| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/SwingWorker.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/SwingUtilities.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/SwingWorker.StateValue.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/SwingWorker.html)    [**NO FRAMES**](http://docs.google.com/SwingWorker.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: FIELD | [CONSTR](#1t3h5sf) | [METHOD](#2s8eyo1) |

## **javax.swing**

Class SwingWorker<T,V>

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **javax.swing.SwingWorker<T,V>**

**Type Parameters:**T - the result type returned by this SwingWorker's doInBackground and get methodsV - the type used for carrying out intermediate results by this SwingWorker's publish and process methods **All Implemented Interfaces:** [Runnable](http://docs.google.com/java/lang/Runnable.html), [Future](http://docs.google.com/java/util/concurrent/Future.html)<T>, [RunnableFuture](http://docs.google.com/java/util/concurrent/RunnableFuture.html)<T>

public abstract class **SwingWorker<T,V>**extends [Object](http://docs.google.com/java/lang/Object.html)implements [RunnableFuture](http://docs.google.com/java/util/concurrent/RunnableFuture.html)<T>

An abstract class to perform lengthy GUI-interacting tasks in a dedicated thread.

When writing a multi-threaded application using Swing, there are two constraints to keep in mind: (refer to  [How to Use Threads](http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html)  for more details):

* Time-consuming tasks should not be run on the *Event Dispatch Thread*. Otherwise the application becomes unresponsive.
* Swing components should be accessed on the *Event Dispatch Thread* only.

These constraints mean that a GUI application with time intensive computing needs at least two threads: 1) a thread to perform the lengthy task and 2) the *Event Dispatch Thread* (EDT) for all GUI-related activities. This involves inter-thread communication which can be tricky to implement.

SwingWorker is designed for situations where you need to have a long running task run in a background thread and provide updates to the UI either when done, or while processing. Subclasses of SwingWorker must implement the [doInBackground()](http://docs.google.com/javax/swing/SwingWorker.html#doInBackground()) method to perform the background computation.

**Workflow**

There are three threads involved in the life cycle of a SwingWorker :

* *Current* thread: The [execute()](http://docs.google.com/javax/swing/SwingWorker.html#execute()) method is called on this thread. It schedules SwingWorker for the execution on a *worker* thread and returns immediately. One can wait for the SwingWorker to complete using the [get](http://docs.google.com/javax/swing/SwingWorker.html#get()) methods.
* *Worker* thread: The [doInBackground()](http://docs.google.com/javax/swing/SwingWorker.html#doInBackground()) method is called on this thread. This is where all background activities should happen. To notify PropertyChangeListeners about bound properties changes use the [firePropertyChange](http://docs.google.com/javax/swing/SwingWorker.html#firePropertyChange(java.lang.String,%20java.lang.Object,%20java.lang.Object)) and [getPropertyChangeSupport()](http://docs.google.com/javax/swing/SwingWorker.html#getPropertyChangeSupport()) methods. By default there are two bound properties available: state and progress.
* *Event Dispatch Thread*: All Swing related activities occur on this thread. SwingWorker invokes the [process](http://docs.google.com/javax/swing/SwingWorker.html#process(java.util.List)) and [done()](http://docs.google.com/javax/swing/SwingWorker.html#done()) methods and notifies any PropertyChangeListeners on this thread.

Often, the *Current* thread is the *Event Dispatch Thread*.

Before the doInBackground method is invoked on a *worker* thread, SwingWorker notifies any PropertyChangeListeners about the state property change to StateValue.STARTED. After the doInBackground method is finished the done method is executed. Then SwingWorker notifies any PropertyChangeListeners about the state property change to StateValue.DONE.

SwingWorker is only designed to be executed once. Executing a SwingWorker more than once will not result in invoking the doInBackground method twice.

**Sample Usage**

The following example illustrates the simplest use case. Some processing is done in the background and when done you update a Swing component.

