String Handling

The String Constructors

1.To create an empty String, call the default constructor.

String s = new String();

will create an instance of String with no characters in it.

2.To create a String initialized by an array of characters, use the constructor shown here:

String(char chars[ ])

char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);

This constructor initializes s with the string "abc".

3.You can specify a subrange of a character array as an initializer using the following constructor:

String(char chars[ ], int startIndex, int numChars)

Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use.

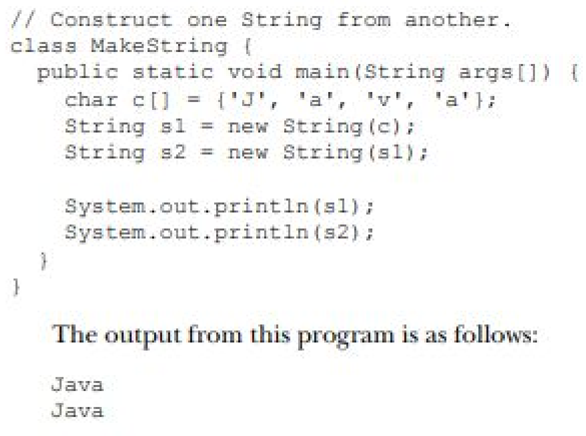
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3);

This initializes s with the characters cde.

4.You can construct a String object that contains the same character sequence as another String object using this constructor:

String(String strObj)

Here, strObj is a String object.



```
// Construct one String from another.
class MakeString {
  public static void main(String args[]) {
    char c[] = {'J', 'a', 'v', 'a'};
    String s1 = new String(c);
    String s2 = new String(s1);

    System.out.println(s1);
    System.out.println(s2);
  }
}
```

The output from this program is as follows:

```
Java
Java
```

5. the String class provides constructors that initialize a string when given a byte array. Two forms are shown here:

String(byte asciiChars[ ])
String(byte asciiChars[ ], int startIndex, int numChars)

```
// Construct string from subset of char array.
class SubStringCons {
  public static void main(String args[]) {
    byte ascii[] = {65, 66, 67, 68, 69, 70 };

    String s1 = new String(ascii);
    System.out.println(s1);

    String s2 = new String(ascii, 2, 3);
    System.out.println(s2);
  }
}
```

This program generates the following output:

```
ABCDEF
CDE
```

String Length

The length of a string is the number of characters that it contains. To obtain this value, call the length( ) method, shown here:

int length( )

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
System.out.println(s.length());
```

Special String Operations

1.String Literals

For each string literal in your program, Java automatically constructs a String object. Thus, you can use a string literal to initialize a String object.

```
char chars[] = { 'a', 'b', 'c' };
String s1 = new String(chars);

String s2 = "abc"; // use string literal
```

Because a String object is created for every string literal, you can use a string literal any place you can use a String object. For example, you can call methods directly on a quoted string as if it were an object reference, as the following statement shows. It calls the length( ) method on the string "abc".

```
System.out.println("abc".length());
```

2.String Concatenation

The + operator, concatenates two strings,

```
String age = "9";
String s = "He is " + age + " years old.";
System.out.println(s);
```

This displays the string "He is 9 years old."

3.String Concatenation with Other Data Types

You can concatenate strings with other types of data.

```
int age = 9;
String s = "He is " + age + " years old.";
System.out.println(s);
```

Output

He is 9 years old.

Character Extraction

1.charAt( )

To extract a single character from a String, you can refer directly to an individual character via the charAt( ) method. It has this general form:

char charAt(int where)

```
char ch;
ch = "abc".charAt(1);
```

assigns the value **b** to **ch.**

2.getChars( )

If you need to extract more than one character at a time, you can use the getChars( ) method. It has this general form:

void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

Here, sourceStart specifies the index of the beginning of the substring, and sourceEnd specifies an index that is one past the end of the desired substring. Thus, the substring contains the characters from sourceStart through sourceEnd–1. The array that will receive the characters is specified by target. The index within target at which the substring will be copied is passed in targetStart.

```
class getCharsDemo {
  public static void main(String args[]) {
    String s = "This is a demo of the getChars method.";
    int start = 10;
    int end = 14;
    char buf[] = new char[end - start];

    s.getChars(start, end, buf, 0);
    System.out.println(buf);
  }
}
```

Here is the output of this program:

```
demo
```

3.getBytes( )

getBytes( ), and it uses the default character-to-byte conversions provided by the platform. Here is its simplest form:

byte[ ] getBytes( )

4.toCharArray( )

If you want to convert all the characters in a String object into a character array, the easiest way is to call toCharArray( ). It returns an array of characters for the entire string. It has this general form:

char[ ] toCharArray( )

String Comparison

1.equals( )

To compare two strings for equality, use equals( ). It has this general form:

boolean equals(Object str)

2.equalsIgnoreCase( )

To perform a comparison that ignores case differences, call equalsIgnoreCase( ).

boolean equalsIgnoreCase(String str)

```
// Demonstrate equals() and equalsIgnoreCase().
class equalsDemo {
  public static void main(String args[]) {
    String s1 = "Hello";
    String s2 = "Hello";
    String s3 = "Good-bye";
    String s4 = "HELLO";
    System.out.println(s1 + " equals " + s2 + " -> " +
                       s1.equals(s2));
    System.out.println(s1 + " equals " + s3 + " -> " +
                       s1.equals(s3));
    System.out.println(s1 + " equals " + s4 + " -> " +
                       s1.equals(s4));
    System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
                       s1.equalsIgnoreCase(s4));
  }
}
```

The output from the program is shown here:

