**Topic Content**

* Type Of References.
* Reachability of the Object and when ReferenceObjects gets associated with Reference Queue.
* Reference Queue poll and remove example.
* Few IMP Points to note.
* Reference Links

* 1. **Type of References.**

<<Types of java references article.pdf>>

* 1. **Reachability of the Object and when ReferenceObjects gets associated with Reference Queue.**

**Ignore Usage and Idioms Topic in the below PDF.**

<<java-references-IBM-artical-from-2002-appplicable for java8.pdf>>

* 1. **Reference Queue poll and remove example.**

<<Reference Queues-example-poll-and-remove.pdf>>

* 1. **Few IMP Points to note.**

* + The reference queue has poll, remove and remove(long timeout) methods.
  + If we use poll method it will just check if there is any reference object, if yes then remove and return the same else simply return null.
  + If we use remove method it will block itself (similar to object.wait(), to let garbage collection take place) until any reference object is enqueued to the reference queue by garbage collector.
  + If we use remove method with timeout it will wait till timeout until any object gets queued then it will remove and return else if not queued until timeout it will return null.
  + We can register multiple objects with ReferenceQueue.
  + 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.
  + ReferenceObjects get method returning null does not mean there memory is freed this is the reason the ReferenceQueue are used (please refer PDF for point 2)

Reference Queue are ways to notify program that the garbage collector has freed the memory (in case of weak and soft reference) or is about to free the memory (in case of phantom reference), so that we can effectively do some clean up required and gives better flexibility than finalize method.

Soft Reference is used for Memory Sensitive Cache as mostly soft references are garbage collected when there is shortage of memory.

Weak Reference is used in WeakHashMap as in normal hashmap Strong keys does not allow to garbage collect the Values.

If the keys are weak references inside the hashmap so it allows the garbage collection of keys and values once the keys are not strongly reachable from the program.

Weakly reachable object does not survive next garbage collection (depends if they might have earlier survived several minor GCs due to strong reference and moved to some higher GC space S0, S1, Old Gen).

Phantom References are used in the case like I no more need this object but I want to know when it has been finalized and the memory will be free (Providing more flexible way than finalize method). Hence Phantom References return null values for get method.

**Example of Registering different ReferenceObjects with ReferenceQueue:**

public class SimpleGCExample {  
 public static void main(String[] args) throws InterruptedException {  
 ReferenceQueue<Object> queue=new ReferenceQueue<>();  
 SimpleGCExample e = new SimpleGCExample();  
 Reference<Object> pRef=new PhantomReference<>(e, queue),  
 wRef=new WeakReference<>(e, queue);  
 e = null;  
 for(int count=0, collected=0; collected<2; ) {  
 Reference ref=queue.remove(100);  
 if(ref==null) {  
 System.gc();  
 count++;  
 }  
 else {  
 collected++;  
 System.out.println((ref==wRef? "weak": "phantom")  
 +" reference enqueued after "+count+" gc polls");  
 }  
 }  
 }

@Override  
 protected void finalize() throws Throwable {  
 System.out.println("finalizing the object in "+Thread.currentThread());  
 Thread.sleep(100);  
 System.out.println("done finalizing.");  
 }  
}

On my system, it prints

weak reference enqueued after 1 gc polls  
finalizing the object in Thread[Finalizer,8,system]  
done finalizing.  
phantom reference enqueued after 2 gc polls

Or

finalizing the object in Thread[Finalizer,8,system]  
weak reference enqueued after 1 gc polls  
done finalizing.  
phantom reference enqueued after 2 gc polls

The key point is

* + soft and weak references are cleared and enqueued before or right when starting finalization (but in PDF 2 it is queued for soft and weak after freeing up the memory, this might be due to Phantom Reference was pointing to Same Weak Reference hence object memory was not freed and hence Weak reference were queued as is).

* + phantom references are enqueued after finalization, assuming that the object has not leaked the finalize method, otherwise they are enqueued after the object has become unreachable again

*From <*[*https://stackoverflow.com/questions/41378933/meaning-of-referencequeue*](https://stackoverflow.com/questions/41378933/meaning-of-referencequeue)*>*

* + **Reference Links**

<https://docs.oracle.com/javase/8/docs/api/java/lang/ref/package-summary.html#reachability>

<https://docs.oracle.com/javase/8/docs/api/java/lang/ref/SoftReference.html>

<https://docs.oracle.com/javase/8/docs/api/java/lang/ref/WeakReference.html>

<https://docs.oracle.com/javase/8/docs/api/java/lang/ref/PhantomReference.html>

<https://docs.oracle.com/javase/8/docs/api/java/lang/ref/ReferenceQueue.html>

<https://www.ibm.com/developerworks/library/j-refs/>

<https://iq.opengenus.org/types-of-references-in-java/>

<http://learningviacode.blogspot.com/2014/02/reference-queues.html>