Say we want to find the "Meaning of Life" and display the result in a JLabel.

final JLabel label;  
 class MeaningOfLifeFinder extends SwingWorker<String, Object> {  
 @Override  
 public String doInBackground() {  
 return findTheMeaningOfLife();  
 }  
  
 @Override  
 protected void done() {  
 try {   
 label.setText(get());  
 } catch (Exception ignore) {  
 }  
 }  
 }  
   
 (new MeaningOfLifeFinder()).execute();

The next example is useful in situations where you wish to process data as it is ready on the *Event Dispatch Thread*.

Now we want to find the first N prime numbers and display the results in a JTextArea. While this is computing, we want to update our progress in a JProgressBar. Finally, we also want to print the prime numbers to System.out.

class PrimeNumbersTask extends   
 SwingWorker<List<Integer>, Integer> {  
 PrimeNumbersTask(JTextArea textArea, int numbersToFind) {   
 //initialize   
 }  
  
 @Override  
 public List<Integer> doInBackground() {  
 while (! enough && ! isCancelled()) {  
 number = nextPrimeNumber();  
 publish(number);  
 setProgress(100 \* numbers.size() / numbersToFind);  
 }  
 }  
 return numbers;  
 }  
  
 @Override  
 protected void process(List<Integer> chunks) {  
 for (int number : chunks) {  
 textArea.append(number + "\n");  
 }  
 }  
 }  
  
 JTextArea textArea = new JTextArea();  
 final JProgressBar progressBar = new JProgressBar(0, 100);  
 PrimeNumbersTask task = new PrimeNumbersTask(textArea, N);  
 task.addPropertyChangeListener(  
 new PropertyChangeListener() {  
 public void propertyChange(PropertyChangeEvent evt) {  
 if ("progress".equals(evt.getPropertyName())) {  
 progressBar.setValue((Integer)evt.getNewValue());  
 }  
 }  
 });  
  
 task.execute();  
 System.out.println(task.get()); //prints all prime numbers we have got

Because SwingWorker implements Runnable, a SwingWorker can be submitted to an [Executor](http://docs.google.com/java/util/concurrent/Executor.html) for execution.

**Since:** 1.6

| **Nested Class Summary** | |
| --- | --- |
| static class | [**SwingWorker.StateValue**](http://docs.google.com/javax/swing/SwingWorker.StateValue.html)            Values for the state bound property. |

| **Constructor Summary** | |
| --- | --- |
| [**SwingWorker**](http://docs.google.com/javax/swing/SwingWorker.html#SwingWorker())()            Constructs this SwingWorker. |

