**JAVA 5**

1. Generics for Collections
2. Enhanced for Loop (for-each loop)
3. Autoboxing/Unboxing
4. Typesafe Enums
5. Varargs/Vargs (Variable-length Argument Lists)
6. Static Import
7. Metadata (Annotations)
8. Formatting
9. Scanner

**JAVA 6**

1. Pluggable Annotation Processing API (JSR 269)
2. Common Annotations (JSR 250)
3. Java API for XML Based Web Services - 2.0 (JSR 224)
4. JAXB 2.0 (JSR 222)
5. Web Services Metadata (JSR 181)
6. Streaming API for XML (JSR 173)
7. XML Digital Signature (JSR 105)
8. Java Class File Specification Update (JSR 202)
9. Java Compiler API (JSR 199)
10. JDBC 4.0 (JSR 221)
11. Scripting in the Java Platform (JSR 223)

**JAVA 7**

1. Allow using String in Switch case
2. Multiple Exception in one catch block
3. Automatic resource management or ARM blocks
4. Fork Join framework in Java 7
5. Diamond operator <> for type inference

**JAVA 8**

**(http://www.journaldev.com/2389/java-8-features-with-examples)**

1.     forEach() method in Iterable interface

2.     default and static methods in Interfaces

3.     Functional Interfaces and Lambda Expressions

4.     Java Stream API for Bulk Data Operations on Collections

5.     Java Time API

6.     Collection API improvements

7.     Concurrency API improvements

8.     Java IO improvements

9.     Miscellaneous Core API improvements

**JAVA 9**

**(http://www.journaldev.com/2389/java-8-features-with-examples)**

1. Java 9 REPL (JShell)
2. Factory Methods for Immutable List, Set, Map and Map.Entry
3. Private methods in Interfaces
4. Java 9 Module System
5. Process API Improvements
6. Try With Resources Improvement
7. CompletableFuture API Improvements
8. Reactive Streams
9. Diamond Operator for Anonymous Inner Class
10. Optional Class Improvements
11. Stream API Improvements
12. Enhanced @Deprecated annotation
13. HTTP 2 Client
14. Мulti-Resolution Image API
15. Miscellaneous Java 9 Features

**How many types you can create object of a class?**

There are four different ways to create objects in java:

**A**. Using new keyword  
This is the most common way to create an object in java. Almost 99% of objects are created in this way.

MyObject object = new MyObject();

**B**. Using Class.forName()  
If we know the name of the class & if it has a public default constructor we can create an object in this way.

MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();

**C**. Using clone()  
The clone() can be used to create a copy of an existing object.

MyObject anotherObject = new MyObject();

MyObject object = anotherObject.clone();

**D**. Using object deserialization  
Object deserialization is nothing but creating an object from its serialized form.

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyObject object = (MyObject) inStream.readObject();

### Generic Qns

**1. What is generics in Java ? What are advantages of using Generics?**

Those who are coming from prior to Java 5 background knows that how inconvenient it was to store object in Collection and then cast it back to correct Type before using it. Generics prevents from those. it provides [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) type-safety and ensures that you only insert correct Type in collection and avoids ClassCastException in runtime.

**2. How Generics works in Java ? What is type erasure ?**

Generics is implemented using **Type erasure**, compiler erases all type related information during compile time and no type related information is available during runtime. for example List<String> is represented by only List at runtime. This was done to ensure binary compatibility with the libraries which were developed prior to Java 5. you don’t have access to Type argument at runtime and **Generic type** is translated to **Raw type** by compiler during runtime. you can get lot of follow up question based on this Generic interview question based upon your response e.g. Why Generics is implemented using Type erasure or presenting some invalid generic code which results in compiler error. read my post [How Generics works in Java](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html) for more details

**3. What is Bounded and Unbounded wildcards in Generics ?**

Bounded Wildcards are those which impose bound on Type. There are two kinds of Bounded wildcards <? extends T> which impose an upper bound by ensuring that type must be sub class of T and <? super T> where its imposing lower bound by ensuring Type must be super class of T. This Generic Type must be instantiated with Type within bound otherwise it will result in compilation error. On the other hand <?> represent and unbounded type because <?> can be replace with any Type.

**4. What is difference between List<? extends T> and List <? super T>** **?**

Both of List declaration is example of bounded wildcards, List<? extends T> will accept any List with Type extending T while List<? super T> will accept any List with type super class of T. for Example List<? extends Number> can accept List<Integer> or List<Float> .

**5. How to write a generic method which accepts generic argument and return Generic Type?**

writing generic method is not difficult, instead of using raw type you need to use Generic Type like T, E or K,V which are well known placeholders for Type, Element and Key, Value. Look on Java Collection framework for examples of generics methods. In simplest form a generic method would look like:

|  |  |
| --- | --- |
| 1 | public V put(K key, V value) { |
| 2 | return cache.put(key, value); | |

|  |  |
| --- | --- |
| 3 | } |

**6. How to write parametrized class in Java using Generics ?**

**class** Wrapper<T> {

…………

……….  
}  
  
This is an extension of previous Java generics interview question. Instead of asking to write Generic method Interviewer may ask to write a type safe class using generics. again key is instead of using raw types you need to used generic types and always use standard place holder used in JDK.

**7. Write a program to implement LRU cache using Generics ?**

This is an exercise for anyone who like Coding in Java. One hint is that LinkedHashMap can be used implement fixed size LRU cache where one needs to remove eldest entry when Cache is full. LinkedHashMap provides a method called removeEldestEntry() which is called by put() and putAll() and can be used to instruct to remove eldest entry. you are free to come up with your own implementation as long as you have a written a working version along with Unit test.

**8. Can you pass List<String> to a method which accepts List<Object>**

It looks like String is object so List<String> can be used where List<Object> is required but this is not true. It will result in compilation error. It does make sense if you go one step further because List<Object> can store anything including [String, Integer](http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html) etc but List<String> can only store Strings.

|  |  |
| --- | --- |
| 1 | List<Object> objectList; |
| 2 | List<String> stringList; |

|  |  |
| --- | --- |
| 3 |  |
| 4 | objectList = stringList;  //compilation error incompatible types | |

**9. Can we use Generics with Array?**

If you know the fact that **Array doesn’t support** Generics and that’s why Joshua bloach suggested preferring List over Array because List can provide compile time type-safety over Array.

**10. How can you suppress unchecked warning in Java ?**

javac compiler for Java 5 generates unchecked warnings if you use combine raw types and generics types e.g.

|  |  |  |
| --- | --- | --- |
| 1 | List<String> rawList = new ArrayList() | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | Note: Hello.java uses unchecked or unsafe operations.; |

Which can be suppressed by using @SuppressWarnings(“unchecked”) annotation.

### Collection framework qns

**Java Concurrent Collection Classes**

* [BlockingQueue](http://javapapers.com/java/java-blockingqueue/) – an interface that is at the base of all Queuebased concurrent collections. While adding an element to a BlockingQueue, if there is no space it can wait till it becomes available and when retrieving, it will wait till an element is available if it is empty.
* [ArrayBlockingQueue](http://javapapers.com/java/java-arrayblockingqueue/) – a blocking queue class based on bounded Java Array. Once instantiated, cannot be resized.
* [SynchronousQueue](http://javapapers.com/java/java-synchronousqueue/) – a blocking queue class with capacity of zero always.
* [PriorityBlockingQueue](http://javapapers.com/java/java-priorityblockingqueue/) – a priority queue based blocking queue. It is an unbounded concurrent collection.
* [LinkedBlockingQueue](http://javapapers.com/java/java-linkedblockingqueue/) – an optionally bounded Java concurrent collection. Orders elements based on FIFO order.
* [DelayQueue](http://javapapers.com/java/java-delayqueue/) – a queue where only delay expired elements can be taken out. Its an unbounded concurrent collection.
* [BlockingDeque](http://javapapers.com/java/java-blockingdeque/) – an interface that extends BlockingQueueand adds the operations of Deque.
* [LinkedBlockingDeque](http://javapapers.com/java/java-linkedblockingdeque/) – an implementation class of BlockingDequeue.
* [TransferQueue](http://javapapers.com/java/java-transferqueue/) – a Java concurrent collection interface that extends BlockingQueue and adds method where the producer will wait for the consumer to receive elements.
* [LinkedTransferQueue](http://javapapers.com/java/java-linkedtransferqueue/) – an implementation class of TransferQueue.
* ConcurrentMap – a Java concurrent collection interface and a type of Map which provides thread safety and atomicity guarantees.
* [ConcurrentHashMap](http://javapapers.com/java/java-concurrenthashmap/) – an implementation class of ConcurrentMap.
* ConcurrentNavigableMap – a Java concurrent collection interface that extends ConcurrentMap and adds operations of NavigableMap.
* [ConcurrentSkipListMap](http://javapapers.com/java/java-concurrentskiplistmap/) – an implementation class of ConcurrentNavigableMap.

**http://www.journaldev.com/1260/collections-in-java-tutorial#iterator-interface**

**2) Difference between HashMap and ConcurrentHashMap**

To better visualize the ConcurrentHashMap, let it consider as a group of HashMaps. To get and put key-value pairs from hashmap, you have to calculate the hashcode and look for correct bucket location in array of Collection.Entry. Rest you have read on previous related article on how hashmap works.

