| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.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 PACKAGE**](http://docs.google.com/java/lang/management/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/java/lang/reflect/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/ref/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

## Package java.lang.ref

Provides reference-object classes, which support a limited degree of interaction with the garbage collector.

**See:**

[**Description**](#3znysh7)

| **Class Summary** | |
| --- | --- |
| [**PhantomReference<T>**](http://docs.google.com/java/lang/ref/PhantomReference.html) | Phantom reference objects, which are enqueued after the collector determines that their referents may otherwise be reclaimed. |
| [**Reference<T>**](http://docs.google.com/java/lang/ref/Reference.html) | Abstract base class for reference objects. |
| [**ReferenceQueue<T>**](http://docs.google.com/java/lang/ref/ReferenceQueue.html) | Reference queues, to which registered reference objects are appended by the garbage collector after the appropriate reachability changes are detected. |
| [**SoftReference<T>**](http://docs.google.com/java/lang/ref/SoftReference.html) | Soft reference objects, which are cleared at the discretion of the garbage collector in response to memory demand. |
| [**WeakReference<T>**](http://docs.google.com/java/lang/ref/WeakReference.html) | Weak reference objects, which do not prevent their referents from being made finalizable, finalized, and then reclaimed. |

## Package java.lang.ref Description

Provides reference-object classes, which support a limited degree of interaction with the garbage collector. A program may use a reference object to maintain a reference to some other object in such a way that the latter object may still be reclaimed by the collector. A program may also arrange to be notified some time after the collector has determined that the reachability of a given object has changed.

## Package Specification

A *reference object* encapsulates a reference to some other object so that the reference itself may be examined and manipulated like any other object. Three types of reference objects are provided, each weaker than the last: *soft*, *weak*, and *phantom*. Each type corresponds to a different level of reachability, as defined below. Soft references are for implementing memory-sensitive caches, weak references are for implementing canonicalizing mappings that do not prevent their keys (or values) from being reclaimed, and phantom references are for scheduling pre-mortem cleanup actions in a more flexible way than is possible with the Java finalization mechanism.

Each reference-object type is implemented by a subclass of the abstract base [Reference](http://docs.google.com/java/lang/ref/Reference.html) class. An instance of one of these subclasses encapsulates a single reference to a particular object, called the *referent*. Every reference object provides methods for getting and clearing the reference. Aside from the clearing operation reference objects are otherwise immutable, so no set operation is provided. A program may further subclass these subclasses, adding whatever fields and methods are required for its purposes, or it may use these subclasses without change.

### Notification

A program may request to be notified of changes in an object's reachability by *registering* an appropriate reference object with a *reference queue* at the time the reference object is created. Some time after the garbage collector determines that the reachability of the referent has changed to the value corresponding to the type of the reference, it will add the reference to the associated queue. At this point, the reference is considered to be *enqueued*. The program may remove references from a queue either by polling or by blocking until a reference becomes available. Reference queues are implemented by the [ReferenceQueue](http://docs.google.com/java/lang/ref/ReferenceQueue.html) class.

The relationship between a registered reference object and its queue is one-sided. That is, a queue does not keep track of the references that are registered with it. If a registered reference becomes unreachable itself, then it will never be enqueued. It is the responsibility of the program using reference objects to ensure that the objects remain reachable for as long as the program is interested in their referents.

While some programs will choose to dedicate a thread to removing reference objects from one or more queues and processing them, this is by no means necessary. A tactic that often works well is to examine a reference queue in the course of performing some other fairly-frequent action. For example, a hashtable that uses weak references to implement weak keys could poll its reference queue each time the table is accessed. This is how the [WeakHashMap](http://docs.google.com/java/util/WeakHashMap.html) class works. Because the [ReferenceQueue.poll](http://docs.google.com/java/lang/ref/ReferenceQueue.html#poll()) method simply checks an internal data structure, this check will add little overhead to the hashtable access methods.

### Automatically-cleared references

Soft and weak references are automatically cleared by the collector before being added to the queues with which they are registered, if any. Therefore soft and weak references need not be registered with a queue in order to be useful, while phantom references do. An object that is reachable via phantom references will remain so until all such references are cleared or themselves become unreachable.

### Reachability

Going from strongest to weakest, the different levels of reachability reflect the life cycle of an object. They are operationally defined as follows:

* An object is *strongly reachable* if it can be reached by some thread without traversing any reference objects. A newly-created object is strongly reachable by the thread that created it.
* An object is *softly reachable* if it is not strongly reachable but can be reached by traversing a soft reference.
* An object is *weakly reachable* if it is neither strongly nor softly reachable but can be reached by traversing a weak reference. When the weak references to a weakly-reachable object are cleared, the object becomes eligible for finalization.
* An object is *phantom reachable* if it is neither strongly, softly, nor weakly reachable, it has been finalized, and some phantom reference refers to it.
* Finally, an object is *unreachable*, and therefore eligible for reclamation, when it is not reachable in any of the above ways.

**Since:** 1.2

| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.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 PACKAGE**](http://docs.google.com/java/lang/management/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/java/lang/reflect/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/ref/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

[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).