javax.swing

Class SwingWorker<T,V>

- java.lang.Object
  - javax.swing.SwingWorker<T,V>

- **Type Parameters:**

  `T` - the result type returned by this `SwingWorker's doInBackground` and `get` methods
  `V` - the type used for carrying out intermediate results by this `SwingWorker's publish` and `process` methods

All Implemented Interfaces:

Runnable, Future<T>, RunnableFuture<T>

```
public abstract class SwingWorker<T,V>
extends Object
implements RunnableFuture<T>
```

An abstract class to perform lengthy GUI-interaction tasks in a background thread. Several background threads can be used to execute such tasks. However, the exact strategy of choosing a thread for any particular `SwingWorker` is unspecified and should not be relied on.

When writing a multi-threaded application using Swing, there are two constraints to keep in mind: (refer to How to Use Threads 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()` method to perform the background computation.

Workflow

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

- *Current* thread: The `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` methods.
- *Worker* thread: The `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` and `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` and `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.

Nested Class Summary

Modifier and Type	Class and Description
`static class`	**`SwingWorker.StateValue`** Values for the `state` bound property.

Constructor Summary

Constructor and Description
`SwingWorker``()` Constructs this `SwingWorker`.

Method Summary

Modifier and Type	Method and Description
`void`	**`addPropertyChangeListener`**`(PropertyChangeListener listener)` Adds a `PropertyChangeListener` to the listener list.
`boolean`	**`cancel`**`(boolean mayInterruptIfRunning)` Attempts to cancel execution of this task.
`protected abstract T`	**`doInBackground`**`()` Computes a result, or throws an exception if unable to do so.
`protected void`	**`done`**`()` Executed on the *Event Dispatch Thread* after the `doInBackground` method is finished.
`void`	**`execute`**`()` Schedules this `SwingWorker` for execution on a *worker* thread.
`void`	**`firePropertyChange`**`(String propertyName, Object oldValue, Object newValue)` Reports a bound property update to any registered listeners.
`T`	**`get`**`()` Waits if necessary for the computation to complete, and then retrieves its result.
`T`	**`get`**`(long timeout, TimeUnit unit)` Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.
`int`	**`getProgress`**`()` Returns the `progress` bound property.
`PropertyChangeSupport`	**`getPropertyChangeSupport`**`()` Returns the `PropertyChangeSupport` for this `SwingWorker`.
`SwingWorker.StateValue`	**`getState`**`()`

	Returns the `SwingWorker` state bound property.
`boolean`	**isCancelled**`()` Returns `true` if this task was cancelled before it completed normally.
`boolean`	**isDone**`()` Returns `true` if this task completed.
`protected void`	**process**`(List<V> chunks)` Receives data chunks from the `publish` method asynchronously on the *Event Dispatch Thread*.
`protected void`	**publish**`(V... chunks)` Sends data chunks to the `process(java.util.List<V>)` method.
`void`	**removePropertyChangeListener**`(PropertyChangeListener listener)` Removes a `PropertyChangeListener` from the listener list.
`void`	**run**`()` Sets this `Future` to the result of computation unless it has been canceled.
`protected void`	**setProgress**`(int progress)` Sets the `progress` bound property.

- **Methods inherited from class java.lang.Object**

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