In concurrentHashMap, the **difference lies in internal structure to store these key-value pairs**. ConcurrentHashMap has an addition concept of segments. It will be easier to understand if you think of one segment equal to one HashMap [conceptually]. A concurrentHashMap is divided into number of segments [default 16] on initialization. ConcurrentHashMap allows similar number (16) of threads to access these segments concurrently so that each thread work on a specific segment during high concurrency.

This way, if when your key-value pair is stored in segment 10; code does not need to block other 15 segments additionally. This structure provides a very high level of concurrency.

In other words, **ConcurrentHashMap uses a multitude of locks, each lock controls one segment of the map**. When setting data in a particular segment, the lock for that segment is obtained. So essentially **update operations are synchronized**.  
**When getting data, a volatile read is used** without any synchronization. If the volatile read results in a miss, then the lock for that segment is obtained and entry is again searched in synchronized block.

I will go further deeper into this concept in my coming post. I will suggest you to subscribe email updates to get notified.

**3) Difference between HashMap and Collections.synchronizedMap(HashMap)**

HashMap is non-synchronized and Collections.synchronizedMap() returns a wrapped instance of HashMap which has all get, put methods synchronized.

Essentially, **Collections.synchronizedMap() returns the reference of internally created inner-class “SynchronizedMap”**, which contains key-value pairs of input HashMap, passed as argument.

This instance of inner class has nothing to do with original parameter HashMap instance and is completely independent.

**4) Difference between ConcurrentHashMap and Collections.synchronizedMap( HashMap )**

Both are synchronized version of HashMap, with difference in their core functionality and internal structure.

As stated above, ConcurrentHashMap is consisting of internal segments which can be viewed as independent HashMaps, conceptually. All such segments can be locked by separate threads in high concurrent executions. In this way, **multiple threads can get/put key-value pairs from ConcurrentHashMap without blocking/waiting for each other**.

In Collections.synchronizedMap(), we get a synchronized version of HashMap and **it is accessed in blocking manner**. This means if multiple threads try to access synchronizedMap at same time, they will be allowed to get/put key-value pairs one at a time in synchronized manner.

**5) Difference between HashMap and HashTable**

The major difference is that **HashTable is synchronized and HashMap is not**.

If asked for other reasons, tell them, **HashTable is legacy class** (part of JDK 1.0) which was promoted into collections framework by implementing Map interface later. It still has some **extra features like Enumerator** with it, which HashMap lacks.

Another minor reason can be: **HashMap supports null key** (mapped to zero bucket), HashTable does not support null keys and throws NullPointerException on such attempt.

**6) Difference between HashTable and Collections.synchronized(HashMap)**

So far you must have got the core idea of the similarities between them. Both are synchronized version of collection. Both have synchronized methods inside class. Both are blocking in nature i.e. multiple threads will need to wait for getting the lock on instance before putting/getting anything out of it.

So what is the difference? Well, **NO major difference** for above said reasons. Performance is also same for both collections.

Only thing, which separates them, is the fact **HashTable is legacy** class promoted into collection framework. It got its own extra features like enumerators.

**What is Java Collections Framework? List out some benefits of Collections framework?**

A collections framework is a unified architecture for representing and manipulating collections of objects.

Collections are used in every programming language and initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **and Array**. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.  
Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package.  
Some of the benefits of collections framework are;

* + Reduced development effort by using core collection classes rather than implementing our own collection classes.
  + Code quality is enhanced with the use of well tested collections framework classes.
  + Reduced effort for code maintenance by using collection classes shipped with JDK.
  + Reusability and Interoperability

**What is the benefit of Generics in Collections Framework?**

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.  
This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don’t need to use casting and instanceof operator.

**What are the basic interfaces of Java Collections Framework?**

[Collection](http://www.journaldev.com/1260/java-collections-framework-tutorial#collection-interface) is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

[Set](http://www.journaldev.com/1260/java-collections-framework-tutorial#set-interface) is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

[List](http://www.journaldev.com/1260/java-collections-framework-tutorial#list-interface) is an ordered collection and can contain duplicate elements. You can access any element from its index. List is more like array with dynamic length.

A [Map](http://www.journaldev.com/1260/java-collections-framework-tutorial#map-interface) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are [Queue](http://www.journaldev.com/1260/java-collections-framework-tutorial#queue-interface), [Dequeue](http://www.journaldev.com/1260/java-collections-framework-tutorial#dequeue-interface), [Iterator](http://www.journaldev.com/1260/java-collections-framework-tutorial#iterator-interface), [SortedSet](http://www.journaldev.com/1260/java-collections-framework-tutorial#sortedset-interface), [SortedMap](http://www.journaldev.com/1260/java-collections-framework-tutorial#sortedmap-interface) and [ListIterator](http://www.journaldev.com/1260/java-collections-framework-tutorial#listiterator-interface).

**Why Collection doesn’t extend Cloneable and Serializable interfaces?**

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don’t.  
A lot of the Collection implementations have a public clone method. However, it doesn’t really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation.  
The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.  
So mandating cloning and serialization in all implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

**5.Why Map interface doesn’t extend Collection interface?**

Although Map interface and its implementations are part of Collections Framework, Map is not collections and collections are not Map. Hence it doesn’t make sense for Map to extend Collection or vice versa.  
If Map extends Collection interface, then where are the elements? Map contains key-value pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

**What is an Iterator?**

Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using iterator () method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Java Collection iterator provides a generic way for traversal through the elements of a collection and implements [**Iterator Design Pattern**](http://www.journaldev.com/1716/iterator-design-pattern-in-java-example-tutorial).

**What is difference between Enumeration and Iterator interface?**

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object, which is being iterated by it.  
Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make it’s functionality clear.

**Why there is not method like Iterator.add() to add elements to the collection?**

The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.

**Why Iterator don’t have a method to get next element directly without moving the cursor?**

It can be implemented on top of current Iterator interface but since it’s use will be rare, it doesn’t make sense to include it in the interface that everyone has to implement.

**What is different between Iterator and ListIterator?**

* + We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
  + Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
  + ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

**What are different ways to iterate over a list?**

We can iterate over a list in two different ways – using iterator and using for-each loop.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | List<String> strList = new ArrayList<>();  //using for-each loop  for(String obj : strList){      System.out.println(obj);  }  //using iterator  Iterator<String> it = strList.iterator();  while(it.hasNext()){      String obj = it.next();      System.out.println(obj);  }  //using listiterator  ListIterator<String> it = strList.listIterator();  while(it.hasNext()){      String obj = it.next();      System.out.println(obj);  } |

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw ConcurrentModificationException.

**What do you understand by iterator fail-fast property?**

Iterator fail-fast property checks for any modification in the structure of the underlying collection every time we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

**What is difference between fail-fast and fail-safe?**

Iterator fail-safe property work with the clone of underlying collection, hence it’s not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe.  
Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException.

**How to avoid ConcurrentModificationException while iterating a collection?**

We can use concurrent collection classes to avoid ConcurrentModificationException while iterating over a collection, for example CopyOnWriteArrayList instead of ArrayList, ConcurrentHashMap instead of HashMap.

**Why there are no concrete implementations of Iterator interface?**

Iterator interface declare methods for iterating a collection but its implementation is responsibility of the Collection implementation classes. Every collection class that returns an iterator for traversing has its own Iterator implementation nested class.  
This allows collection classes to choose whether iterator is fail-fast or fail-safe. For example ArrayList iterator is fail-fast whereas CopyOnWriteArrayList iterator is fail-safe.

**What is UnsupportedOperationException?**

UnsupportedOperationException is the exception used to indicate that the operation is not supported. It’s used extensively in [JDK](http://www.journaldev.com/546/difference-between-jdk-jre-and-jvm-in-java) classes, in collections framework java.util.Collections.UnmodifiableCollection throws this exception for all add and remove operations.

**How HashMap works in Java?**

HashMap stores key-value pair in Map. Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.

When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.

When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return its value. Below image will explain these details clearly.

The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is 32 and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry, if map size is greater than threshold, HashMap rehashes the contents of map into a new array with a larger capacity. The capacity is always power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it’s good idea to initialize the HashMap with correct capacity and load factor.

**What is the importance of hashCode() and equals() methods?**

HashMap uses Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key’s might produce same hashCode() and equals() output and in that case rather than storing it at different location, HashMap will consider them same and overwrite them.

Similarly all the collection classes that doesn’t store duplicate data use hashCode() and equals() to find duplicates, so it’s very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.

* + If o1.equals(o2), then o1.hashCode() == o2.hashCode()should always be true.
  + If o1.hashCode() == o2.hashCode is true, it doesn’t mean that o1.equals(o2) will be true.

**Can we use any class as Map key?**

We can use any class as Map Key, however following points should be considered before using them.

* + If the class overrides equals() method, it should also override hashCode() method.
  + The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.
  + If a class field is not used in equals(), you should not use it in hashCode() method.
  + Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.  
    For example, let’s say I have a class MyKey that I am using for HashMap key.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //MyKey name argument passed is used for equals() and hashCode()  MyKey key = new MyKey("Pankaj"); //assume hashCode=1234  myHashMap.put(key, "Value");    // Below code will change the key hashCode() and equals()  // but it's location is not changed.  key.setName("Amit"); //assume new hashCode=7890    //below will return null, because HashMap will try to look for key  //in the same index as it was stored but since key is mutated,  //there will be no match and it will return null.  myHashMap.get(new MyKey("Pankaj")); |

* + This is the reason why String and Integer are mostly used as HashMap keys.

