Types of Reference in Java:

Many Java developers are not aware that Java has four main types of references.

- Strong Reference
- Weak Reference
- Soft Reference
- Phantom Reference

But why there are different types of reference? What are they used for? To understand that I will go over an example:

Suppose in an application, one needs to fetch data from a MASTER TABLE. As a clever developer, we certainly don't want the application to call the database every time, it will degrade application performance.

Obviously, the choice is to use the cache. Cache is a class (Internally a Map), and first an application will check the cache to check if data is available there, otherwise it will check the database and put the entry in cache, so next time it can be found in the cache to skip a database call.

# Is it Going to Improve Performance?

It will depend on the situation, If the master table has fewer entries this will work fine and certainly increase the performance. But if Master Table has huge entries, it will create a problem as the Cache map is growing as entries load  from Master Table. Now instead of providing better performance it may lead to  out of memory. Imagine a situation where all rows in this huge master table have been loaded to cache. Think about the size of the cache, it can either take most of the JVM memory or it can produce an out of memory error.

# So What Should We Do?

One thing we can do is to restrict the number of entries in the Cache and delete old entries. It will partially solve the problem, but it will take

constant memory in JVM, although some of the objects in the cache will not be used for a long time.

## What is the Ideal Solution?

The ideal solution would be if we can make a type of cache which is dynamic in nature, and can grow and shrink as needed. So we need some kind of technique where we can delete those entries sitting in the cache which will not be used for a long time.

To achieve it in Java, we provide different types of reference in the ***java.lang.ref*** package.

**Strong Reference**: We use Strong references in Java everywhere: we can create an object and then assign it to a reference. Note that if the object has a strong reference, this object is never be garbage collected.

**Example:**

```
HelloWorld hello = new HelloWorld();
```

Here hello is the strong reference to HelloWorld Object.

**Soft Reference**: If an object has no strong reference but has a soft reference, then the garbage collector reclaims this object's memory when GC needs to free up some memory. To get Object from a soft reference, one can invoke the `get()` method. If the object is not GCed, it returns the object, otherwise , it returns null.

**Weak Reference:** If an object has no strong reference but has a weak reference then GC reclaims this object's memory in next run even though there is enough memory.

**Phantom Reference**: If an object does not have any of the above references then it may have a phantom reference. Phantom references can't be accessed directly. When using a `get()` method it will always return null.

Phantom Reference can be used in situations, where sometimes using finalize() is not  sensible.This is a special reference which says that the object was already finalized, and the garbage collector is ready to reclaim its memory.

Coming to the example so we can initialize the cache as WeakHashMap, so that if the key does not have any strong reference it will be deleted by GC. so it will be dynamic in nature.

## An Example of the Different Type of References

```
package com.example.reference;
import java.lang.ref.PhantomReference;
import java.lang.ref.ReferenceQueue;
import java.lang.ref.SoftReference;
import java.lang.ref.WeakReference;
public class ReferenceExample {
	private String status ="Hi I am active";
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
	@Override
	public String toString() {
		return "ReferenceExample [status=" + status + "]";
	}
	public void strongReference()
	{
		ReferenceExample ex = new ReferenceExample();
		System.out.println(ex);
	}
	public void softReference()
	{
		SoftReference<ReferenceExample> ex = new
SoftReference<ReferenceExample>(getRefrence());
		System.out.println("Soft refrence :: " + ex.get());
	}
	public void weakReference()
	{
		int counter=0;
		WeakReference<ReferenceExample> ex = new
WeakReference<ReferenceExample>(getRefrence());
		while(ex.get()!=null)
```

```
                {
                        counter++;
                        System.gc();
                        System.out.println("Weak reference deleted  after:: " +
counter + ex.get());
                }
        }
        public void phantomReference() throws InterruptedException
        {
                final ReferenceQueue queue = new ReferenceQueue();
                PhantomReference<ReferenceExample> ex = new
PhantomReference<ReferenceExample>(getRefrence(),queue);
                System.gc();
                queue.remove();
                System.out.println("Phantom reference deleted  after");
        }
        private ReferenceExample getRefrence()
        {
                return new ReferenceExample();
        }
        public static void main(String[] args) {
                ReferenceExample ex = new ReferenceExample();
                ex.strongReference();
                ex.softReference();
                ex.weakReference();
                try {
                        ex.phantomReference();
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}
Output :
ReferenceExample [status=Hi I am active]
Soft refrence :: ReferenceExample [status=Hi I am active]
Weak reference deleted  after:: 1null
Phantom reference deleted  after
```

Look at the `softReference()` method. Here we create a soft reference, as memory is available so a reference is not GCed.

For weakRefrence(), reference is GCed immediately as there is no strong reference. Same for the Phantom reference.