**Top 25  Java Interview Questions :**  
 **1. Which two method you need to implement for key Object in HashMap ?**

In order to use any object as Key in HashMap, it must implements equals and hashcode method in Java. Read [**How HashMap works in Java**](http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html)  for detailed explanation on how equals and hashcode method is used to put and get object from HashMap.   
  
**2. What is immutable object? Can you write immutable object?**Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object result in new object. For example is String is immutable in Java. Mostly Immutable are also final in Java, in order to prevent sub class from overriding methods in Java which can compromise Immutability. You can achieve same functionality by making member as non final but private and not modifying them except in constructor.

**3. What is the difference between creating String as new() and literal?**

When we create string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

String s = new String("Test");  
   
does not  put the object in String pool , we need to call String.intern() method which is used to put  them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.

**4. What is**[**difference between StringBuffer and StringBuilder**](http://javahungry.blogspot.com/2013/06/difference-between-string-stringbuilder.html)**in Java ?**

Classic Java questions which some people thing tricky and some consider very easy. StringBuilder in Java is introduced in Java 5 and only difference between both of them is that Stringbuffer methods are synchronized while StringBuilder is non synchronized. See [StringBuilder vs StringBuffer](http://javahungry.blogspot.com/2013/06/difference-between-string-stringbuilder.html) for more differences.

**5.  Write code to find the First non repeated character in the String  ?**

Another good Java interview question, This question is mainly asked by Amazon and equivalent companies. See [first non repeated character in the string : Amazon interview question](http://javahungry.blogspot.com/2013/12/first-non-repeated-character-in-string-java-program-code-example.html)

**6. What is the difference between ArrayList and Vector ?**  
This question is mostly used as a start up question in Technical interviews  on the topic of Collection framework . Answer is explained in detail here [Difference between ArrayList and Vector](http://javahungry.blogspot.com/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html) .

**7. How do you handle error condition  while writing stored procedure or accessing stored procedure from java?**

This is one of the tough Java interview question and its open for all, my friend didn't know the answer so he didn't mind telling me. my take is that stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

**8. What is difference between Executor.submit() and Executer.execute() method ?**

*There is a difference when looking at exception handling. If your tasks throws an exception and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with an exception, the Future.get will re-throw this exception, wrapped in an ExecutionException.*

**9. What is the difference between factory and abstract factory pattern?**

*Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for creation of objects in that genre.*

You can also refer What is Factory method design pattern in Java to know more details.

**10. What is Singleton? is it better to make whole method synchronized or only critical section synchronized ?**

[Singleton in Java is a class with just one instance in whole Java application](http://javahungry.blogspot.com/2013/08/singleton-design-pattern-use-in-java.html), for example java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum its very easy. see my article [How to create thread-safe Singleton in Java](http://javahungry.blogspot.com/2013/08/singleton-design-pattern-use-in-java.html) for more details on writing Singleton using enum and double checked locking which is purpose of this Java interview question.

**11. Can you write critical section code for singleton?**

This core Java question is followup of previous question and expecting candidate to write Java singleton using double checked locking. Remember to use volatile variable to make Singleton thread-safe.

**12. Can you write code for**[**iterating**](http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html)**over hashmap in Java 4 and Java 5 ?**

Tricky one but he managed to write using while and for loop.  
  
**13. When do you override hashcode and equals() ?**  
Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap.  
  
**14. What will be the problem if you don't override hashcode() method ?**  
You will not be able to recover your object from hash Map if that is used as key in HashMap.  
See here  [How HashMap works in Java](http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html) for detailed explanation.  
  
**15. Is it better to synchronize critical section of getInstance() method or whole getInstance() method ?**  
Answer is critical section because if we lock whole method than every time some one call this method will have to wait even though we are not creating any object)  
  
**16. What is the difference when String is gets created using literal or new() operator ?**  
When we create string with new() its created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap.  
  
**17. Does not overriding hashcode() method has any performance implication ?**  
This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.  
  
**18. What’s wrong using HashMap in multithreaded environment? When get() method go to infinite loop ?**  
Another good question. His answer was during concurrent access and re-sizing.  
  