**What are different Collection views provided by Map interface?**

Map interface provides three collection views:

* + **Set keySet()**: Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.
  + **Collection values()**: Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.
  + **Set<Map.Entry<K, V>> entrySet()**: Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

**What is difference between HashMap and Hashtable?**

HashMap and Hashtable both implements Map interface and looks similar, however there are following difference between HashMap and Hashtable.

* + HashMap allows null key and values whereas Hashtable doesn’t allow null key and values.
  + Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.
  + LinkedHashMap was introduced in Java 1.4 as a subclass of HashMap, so incase you want iteration order, you can easily switch from HashMap to LinkedHashMap but that is not the case with Hashtable whose iteration order is unpredictable.
  + HashMap provides Set of keys to iterate and hence it’s fail-fast but Hashtable provides Enumeration of keys that doesn’t support this feature.
  + Hashtable is considered to be legacy class and if you are looking for modifications of Map while iterating, you should use ConcurrentHashMap.

**How to decide between HashMap and TreeMap?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

**What are similarities and difference between ArrayList and Vector?**

ArrayList and Vector are similar classes in many ways.

* + Both are index based and backed up by an array internally.
  + Both maintains the order of insertion and we can get the elements in the order of insertion.
  + The iterator implementations of ArrayList and Vector both are fail-fast by design.
  + ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector.

* + Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
  + ArrayList is faster than Vector because it doesn’t have any overhead because of synchronization.
  + ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

**What is difference between Array and ArrayList? When will you use Array over ArrayList?**

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.  
Arrays are fixed size whereas ArrayList size is dynamic.  
Arrays don’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc.

Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

* + If the size of list is fixed and mostly used to store and traverse them.
  + For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.
  + If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

**What is difference between ArrayList and LinkedList?**

ArrayList and LinkedList both implement List interface but there are some differences between them.

* ArrayList is an index based data structure backed by Array, so it provides random access to it’s elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to it’s previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.
* Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.
* LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

**Which collection classes provide random access of it’s elements?**

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to its elements.

**What is EnumSet?**

java.util.EnumSet is Set implementation to use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. EnumSet is not synchronized and null elements are not allowed. It also provides some useful methods like copyOf(Collection c), of(E first, E… rest) and complementOf(EnumSet s).

**Which collection classes are thread-safe?**

Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment. Java 1.5 Concurrent API included some collection classes that allows modification of collection while iteration because they work on the clone of the collection, so they are safe to use in multi-threaded environment.

**What are concurrent Collection Classes?**

Java 1.5 Concurrent package (java.util.concurrent) contains thread-safe collection classes that allow collections to be modified while iterating. By design iterator is fail-fast and throws ConcurrentModificationException. Some of these classes are CopyOnWriteArrayList, ConcurrentHashMap,  CopyOnWriteArraySet.

**What is BlockingQueue?**

Java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue as it is handled by implementation classes of BlockingQueue.

Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc.  
**What is Queue and Stack, list their differences?**

Both Queue and Stack are used to store data before processing them. java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it’s not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.  
Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.  
Stack is a class that extends Vector whereas Queue is an interface.

**What is Collections Class?**

java.util.Collections is a utility class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, “wrappers”, which return a new collection backed by a specified collection, and a few other odds and ends.

This class contains methods for collection framework algorithms, such as binary search, sorting, shuffling, reverse etc.

### Comparable and Comparator

**What is Comparable and Comparator interface?**

Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can’t chose the field on which we want to sort the Object.

Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

**What is difference between Comparable and Comparator interface?**

Comparable and Comparator interfaces are used to sort collection or array of objects.

Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic.  
Comparator interface is used to provide different algorithms for sorting and we can choose the comparator we want to use to sort the given collection of objects.

**How can we sort a list of Objects?**

If we need to sort an array of Objects, we can use Arrays.sort(). If we need to sort a list of objects, we can use Collections.sort(). Both these classes have overloaded sort() methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator).  
Collections internally use Arrays sorting method, so both of them have same performance except that Collections take some time to convert list to array.

**While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?**

We can create a read-only collection using **Collections.unmodifiableCollection(Collection c)** method before passing it as argument; this will make sure that any operation to change the collection will throw UnsupportedOperationException.

**How can we create a synchronized collection from given collection?**

We can use **Collections.synchronizedCollection(Collection c)** to get a synchronized (thread-safe) collection backed by the specified collection.

**What are common algorithms implemented in Collections Framework?**

Java Collections Framework provides algorithm implementations that are commonly used such as sorting and searching. Collections class contains these method implementations. Most of these algorithms work on List but some of them are applicable for all kinds of collections.  
Some of them are sorting, searching, shuffling, min-max values.

**What is Big-O notation? Give some examples?**

The Big-O notation describes the performance of an algorithm in terms of number of elements in a data structure. Since Collection classes are actually data structures, we usually tend to use Big-O notation to choose the collection implementation to use based on time, memory and performance.

Example 1: ArrayList get(index i) is a constant-time operation and doesn’t depend on the number of elements in the list. So it’s performance in Big-O notation is O(1).  
Example 2: A linear search on array or list performance is O(n) because we need to search through entire list of elements to find the element.

**3. What is difference between fail-fast and fail-safe Iterators?**

Fail-fast Iterators throws ConcurrentModificationException when one [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure. On the other hand [fail-safe Iterators](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) works on copy of collection instead of original collection.

**4. How do you remove an entry from a Collection? and subsequently what is difference between remove() method of Collection and remove() method of Iterator, which one you will use, while removing elements during iteration?**  
  
Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of this method to remove an entry from Collection, while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

**5. What is difference between Synchronized Collection and Concurrent Collection?**

Java 5 has added several new Concurrent Collection classes e.g. ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.synchronizedMap()Utility function.One Significant difference is that Concurrent Collections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization.

**6. What is difference between Iterator and Enumeration?**

Iterator duplicate functionality of Enumeration with one addition of remove() method and both provide navigation functionally on objects of Collection .Another difference is that Iterator is more safe than Enumeration and doesn't allow another thread to modify collection object during iteration except remove() method and throws ConcurrentModificaitonException.

**7. How does HashSet is implemented in Java, How does it uses Hashing ?**

This is a tricky question in Java, because for hashing you need both key and value and there is no key for store it in a bucket, then how exactly HashSet store element internally. Well, HashSet is built on top of HashMap. If you look at source code of java.util.HashSet class, you will find that that it uses a HashMap with same values for all keys, as shown below :  
  
private transient HashMap map;  
  
// Dummy value to associate with an Object in the backing Map  
private static final Object PRESENT = new Object();  
  
When you call add() method of HashSet, it put entry in HashMap :  
  
public boolean add(E e) {  
  return map.put(e, PRESENT)==null;  
}  
  
Since keys are unique in a HashMap, it provides uniqueness guarantee of Set interface.  
  
  
**8. What do you need to do to use a custom object as key in Collection classes like Map or Set?**  
If you are using any custom object in Map as key, you need to override equals() and hashCode() method, and make sure they follow their contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent to compareTo() method, otherwise those collection will not follow their contracts e.g. Set may allow duplicates.

**10. When do you use ConcurrentHashMap in Java?**

ConcurrentHashMap is better suited for situation where you have multiple readers and one Writer or fewer writers since Map gets locked only during write operation. If you have equal number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in line of Hashtable or synchronized HashMap.

**11. What is difference between Set and List in Java?**

* Set doesn't allowed duplicate while List does.
* List maintains insertion order while Set doesn't.

**12. How do you Sort objects on collection?**

Sorting is implemented using Comparable and Comparator in Java and when you call Collections.sort() it gets sorted based on natural order specified in compareTo() method while Collections.sort(Comparator) will sort objects based on compare() method of Comparator.

**13. What is difference between Vector and ArrayList?**

* Vector is synchronized whereas ArrayList is not.
* The Vector class provides the capability to implement a growable array of objects.
* In vector the data is retrieved using the elementAt() method while in ArrayList, it is done using the get() method.
* ArrayList has no default size while vector has a default size of 10.
* when you want programs to run in multithreading environment then use concept of vector because it is synchronized. But ArrayList is not synchronized so, avoid use of it in a multithreading environment.

**14. What is difference between HashMap and HashSet?**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **HashMap** | **HashSet** |
| Interface | This is core difference among them. HashMap implements Map interface | HashSet implement Set interface |
| Method for storing data | It stores data in a form of key-value pair. So it uses put(key,value) method for storing data | It uses add(value) method for storing data |
| Duplicates | HashMap allows duplicate value but not duplicate keys | HashSet does not allow duplicate values. |
| Performance | It is faster than hashset as values are stored with unique keys | It is slower than HashMap |
| HashCode Calculation | In hash map hashcode value is calculated using key object | In this,hashcode is calculated on the basis of value object. Hashcode can be same for two value object so we have to implement equals() method. If equals() method return false then two objects are different. |

**15) What is NavigableMap in Java ? What is benefit over Map?**  
NavigableMap Map was added in Java 1.6, it adds navigation capability to Map data structure. It provides methods like lowerKey() to get keys which is less than specified key, floorKey() to return keys which is less than or equal to specified key, ceilingKey() to get keys which is greater than or equal to specified key and higherKey() to return keys which is greater specified key from a Map. It also provide similar methods to get entries e.g. lowerEntry(), floorEntry(), ceilingEntry() and higherEntry(). Apart from navigation methods, it also provides utilities to create sub-Map e.g. creating a Map from entries of an exsiting Map like tailMap, headMap and subMap. headMap() method returns a NavigableMap whose keys are less than specified, tailMap() returns a NavigableMap whose keys are greater than the specified and subMap() gives a NavigableMap between a range, specified by toKey to fromKey.    
  