```
Hello equals Hello -> true
Hello equals Good-bye -> false
Hello equals HELLO -> false
Hello equalsIgnoreCase HELLO -> true
```

3.regionMatches( )

The regionMatches( ) method compares a specific region inside a string with another specific region in another string.

- **boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)**
- **boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)**

4.startsWith( )

The startsWith( ) method determines whether a given String begins with a specified string.

- **boolean startsWith(String str)**

"Foobar".startsWith("Foo") returns true.

- **boolean startsWith(String str, int startIndex)**

startIndex specifies the index into the invoking string at which point the search will begin.

"Foobar".startsWith("bar", 3) returns true.

5. endsWith( )

endsWith( ) determines whether the String in question ends with a specified string.

boolean endsWith(String str)

"Foobar".endsWith("bar") returns true.

<u>equals( ) Versus ==</u>

<u>the equals( ) method compares the characters inside a String object. The == operator compares two object references to see whether they refer to the same instance.</u>

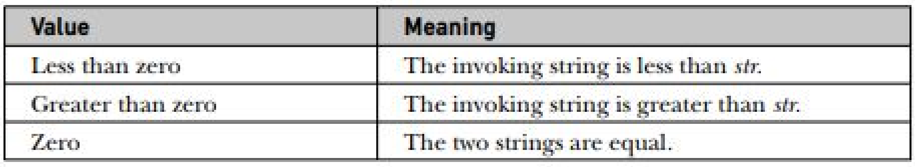
```
// equals() vs ==
class EqualsNotEqualTo {
  public static void main(String args[]) {
    String s1 = "Hello";
    String s2 = new String(s1);

    System.out.println(s1 + " equals " + s2 + " -> " +
                       s1.equals(s2));
    System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
  }
}
```

The variable s1 refers to the String instance created by "Hello". The object referred to by s2 is created with s1 as an initializer. Thus, the contents of the two String objects are identical, but they are distinct objects. This means that s1 and s2 do not refer to the same objects and are, therefore, not ==, as is shown here by the output of the preceding example:
output
Hello equals Hello -> true
Hello == Hello -> false

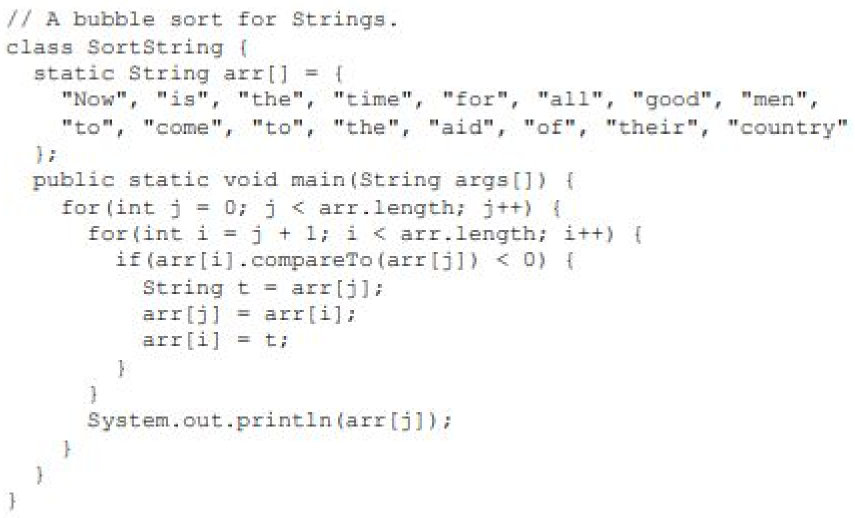
6.compareTo( )
General form:

int **compareTo**(*String str*)

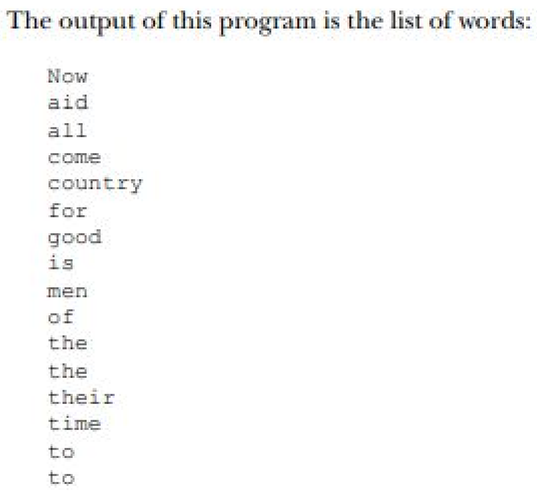
Here, str is the String being compared with the invoking String.



Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.



