1. **What Happened in below scenario :**

We have below class called User which has a property “name”

Public Class User{

Private String name;

}

We are adding user object into map.

Map<User, String> map = new HashMap<User, String>();

map.put(new User(“Mahesh”), “abc);

**Then what is the output of map.get(new User(“Mahesh”));**

**ANS : null**

The user object is not available in the collection previously. If it is available, it would have return the existing one .

1. **What happened with below statement**

Final List<String> list = new ArrayList<String>();

List.add(“abc”);

**Ans**: It just means that you can't re-assign its reference. Attempting to do something like the below will lead to compiler error.

But,We still can add to ArrayList new elements, remove elements and update it.

If We wish, we can prevent insertion, removal etc by using [Collections.unmodifiableList()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#unmodifiableList%28java.util.List%29):

(3)Can declare a variable in interface without initializing it??

Ans: In interface if any variable is declared then no need to initialize.

(4) Consider below scenario

if we have one Base class called Animal. and one sub class for it called Cat.

List<Animal> list = new ArrayList<Cat>();

is the above statement valid?

Ans: No it will not work directly except you define your first list as:

List<? extends Animals> animal;

then you will be able to do:

List<Cat> cat = new ArrayList<Cat>();

animal = cat;

(5) Inner implementation of HashMap

If anybody asks me to describe “**How HashMap works?**“, I simply answer: “**On principle of Hashing**“.

**What is Hashing**

[**Hashing**](http://en.wikipedia.org/wiki/Hash_function) in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties. A true Hashing function must follow this rule:

*Hash function should return the same hash code each and every time, when function is applied on same or equal objects. In other words, two equal objects must produce same hash code consistently.*

*Note: All objects in java inherit a default implementation of hashCode() function defined in Object class. This function produce hash code by typically converting the internal address of the object into an integer, thus producing different hash codes for all different objects.*

1. Data structure to store Entry objects is an array named **table** of type Entry.
2. A particular index location in array is referred as bucket, because it can hold the first element of a LinkedList of Entry objects.
3. Key object’s hashCode() is required to calculate the index location of Entry object.
4. Key object’s equals() method is used to maintain uniqueness of Keys in map.
5. Value object’s hashCode() and equals() method are not used in HashMap’s get() and put() methods.
6. Hash code for null keys is always zero, and such Entry object is always stored in zero index in Entry[].

(6) What is Linked HashMap.

* Ans: LinkedHashMap is a combination of Hash table and linked list implementation of the Map interface, with predictable iteration order.
* It maintains a doubly-linked list running through all of its entries.
* The iteration order is normally the order in which keys were inserted into the map ie insertion order.The insertion order is not affected if a key is re-inserted into the map

(7) How many objects will create in below scenario?

String s1= “abc”

String s2 = “abc”;

String S3= new String(abc”);

**Ans**: 2 objects will be created.  
See whenever you are using the "new" operator the object is created, JVM will not look in string pool it is just going to create the Object, but when you are using the string literals for creating String objects then JVM will perform the task of looking in the string pool

(8)difference between Array and ArrayList

Ans:

* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) can contain primitive or objects, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) can contain only objects.
* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) have fixed size, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) is dynamic.
* An [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)provides more methods and features, such as [addAll](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#addAll(java.util.Collection)), [removeAll](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#removeAll(java.util.Collection)), [iterator](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#iterator()), etc.
* For a list of primitive data types, the collections use autoboxing to reduce the coding effort. However, this approach makes them slower when working on fixed size primitive data types.