**16) Which one you will prefer between Array and ArrayList for Storing object and why?**

Though ArrayList is also backed up by array, it offers some usability advantage over array in Java. Array is fixed length data structure, once created you cannot change its length. On the other hand, ArrayList is dynamic, it automatically allocate a new array and copies content of old array, when it resize. Another reason of using ArrayList over Array is support of Generics. Array doesn't support Generics, and if you store an Integer object on a String array, you will only going to know about it at runtime, when it throws ArrayStoreException. On the other hand, if you use ArrayList, compiler and IDE will catch those errors on the spot. So if you know size in advance and you don't need re-sizing than use array, otherwise use ArrayList.

**17) Can we replace Hashtable with ConcurrentHashMap?**

Yes we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap. But you need to be careful with code which relies on locking behavior of Hashtable. Since Hashtable locks whole Map instead of portion of Map, compound operations like if(Hashtable.get(key) == null) put(key, value) works in Hashtable but not in concurrentHashMap. Instead of this use putIfAbsent() method of ConcurrentHashMap.

**18) What is CopyOnWriteArrayList, how it is different than ArrayList and Vector?**

CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. Better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking. Also CopyOnWriteArrayList doesn't throw any ConcurrentModification Exception. It’s different than ArrayList because it’s thread-safe and ArrayList is not thread safe and it’s different than Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

**19) Why ListIterator has add() method but Iterator doesn't or Why add() method is declared in ListIterator and not on Iterator.**

ListIterator has add() method because of its ability to traverse or iterate in both direction of collection. It maintains two pointers in terms of previous and next call and in position to add new element without affecting current iteration.

**20) When does ConcurrentModificationException occur on iteration?**

When you remove object using Collection's or List's remove method e.g. remove(Object element) or remove(int index), instead of Iterator's remove() method than ConcurrentModificationException occur. As per Iterator's contract, if it detect any structural change in Collection e.g. adding or removing of element, once Iterator begins, it can throw ConcurrentModificationException.

**21) Difference between Set, List and Map Collection classes?**

java.util.Set, java.util.List and java.util.Map defines three of most popular data structure support in Java. Set provides uniqueness guarantee i.e.g you can not store duplicate elements on it, but it's not ordered. On the other hand List is an ordered Collection and also allows duplicates. Map is based on hashing and stores key and value in an Object called entry. It provides O(1) performance to get object, if you know keys, if there is no collision. Popular impelmentation of Set is HashSet, of List is ArrayList and LinkedList, and of Map are HashMap, Hashtable and ConcurrentHashMap. Another key difference between Set, List and Map are that Map doesn't implement Collection interface, while other two does. For a more detailed answer, see Set vs List vs Map in Java

**22) What is BlockingQueue, how it is different than other collection classes?**

BlockingQueue is a Queue implementation available in java.util.concurrent package. It's one of the concurrent Collection class added on Java 1.5, main difference between BlockingQueue and other collection classes is that apart from storage, it also provides flow control. It can be used in inter thread communication and also provides built-in thread-safety by using happens-before guarantee. You can use BlockingQueue to solve Producer Consumer problem, which is what is needed in most of concurrent applications.

25) What is Deque? when do you use it ?

Deque is used is situation where you want to retrieve elements from both head and tail. If you want a simple stack there is no need to go for a deque.

**What are best practices related to Java Collections Framework?**

* + Choosing the right type of collection based on the need, for example if size is fixed, we might want to use Array over ArrayList. If we have to iterate over the Map in order of insertion, we need to use TreeMap. If we don’t want duplicates, we should use Set.
  + Some collection classes allow to specify the initial capacity, so if we have an estimate of number of elements we will store, we can use it to avoid rehashing or resizing.
  + Write program in terms of interfaces not implementations, it allows us to change the implementation easily at later point of time.
  + Always use Generics for type-safety and avoid ClassCastException at runtime.
  + Use immutable classes provided by JDK as key in Map to avoid implementation of hashCode() and equals() for our custom class.
  + Use Collections utility class as much as possible for algorithms or to get read-only, synchronized or empty collections rather than writing own implementation. It will enhance code-reuse with greater stability and low maintainability.

**What is Java Priority Queue?**

PriorityQueue is an unbounded queue based on a priority heap and the elements are ordered in their natural order or we can provide [Comparator](http://www.journaldev.com/780/java-comparable-and-comparator-example-to-sort-objects) for ordering at the time of creation. PriorityQueue doesn’t allow null values and we can’t add any object that doesn’t provide natural ordering or we don’t have any comparator for them for ordering. Java PriorityQueue is not [thread-safe](http://www.journaldev.com/1061/java-synchronization-and-thread-safety-tutorial-with-examples) and provided O(log(n)) time for enqueing and dequeing operations. Check this post for [java priority queue example](http://www.journaldev.com/1642/java-priority-queue-priorityqueue-example).

**Why can’t we write code as List<Number> numbers = new ArrayList<Integer>();?**

Generic doesn’t support sub-typing because it will cause issues in achieving type safety. That’s why List<T> is not considered as a subtype of List<S> where S is the super-type of T. To understanding why it’s not allowed, let’s see what could have happened if it has been supported.

|  |  |
| --- | --- |
| 1  2  3  4 | List<Long> listLong = new ArrayList<Long>();  listLong.add(Long.valueOf(10));  List<Number> listNumbers = listLong; // compiler error  listNumbers.add(Double.valueOf(1.23)); |

As you can see from above code that IF generics would have been supporting sub-typing, we could have easily add a Double to the list of Long that would have caused ClassCastException at runtime while traversing the list of Long.

**Why can’t we create generic array? Or write code as List<Integer>[] array = new ArrayList<Integer>[10];**

We are not allowed to create generic arrays because array carries type information of its elements at runtime. This information is used at runtime to throw ArrayStoreException if elements type doesn’t match to the defined type. Since generics type information gets erased at runtime by Type Erasure, the array store check would have been passed where it should have failed. Let’s understand this with a simple example code.

|  |  |
| --- | --- |
| 1  2  3  4  5 | List<Integer>[] intList = new List<Integer>[5]; // compile error  Object[] objArray = intList;  List<Double> doubleList = new ArrayList<Double>();  doubleList.add(Double.valueOf(1.23));  objArray[0] = doubleList; // this should fail but it would pass because at runtime intList and doubleList both are just List |

Arrays are covariant by nature i.e S[] is a subtype of T[] whenever S is a subtype of T but generics doesn’t support covariance or sub-typing as we saw in last question. So if we would have been allowed to create generic arrays, because of type erasure we would not get array store exception even though both types are not related.

1. **What is difference between poll() and remove() method of Queue interface ?**

Though both poll() and remove() method from Queue is used to remove object and returns head of the queue, there is subtle difference between them. If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example PriorityQueue keeps lowest element as per Comparator or Comparable at head position.

### Tricky 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.   
  
**2. What is immutable object? Can you write immutable object?**

Immutable classes are Java classes whose objects cannot be modified once created. Any modifications 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.

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables
4. Make all **mutable fields final** so that it’s value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

**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 ?**

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.

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

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.

Submit - Returns Future object, which can be used to check result of submitted task. Can be used to cancel or to check isDone etc.

Execute - doesn't return anything.

**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.

**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 see example in eclipse.

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

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

**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.  
  
**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.  
  
**17. Does not overriding hashcode() method has any performance implication ?**  
A poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.

### Thread

**18. What’s wrong using HashMap in multithreaded environment? When get() method go to infinite loop ?**  
answer is 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 ? 

|  |
| --- |
| Thread safe simply means that it may be used from multiple threads at the same time without causing problems. This can mean that access to any resources are synchronized, or whatever. |

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.

}

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

Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

**What is CyclicBarrier?**

CyclicBarrier is used to make threads wait for each other. It is used when different threads process a part of computation and when all threads have completed the execution, the result needs to be combined in the parent thread. In other words, a CyclicBarrier is used when multiple thread carry out different sub tasks and the output of these sub tasks need to be combined to form the final output. After completing its execution, threads call await() method and wait for other threads to reach the barrier. Once all the threads have reached, the barriers then give the way for threads to proceed.

**What is CountDownLatch** **?**

A java.util.concurrent.CountDownLatch is a concurrency construct that allows one or more threads to wait for a given set of operations to complete.

A CountDownLatch is initialized with a given count. This count is decremented by calls to the countDown()method. Threads waiting for this count to reach zero can call one of the await() methods. Calling await() blocks the thread until the count reaches zero.

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

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

### Regular Questions

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

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 ?**

You cannot 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?**

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?**

Answer is you can very well throw super class of RuntimeException in overridden method but you cannot do same if it’s checked Exception.

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

No you can not access non static variable in static context in Java.