```
// A bubble sort for Strings.
class SortString {
  static String arr[] = {
    "Now", "is", "the", "time", "for", "all", "good", "men",
    "to", "come", "to", "the", "aid", "of", "their", "country"
  };
  public static void main(String args[]) {
    for(int j = 0; j < arr.length; j++) {
      for(int i = j + 1; i < arr.length; i++) {
        if(arr[i].compareTo(arr[j]) < 0) {
          String t = arr[j];
          arr[j] = arr[i];
          arr[i] = t;
        }
      }
      System.out.println(arr[j]);
    }
  }
}
```



The output of this program is the list of words:

```
Now
aid
all
come
country
for
good
is
men
of
the
the
their
time
to
to
```

If you want to ignore case differences when comparing two strings, use **compareToIgnoreCase( )**, as shown here:

int **compareToIgnoreCase**(String str)

Searching Strings

1.indexOf( )

Searches for the first occurrence of a character or substring.

int indexOf(char ch)

To search for the first occurrence of a substring

int indexOf(String str)

You can specify a starting point for the search using these forms:

int indexOf(char ch, int startIndex)

int indexOf(String str, int startIndex)

startIndex specifies the index at which point the search begins. For indexOf( ), the search runs from startIndex to the end of the string.

2.lastIndexOf( )

Searches for the last occurrence of a character or substring.

To search for the last occurrence of a character, use

int lastIndexOf(char ch)

To search for the last occurrence of a substring, use

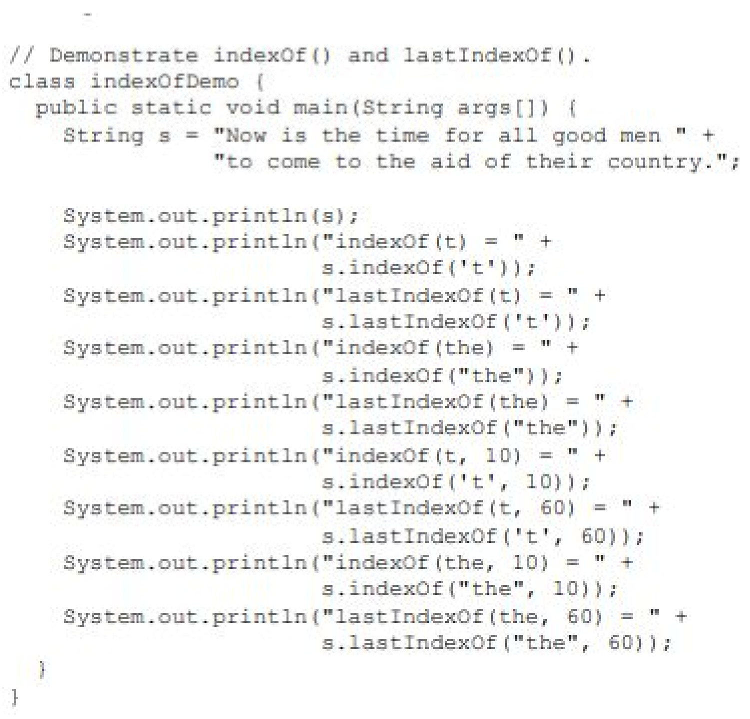
int lastIndexOf(String str)

You can specify a starting point for the search using these forms:

int indexOf(String str, int startIndex)

int lastIndexOf(String str, int startIndex)

For lastIndexOf( ), the search runs from startIndex to zero.



```
// Demonstrate indexOf() and lastIndexOf().
class indexOfDemo {
  public static void main(String args[]) {
    String s = "Now is the time for all good men " +
               "to come to the aid of their country.";

    System.out.println(s);
    System.out.println("indexOf(t) = " +
                       s.indexOf('t'));
    System.out.println("lastIndexOf(t) = " +
                       s.lastIndexOf('t'));
    System.out.println("indexOf(the) = " +
                       s.indexOf("the"));
    System.out.println("lastIndexOf(the) = " +
                       s.lastIndexOf("the"));
    System.out.println("indexOf(t, 10) = " +
                       s.indexOf('t', 10));
    System.out.println("lastIndexOf(t, 60) = " +
                       s.lastIndexOf('t', 60));
    System.out.println("indexOf(the, 10) = " +
                       s.indexOf("the", 10));
    System.out.println("lastIndexOf(the, 60) = " +
                       s.lastIndexOf("the", 60));
  }
}
```

Here is the output of this program:

```
Now is the time for all good men to come to the aid of their country.
indexOf(t) = 7
lastIndexOf(t) = 65
indexOf(the) = 7
lastIndexOf(the) = 55
indexOf(t, 10) = 11
lastIndexOf(t, 60) = 55
indexOf(the, 10) = 44
lastIndexOf(the, 60) = 55
```

Modifying a String

1.substring( )

String substring(int startIndex)

startIndex specifies the index at which the substring will begin

String substring(int startIndex, int endIndex)

startIndex specifies the beginning index, and endIndex specifies the stopping point. The string returned contains all the characters from the beginning index, up to, but not including, the ending index.

```
// Substring replacement.
class StringReplace {
  public static void main(String args[]) {
    String org = "This is a test. This is, too.";
    String search = "is";
    String sub = "was";
    String result = "";
    int i;

    do { // replace all matching substrings
      System.out.println(org);
      i = org.indexOf(search);
      if(i != -1) {
        result = org.substring(0, i);
        result = result + sub;
        result = result + org.substring(i + search.length());
        org = result;
      }
    } while(i != -1);
  }
}
```

The output from this program is shown here:

```
This is a test. This is, too.
Thwas is a test. This is, too.
Thwas was a test. This is, too.
Thwas was a test. Thwas is, too.
Thwas was a test. Thwas was, too.
```

2.concat( )

String concat(String str)

concat( ) performs the same function as +.

String s1 = "one";

String s2 = s1.concat("two");

puts the string "onetwo" into s2.

It generates the same result as the following sequence:

String s1 = "one";
String s2 = s1 + "two";

3.replace( )

String replace(char original, char replacement)

, original specifies the character to be replaced by the character specified by replacement.

String s = "Hello".replace('l', 'w');

puts the string "Hewwo" into s.

String replace(CharSequence original, CharSequence replacement)

replaces one character sequence with another.

4.trim( )

returns a copy of the invoking string from which any leading and trailing whitespace has been removed. It has this general form:

String trim( )

Here is an example:

