VK_F10

```
public final static int VK_F10
```

The F10 key.

VK_F11

```
public final static int VK_F11
```

The F11 key.

VK_F12

```
public final static int VK_F12
```

The F12 key.

VK_FINAL

```
public final static int VK_FINAL
```

This constant is used for Asian keyboards.

VK_G

```
public final static int VK_G
```

The ‘g’ key.

VK_H

```
public final static int VK_H
```

The ‘h’ key.

VK_HELP

```
public final static int VK_HELP
```

The Help key.

VK_HOME

```
public final static int VK_HOME
```

The Home key.

VK_I

```
public final static int VK_I
```

The ‘i’ key.

VK_INSERT

```
public final static int VK_INSERT
```

The Insert key.

VK_J

```
public final static int VK_J
```

The ‘j’ key.

VK_K

```
public final static int VK_K
```

The ‘k’ key.

VK_KANA

```
public final static int VK_KANA
```

This constant is used for Asian keyboards.

VK_KANJI

```
public final static int VK_KANJI
```

This constant is used for Asian keyboards.

VK_L

```
public final static int VK_L
```

The ‘l’ key.

VK_LEFT

```
public final static int VK_LEFT
```

The Left arrow key.

VK_M

`public final static int VK_M`

The ‘m’ key.

VK_MODECHANGE

`public final static int VK_MODECHANGE`

This constant is used for Asian keyboards.

VK_META

`public final static int VK_META`

The Meta key.

VK_MULTIPLY

`public final static int VK_MULTIPLY`

The * key on the numeric keypad.

VK_N

`public final static int VK_N`

The ‘n’ key.

VK_NONCONVERT

`public final static int VK_NONCONVERT`

This constant is used for Asian keyboards.

VK_NUM_LOCK

`public final static int VK_NUM_LOCK`

The Num Lock key.

VK_NUMPAD0

`public final static int VK_NUMPAD0`

The 0 key on the numeric keypad.

VK_NUMPAD1

`public final static int VK_NUMPAD1`

The 1 key on the numeric keypad.

VK_NUMPAD2

```
public final static int VK_NUMPAD2
```

The 2 key on the numeric keypad.

VK_NUMPAD3

```
public final static int VK_NUMPAD3
```

The 3 key on the numeric keypad.

VK_NUMPAD4

```
public final static int VK_NUMPAD4
```

The 4 key on the numeric keypad.

VK_NUMPAD5

```
public final static int VK_NUMPAD5
```

The 5 key on the numeric keypad.

VK_NUMPAD6

```
public final static int VK_NUMPAD6
```

The 6 key on the numeric keypad.

VK_NUMPAD7

```
public final static int VK_NUMPAD7
```

The 7 key on the numeric keypad.

VK_NUMPAD8

```
public final static int VK_NUMPAD8
```

The 8 key on the numeric keypad.

VK_NUMPAD9

```
public final static int VK_NUMPAD9
```

The 9 key on the numeric keypad.

VK_O

```
public final static int VK_O
```

The ‘o’ key.

VK_OPEN_BRACKET

`public final static int VK_OPEN_BRACKET`

The open bracket '[' key.

VK_P

`public final static int VK_P`

The 'p' key.

VK_PAGE_DOWN

`public final static int VK_PAGE_DOWN`

The Page Down key.

VK_PAGE_UP

`public final static int VK_PAGE_UP`

The Page Up key.

VK_PAUSE

`public final static int VK_PAUSE`

The Pause key.

VK_PERIOD

`public final static int VK_PERIOD`

The period (.) key.

VK_PRINTSCREEN

`public final static int VK_PRINTSCREEN`

The Print Screen key.

VK_Q

`public final static int VK_Q`

The 'q' key.

VK_QUOTE

`public final static int VK_QUOTE`

The quotation mark (") key.

VK_R

```
public final static int VK_R
```

The ‘r’ key.

VK_RIGHT

```
public final static int VK_RIGHT
```

The Right arrow key.