### OOPS

**Q) What is the Java API?**  
 The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets.

**Q) Describe the principles of OOPS**  
 There are four main principals of oops which are called Polymorphism, Inheritance and Encapsulation, abstraction

**Q) Explain the Inheritance principle**  
 Inheritance is the process by which one object acquires the properties of another object. Inheritance allows well-tested procedures to be reused and enables changes to make once and have effect in all relevant places

**Q) Explain the Polymorphism principle. Explain the different forms of Polymorphism.**  
 Polymorphism in simple terms means one name many forms. Polymorphism enables one entity to be used as a general category for different types of actions. The specific action is determined by the exact nature of the situation.  
Polymorphism exists in three distinct forms in Java:   
\* Method overloading   
\* Method overriding through inheritance   
\* Method overriding through the Java interface

**Q) What type of parameter passing does Java support?**  
 In Java the arguments (primitives and objects) are always passed by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object.

**Q) Explain the Encapsulation principle.**  
 Encapsulation is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse. Objects allow procedures to be encapsulated with their data to reduce potential interference. One way to think about encapsulation is as a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper.

**Q) What is data encapsulation?**   
 Encapsulation may be used by creating 'get' and 'set' methods in a class (JAVABEAN) which are used to access the fields of the object. Typically the fields are made private while the get and set methods are public. Encapsulation can be used to validate the data that is to be stored, to do calculations on data that is stored in a field or fields, or for use in introspection (often the case when using javabeans in Struts, for instance). Wrapping of data and function into a single unit is called as data encapsulation. Encapsulation is nothing but wrapping up the data and associated methods into a single unit in such a way that data can be accessed with the help of associated methods. Encapsulation provides data security. It is nothing but data hiding.

**Q) What do you understand by a variable?**   
 Variable is a named memory location that can be easily referred in the program. The variable is used to hold the data and it can be changed during the course of the execution of the program.

**Q) Name primitive Java types.**   
 The 8 primitive types are byte, char, short, int, long, float, double, and boolean.

### Access specifiers

**Q) State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.**  
 public : Public class is visible in other packages, field is visible everywhere (class must be public too)  
private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.  
protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature. This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.  
default: What you get by default ie, without any access modifier (ie, public private or protected).It means that it is visible to all within a particular package.

**Q) What if the main method is declared as private?**   
 The program compiles properly but at runtime it will give "Main method not public." message.

**Q) What if I write static public void instead of public static void?**   
 Program compiles and runs properly.

**Q) What environment variables do I need to set on my machine in order to be able to run Java programs?**  
 CLASSPATH and PATH are the two variables.

**Q) How can one prove that the array is not null but empty using one line of code?**   
 Print args.length. It will print 0. That means it is empty. But if it would have been null then it would have thrown a NullPointerException on attempting to print args.length.

**Q) If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?**   
 It is empty. But not null.

**Q) Can I have multiple main methods in the same class?**  
 We can have multiple overloaded main methods but there can be only one main method with the following signature:   
public static void main(String[] args) {}

**Q) Do I need to import java.lang package any time?**   
 No. It is by default loaded internally by the JVM.

**Q) What is Overriding?**   
 When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass.  
When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass. Methods may be overridden to be more public, not more private.

**Q) Can a top level class be private or protected?**   
 No. A top level class can not be private or protected. It can have either "public" or no modifier. If it does not have a modifier it is supposed to have a default access. If a top level class is declared as private the compiler will complain that the "modifier private is not allowed here". This means that a top level class can not be private. Same is the case with protected.

**Q) What is the difference between a while statement and a do statement?**   
 A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.

**Q) What is the difference between static and non-static variables?**   
 A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

**Q) How are this() and super() used with constructors?**   
 This() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.

**Q) What is the first argument of the String array in main method?**   
 The String array is empty. It does not have any element. This is unlike C/C++ where the first element by default is the program name.

**Q) What if I do not provide the String array as the argument to the method?**   
 Program compiles but throws a runtime error "NoSuchMethodError".

**Q) Can an application have multiple classes having main method?**   
 Yes it is possible. While starting the application we mention the class name to be run. The JVM will look for the Main method only in the class whose name you have mentioned. Hence there is not conflict amongst the multiple classes having main method.

**Q) What is the difference between a constructor and a method?**  
 A constructor is a member function of a class that is used to create objects of that class, invoked using the new operator. It has the same name as the class and has no return type. They are only called once, whereas member functions can be called many times. A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator. Constructor will be automatically invoked when an object is created whereas method has to be called explicitly.  
super.method(); is used to call a super class method from a sub class. To call a constructor of the super class, we use the super(); statement as the first line of the subclass constructor.

**Q) What is the difference between declaring a variable and defining a variable?**  
 In declaration we just mention the type of the variable and it's name. We do not initialize it. But defining means declaration + initialization.  
e.g String s; is just a declaration while String s = new String ("abcd"); Or String s = "abcd"; are both definitions.

**Q) Give a simplest way to find out the time a method takes for execution without using any profiling tool?**  
 Read the system time just before the method is invoked and immediately after method returns. Take the time difference, which will give you the time taken by a method for execution. To put it in code.  
Long start = System.currentTimeMillis ();  
method ();  
long end = System.currentTimeMillis ();  
System.out.println ("Time taken for execution is " + (end - start));  
Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amount of processing.

**Q) What are Wrapper Classes? Describe the wrapper classes in Java.**   
 Wrapper classes are classes that allow primitive types to be accessed as objects. Wrapper class is wrapper around a primitive data type.   
Following table lists the primitive types and the corresponding wrapper classes:  
Primitive Wrapper  
Boolean java.lang.Boolean  
Byte java.lang.Byte  
Char java.lang.Character  
double java.lang.Double  
Float java.lang.Float  
Int java.lang.Integer  
Long java.lang.Long  
Short java.lang.Short  
Void java.lang.Void

**Q) Why do we need wrapper classes?**  
 It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these reasons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

**Q) What is the Locale class?**  
 The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region.

**Q) What are some alternatives to inheritance?**  
 Delegation is an alternative to inheritance. Delegation means that you include an instance of another class as an instance variable, and forward messages to the instance. It is often safer than inheritance because it forces you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense. On the other hand, it makes you write more code, and it is harder to re-use (because it is not a subclass).

**Q) What happens if you don't initialize an instance variable of any of the primitive types in Java?**  
 Java by default initializes it to the default value for that primitive type. Thus an int will be initialized to 0, a boolean will be initialized to false.

|  |  |
| --- | --- |
| **Data Type** | **Default Value (for fields)** |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

**Q) What will be the initial value of an object reference which is defined as an instance variable?**   
 The object references are all initialized to null in Java. However in order to do anything useful with these references, you must set them to a valid object, else you will get NullPointerExceptions everywhere you try to use such default initialized references.

**Q) What are the different scopes for Java variables?**   
 The scope of a Java variable is determined by the context in which the variable is declared. Thus a java variable can have one of the three scopes at any given point in time.  
1. Instance : - These are typical object level variables, they are initialized to default values at the time of creation of object, and remain accessible as long as the object accessible.  
2. Local : - These are the variables that are defined within a method. They remain accessible only during the course of method execution. When the method finishes execution, these variables fall out of scope.  
3. Static: - These are the class level variables. They are initialized when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance.

**Q) What is the default value of the local variables?**  
 The local variables are not initialized to any default value, neither primitives nor object references. If you try to use these variables without initializing them explicitly, the java compiler will not compile the code. It will complain about the local variable not being initialized.

**Q) What will be the default values of all the elements of an array defined as an instance variable?**  
 If the array is an array of primitive types, then all the elements of the array will be initialized to the default value corresponding to that primitive type. e.g. All the elements of an array of int will be initialized to 0, while that of boolean type will be initialized to false. Whereas if the array is an array of references (of any type), all the elements will be initialized to null.

**Q) What gives java it's "write once and run anywhere" nature?**   
 All Java programs are compiled into class files that contain bytecodes. These byte codes can be run in any platform and hence java is said to be platform independent.

**Q) Expain the reason for each keyword of public static void main(String args[])?**   
 public: The method can be accessed outside the class / package   
static: You need not have an instance of the class to access the method   
void: Your application need not return a value, as the JVM launcher would return the value when it exits   
main(): This is the entry point for the application   
If the main() was not static, you would require an instance of the class in order to execute the method.

Q) What is System.out.println

System is a predefined final class, out is a PrintStream object and println is a built-in overloaded method in the out object.

System.out.println prints the argument passed, into the System.out which is generally stdout.

* System – is a final class and cannot be inherited. As per javadoc, “…Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an [array](http://javapapers.com/core-java/java-array/)…”
* out – is a [static](http://javapapers.com/core-java/explain-the-java-static-modifier/)member field of [System](http://docs.oracle.com/javase/6/docs/api/java/lang/System.html)class and is of type [PrintStream](http://docs.oracle.com/javase/1.3/docs/api/java/io/PrintStream.html). Its [access specifiers](http://javapapers.com/core-java/access-modifiers-in-java-explain/)are public [final](http://javapapers.com/core-java/explain-the-final-keyword-in-java/). This gets instantiated during startup and gets mapped with standard output console of the host. This stream is open by itself immediately after its instantiation and ready to accept data.
* println – println prints the argument passed to the standard console and a newline. There are multiple println methods with different arguments ([overloading](http://javapapers.com/core-java/overloading-and-overriding/)). Every println makes a call to print method and adds a newline. print callswrite() and the story goes on like that.