**19.  What do you understand by thread-safety ? Why is it required ? And finally, how to achieve thread-safety in Java Applications ?**   
  
Java Memory Model defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably see writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.  
  
Let's first discuss about Memory Barrier which are the base for our further discussions. There are two type of memory barrier instructions in JMM - read barriers and write barrier.  
  
A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.  
A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.  
**JMM semantics for synchronized**  
When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)  
Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.  
  
**JMM semantics for Volatile  fields**  
Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code

Data data = null;

volatile boolean flag = false;

Thread A

-------------

data = new Data();

flag = true; <-- writing to volatile will flush data as well as flag to main memory

Thread B

-------------

if(flag==true){ <-- as="" barrier="" data.="" flag="" font="" for="" from="" perform="" read="" reading="" volatile="" well="" will="">

use data; <!--- data is guaranteed to visible even though it is not declared volatile because of the JMM semantics of volatile flag.

}

**20.  What will happen if you call return statement or System.exit on try or catch block ? will finally block execute?**

This is a very *popular tricky Java question* and its tricky because many programmer think that finally block always executed. This question challenge that concept by putting return statement in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put return statement in try block or catch block but finally block won't run if you call System.exit form try or catch.

**19. Can you override private or static method in Java ?**

Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java.  Anyway, you can not override private or static method in Java, if you create similar method with same return type and same method arguments that's called method hiding. 

**20. What will happen if we put a key object in a HashMap which is already there ?**

This tricky Java questions is part of [How HashMap works in Java](http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html), which is also a popular topic to create confusing and tricky question in Java. well if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys.

**21. If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException?**

One more tricky Java questions from overloading and overriding concept. Answer is you can very well throw super class of RuntimeException in overridden method but you can not do same if its checked Exception.

**22. What is the issue with following implementation of compareTo() method in Java**

public int compareTo(Object o){

   Employee emp = (Employee) emp;

   return this.id - o.id;

}

**23. How do you ensure that N thread can access N resources without deadlock**

If you are not well versed in writing multi-threading code then this is real tricky question for you. This Java question can be tricky even for experienced and senior programmer, who are not really exposed to deadlock and race conditions. Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

**24. What is difference between CyclicBarrier and CountDownLatch in Java**

Relatively newer Java tricky question, only been introduced form Java 5. Main difference between both of them is that you can reuse CyclicBarrier even if Barrier is broken but you can not reuse CountDownLatch in Java. See CyclicBarrier vs CountDownLatch in Java for more differences.

**25. Can you access non static variable in static context?**

Another tricky Java question from Java fundamentals. No you can not access static variable in non static context in Java. Read why you can not access non-static variable from static method to learn more about this tricky Java questions.

[Does adding a duplicate value to a HashSet/HashMap replace the previous value](http://stackoverflow.com/questions/12940663/does-adding-a-duplicate-value-to-a-hashset-hashmap-replace-the-previous-value)

In the case of [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html#put%28K,%20V%29), it replaces the old value with the new one.

In the case of [HashSet](http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html#add%28E%29), the item isn't inserted.

**HashSet Doesn’t gurantee ORDER.**

**LinkedHashSet gurantee ORDER of insertion**

**TreeSet – elements stored in sorted manner.**

**SortedMap – maintains order based on keys – can use comparator for sorting based on field**

**Queue Deque and blockingqueue**

## What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. I know it sounds confusing. Don’t worry we will discuss this in detail.

* It is a  feature that allows one interface to be used for a general class of  actions.
* An operation may exhibit different behavior in different instances.
* The behavior depends on the types of data used in the operation.
* It plays an important role in allowing objects having different internal structures to share the same external interface.
* Polymorphism is extensively used in implementing inheritance.

Following concepts demonstrate different types of polymorphism in java.  
1) [**Method Overloading**](http://beginnersbook.com/2013/05/method-overloading/)  
2) [**Method Overriding**](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/)

**Method Definition:**  
A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method’s name.

1)**Method Overloading:**  
In Java, it is possible to define two or more methods of same name in a class, provided that there argument list or parameters are different. This concept is known as Method Overloading.

I have covered method overloading and Overriding below. To know more about polymorphism types refer my post [**Types of polymorphism in java: Static, Dynamic, Runtime and Compile time Polymorphism**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/).