| **Method Summary** | |
| --- | --- |
| void | [**addPropertyChangeListener**](http://docs.google.com/javax/swing/SwingWorker.html#addPropertyChangeListener(java.beans.PropertyChangeListener))([PropertyChangeListener](http://docs.google.com/java/beans/PropertyChangeListener.html) listener)            Adds a PropertyChangeListener to the listener list. |
| boolean | [**cancel**](http://docs.google.com/javax/swing/SwingWorker.html#cancel(boolean))(boolean mayInterruptIfRunning)            Attempts to cancel execution of this task. |
| protected abstract  [T](http://docs.google.com/javax/swing/SwingWorker.html) | [**doInBackground**](http://docs.google.com/javax/swing/SwingWorker.html#doInBackground())()            Computes a result, or throws an exception if unable to do so. |
| protected  void | [**done**](http://docs.google.com/javax/swing/SwingWorker.html#done())()            Executed on the *Event Dispatch Thread* after the doInBackground method is finished. |
| void | [**execute**](http://docs.google.com/javax/swing/SwingWorker.html#execute())()            Schedules this SwingWorker for execution on a *worker* thread. |
| void | [**firePropertyChange**](http://docs.google.com/javax/swing/SwingWorker.html#firePropertyChange(java.lang.String,%20java.lang.Object,%20java.lang.Object))([String](http://docs.google.com/java/lang/String.html) propertyName, [Object](http://docs.google.com/java/lang/Object.html) oldValue, [Object](http://docs.google.com/java/lang/Object.html) newValue)            Reports a bound property update to any registered listeners. |
| [T](http://docs.google.com/javax/swing/SwingWorker.html) | [**get**](http://docs.google.com/javax/swing/SwingWorker.html#get())()            Waits if necessary for the computation to complete, and then retrieves its result. |
| [T](http://docs.google.com/javax/swing/SwingWorker.html) | [**get**](http://docs.google.com/javax/swing/SwingWorker.html#get(long,%20java.util.concurrent.TimeUnit))(long timeout, [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)            Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available. |
| int | [**getProgress**](http://docs.google.com/javax/swing/SwingWorker.html#getProgress())()            Returns the progress bound property. |
| [PropertyChangeSupport](http://docs.google.com/java/beans/PropertyChangeSupport.html) | [**getPropertyChangeSupport**](http://docs.google.com/javax/swing/SwingWorker.html#getPropertyChangeSupport())()            Returns the PropertyChangeSupport for this SwingWorker. |
| [SwingWorker.StateValue](http://docs.google.com/javax/swing/SwingWorker.StateValue.html) | [**getState**](http://docs.google.com/javax/swing/SwingWorker.html#getState())()            Returns the SwingWorker state bound property. |
| boolean | [**isCancelled**](http://docs.google.com/javax/swing/SwingWorker.html#isCancelled())()            Returns true if this task was cancelled before it completed normally. |
| boolean | [**isDone**](http://docs.google.com/javax/swing/SwingWorker.html#isDone())()            Returns true if this task completed. |
| protected  void | [**process**](http://docs.google.com/javax/swing/SwingWorker.html#process(java.util.List))([List](http://docs.google.com/java/util/List.html)<[V](http://docs.google.com/javax/swing/SwingWorker.html)> chunks)            Receives data chunks from the publish method asynchronously on the *Event Dispatch Thread*. |
| protected  void | [**publish**](http://docs.google.com/javax/swing/SwingWorker.html#publish(V...))([V](http://docs.google.com/javax/swing/SwingWorker.html)... chunks)            Sends data chunks to the [process(java.util.List)](http://docs.google.com/javax/swing/SwingWorker.html#process(java.util.List)) method. |
| void | [**removePropertyChangeListener**](http://docs.google.com/javax/swing/SwingWorker.html#removePropertyChangeListener(java.beans.PropertyChangeListener))([PropertyChangeListener](http://docs.google.com/java/beans/PropertyChangeListener.html) listener)            Removes a PropertyChangeListener from the listener list. |
| void | [**run**](http://docs.google.com/javax/swing/SwingWorker.html#run())()            Sets this Future to the result of computation unless it has been cancelled. |
| protected  void | [**setProgress**](http://docs.google.com/javax/swing/SwingWorker.html#setProgress(int))(int progress)            Sets the progress bound property. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### SwingWorker

public **SwingWorker**()

Constructs this SwingWorker.

| **Method Detail** |
| --- |

### doInBackground

protected abstract [T](http://docs.google.com/javax/swing/SwingWorker.html) **doInBackground**()  
 throws [Exception](http://docs.google.com/java/lang/Exception.html)

Computes a result, or throws an exception if unable to do so.

Note that this method is executed only once.

Note: this method is executed in a background thread.

**Returns:**the computed result **Throws:** [Exception](http://docs.google.com/java/lang/Exception.html) - if unable to compute a result

### run

public final void **run**()

Sets this Future to the result of computation unless it has been cancelled.

**Specified by:**[run](http://docs.google.com/java/lang/Runnable.html#run()) in interface [Runnable](http://docs.google.com/java/lang/Runnable.html)**Specified by:**[run](http://docs.google.com/java/util/concurrent/RunnableFuture.html#run()) in interface [RunnableFuture](http://docs.google.com/java/util/concurrent/RunnableFuture.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **See Also:**[Thread.run()](http://docs.google.com/java/lang/Thread.html#run())

### publish

protected final void **publish**([V](http://docs.google.com/javax/swing/SwingWorker.html)... chunks)

Sends data chunks to the [process(java.util.List)](http://docs.google.com/javax/swing/SwingWorker.html#process(java.util.List)) method. This method is to be used from inside the doInBackground method to deliver intermediate results for processing on the *Event Dispatch Thread* inside the process method.