**Q) what is difference between == and equals?**  
 The == operator compares two objects to determine if they are the same object in memory i.e. present in the same memory location. It is possible for two String objects to have the same value, but located in different areas of memory.  
== compares references while .equals compares contents. The method public boolean equals(Object obj) is provided by the Object class and can be overridden. The default implementation returns true only if the object is compared with itself, which is equivalent to the equality operator == being used to compare aliases to the object. String, BitSet, Date, and File override the equals() method. For two String objects, value equality means that they contain the same character sequence. For the Wrapper classes, value equality means that the primitive values are equal.

**Q) What is the ResourceBundle class?**   
 The ResourceBundle class is used to store locale-specific resources that can be loaded by a program to tailor the program's appearance to the particular locale in which it is being run

**Q) Why there are no global variables in Java?**  
 Global variables are globally accessible. Java does not support globally accessible variables due to following reasons:   
-The global variables breaks the referential transparency  
-Global variables creates collisions in namespace.

**Q) How to convert String to Number in java program?**  
 The valueOf() function of Integer class is is used to convert string to Number. Here is the code example:  
String numString = "1000";   
int id=Integer.valueOf(numString).intValue();

**Q) What is implicit casting?**  
 Implicit casting is the process of simply assigning one entity to another without any transformation guidance to the compiler. This type of casting is not permitted in all kinds of transformations and may not work for all scenarios.  
Example :  
int i = 1000;  
long j = i; //Implicit casting

**Q) What is explicit casting?**  
 Explicit casting in the process in which the complier are specifically informed to about transforming the object.  
Example :  
long i = 700.20;  
int j = (int) i; //Explicit casting

**Q) What is a native method?**  
 A native method is a method that is implemented in a language other than Java

**Q) What is the difference between the Boolean & operator and the && operator?**   
 If an expression involving the Boolean “&” operator is evaluated, both operands are evaluated. Then the “&” operator is applied to the operand. When an expression involving the “&&” operator is evaluated, the first operand is evaluated. If the first operand returns a value of true then the second operand is evaluated. The “&&” operator is then applied to the first and second operands. If the first operand evaluates to false, the evaluation of the second operand is skipped. Operator “&” has no chance to skip both sides evaluation and “&&” operator does.

**Q) How can I swap two variables without using a third variable?**  
 Add two variables and assign the value into First variable. Subtract the Second value with the result Value. and assign to Second variable. Subtract the Result of First Variable With Result of Second Variable and Assign to First Variable. Example:  
int a=5,b=10;a=a+b; b=a-b; a=a-b;  
An other approach to the same question  
You use an XOR swap.  
for example:  
int a = 5; int b = 10;  
a = a ^ b;  
b = a ^ b;  
a = a ^ b;

**Q) Explain working of Java Virtual Machine (JVM)**  
 The goal of a runtime instance of JVM is to run one Java application. i.e. it is an abstract computing machine like any other real computing machine which first converts “.java” file into “.class” file by using Compiler (.class is nothing but byte code file.) and Interpreter reads byte codes. The instance starts by invoking the main() method of some initial class. The definition of the method is  
public static void main(String[] args){  
// do something...  
}  
The main() method serves as a starting point for that application's initial thread. The initial thread can spawn other threads. There are deamon and nondeamon threads.The Java virtual machine organizes the memory it needs to execute a program into several runtime data areas including method area, heap, Java stacks, pc registers, and native method stacks. Each thread gets its own pc register and Java stack. JVM's stack-based architecture facilitates JIT and dynamic compiler's optimization work. The range of values for each data type are defined in the JVM as follows:  
byte 8-bit signed two's complement integer (-27 to 27 - 1, inclusive)  
short 16-bit signed two's complement integer (-215 to 215 - 1, inclusive)  
int 32-bit signed two's complement integer (-231 to 231 - 1, inclusive)  
long 64-bit signed two's complement integer (-263 to 263 - 1, inclusive)  
char 16-bit unsigned Unicode character (0 to 216 - 1, inclusive)  
float 32-bit IEEE 754 single-precision float  
double 64-bit IEEE 754 double-precision float

**Q) What is reflection API? How are they implemented?**  
 Reflection is the process of introspecting the features and state of a class at runtime and dynamically manipulate at run time. This is supported using Reflection API with built-in classes like Class, Method, Fields, Constructors etc. Example: Using Java Reflection API we can get the class name, by using the getName method.

**Q) What is difference between String and StringTokenizer?**  
 A StringTokenizer is utility class used to break up string.  
Example:  
StringTokenizer st = new StringTokenizer("Hello World");  
while (st.hasMoreTokens()) {  
System.out.println(st.nextToken());  
}  
Output:  
Hello  
World

**Q) What is the difference between String and StringBuffer**   
 String is an immutable class, i.e you cannot change the values of that class   
Example:   
String str = "java"; // address in memory say 12345   
And now if you assign a new value to the variable str then   
str = "core java"; then the value of the variable at address 12345 will not change but a new memory is allocated for this variable say 54321   
So in the memory address 12345 will have value "java"   
And the memory address 54321 will have value "core java" and the variable str will now be pointing to address 54321 in memory   
  
StringBuffer can be modified dynamically   
Example:   
StringBuffer strt ="java" // address in memory is say 12345   
And now if you assign a new value to the variable str then   
Str = "core java"; then value in the address of memory will get replaced, a new memory address is not allocated in this case.

**Q) Does JVM maintain a cache by itself? Does the JVM allocate objects in heap? Is this the OS heap or the heap maintained by the JVM? Why**  
 yes, the JVM maintains a cache by itself. It creates the Objects on the HEAP, but references to those objects are on the STACK.

**Q) Does java Support multiple inheritance**   
 Java does not support multiple inheritance directly like C++, because then it is prone to ambiguity, example if a class extends 2 other classes and these 2 parent classes have same method names then there is ambiguity. Hence in Java Multiple inheritance is supported using Interfaces.

**Q) Where and how can you use a private constructor?**  
 Private constructor can be used if you do not want any other class to instantiate the class. This concept is generally used in Singleton Design Pattern. The instantiation of such classes is done from a static public method.

**Q) Differentiate between a Class and an Object?**  
 The Object class is the highest-level class in the Java class hierarchy. The Class class is used to represent the classes and interfaces that are loaded by a Java program. The Class class is used to obtain information about an object's design. A Class is only a definition or prototype of real life object. Whereas an object is an instance or living representation of real life object. Every object belongs to a class and every class contains one or more related objects.

**Q) What is a singleton class**  
 A singleton is an object that cannot be instantiated. The restriction on the singleton is that there can be only one instance of a singleton created by the Java Virtual Machine (JVM) - by prevent direct instantiation we can ensure that developers don't create a second copy. You can achieve this by having the private constructor in the class and having a getter method which returns an object of the class and creates one for the first time if it’s null.

**Q) What is method overloading and method overriding?**  
 Method overloading: When 2 or more methods in a class have the same method names with different arguments, it is said to be method overloading. Overloading does not block inheritance from the superclass. Overloaded methods must have different method signatures  
  
Method overriding : When a method in a class has the same method name with same arguments as that of the superclass,it is said to be method overriding. Overriding blocks inheritance from the superclass. Overridden methods must have same signature.Basically overloading and overriding are different aspects of polymorphism.  
  
static/early binding polymorphism: overloading   
dynamic/late binding polymorphism: overriding

**Q) What restrictions are placed on method overriding?**  
 Restrictions placed on method overriding:  
\* Overridden methods must have the same name, argument list, and return type.   
\* The overriding method may not limit the access of the method it overrides. Methods may be overridden to be more public, not more private.   
\* The overriding method may not throw any exceptions that may not be thrown by the overridden method.

**Q) Why default constructor of base class will be called first in java?**  
 A subclass inherits all the methods and fields (eligible one) from the base class, so base class is constructed in the process of creation of subclass object (subclass is also an object of superclass). Hence before initializing the default value of sub class the super class should be initialized using the default constructor.

**Q) What is the difference between instance, object, reference and a class?**  
 Class: A class is a user defined data type with set of data members & member functions  
Object: An Object is an instance of a class  
Reference: A reference is just like a pointer pointing to an object  
Instance: This represents the values of data members of a class at a particular time

**Q) How could Java classes direct program messages to the system console, but error messages, say to a file?**  
 The class System has a variable out that represents the standard output, and the variable err that represent the standard error device. By default, they both point at the system console. This how the standard output could be re-directed:   
Stream st = new Stream (new FileOutputStream ("techinterviews\_com.txt"));  
System.setErr(st);  
System.setOut(st);

**Q) How do you know if an explicit object casting is needed?**  
 If you assign a superclass object to a variable of a subclass's data type, you need to do explicit casting. For example:  
Object a; Customer b; b = (Customer) a;  
When you assign a subclass to a variable having a superclass type, the casting is performed automatically

**Q) For concatenation of strings, which method is good, StringBuffer or String ?**  
 StringBuffer is faster than String for concatenation.

**Q) What is the difference between a break statement and a continue statement?**   
 A break statement results in the termination of the statement to which it applies (switch, for, do, or while). A continue statement is used to end the current loop iteration and return control to the loop statement.

