Difference between WeakReference vs SoftReference vs PhantomReference vs Strong reference in Java

`WeakReference` and `SoftReference` were added into Java API from long time but not every Java programmer is familiar with it. Which means there is a gap between where and *how to use WeakReference and SoftReference in Java*. Reference classes are particularly important in context of How Garbage collection works. As we all know that Garbage Collector reclaims memory from objects which are eligible for garbage collection, but not many programmer knows that this eligibility is decided based upon which kind of references are pointing to that object. This is also main *difference between WeakReference and SoftReference in Java*. Garbage collector can collect an object if only weak references are pointing towards it and they are eagerly collected, on the other hand Objects with `SoftReference` are collected when JVM absolutely needs memory. These special behaviour of `SoftReference` and `WeakReference`makes them useful in certain cases e.g. `SoftReference` looks perfect for implementing caches, so when JVM needs memory it removes object which have only `SoftReference` pointing towards them. On the other hand `WeakReference` is great for storing meta data e.g. storing `ClassLoader` reference. If no class is loaded then no point in keeping reference of `ClassLoader`, a `WeakReference` makes ClassLoader eligible for Garbage collection as soon as last strong reference removed. In this article we will explore some more about various reference in Java e.g. Strong reference and Phantom reference.

WeakReference vs SoftReference in Java

For those who don't know there are four kind of reference in Java :

1. Strong reference
2. Weak Reference
3. Soft Reference
4. Phantom Reference

Strong Reference is most simple as we use it in our day to day programming life e.g. in the code, `String s = "abc"` , reference variable **`s`** has strong reference to String object `"abc"`. Any object which has Strong reference attached to it is *not eligible for garbage collection*. Obviously these are objects which is needed by Java program. Weak Reference are represented using `java.lang.ref.WeakReference` class and you can create Weak Reference by using following code :

```
Counter counter = new Counter(); // strong reference - line 1
WeakReference<Counter> weakCounter = new WeakReference<Counter>(counter);
//weak reference
counter = null; // now Counter object is eligible for garbage collection
```

Now as soon as you make strong reference `counter = null`, counter object created on line 1 becomes eligible for garbage collection; because it doesn't have any more Strong reference and Weak reference by reference variable `weakCounter` can not prevent `Counter` object from being garbage collected. On the other hand, had this been Soft Reference, Counter object is not garbage collected until JVM absolutely needs memory. Soft reference in Java is represented

using `java.lang.ref.SoftReference` class. You can use following code to create a `SoftReference` in Java

```
Counter prime = new Counter();  // prime holds a strong reference - line 2
SoftReference<Counter> soft = new SoftReference<Counter>(prime) ; //soft
reference variable has SoftReference to Counter Object created at line 2

prime = null;  // now Counter object is eligible for garbage collection but
only be collected when JVM absolutely needs memory
```

After making strong reference `null`, `Counter` object created on line 2 only has one soft reference which cannot prevent it from being garbage collected but it can delay collection, which is eager in case of `WeakReference`. Due to this major *difference between SoftReference and WeakReference*, SoftReference are more suitable for **caches** and WeakReference are more suitable for storing **meta data**. One convenient example of `WeakReference` is `WeakHashMap`, which is another implementation of Map interface like HashMap or TreeMap but with one unique feature. `WeakHashMap` wraps keys as `WeakReference` which means once strong reference to actual object removed, `WeakReference` present internally on `WeakHashMap` doesn't prevent them from being Garbage collected.

Phantom reference is third kind of reference type available in `java.lang.ref` package. Phantom reference is represented by `java.lang.ref.PhantomReference` class. Object which only has Phantom reference pointing them can be collected whenever Garbage Collector likes it. Similar to `WeakReference` and`SoftReference` you can create `PhantomReference` by using following code :

```
DigitalCounter digit = new DigitalCounter(); // digit reference variable has
strong reference - line 3
PhantomReference<DigitalCounter> phantom = new
PhantomReference<DigitalCounter>(digit); // phantom reference to object
created at line 3

digit = null;
```