Because the process method is invoked asynchronously on the *Event Dispatch Thread* multiple invocations to the publish method might occur before the process method is executed. For performance purposes all these invocations are coalesced into one invocation with concatenated arguments.

For example:

publish("1");  
 publish("2", "3");  
 publish("4", "5", "6");

might result in:

process("1", "2", "3", "4", "5", "6")

**Sample Usage**. This code snippet loads some tabular data and updates DefaultTableModel with it. Note that it safe to mutate the tableModel from inside the process method because it is invoked on the *Event Dispatch Thread*.

class TableSwingWorker extends   
 SwingWorker<DefaultTableModel, Object[]> {  
 private final DefaultTableModel tableModel;  
   
 public TableSwingWorker(DefaultTableModel tableModel) {  
 this.tableModel = tableModel;  
 }  
   
 @Override  
 protected DefaultTableModel doInBackground() throws Exception {  
 for (Object[] row = loadData();   
 ! isCancelled() && row != null;   
 row = loadData()) {  
 publish((Object[]) row);  
 }  
 return tableModel;  
 }  
   
 @Override  
 protected void process(List<Object[]> chunks) {  
 for (Object[] row : chunks) {  
 tableModel.addRow(row);  
 }  
 }  
 }

**Parameters:**chunks - intermediate results to process**See Also:**[process(java.util.List)](http://docs.google.com/javax/swing/SwingWorker.html#process(java.util.List))

### process

protected void **process**([List](http://docs.google.com/java/util/List.html)<[V](http://docs.google.com/javax/swing/SwingWorker.html)> chunks)

Receives data chunks from the publish method asynchronously on the *Event Dispatch Thread*.

Please refer to the [publish(V...)](http://docs.google.com/javax/swing/SwingWorker.html#publish(V...)) method for more details.

**Parameters:**chunks - intermediate results to process**See Also:**[publish(V...)](http://docs.google.com/javax/swing/SwingWorker.html#publish(V...))

### done

protected void **done**()

Executed on the *Event Dispatch Thread* after the doInBackground method is finished. The default implementation does nothing. Subclasses may override this method to perform completion actions on the *Event Dispatch Thread*. Note that you can query status inside the implementation of this method to determine the result of this task or whether this task has been cancelled.

**See Also:**[doInBackground()](http://docs.google.com/javax/swing/SwingWorker.html#doInBackground()), [isCancelled()](http://docs.google.com/javax/swing/SwingWorker.html#isCancelled()), [get()](http://docs.google.com/javax/swing/SwingWorker.html#get())

### setProgress

protected final void **setProgress**(int progress)

Sets the progress bound property. The value should be from 0 to 100.

Because PropertyChangeListeners are notified asynchronously on the *Event Dispatch Thread* multiple invocations to the setProgress method might occur before any PropertyChangeListeners are invoked. For performance purposes all these invocations are coalesced into one invocation with the last invocation argument only.

For example, the following invokations:

setProgress(1);  
 setProgress(2);  
 setProgress(3);

might result in a single PropertyChangeListener notification with the value 3.

**Parameters:**progress - the progress value to set **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - is value not from 0 to 100

### getProgress

public final int **getProgress**()

Returns the progress bound property.

**Returns:**the progress bound property.

### execute

public final void **execute**()

Schedules this SwingWorker for execution on a *worker* thread. There are a number of *worker* threads available. In the event all *worker* threads are busy handling other SwingWorkers this SwingWorker is placed in a waiting queue.

Note: SwingWorker is only designed to be executed once. Executing a SwingWorker more than once will not result in invoking the doInBackground method twice.