VK_S

```
public final static int VK_S
```

The ‘s’ key.

VK_SCROLL_LOCK

```
public final static int VK_SCROLL_LOCK
```

The Scroll Lock key.

VK_SEMICOLON

```
public final static int VK_SEMICOLON
```

The semicolon (;) key.

VK_SEPARATER

```
public final static int VK_SEPARATER
```

The numeric separator key on the numeric keypad (i.e., the locale-dependent key used to separate groups of digits). A misspelling of VK_SEPARATOR.

VK_SHIFT

```
public final static int VK_SHIFT
```

The Shift key.

VK_SLASH

```
public final static int VK_SLASH
```

The slash (/) key.

VK_SPACE

```
public final static int VK_SPACE
```

The space key.

VK_SUBTRACT

```
public final static int VK_SUBTRACT
```

The subtract (–) key on the numeric keypad.

VK_T

```
public final static int VK_T
```

The ‘t’ key.

VK_TAB

```
public final static int VK_TAB
```

The Tab key.

VK_U

```
public final static int VK_U
```

The ‘u’ key.

VK_UNDEFINED

```
public final static int VK_UNDEFINED
```

An undefined key.

VK_UP

```
public final static int VK_UP
```

The Up arrow key.

VK_V

```
public final static int VK_V
```

The ‘v’ key.

VK_W

```
public final static int VK_W
```

The ‘w’ key.

VK_X

```
public final static int VK_X
```

The ‘x’ key.

VK_Y

```
public final static int VK_Y
```

The ‘y’ key.

VK_Z

```
public final static int VK_Z
```

The ‘z’ key.

Constructors

KeyEvent

```
public KeyEvent (Component source, int id, long when, int
modifiers, int keyCode, char keyChar)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
	when	When the event occurred, in milliseconds from the epoch.
	modifiers	What modifier keys were pressed with this key.
	keyCode	The code of the key.
	keyChar	The character for this key.

Description Constructs a `KeyEvent` with the given characteristics.

Class Methods

getKeyModifiersText

```
public static String getKeyModifiersText(int modifiers)
```

Parameters	*modifiers*	One or more modifier keys.

Returns A string describing the modifiers.

getKeyText

```
public static String getKeyText(int keyCode)
```

Parameters	*keyCode*	One of the key codes.

Returns A string describing the given key.

Instance Methods

getKeyChar

```
public char getKeyChar()
```

Returns	The character corresponding to this event. KEY_TYPED events have characters.

getKeyCode

```
public int getKeyCode()
```

Returns	The integer key code corresponding to this event. This will be one of the constants defined above. KEY_PRESSED and KEY_RELEASED events have codes. Key codes are virtual keys, not actual. Pressing the 'a' key is identical to 'A', but has different modifiers. Same for '/' and '?' on a standard keyboard.

isActionKey

```
public boolean isActionKey()
```

Returns	`true` if this event is for one of the action keys; `false` otherwise.
Description	In general, an action key is a key that causes an action but has no printing equivalent. The action keys are the function keys, the arrow keys, Caps Lock, End, Home, Insert, Num Lock, Pause, Page Down, Page Up, Print Screen, and Scroll Lock. They do not generate a KEY_TYPED event, only KEY_PRESSED and KEY_RELEASED.

paramString

```
public String paramString()
```

Returns	A string with current settings of the `KeyEvent`.
Overrides	`ComponentEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

setKeyChar

```
public void setKeyChar(char keyChar)
```

Parameters	*keyChar*	The new key character.
Description	Sets the character code of this `KeyEvent`.	

setKeyCode

```
public void setKeyCode (int keyCode)
```

Parameters	*keyCode*	The new key code.
Description	Sets the key code of this `KeyEvent`.	

setModifiers