### 1) Method Overloading

1. To call an overloaded method in Java, it is must to use the type and/or number of arguments to determine which version of the overloaded method to actually call.
2. Overloaded methods may have different return types; the return type alone is insufficient to distinguish two versions of a method. .
3. When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.
4. It allows the user to achieve compile time polymorphism.
5. An overloaded method can throw different exceptions.
6. It can have different access modifiers.

Example:

class Overload

{

void demo (int a)

{

System.out.println ("a: " + a);

}

void demo (int a, int b)

{

System.out.println ("a and b: " + a + "," + b);

}

double demo(double a) {

System.out.println("double a: " + a);

return a\*a;

}

}

class MethodOverloading

{

public static void main (String args [])

{

Overload Obj = new Overload();

double result;

Obj .demo(10);

Obj .demo(10, 20);

result = Obj .demo(5.5);

System.out.println("O/P : " + result);

}

}

Here the method demo() is overloaded 3 times: first having 1 int parameter, second one has 2 int parameters and third one is having double arg. The methods are invoked or called with the same type and number of parameters used.

**Output:**

a: 10

a and b: 10,20

double a: 5.5

O/P : 30.25

**Rules for Method Overloading**

1. Overloading can take place in the same class or in its sub-class.
2. Constructor in Java can be overloaded
3. Overloaded methods must have a different argument list.
4. Overloaded method  (can also take place in sub class), with same name but different parameters.
5. The parameters may differ in their type or number, or in both.
6. They may have the same or different return types.
7. It is also known as compile time polymorphism.

### 2) Method Overriding

Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.  
Example:

public class BaseClass

{

public void methodToOverride() //Base class method

{

System.out.println ("I'm the method of BaseClass");

}

}

public class DerivedClass extends BaseClass

{

public void methodToOverride() //Derived Class method

{

System.out.println ("I'm the method of DerivedClass");

}

}

public class TestMethod

{

public static void main (String args []) {

// BaseClass reference and object

BaseClass obj1 = new BaseClass();

// BaseClass reference but DerivedClass object

BaseClass obj2 = new DerivedClass();

// Calls the method from BaseClass class

obj1.methodToOverride();

//Calls the method from DerivedClass class

obj2.methodToOverride();

}

}

**Output:**

I'm the method of BaseClass

I'm the method of DerivedClass

**Rules for Method Overriding:**

1. applies only to inherited methods
2. object type (NOT reference variable type) determines which overridden method will be used at runtime
3. Overriding method can have different return type ([**refer this**](http://stackoverflow.com/questions/14694852/can-overridden-methods-differ-in-return-type))
4. Overriding method must not have more restrictive access modifier
5. Abstract methods must be overridden
6. Static and final methods cannot be overridden
7. Constructors cannot be overridden
8. It is also known as Runtime polymorphism.

### super keyword in Overriding:

When invoking a superclass version of an overridden method the super keyword is used.  
Example:

class Vehicle {

public void move () {

System.out.println ("Vehicles are used for moving from one place to another ");

}

}

class Car extends Vehicle {

public void move () {

super. move (); // invokes the super class method

System.out.println ("Car is a good medium of transport ");

}

}

public class TestCar {

public static void main (String args []){

Vehicle b = new Car (); // Vehicle reference but Car object

b.move (); //Calls the method in Car class

}

}

**Output:**

Vehicles are used for moving from one place to another

Car is a good medium of transport

<http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples>

package com.journaldev.singleton;

public class BillPughSingleton {

private BillPughSingleton(){}

private static class SingletonHelper{

private static final BillPughSingleton INSTANCE = new BillPughSingleton();

}

public static BillPughSingleton getInstance(){

return SingletonHelper.INSTANCE;

}

}

Notice the **private inner static class** that contains the instance of the singleton lass. When the singleton class is loaded, SingletonHelper class is not loaded into memory and only when someone calls the getInstance method, this class gets loaded and creates the Singleton class instance.

Modifier | Class | Package | Subclass | World

public | Y | Y | Y | Y

protected | Y | Y | Y | N

no modifier | Y | Y | N | N

private | Y | N | N | N