```
String s = "   Hello World    ".trim();
```

This puts the string "Hello World" into s.

Changing the Case of Characters Within a String

1. toLowerCase( )

converts all the characters in a string from uppercase to lowercase.

String toLowerCase( )

2. toUpperCase( )

converts all the characters in a string from lowercase to uppercase.

String toUpperCase( )

```
// Demonstrate toUpperCase() and toLowerCase().

class ChangeCase {
  public static void main(String args[])
  {
    String s = "This is a test.";

    System.out.println("Original: " + s);

    String upper = s.toUpperCase();
    String lower = s.toLowerCase();

    System.out.println("Uppercase: " + upper);
    System.out.println("Lowercase: " + lower);
  }
}
```

The output produced by the program is shown here:

```
Original: This is a test.
Uppercase: THIS IS A TEST.
Lowercase: this is a test.
```

Event Handling

The Delegation Event Model

- Defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event.
- Once an event is received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A user interface element is able to “delegate” the processing of an event to a separate piece of code.
- In the delegation event model, listeners must register with a source in order to receive an event notification.
- This provides an important benefit: notifications are sent only to listeners that want to receive them.an event was propagated up the containment hierarchy until it was handled by a component.
- This required components to receive events that they did not process, and it wasted valuable time.
- The delegation event model eliminates this overhead.

Events
an event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
Event Sources

- A source is an object that generates an event. This occurs when the internal state of that object changes in some way.

- Each type of event has its own registration method. Here is the general form:
  public void addTypeListener (TypeListener el)
- Type is the name of the event, and el is a reference to the event listener
- For example, the method that registers a keyboard event listener is called addKeyListener( ).
- When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as **multicasting** the event.
- Some sources may allow only one listener to register. The general form of such a method is this:
  public void addTypeListener(TypeListener el) throws
  java.util.TooManyListenersException
- When such an event occurs, the registered listener is notified. This is known as **unicasting** the event.
- A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:
  public void removeTypeListener(TypeListener el)

Event Listeners

- A listener is an object that is notified when an event occurs.
- It has two major requirements.

- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.

Event Classes

At the root of the Java event class hierarchy is **EventObject**, which is in java.util.
Constructor

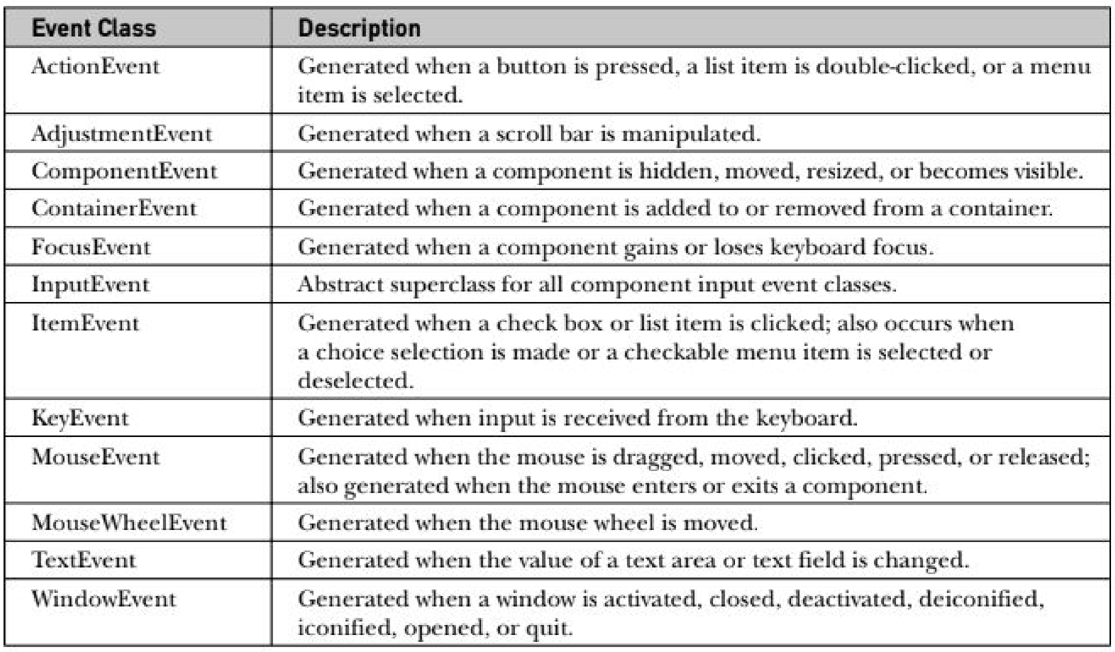
EventObject(Object src)

src is the object that generates this event.

Methods

Object getSource( )-returns the source of the event.