```
public void setModifiers (int modifiers)
```

Parameters	*modifiers*	The new modifiers.
Description	This is a combination of the mask constants defined in `java.awt.event.InputEvent`.	

See Also

`Component`, `ComponentEvent`, `InputEvent`, `KeyAdapter`, `KeyListener`

21.19 KeyListener ★

Description

Objects that implement the `KeyListener` interface can receive `KeyEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.KeyListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void keyPressed (KeyEvent e);
  public abstract void keyReleased (KeyEvent e);
  public abstract void keyTyped (KeyEvent e);
}
```

Interface Methods

keyPressed

```
public abstract void keyPressed (KeyEvent e)
```

Parameters	*e*	The key event that occurred.
Description	Notifies the `KeyListener` that a key was pressed.	

keyReleased

```
public abstract void keyReleased (KeyEvent e)
```

Parameters	*e*	The key event that occurred.
Description	Notifies the `KeyListener` that a key was released.	

keyTyped

```
public abstract void keyTyped (KeyEvent e)
```

Parameters	*e*	The key event that occurred.
Description	Notifies the `KeyListener` that a key was typed (pressed and released).	

See Also

`AWTEventMulticaster`, `EventListener`, `KeyEvent`, `KeyListener`

21.20 MouseAdapter ★

Description

The `MouseAdapter` class implements the methods of `MouseListener` with empty functions. It may be easier for you to extend `MouseAdapter`, overriding only those methods you are interested in, than to implement `MouseListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.MouseAdapter
   extends java.lang.Object
   implements java.awt.event.MouseListener {

  // Instance Methods
  public void mouseClicked (MouseEvent e);
  public void mouseEntered (MouseEvent e);
  public void mouseExited (MouseEvent e);
  public void mousePressed (MouseEvent e);
  public void mouseReleased (MouseEvent e);
}
```

Instance Methods

mouseClicked

```
public void mouseClicked (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the mouse button is clicked (pressed and released).	

mouseEntered

```
public void mouseEntered (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the user moves the mouse cursor into a component.	

mouseExited

```
public void mouseExited (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the moves the mouse cursor out of a component.	

mousePressed

```
public void mousePressed (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the mouse button is pressed.	

mouseReleased

```
public void mouseReleased (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the mouse button is released.	

See Also

`MouseEvent`, `MouseListener`

21.21 MouseEvent ★

Description

Mouse events are generated when the user moves and clicks the mouse.

Class Definition

```
public class java.awt.event.MouseEvent
    extends java.awt.event.InputEvent {

  // Constants
  public final static int MOUSE_CLICKED;
  public final static int MOUSE_DRAGGED;
  public final static int MOUSE_ENTERED;
  public final static int MOUSE_EXITED;
  public final static int MOUSE_FIRST;
  public final static int MOUSE_LAST;
  public final static int MOUSE_MOVED;
  public final static int MOUSE_PRESSED;
  public final static int MOUSE_RELEASED;

  // Constructors
  public MouseEvent (Component source, int id, long when, int modifiers, int x,
    int y, int clickCount, boolean popupTrigger);

  // Instance Methods
  public int getClickCount();
  public synchronized Point getPoint();
  public int getX();
  public int getY();
  public boolean isPopupTrigger();
  public String paramString();
  public synchronized void translatePoint (int x, int y);
}
```

Constants

MOUSE_CLICKED

```
public final static int MOUSE_CLICKED
```

An event type ID indicating a mouse click.

MOUSE_DRAGGED

```
public final static int MOUSE_DRAGGED
```

An event type ID indicating a mouse move with the button held down.

MOUSE_ENTERED

```
public final static int MOUSE_ENTERED
```

An event type ID indicating that a mouse entered a component.

MOUSE_EXITED

```
public final static int MOUSE_EXITED
```

An event type ID indicating that a mouse left a component.

MOUSE_FIRST

```
public final static int MOUSE_FIRST
```

Specifies the beginning range of mouse event ID values.

MOUSE_LAST

```
public final static int MOUSE_LAST
```

Specifies the ending range of mouse event ID values.