### cancel

public final boolean **cancel**(boolean mayInterruptIfRunning)

Attempts to cancel execution of this task. This attempt will fail if the task has already completed, has already been cancelled, or could not be cancelled for some other reason. If successful, and this task has not started when cancel is called, this task should never run. If the task has already started, then the mayInterruptIfRunning parameter determines whether the thread executing this task should be interrupted in an attempt to stop the task.

After this method returns, subsequent calls to [Future.isDone()](http://docs.google.com/java/util/concurrent/Future.html#isDone()) will always return true. Subsequent calls to [Future.isCancelled()](http://docs.google.com/java/util/concurrent/Future.html#isCancelled()) will always return true if this method returned true.

**Specified by:**[cancel](http://docs.google.com/java/util/concurrent/Future.html#cancel(boolean)) in interface [Future](http://docs.google.com/java/util/concurrent/Future.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **Parameters:**mayInterruptIfRunning - true if the thread executing this task should be interrupted; otherwise, in-progress tasks are allowed to complete **Returns:**false if the task could not be cancelled, typically because it has already completed normally; true otherwise

### isCancelled

public final boolean **isCancelled**()

Returns true if this task was cancelled before it completed normally.

**Specified by:**[isCancelled](http://docs.google.com/java/util/concurrent/Future.html#isCancelled()) in interface [Future](http://docs.google.com/java/util/concurrent/Future.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **Returns:**true if this task was cancelled before it completed

### isDone

public final boolean **isDone**()

Returns true if this task completed. Completion may be due to normal termination, an exception, or cancellation -- in all of these cases, this method will return true.

**Specified by:**[isDone](http://docs.google.com/java/util/concurrent/Future.html#isDone()) in interface [Future](http://docs.google.com/java/util/concurrent/Future.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **Returns:**true if this task completed

### get

public final [T](http://docs.google.com/javax/swing/SwingWorker.html) **get**()  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html),  
 [ExecutionException](http://docs.google.com/java/util/concurrent/ExecutionException.html)

Waits if necessary for the computation to complete, and then retrieves its result.

Note: calling get on the *Event Dispatch Thread* blocks *all* events, including repaints, from being processed until this SwingWorker is complete.

When you want the SwingWorker to block on the *Event Dispatch Thread* we recommend that you use a *modal dialog*.

For example:

class SwingWorkerCompletionWaiter extends PropertyChangeListener {  
 private JDialog dialog;  
   
 public SwingWorkerCompletionWaiter(JDialog dialog) {  
 this.dialog = dialog;  
 }  
   
 public void propertyChange(PropertyChangeEvent event) {  
 if ("state".equals(event.getPropertyName())  
 && SwingWorker.StateValue.DONE == event.getNewValue()) {  
 dialog.setVisible(false);  
 dialog.dispose();  
 }  
 }  
 }  
 JDialog dialog = new JDialog(owner, true);  
 swingWorker.addPropertyChangeListener(  
 new SwingWorkerCompletionWaiter(dialog));  
 swingWorker.execute();  
 //the dialog will be visible until the SwingWorker is done  
 dialog.setVisible(true);

**Specified by:**[get](http://docs.google.com/java/util/concurrent/Future.html#get()) in interface [Future](http://docs.google.com/java/util/concurrent/Future.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **Returns:**the computed result **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting [ExecutionException](http://docs.google.com/java/util/concurrent/ExecutionException.html) - if the computation threw an exception

### get

public final [T](http://docs.google.com/javax/swing/SwingWorker.html) **get**(long timeout,  
 [TimeUnit](http://docs.google.com/java/util/concurrent/TimeUnit.html) unit)  
 throws [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html),  
 [ExecutionException](http://docs.google.com/java/util/concurrent/ExecutionException.html),  
 [TimeoutException](http://docs.google.com/java/util/concurrent/TimeoutException.html)

Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.

Please refer to [get()](http://docs.google.com/javax/swing/SwingWorker.html#get()) for more details.