toString( )- returns the string equivalent of the event.

- **EventObject is a superclass of all events.**
- **AWTEvent is a superclass of all AWT events that are handled by the delegation event model.**



Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 23-1 Commonly Used Event Classes in **java.awt.event**

1.The ActionEvent Class

Constructor

- **ActionEvent(Object src, int type, String cmd)**
- **ActionEvent(Object src, int type, String cmd, int modifiers)**
- **ActionEvent(Object src, int type, String cmd, long when, int modifiers)**

Here, src is a reference to the object that generated this event. The type of the event is specified by type, and its command string is cmd. The argument modifiers indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated.

The when parameter specifies when the event occurred.

Methods

- **String getActionCommand( )**-obtain the command name for the invoking ActionEvent object.
- **int getModifiers( )**-returns a value that indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated.
- **long getWhen( )**-returns the time at which the event took place.

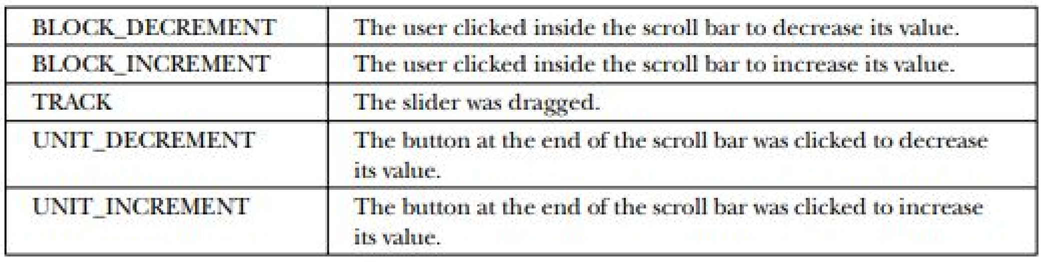
2.The AdjustmentEvent Class

Constructors

AdjustmentEvent(Adjustable src, int id, int type, int data)
Here, src is a reference to the object that generated this event. The id specifies the event. The type of the adjustment is specified by type, and its associated data is data.

METHODS
- **Adjustable getAdjustable( )**- returns the object that generated the event
- **int getAdjustmentType( )**-returns one of the constants defined by AdjustmentEvent.

CONSTANTS



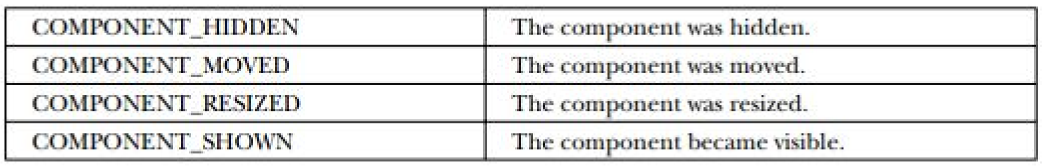
BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged.
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

int getValue( )-The amount of the adjustment can be obtained .

3.The ComponentEvent Class
A ComponentEvent is generated when the size, position, or visibility of a component is changed.
CONSTRUCTORS
ComponentEvent(Component src, int type)
src is a reference to the object that generated this event. The type of the event is specified by type

METHODS
- Component getComponent( )- returns the component that generated the event.

CONSTANTS



COMPONENT_HIDDEN	The component was hidden.
COMPONENT_MOVED	The component was moved.
COMPONENT_RESIZED	The component was resized.
COMPONENT_SHOWN	The component became visible.

4.The ContainerEvent Class
A Container Event is generated when a component is added to or removed from a container.

CONSTRUCTORS
ContainerEvent(Component src, int type, Component comp)
src is a reference to the container that generated this event. The type of the event is specified by type, and the component that has been added to or removed from the container is comp.

METHODS
- **Container getContainer( )**- obtain a reference to the container that generated this event.
- **Component getChild( )**-d returns a reference to the component that was added to or removed from the container.

CONSTANTS

- COMPONENT_ADDED
- COMPONENT_REMOVED.

5.The FocusEvent Class

CONSTRUCTORS
- FocusEvent(Component src, int type)