MOUSE_MOVED

```
public final static int MOUSE_MOVED
```

An event type ID indicating a mouse move.

MOUSE_PRESSED

```
public final static int MOUSE_PRESSED
```

An event type ID indicating a mouse button press.

MOUSE_RELEASED

```
public final static int MOUSE_RELEASED
```

An event type ID indicating a mouse button release.

Constructors

MouseEvent

```
public MouseEvent (Component source, int id, long when,
int modifiers, int x, int y, int clickCount, boolean
popupTrigger)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
	when	When the event occurred, in milliseconds from the epoch.
	modifiers	What modifier keys were pressed with this key.
	x	The horizontal location of the event.
	y	The vertical location of the event.
	clickCount	The number of times the mouse button has been clicked.
	popupTrigger	A flag indicating if this event is a popup trigger event.

Description Constructs a `MouseEvent` with the given characteristics.

Instance Methods

getClickCount

```
public int getClickCount()
```

Returns The number of consecutive mouse button clicks for this event.

getPoint

```
public synchronized Point getPoint()
```

Returns The location where the event happened.

getX

```
public int getX()
```

Returns The horizontal location where the event happened.

getY

```
public int getY()
```

Returns The vertical location where the event happened.

isPopupTrigger

```
public boolean isPopupTrigger()
```

Returns Returns `true` if this event is the popup menu event for the run-time system.

paramString

`public String paramString()`

Returns	String with current settings of the `MouseEvent`.
Overrides	`ComponentEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

translatePoint

`public synchronized void translatePoint (int x, int y)`

Parameters	*x*	The horizontal amount of translation.
	y	The vertical amount of translation.
Description	Translates the location of the event by the given amounts.	

See Also

`Component`, `ComponentEvent`, `InputEvent`, `MouseAdapter`, `MouseListener`, `Point`

21.22 MouseListener ★

Description

Objects that implement the `MouseListener` interface can receive non-motion oriented `MouseEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.MouseListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void mouseClicked (MouseEvent e);
  public abstract void mouseEntered (MouseEvent e);
  public abstract void mouseExited (MouseEvent e);
  public abstract void mousePressed (MouseEvent e);
  public abstract void mouseReleased (MouseEvent e);
}
```

Interface Methods

`mouseClicked`

`public abstract void mouseClicked (MouseEvent e)`

Parameters	*e*	The key event that occurred.
Description	Notifies the `MouseListener` that the mouse button was clicked (pressed and released).	

`mouseEntered`

`public abstract void mouseEntered (MouseEvent e)`

Parameters	*e*	The key event that occurred.
Description	Notifies the `MouseListener` that the mouse cursor has been moved into a component's coordinate space.	

`mouseExited`

`public abstract void mouseExited (MouseEvent e)`

Parameters	*e*	The key event that occurred.
Description	Notifies the `MouseListener` that the mouse cursor has been moved out of a component's coordinate space.	

`mousePressed`

`public abstract void mousePressed (MouseEvent e)`

Parameters	*e*	The key event that occurred.
Description	Notifies the `MouseListener` that the mouse button was pressed.	

`mouseReleased`

`public abstract void mouseReleased (MouseEvent e)`

Parameters	*e*	The key event that occurred.
Description	Notifies the `MouseListener` that the mouse button was released.	

See Also

`EventListener`, `MouseAdapter`, `MouseEvent`

21.23 MouseMotionAdapter ★

Description

The `MouseMotionAdapter` class implements the methods of `MouseMotionListener` with empty functions. It may be easier for you to extend `MouseMotionAdapter`, overriding only those methods you are interested in, than to implement `MouseMotionListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.MouseMotionAdapter
    extends java.lang.Object
    implements java.awt.event.MouseMotionListener {

  // Instance Methods
  public void mouseDragged (MouseEvent e);
  public void mouseMoved (MouseEvent e);
}
```

Instance Methods

mouseDragged

```
public void mouseDragged (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the mouse is dragged.	