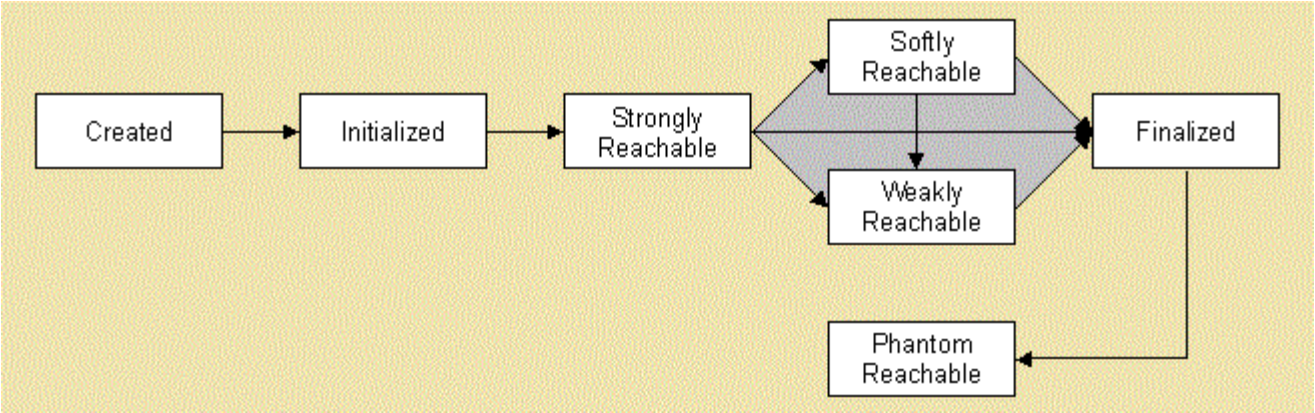
As soon as you remove Strong reference, `DigitalCounter` object created at line 3 can be garbage collected at any time as it only has one more `PhantomReference` pointing towards it, which can not prevent it from GC'd.

Apart from knowing about `WeakReference`, `SoftReference`, `PhantomReference` and `WeakHashMap` there is one more class called `ReferenceQueue` which is worth knowing. You can supply a `ReferenceQueue` instance while creating any `WeakReference`, `SoftReference` or `PhantomReference` as shown in following code :

```
ReferenceQueue refQueue = new ReferenceQueue(); //reference will be stored in
this queue for cleanup

DigitalCounter digit = new DigitalCounter();
PhantomReference<DigitalCounter> phantom = new
PhantomReference<DigitalCounter>(digit, refQueue);
```

Reference of instance will be appended to `ReferenceQueue` and you can use it to perform any clean-up by polling `ReferenceQueue`. An Object's life-cycle is nicely summed up by this diagram.



That's all on **Difference between WeakReference and SoftReference in Java**. We also learned basics of reference classes e.g. Weak, soft and phantom reference in Java and `WeakHashMap` and `ReferenceQueue`. Careful use of reference can assist Garbage Collection and result in better memory management in Java.

Also a small program to bring more clarity and understanding ....

weak reference is related to garbage collection. Normally, object having one or more reference will not be eligible for garbage collection. Thea above rule is not applicable when it is weak reference. It an object has only weak reference with other objects, then its ready for garbage collection.

Let's now look at the below example. We have an Map with Objects where Key is Some Class Reference object.

```
import java.util.HashMap;
public class Test {

public static void main(String args[]) {
HashMap aMap = new
HashMap();

Employee emp = new Employee("SARAL");
EmployeeVal val = new EmployeeVal("Developer");

aMap.put(emp, val);

emp = null;

System.gc();
System.out.println("Size of Map" + aMap.size());

}
}
```

Now, during the execution of the program we have made emp = null. The Map holding the the key makes no sense here as the it is null. In the above situaton, the object is not garbage collected.

WeakHashMap

WeakHashMap is one where the entries (key-to-value mappings) will be removed when it is no longer possible to retrieve them from the map.

Let me show the above example same with WeakHashMap

```
import java.util.WeakHashMap;

public class Test {

public static void main(String args[]) {
WeakHashMap aMap =
new WeakHashMap();

Employee emp = new Employee("SARAL");
EmployeeVal val = new EmployeeVal("Developer");

aMap.put(emp, val);

emp = null;

System.gc();
int count = 0;
while (0 != aMap.size()) {
++count;
System.gc();
}
System.out.println("Took " + count
+ " calls to System.gc() to result in weakHashMap size of : "
+ aMap.size());
}
}
```

Took 20 calls to System.gc() to result in aMap size of : 0.

WeakHashMap has only weak references to the keys, not strong references like other Map classes. There are situations which you have to take care when the value or key is strongly referenced though you have used WeakHashMap. This can avoided by wrapping the object in a WeakReference.

```
import java.lang.ref.WeakReference;
import java.util.HashMap;

public class Test {

public static void main(String args[]) {
HashMap map =
new HashMap();
WeakReference> aMap =
new WeakReference>(
map);
```

```
map = null;

while (null != aMap.get()) {
aMap.get().put(new Employee("SARAL"),
new EmployeeVal("Developer"));
System.out.println("Size of aMap " + aMap.get().size());
System.gc();
}
System.out.println("Its garbage collected");
}
}
```

Soft References.

Soft Reference is slightly stronger that weak reference. Soft reference allows for garbage collection, but begs the garbage collector to clear it only if there is no other option.

The garbage collector does not aggressively collect softly reachable objects the way it does with weakly reachable ones -- instead it only collects softly reachable objects if it really "needs" the memory. Soft references are a way of saying to the garbage collector, "As long as memory isn't too tight, I'd like to keep this object around. But if memory gets really tight, go ahead and collect it and I'll deal with that." The garbage collector is required to clear all soft references before it can throw OutOfMemoryError