- FocusEvent(Component src, int type, boolean temporaryFlag)
- FocusEvent(Component src, int type, boolean temporaryFlag, Component other)

src is a reference to the component that generated this event. The type of the event is specified by type. The argument temporaryFlag is set to true if the focus event is temporary. Otherwise, it is set to false.

METHODS
- Component getOppositeComponent( )-The opposite component is returned.
- boolean isTemporary( )-returns true if the change is temporary. Otherwise, it returns false

CONSTANTS
- FOCUS_GAINED
- FOCUS_LOST

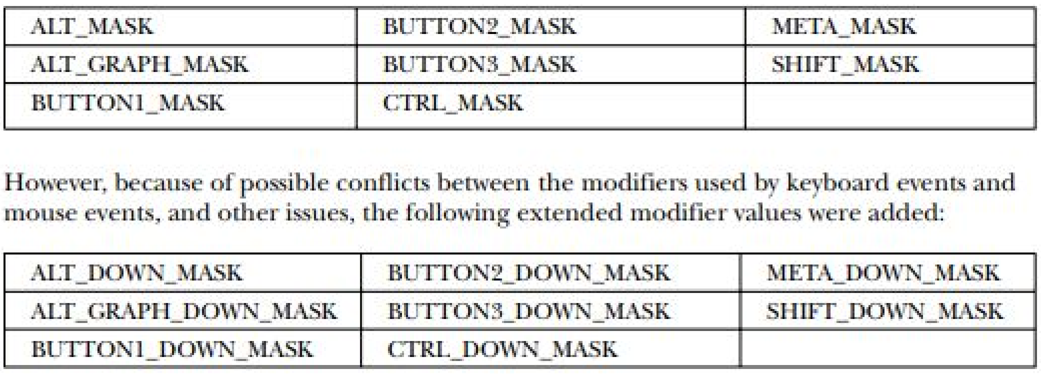
6.The InputEvent Class

The abstract class InputEvent is a subclass of ComponentEvent and is the superclass for component input events. Its subclasses are KeyEvent and MouseEvent.

METHODS
- boolean isAltDown( )
- boolean isAltGraphDown( )
- boolean isControlDown( )

- boolean isMetaDown( )
- boolean isShiftDown( )
- int getModifiers( )- obtain a value that contains all of the original modifier flags
- int getModifiersEx( )- obtain the extended modifiers.

CONSTANTS



ALT_MASK	BUTTON2_MASK	META_MASK
ALT_GRAPH_MASK	BUTTON3_MASK	SHIFT_MASK
BUTTON1_MASK	CTRL_MASK	

However, because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, the following extended modifier values were added:

ALT_DOWN_MASK	BUTTON2_DOWN_MASK	META_DOWN_MASK
ALT_GRAPH_DOWN_MASK	BUTTON3_DOWN_MASK	SHIFT_DOWN_MASK
BUTTON1_DOWN_MASK	CTRL_DOWN_MASK	

7.The ItemEvent Class
generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected

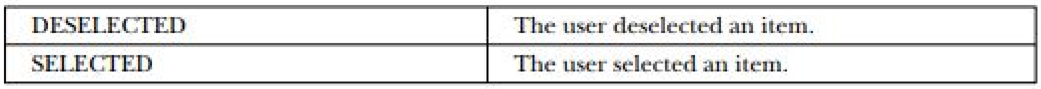
CONSTRUCTORS
ItemEvent(ItemSelectable src, int type, Object entry, int state)

Here, src is a reference to the component that generated this event. The type of the event is specified by type. The specific item that generated the item event is passed in entry. The current state of that item is in state.

METHODS

- Object getItem( )-obtain a reference to the item that generated an event.
- ItemSelectable getItemSelectable( )- obtain a reference to the ItemSelectable object that generated an event.
- int getStateChange( )- returns the state change (that is, SELECTED or DESELECTED) for the event.

CONSTANTS



DESELECTED	The user deselected an item.
SELECTED	The user selected an item.

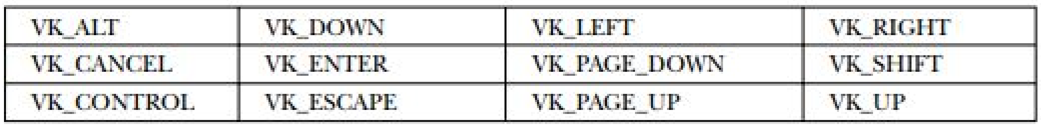
8.The KeyEvent Class
generated when keyboard input occurs.

CONSTRUCTOR
KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)
src is a reference to the component that generated the event. The type of the event is specified by type. The system time at which the key was pressed is passed in when. The modifiers argument indicates which modifiers were pressed when this key event occurred.

METHODS

- char getKeyChar( )- returns the character that was entered. If no valid character is available, then getKeyChar( ) returns CHAR_UNDEFINED. When a KEY_TYPED event occurs, getKeyCode( ) returns VK_UNDEFINED
- int getKeyCode( )-returns the key code.

CONSTANTS



VK_ALT	VK_DOWN	VK_LEFT	VK_RIGHT
VK_CANCEL	VK_ENTER	VK_PAGE_DOWN	VK_SHIFT
VK_CONTROL	VK_ESCAPE	VK_PAGE_UP	VK_UP

- KEY_PRESSED,
- KEY_RELEASED,
- KEY_TYPED.