mouseMoved

```
public void mouseEntered (MouseEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when the mouse moves.	

See Also

`MouseEvent`, `MouseMotionListener`

21.24 MouseMotionListener ★

Description

Objects that implement the MouseMotionListener interface can receive motion-oriented MouseEvent objects. Listeners must first register themselves with objects that produce events. When events occur, they are automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.MouseMotionListener
   extends java.util.EventListener {

  // Instance Methods
  public abstract void mouseDragged (MouseEvent e);
  public abstract void mouseMoved (MouseEvent e);
}
```

Interface Methods

mouseDragged

```
public abstract void mouseDragged (MouseEvent e)
```

Parameters	*e*	The key event that occurred.
Description	Notifies the MouseMotionListener that the mouse has been dragged.	

mouseMoved

```
public abstract void mouseMoved (MouseEvent e)
```

Parameters	*e*	The key event that occurred.
Description	Notifies the MouseMotionListener that the mouse has been moved.	

See Also

AWTEventMulticaster, EventListener, MouseEvent, MouseMotionAdapter

21.25 PaintEvent ★

Description

The `PaintEvent` class represents the paint and update operations that the AWT performs on components. There is no `PaintListener` interface, so the only way to catch these events is to override `paint(Graphics)` and `update(Graphics)` in `Component`. This class exists so that paint events will get serialized properly.

Class Definition

```
public class java.awt.event.PaintEvent
   extends java.awt.event.ComponentEvent {

  // Constants
  public final static int PAINT;
  public final static int PAINT_FIRST;
  public final static int PAINT_LAST;
  public final static int UPDATE;

  // Constructor
  public PaintEvent (Component source, int id, Rectangle updateRect);

  // Instance Methods
  public Rectangle getUpdateRect();
  public String paramString();
  public void setUpdateRect (Rectangle updateRect);
}
```

Class Definition

```
public class java.awt.event.PaintEvent
        extends java.awt.event.ComponentEvent {

  // Constants
  public final static int PAINT;
  public final static int PAINT_FIRST;
  public final static int PAINT_LAST;
  public final static int UPDATE;

  //Constructor
  public PaintEvent (Component source, int id, Rectangle updateRect);

  // Instance Methods
  public Rectangle getUpdateRect();
  public String paramString();
  public void setUpdateRect (Rectangle updateRect);
}
```

Constants

PAINT

```
public final static int PAINT
```

The paint event type.

PAINT_FIRST

```
public final static int PAINT_FIRST
```

Specifies the beginning range of paint event ID values.

PAINT_LAST

```
public final static int PAINT_LAST
```

Specifies the ending range of paint event ID values.

UPDATE

```
public final static int UPDATE
```

The update event type.

Constructor

PaintEvent

```
public PaintEvent (Component source, ind id, Rectangle
updateRect)
```

Parameters	*source*	The source of the event.
	id	The event type ID.
	g	The rectangular area to paint.
Description	Constructs a `PaintEvent` with the given characteristics.	

Instance Methods

getUpdateRect

```
public Rectangle getUpdateRect()
```

Returns	The rectangular area that needs painting.

paramString

```
public String paramString()
```

Returns	String with current settings of the `PaintEvent`.
Overrides	`ComponentEvent.paramString()`

Description Helper method for `toString()` to generate string of current settings.

`setUpdateRect`

`public void setUpdateRect (Rectangle updateRect)`

Parameters	*updateRect*	The rectangular area to paint.
Description	Changes the rectangular area that this `PaintEvent` will paint.	

See Also

`Component`, `ComponentEvent`, `Graphics`

21.26 TextEvent ★

Description

Text events are generated by text components when their contents change, either programmatically or by a user typing.