**Q) What is the difference between the prefix and postfix forms of the ++ operator?**  
 The prefix form performs the increment operation and returns the value of the increment operation. The postfix form returns the current value all of the expression and then performs the increment operation on that value.

**Q) What is the difference between an “if” statement and a “switch” statement?**  
 The “if” statement is used to select among two alternatives. It uses a boolean expression to decide which alternative should be executed. The switch statement is used to select among multiple alternatives. It uses an int expression to determine which alternative should be executed.

**Q) Explain the principles of object oriented programming**   
 1. Abstraction   
It means hiding the details and only exposing the essential parts   
2. Polymorphism   
Polymorphism means having many forms. In java you can see polymorphism when you have multiple methods with the same name  
3. Inheritance   
Inheritance means the child class inherits the non private properties of the parent class   
4. Encapsulation   
It means data hiding. In java with encapsulate the data by making it private and even we want some other class to work on that data then the setter and getter methods are provided.

**Q) Difference between procedural and object oriented language**  
 In procedural programming the instructions are executed one after another and the data is exposed to the whole program. In OOPs programming the unit of program is an object which is nothing but combination of data and code and the data is not exposed outside the object

**Q) What is the difference between parameters and arguments**  
 While defining method, variables passed in the method are called parameters. While using those methods, values passed to those variables are called arguments.

**Q) What are null or Marker interfaces in Java**   
 The null interfaces are marker interfaces, they do not have function declarations in them, they are empty interfaces, this is to convey the compiler that they have to be treated differently

**Q) Why we should not have instance variable in an interface**  
 Since all data fields and methods in an Interface are public by default, then we implement that interface in our class then we have public members in our class and this class will expose these data members and this is violation of encapsulation as now the data is not secure

**Q) What is the difference between while and do while loop**  
 Do while loop always executes the body of the loop at least once, since the test is performed at the end of the body.

**Q) What is JVM?**  
 When we install a java package. It contains 2 things   
\* The Java Runtime Environment (JRE)   
\* The Java Development Kit (JDK)   
The JRE provides runtime support for Java applications. The JDK provides the Java compiler and other development tools. The JDK includes the JRE. Both the JRE and the JDK include a Java Virtual Machine (JVM). This is the application that executes a Java program. A Java program requires a JVM to run on a particular platform

**Q) Does JAVA has goto?**  
 No, Java does not have Goto statements because  
1.Goto statements form infinite loops which are drawbacks in a program.  
2. Goto statements decrease the reliability of a program.  
3. Goto statements make documentation of a program very difficult. Documentation means preserving a copy of a program for future use.

**Q) Disadvantages of Java**  
 Development time is much longer than using scripting languages.  
It's a bit slower than C.  
String manipulation in Perl takes much less lines of code.

**Q) How to use C++ code in Java Program?**  
 By using a native method. A native method can specify checked exceptions in a throws clause. JNI API allows java methods to invoke native functions implemented in C.  
class Native{  
/\*  
\* Loading the native library in a static block ensures that it's loaded  
\* before the native method call.  
\*/  
static{System.loadLibrary("LibraryOfNativeMethods");}  
native void nativeMethod();  
}  
  
class Client{  
public static void main(String[] args){  
Native nativeOne = new Native();  
nativeONe.nativeMethod();  
}  
}

**Q) What is composition?**  
 Holding the reference of the other class within some other class is known as composition.

**Q) What is aggregation?**  
 It is a special type of composition. If you expose all the methods of a composite class and route the method call to the composite method through its reference, then it is called aggregation.

**Q) What is a lightweight component?**  
 Lightweight components are the one which doesn't go with the native call to obtain the graphical units. They share their parent component graphical units to render them. Example, Swing components

**Q) What is a heavyweight component?**  
 For every paint call, there will be a native call to get the graphical units. Example, AWT.

**Q) What is constructor chaining and how is it achieved in Java?**   
 A child object constructor always first needs to construct its parent (which in turn calls its parent constructor.). In Java it is done via an implicit call to the no-args constructor as the first statement.

**Q) What is an abstract class?**  
 Abstract class must be extended/subclassed (to be useful). It serves as a template. A class that is abstract may not be instantiated (ie, you may not call its constructor), abstract class may contain static data. Any class with an abstract method is automatically abstract itself, and must be declared as such.  
A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.

**Q) What modifiers are allowed for methods in an Interface?**  
 Only public and abstract modifiers are allowed for methods in interfaces.

**Q) What is the difference between Abstract class and Interface**  
 1.An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation.  
2. Abstract class definition begins with the keyword "abstract" keyword followed by Class definition. An Interface definition begins with the keyword "interface".  
3. Abstract classes are useful in a situation when some general methods should be implemented and specialization behavior should be implemented by subclasses. Interfaces are useful in a situation when all its properties need to be implemented by subclasses  
4. All variables in an Interface are by default - **public static final** while an abstract class can have instance variables.  
5. An interface is also used in situations when a class needs to extend another class apart from the abstract class. In such situations it’s not possible to have multiple inheritance of classes. An interface on the other hand can be used when it is required to implement one or more interfaces. Abstract class does not support Multiple Inheritance whereas an Interface supports multiple Inheritance.  
6. An Interface can only have public members whereas an abstract class can contain private as well as protected members.  
7. A class implementing an interface must implement all of the methods defined in the interface, while a class extending an abstract class need not implement any of the methods defined in the abstract class.  
8. The problem with an interface is, if you want to add a new feature (method) in its contract, then you MUST implement those methods in all of the classes which implement that interface. However, in the case of an abstract class, the method can be simply implemented in the abstract class and the same can be called by its subclass  
9. Interfaces are slow as it requires extra indirection to find corresponding method in the actual class. Abstract classes are fast  
10.Interfaces are often used to describe the peripheral abilities of a class, and not its central identity, E.g. an Automobile class might implement the Recyclable interface, which could apply to many otherwise totally unrelated objects.  
Note: There is no difference between a fully abstract class (all methods declared as abstract and all fields are public static final) and an interface.  
Note: If the various objects are all of-a-kind, and share a common state and behavior, then tend towards a common base class. If all they share is a set of method signatures, then tend towards an interface.  
Similarities:   
Neither Abstract classes nor Interface can be instantiated.

**Q) Can you make an instance of an abstract class?**  
 Abstract classes can contain abstract and concrete methods. Abstract classes cannot be instantiated directly i.e. we cannot call the constructor of an abstract class directly nor we can create an instance of an abstract class by using "Class.forName().newInstance()" (Here we get java.lang.InstantiationException). However, if we create an instance of a class that extends an Abstract class, compiler will initialize both the classes. Here compiler will implicitly call the constructor of the Abstract class. Any class that contain an abstract method must be declared "abstract" and abstract methods can have definitions only in child classes. By overriding and customizing the abstract methods in more than one subclass makes "Polymorphism" and through Inheritance we define body to the abstract methods. Basically an abstract class serves as a template. Abstract class must be extended/subclassed for it to be implemented. A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated. Abstract class is a class that provides some general functionality but leaves specific implementation to its inheriting classes.   
Example of Abstract class:  
  
abstract class AbstractClassExample{   
protected String name;   
public String getname() {   
return name;  
}   
public abstract void function();  
}  
  
Example: Vehicle is an abstract class and Bus Truck, car etc are specific implementations No! You cannot make an instance of an abstract class. An abstract class has to be sub-classed.If you have an abstract class and you want to use a method which has been implemented, you may need to subclass that abstract class, instantiate your subclass and then call that method.

**Q) Can you make an instance of an abstract class? For example - java.util.Calender is an abstract class with a method getInstance() which returns an instance of the Calender class.**   
 No! You cannot make an instance of an abstract class. An abstract class has to be sub-classed. If you have an abstract class and you want to use a method which has been implemented, you may need to subclass that abstract class, instantiate your subclass and then call that method.

**Q) How to define an Interface?**  
 In Java Interface defines the methods but does not implement them. Interface can include constants.A class that implements the interfaces is bound to implement all the methods defined in Interface.  
Example of Interface:  
  
public interface SampleInterface {  
public void functionOne();  
public long CONSTANT\_ONE = 1000;   
}

**Q) Can Abstract Class have constructors? Can interfaces have constructors?**  
 Abstract classes can have a constructor, but you cannot access it through the object, since you cannot instantiate abstract class. To access the constructor create a sub class and extend the abstract class which is having the constructor.  
  
Example  
public abstract class AbstractExample {  
public AbstractExample(){  
System.out.println("In AbstractExample()");  
}  
}  
  
public class Test extends AbstractExample{  
public static void main(String args[]){  
Test obj=new Test();  
}  
}

**Q) If interface & abstract class have same methods and those methods contain no implementation, which one would you prefer?**  
 Obviously one should ideally go for an interface, as we can only extend one class. Implementing an interface for a class is very much effective rather than extending an abstract class because we can extend some other useful class for this subclass

**Q) How to define an Abstract class?**  
 A class containing abstract method is called Abstract class. An Abstract class can't be instantiated.   
Example of Abstract class:  
abstract class testAbstractClass {   
protected String myString;   
public String getMyString() {   
return myString;   
}   
public abstract string anyAbstractFunction();  
}

**Q) Explain modifier final**  
 Final can be applied to classes, methods and variables.

* Variables values cannot be changed.
* Final class cannot be sub classed.
* Methods cannot be overridden

**Q) Can an abstract class be final?**  
 An abstract class may not be declared as final.

**Q) When does the compiler insist that the class must be abstract ?**

If one or more methods of the class are abstract.   
If class inherits one or more abstract methods from the parent abstract class and no implementation is provided for that method.  
If class implements an interface and provides no implementation for those methods

**Q) Where can static modifiers be used ?**  
 They can be applied to variables, methods and even a block of code, static methods and variables are not associated with any instance of class

**Q) What is interface and its use?**  
 Interface is similar to a class which may contain method's signature only but not bodies and it is a formal set of method and constant declarations that must be defined by the class that implements it. Interfaces are useful for:

* Declaring methods that one or more classes are expected to implement.
* Capturing similarities between unrelated classes without forcing a class relationship.
* Determining an object's programming interface without revealing the actual body of the class.