- VK_0 through VK_9
- VK_A through VK_Z

VK constants specify virtual key codes

9.The MouseEvent Class

CONSTRUCTORS

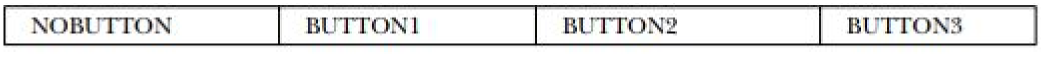
MouseEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup)

src is a reference to the component that generated the event. The type of the event is specified by type. The system time at which the mouse event occurred is passed in when. The modifiers argument indicates which modifiers were pressed when a mouse event occurred. The coordinates of the mouse are passed in x and y. The click count is passed in clicks. The triggersPopup flag indicates if this event causes a pop-up menu to appear on this platform

METHODS

- int getX( )- return the X coordinates
- int getY( )-return the Y coordinates
- Point getPoint( )- obtain the coordinates of the mouse.
- void translatePoint(int x, int y) - changes the location of the event.
- int getClickCount( )- obtains the number of mouse clicks for this event
- boolean isPopupTrigger( )- tests if this event causes a pop-up menu to appear on this platform.
- int getButton( )- returns a value that represents the button that caused the event.

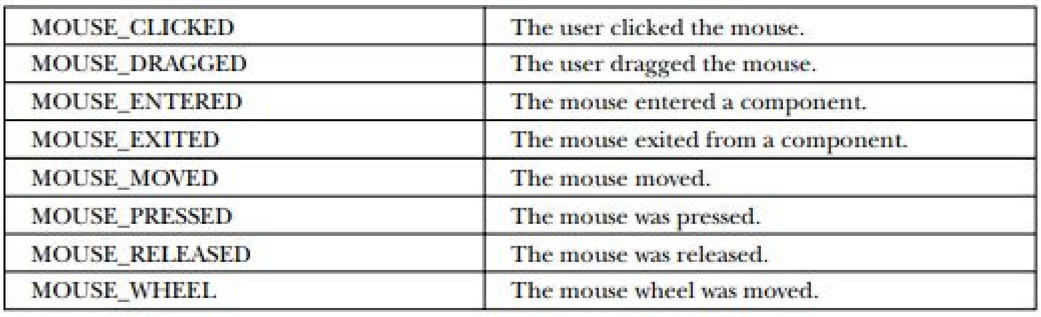
The return value will be one of these constants defined by MouseEvent:



NOBUTTON	BUTTON1	BUTTON2	BUTTON3

- Point getLocationOnScreen( )
- int getXOnScreen( )
- int getYOnScreen( )

CONSTANTS



MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse.
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved.

10.The MouseWheelEvent Class

CONSTRUCTORS
MouseWheelEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup, int scrollHow, int amount, int count)

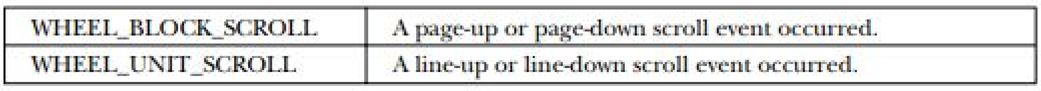
src is a reference to the object that generated the event. The type of the event is specified by type. The system time at which the mouse event occurred is passed in when. The modifiers argument indicates which modifiers were pressed when the event occurred. The coordinates of the mouse are passed in x and y. The number of clicks is passed in clicks. The triggersPopup flag indicates if this event causes a pop-up menu to appear on this platform.

The scrollHow value must be either WHEEL_UNIT_SCROLL or WHEEL_BLOCK_ SCROLL. The number of units to scroll is passed in amount. The count parameter indicates the number of rotational units that the wheel moved

METHODS

- int getWheelRotation( )-returns the number of rotational units. If the value is positive, the wheel moved counterclockwise. If the value is negative, the wheel moved clockwise.
- int getScrollType( )-It returns either WHEEL_UNIT_SCROLL or WHEEL_BLOCK_SCROLL.
- int getScrollAmount( )-If the scroll type is WHEEL_UNIT_SCROLL, you can obtain the number of units to scroll by calling getScrollAmount( ).

CONSTANTS



WHEEL_BLOCK_SCROLL	A page-up or page-down scroll event occurred.
WHEEL_UNIT_SCROLL	A line-up or line-down scroll event occurred.

11.The TextEvent Class

CONSTRUCTORS
TextEvent(Object src, int type)
src is a reference to the object that generated this event. The type of the event is specified by type.

CONSTANTS
TEXT_VALUE_CHANGED.