Class Definition

```
public class java.awt.event.TextEvent
   extends java.awt.AWTEvent {

  // Constants
  public final static int TEXT_FIRST;
  public final static int TEXT_LAST;
  public final static int TEXT_VALUE_CHANGED;

  // Constructors
  public TextEvent (Object source, int id);

  // Instance Methods
  public String paramString();
}
```

Constants

`TEXT_FIRST`

`public final static int TEXT_FIRST`

Specifies the beginning range of text event ID values.

`TEXT_LAST`

`public final static int TEXT_LAST`

Specifies the ending range of text event ID values.

`TEXT_VALUE_CHANGED`

`public final static int TEXT_VALUE_CHANGED`

The only text event type; it indicates that the contents of something have changed.

Constructors

`TextEvent`

`public TextEvent (Object source, int id)`

Parameters	*source*	The object that generated the event.
	id	The type ID of the event.
Description	Constructs a `TextEvent` with the given characteristics.	

Instance Methods

`paramString`

`public String paramString()`

Returns	String with current settings of the `TextEvent`.
Overrides	`AWTEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`AWTEvent`, `TextListener`

21.27 TextListener ★

Description

Objects that implement the `TextListener` interface can receive `TextEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.TextListener
   extends java.util.EventListener {

  // Interface Methods
  public abstract void textValueChanged (TextEvent e);
}
```

Interface Methods

`textValueChanged`

`public abstract void textValueChanged (TextEvent e)`

Parameters	*e*	The text event that occurred.
Description	Notifies the `TextListener` that an event occurred.	

See Also

`AWTEventMulticaster`, `EventListener`, `TextEvent`

21.28 WindowAdapter ★

Description

The `WindowAdapter` class implements the methods of `WindowListener` with empty functions. It may be easier for you to extend `WindowAdapter`, overriding only those methods you are interested in, than to implement `WindowListener` and provide the empty functions yourself.

Class Definition

```
public abstract class java.awt.event.WindowAdapter
   extends java.lang.Object
   implements java.awt.event.WindowListener {

  // Instance Methods
  public void windowActivated (WindowEvent e);
  public void windowClosed (WindowEvent e);
  public void windowClosing (WindowEvent e);
  public void windowDeactivated (WindowEvent e);
  public void windowDeiconified (WindowEvent e);
  public void windowIconified (WindowEvent e);
  public void windowOpened (WindowEvent e);
}
```

Instance Methods

`windowActivated`

```
public void windowActivated (WindowEvent e)
```

Parameters	*e*	The event that has occurred.

Description Does nothing. Override this function to be notified when a window is activated.

`windowClosed`

```
public void windowClosed (WindowEvent e)
```

Parameters	*e*	The event that has occurred.

Description Does nothing. Override this function to be notified when a window is closed.

`windowClosing`

```
public void windowClosing (WindowEvent e)
```

Parameters	*e*	The event that has occurred.

Description Does nothing. Override this function to be notified when a window is in the process of closing.

`windowDeactivated`

```
public void windowDeactivated (WindowEvent e)
```

Parameters	*e*	The event that has occurred.

Description Does nothing. Override this function to be notified when a window is deactivated.

`windowDeiconified`

```
public void windowDeiconified (WindowEvent e)
```

Parameters	*e*	The event that has occurred.

Description Does nothing. Override this function to be notified when an iconified window is restored.

windowIconified

```
public void windowIconified (WindowEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a window is iconified (minimized).	

windowOpened

```
public void windowOpened (WindowEvent e)
```

Parameters	*e*	The event that has occurred.
Description	Does nothing. Override this function to be notified when a window is opened.	

See Also

`WindowEvent`, `WindowListener`

21.29 WindowEvent ★

Description

Window events are generated when a window is opened, closed, iconified, or deiconified.