**Specified by:**[get](http://docs.google.com/java/util/concurrent/Future.html#get(long,%20java.util.concurrent.TimeUnit)) in interface [Future](http://docs.google.com/java/util/concurrent/Future.html)<[T](http://docs.google.com/javax/swing/SwingWorker.html)> **Parameters:**timeout - the maximum time to waitunit - the time unit of the timeout argument **Returns:**the computed result **Throws:** [InterruptedException](http://docs.google.com/java/lang/InterruptedException.html) - if the current thread was interrupted while waiting [ExecutionException](http://docs.google.com/java/util/concurrent/ExecutionException.html) - if the computation threw an exception [TimeoutException](http://docs.google.com/java/util/concurrent/TimeoutException.html) - if the wait timed out

### addPropertyChangeListener

public final void **addPropertyChangeListener**([PropertyChangeListener](http://docs.google.com/java/beans/PropertyChangeListener.html) listener)

Adds a PropertyChangeListener to the listener list. The listener is registered for all properties. The same listener object may be added more than once, and will be called as many times as it is added. If listener is null, no exception is thrown and no action is taken.

Note: This is merely a convenience wrapper. All work is delegated to PropertyChangeSupport from [getPropertyChangeSupport()](http://docs.google.com/javax/swing/SwingWorker.html#getPropertyChangeSupport()).

**Parameters:**listener - the PropertyChangeListener to be added

### removePropertyChangeListener

public final void **removePropertyChangeListener**([PropertyChangeListener](http://docs.google.com/java/beans/PropertyChangeListener.html) listener)

Removes a PropertyChangeListener from the listener list. This removes a PropertyChangeListener that was registered for all properties. If listener was added more than once to the same event source, it will be notified one less time after being removed. If listener is null, or was never added, no exception is thrown and no action is taken.

Note: This is merely a convenience wrapper. All work is delegated to PropertyChangeSupport from [getPropertyChangeSupport()](http://docs.google.com/javax/swing/SwingWorker.html#getPropertyChangeSupport()).

**Parameters:**listener - the PropertyChangeListener to be removed

### firePropertyChange

public final void **firePropertyChange**([String](http://docs.google.com/java/lang/String.html) propertyName,  
 [Object](http://docs.google.com/java/lang/Object.html) oldValue,  
 [Object](http://docs.google.com/java/lang/Object.html) newValue)

Reports a bound property update to any registered listeners. No event is fired if old and new are equal and non-null.

This SwingWorker will be the source for any generated events.

When called off the *Event Dispatch Thread* PropertyChangeListeners are notified asynchronously on the *Event Dispatch Thread*.

Note: This is merely a convenience wrapper. All work is delegated to PropertyChangeSupport from [getPropertyChangeSupport()](http://docs.google.com/javax/swing/SwingWorker.html#getPropertyChangeSupport()).

**Parameters:**propertyName - the programmatic name of the property that was changedoldValue - the old value of the propertynewValue - the new value of the property

### getPropertyChangeSupport

public final [PropertyChangeSupport](http://docs.google.com/java/beans/PropertyChangeSupport.html) **getPropertyChangeSupport**()

Returns the PropertyChangeSupport for this SwingWorker. This method is used when flexible access to bound properties support is needed.

This SwingWorker will be the source for any generated events.

Note: The returned PropertyChangeSupport notifies any PropertyChangeListeners asynchronously on the *Event Dispatch Thread* in the event that firePropertyChange or fireIndexedPropertyChange are called off the *Event Dispatch Thread*.

**Returns:**PropertyChangeSupport for this SwingWorker

### getState

public final [SwingWorker.StateValue](http://docs.google.com/javax/swing/SwingWorker.StateValue.html) **getState**()

Returns the SwingWorker state bound property.

**Returns:**the current state

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/SwingWorker.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/SwingUtilities.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/SwingWorker.StateValue.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/SwingWorker.html)    [**NO FRAMES**](http://docs.google.com/SwingWorker.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: FIELD | [CONSTR](#1t3h5sf) | [METHOD](#2s8eyo1) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).