12.The WindowEvent Class

CONSTRUCTORS

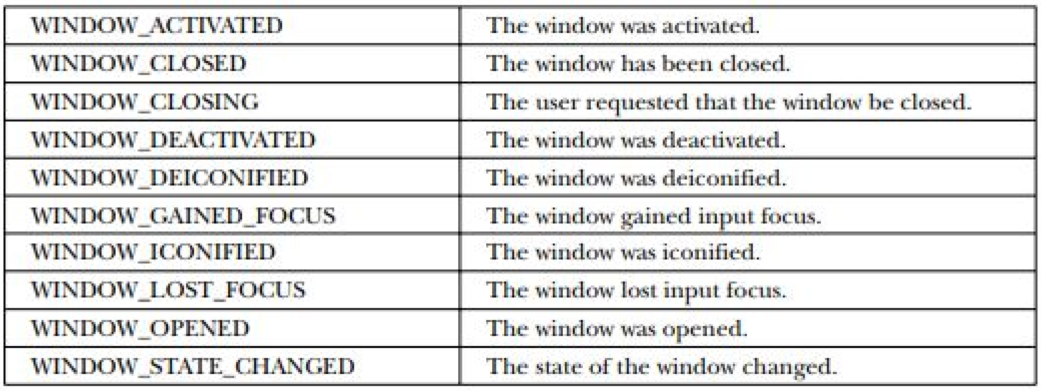
- WindowEvent(Window src, int type, Window other)
- WindowEvent(Window src, int type, int fromState, int toState)
- WindowEvent(Window src, int type, Window other, int fromState, int toState)

other specifies the opposite window when a focus or activation event occurs. The fromState specifies the prior state of the window, and toState specifies the new state that the window will have when a window state change occurs.

METHODS

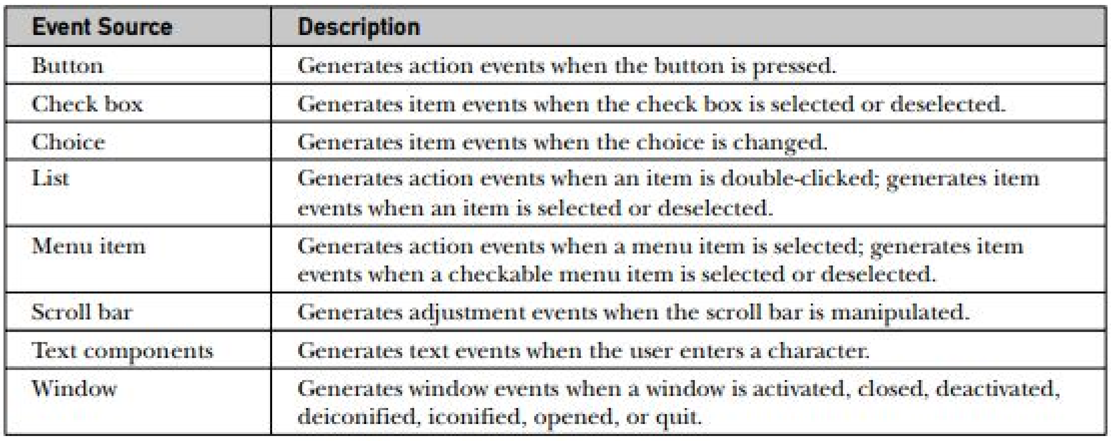
- Window getWindow( )-returns the Window object that generated the event.
- Window getOppositeWindow( )-return the opposite window (when a focus or activation event has occurred),
- int getOldState( ) -return the previous window state,
- int getNewState( )- return the current window state.

CONSTANTS



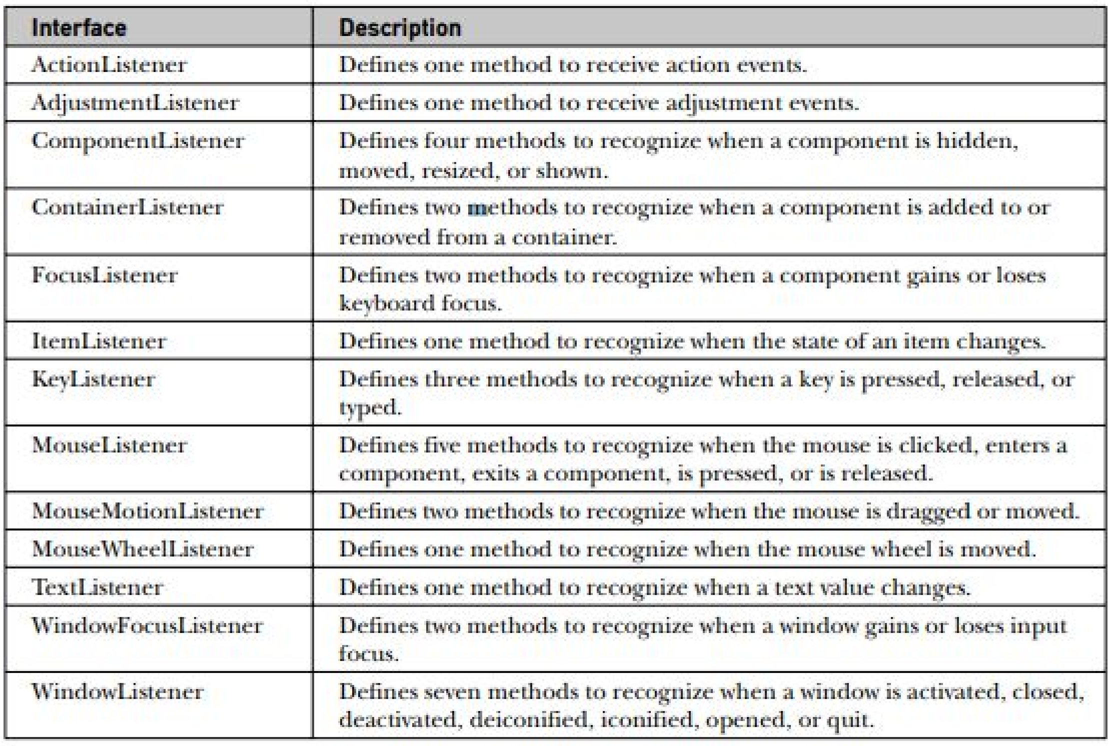
WINDOW_ACTIVATED	The window was activated.
WINDOW_CLOSED	The window has been closed.
WINDOW_CLOSING	The user requested that the window be closed.
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window gained input focus.
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.

Sources of Events



Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces



Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 23-3 Commonly Used Event Listener Interfaces

1.The ActionListener Interface

METHODS

- void actionPerformed(ActionEvent ae)

2.The AdjustmentListener Interface

METHODS

- void adjustmentValueChanged(AdjustmentEvent ae)

3.The ComponentListener Interface
METHODS

- void componentResized(ComponentEvent ce)
- void componentMoved(ComponentEvent ce)
- void componentShown(ComponentEvent ce)
- void componentHidden(ComponentEvent ce)

4.The ContainerListener Interface
METHODS

- void componentAdded(ContainerEvent ce)
- void componentRemoved(ContainerEvent ce)

5.The FocusListener Interface
METHODS

- void focusGained(FocusEvent fe)
- void focusLost(FocusEvent fe)

6.The ItemListener Interface
METHODS

- void itemStateChanged(ItemEvent ie)

7.The KeyListener Interface
METHODS

- void keyPressed(KeyEvent ke)
- void keyReleased(KeyEvent ke)
- void keyTyped(KeyEvent ke)

8.The MouseListener Interface
METHODS

- void mouseClicked(MouseEvent me)
- void mouseEntered(MouseEvent me)
- void mouseExited(MouseEvent me)
- void mousePressed(MouseEvent me)
- void mouseReleased(MouseEvent me)

9.The MouseMotionListener Interface
METHODS

- void mouseDragged(MouseEvent me)
- void mouseMoved(MouseEvent me)

10.The MouseWheelListener Interface
METHODS

- void mouseWheelMoved(MouseWheelEvent mwe)

11.The TextListener Interface
METHODS

- void textChanged(TextEvent te)

12.The WindowFocusListener Interface
METHODS

- void windowGainedFocus(WindowEvent we)
- void windowLostFocus(WindowEvent we)

13.The WindowListener Interface
METHODS

- void windowActivated(WindowEvent we)
- void windowClosed(WindowEvent we)
- void windowClosing(WindowEvent we)

- void windowDeactivated(WindowEvent we)
- void windowDeiconified(WindowEvent we)
- void windowIconified(WindowEvent we)
- void windowOpened(WindowEvent we)