Class Definition

```
public class java.awt.event.WindowEvent
   extends java.awt.event.ComponentEvent {

  // Constants
  public final static int WINDOW_ACTIVATED;
  public final static int WINDOW_CLOSED;
  public final static int WINDOW_CLOSING;
  public final static int WINDOW_DEACTIVATED;
  public final static int WINDOW_DEICONIFIED;
  public final static int WINDOW_FIRST;
  public final static int WINDOW_ICONIFIED;
  public final static int WINDOW_LAST;
  public final static int WINDOW_OPENED;

  // Constructors
  public WindowEvent (Window source, int id);

  // Instance Methods
  public Window getWindow();
  public String paramString();
}
```

Constants

`WINDOW_ACTIVATED`

```
public final static int WINDOW_ACTIVATED
```

Event type ID indicating the window has been activated, brought to the foreground.

`WINDOW_CLOSED`

```
public final static int WINDOW_CLOSED
```

Event type ID indicating the window has closed.

`WINDOW_CLOSING`

```
public final static int WINDOW_CLOSING
```

Event type ID indicating the window is closing.

`WINDOW_DEACTIVATED`

```
public final static int WINDOW_DEACTIVATED
```

Event type ID indicating the window has been deactivated, placed in the background.

`WINDOW_DEICONIFIED`

```
public final static int WINDOW_DEICONIFIED
```

Event type ID indicating the window has been restored from an iconified state.

`WINDOW_FIRST`

```
public final static int WINDOW_FIRST
```

Specifies the beginning range of window event ID values.

`WINDOW_ICONIFIED`

```
public final static int WINDOW_ICONIFIED
```

Event type ID indicating the window has been iconified (minimized).

`WINDOW_LAST`

```
public final static int WINDOW_LAST
```

Specifies the ending range of window event ID values.

WINDOW_OPENED

```
public final static int WINDOW_OPENED
```

Event type ID indicating the window has opened.

Constructors

WindowEvent

```
public WindowEvent (Window source, int id)
```

Parameters	*source*	The object that generated the event.
	id	The event type ID of the event.
Description	Constructs a `WindowEvent` with the given characteristics.	

Instance Methods

getWindow

```
public Window getWindow()
```

Returns	The window that generated this event.

paramString

```
public String paramString()
```

Returns	String with current settings of the `WindowEvent`.
Overrides	`ComponentEvent.paramString()`
Description	Helper method for `toString()` to generate string of current settings.

See Also

`ComponentEvent`, `Window`, `WindowAdapter`, `WindowListener`

21.30 WindowListener ★

Description

Objects that implement the `WindowListener` interface can receive `WindowEvent` objects. Listeners must first register themselves with objects that produce events. When events occur, they are then automatically propagated to all registered listeners.

Interface Definition

```
public abstract interface java.awt.event.WindowListener
    extends java.util.EventListener {

   // Instance Methods
```

```
    public abstract void windowActivated (WindowEvent e);
    public abstract void windowClosed (WindowEvent e);
    public abstract void windowClosing (WindowEvent e);
    public abstract void windowDeactivated (WindowEvent e);
    public abstract void windowDeiconified (WindowEvent e);
    public abstract void windowIconified (WindowEvent e);
    public abstract void windowOpened (WindowEvent e);
  }
```

Interface Methods

windowActivated

`public abstract void windowActivated (WindowEvent e)`

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window has been activated.	

windowClosed

`public abstract void windowClosed (WindowEvent e)`

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window has closed.	

windowClosing

`public abstract void windowClosing (WindowEvent e)`

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window is closing.	

windowDeactivated

`public abstract void windowDeactivated (WindowEvent e)`

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window has been deactivated.	

windowDeiconified

`public abstract void windowDeiconified (WindowEvent e)`

Parameters	*e*	The event that occurred.

Description	Notifies the `WindowListener` that a window has been restored from an iconified state.	

`windowIconified`

```
public abstract void windowIconified (WindowEvent e)
```

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window has iconified (minimized).	

`windowOpened`

```
public abstract void windowOpened (WindowEvent e)
```

Parameters	*e*	The event that occurred.
Description	Notifies the `WindowListener` that a window has opened.	

See Also

`AWTEventMulticaster`, `EventListener`, `Window`, `WindowAdapter`, `Window-Event`