**Q) What is static in java?**  
 Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class. Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object. A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

**Q) How can you reference static variables?**  
 Via reference to any instance of the class   
Computer comp = new Computer ();   
comp.harddisk where hardisk is a static variable   
comp.compute() where compute is a method   
Via the class name   
Computer.harddisk   
Computer.compute()

**Q) Can static method use non static features of their class ?**  
 No they are not allowed to use non static features of the class, they can only call static methods and can use static data

**Q) What is static initializer code ?**  
 A class can have a block of initializer code that is simply surrounded by curly braces and labeled as static e.g. public class Demo{   
static int =10;   
static{   
System.out.println("Hello world');   
}   
}   
And this code is executed exactly once at the time of class load Where is native modifier used It can refer only to methods and it indicates that the body of the method is to be found else here and it is usually written in non-java language

**Q) What is the difference between static and non-static variables?**  
 A static variable is associated with the class as a whole rather than with specific instances of a class. Each object will share a common copy of the static variables i.e. there is only one copy per class, no matter how many objects are created from it. Class variables or static variables are declared with the static keyword in a class. These are declared outside a class and stored in static memory. Class variables are mostly used for constants. Static variables are always called by the class name. This variable is created when the program starts and gets destroyed when the programs stops. The scope of the class variable is same an instance variable. Its initial value is same as instance variable and gets a default value when it’s not initialized corresponding to the data type. Similarly, a static method is a method that belongs to the class rather than any object of the class and doesn't apply to an object or even require that any objects of the class have been instantiated. Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object. A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a non-static method. In other words, you can't change a static method into an instance method in a subclass. Non-static variables take on unique values with each object instance

**Q) Can a method be static and synchronized?**   
 A static method can be synchronized. If you do so, the JVM will obtain a lock on the java.lang.Class instance associated with the object. It is similar to saying:   
synchronized(XYZ.class) {   
}

**Q) What are the Final fields & Final Methods ?**  
 Fields and methods can also be declared final. A final method cannot be overridden in a subclass. A final field is like a constant: once it has been given a value, it cannot be assigned to again.  
Code : private static final int MAXATTEMPTS = 10;

**Q) Give example of a final class**  
 Math class is final class and hence cannot be extended.

**Q) What is thread? What are the high-level thread states?**  
 A thread is an independent path of execution in a system. The high-level thread states are ready, running, waiting and dead.

**Q) When a thread is created and started, what is its initial state?**   
 A thread is in the ready state after it has been created and started.

**Q) Explain different way of using thread?**   
 The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, cause when you are going for multiple inheritance, the only interface can help.

**Q) What is synchronization and why is it important?**   
 With respect to multithreading, Synchronization is a process of controlling the access of shared resources by the multiple threads in such a manner that only one thread can access a particular resource at a time. In non-synchronized multithreaded application, it is possible for one thread to modify a shared object while another thread is in the process of using or updating the object's value. Synchronization prevents such type of data corruption which may otherwise lead to dirty reads and significant errors.   
E.g. synchronizing a function:  
public synchronized void Method1 () {  
// method code.   
}  
E.g. synchronizing a block of code inside a function:  
public Method2 (){  
synchronized (this) {   
// synchronized code here.  
}  
}

**Q) What are synchronized methods and synchronized statements?**  
 Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

**Q) What is daemon thread and which method is used to create the daemon thread?**  
 Daemon threads are threads with low priority and runs in the back ground doing the garbage collection operation for the java runtime system. The setDaemon() method is used to create a daemon thread. These threads run without the intervention of the user. To determine if a thread is a daemon thread, use the accessor method isDaemon() When a standalone application is run then as long as any user threads are active the JVM cannot terminate, otherwise the JVM terminates along with any daemon threads which might be active. Thus a daemon thread is at the mercy of the runtime system. Daemon threads exist only to serve user threads.

**Q) What method must be implemented by all threads?**  
 All tasks must implement the run() method, whether they are a subclass of Thread or implement the Runnable interface.

**Q) What are different ways in which a thread can enter the waiting state?**  
 A thread can enter the waiting state by the following ways:   
1. Invoking its sleep() method,  
2. By blocking on I/O  
3. By unsuccessfully attempting to acquire an object's lock  
4. By invoking an object's wait() method.   
5. It can also enter the waiting state by invoking its (deprecated) suspend() method.

**Q) What is the difference between yielding and sleeping?**  
 When a task invokes its yield() method, it returns to the ready state, either from waiting, running or after its creation. When a task invokes its sleep() method, it returns to the waiting state from a running state.

**Q) What is mutual exclusion? How can you take care of mutual exclusion using Java threads?**  
 Mutual exclusion is a phenomenon where no two processes can access critical regions of memory at the same time. Using Java multithreading we can arrive at mutual exclusion. For mutual exclusion, you can simply use the synchronized keyword and explicitly or implicitly provide an Object, any Object, to synchronize on. The synchronized keyword can be applied to a class, to a method, or to a block of code. There are several methods in Java used for communicating mutually exclusive threads such as wait( ), notify( ), or notifyAll( ). For example, the notifyAll( ) method wakes up all threads that are in the wait list of an object.

**Q) How to create multithreaded program? Explain different ways of using thread? When a thread is created and started, what is its initial state?**  
 You have two ways to do so. First, making your class "extends" Thread class. The other way is making your class implement "Runnable" interface. The latter is more advantageous, cause when you are going for multiple inheritance, then only interface can help. . If you are already inheriting a different class, then you have to go for Runnable Interface. Otherwise you can extend Thread class. Also, if you are implementing interface, it means you have to implement all methods in the interface. Both Thread class and Runnable interface are provided for convenience and use them as per the requirement. But if you are not extending any class, better extend Thread class as it will save few lines of coding. Otherwise performance wise, there is no distinguishable difference. A thread is in the ready state after it has been created and started.

**Q) What is the difference between preemptive scheduling and time slicing?**   
 Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then re-enters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

**Q) What invokes a thread's run() method?**  
 After a thread is started, via its start() method of the Thread class, the JVM invokes the thread's run() method when the thread is initially executed.

**Q) What is the purpose of the wait(), notify(), and notifyAll() methods?**  
 The wait(),notify(), and notifyAll() methods are used to provide an efficient way for threads to wait for a shared resource. When a thread executes an object's wait() method, it enters the waiting state. It only enters the ready state after another thread invokes the object's notify() or notifyAll() methods.

**Q) What is deadlock?**   
 When two threads are waiting for each other and can't proceed until the first thread obtains a lock on the other thread or vice versa, the program is said to be in a deadlock.

**Q) How does multithreading take place on a computer with a single CPU?**   
 The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.

**Q) Can Java object be locked down for exclusive use by a given thread?**  
 Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it. If a thread attempts to execute a synchronized method or synchronized statement and is unable to acquire an object's lock, it enters the waiting state until the lock becomes available.

**Q) What's the difference between the methods sleep() and wait()?**   
 **sleep()** is a method which is used to pause the process for few seconds or the time we want to. But in case of **wait()** method, thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases the lock or monitor while waiting. wait() is used for inter-thread communication while sleep() is used to introduce pause on execution, generally.

**Q) What is an object's lock and which objects have locks?**  
 An object's lock is a mechanism that is used by multiple threads to obtain synchronized access to the object. A thread may execute a synchronized method of an object only after it has acquired the object's lock. All objects and classes have locks. A class's lock is acquired on the class's Class object.

**Q) Can a lock be acquired on a class?**  
 Yes, a lock can be acquired on a class. This lock is acquired on the class's Class object.

**Q) What is the difference between process and thread?**  
**Threads** are used for small tasks, whereas **processes** are used for more 'heavyweight' tasks – basically the execution of applications. Another **difference between** a **thread** and a **process** is that **threads** within the same **process** share the same address space, whereas different **processes** do not.

**Q) When you will synchronize a piece of your code?**  
 When you expect that your shared code will be accessed by different threads and these threads may change a particular data causing data corruption, then they are placed in a synchronized construct or a synchronized method.

**Q) How would you implement a thread pool?**

1. Create a task(Runnable Object) to execute

2. Create Executor Pool using ExecutorService

3. Pass tasks to Executor Pool

4. Shutdown the Executor Pool

**Q) Is there a separate stack for each thread in Java?**  
 Yes. Every thread maintains its own separate stack, called Runtime Stack but they share the same memory. Elements of the stack are the method invocations, called activation records or stack frame. The activation record contains pertinent information about a method like local variables.

**Q) What is Runnable interface? Are there any other ways to make a java program as multithread java program?**  
 There are two ways to create new kinds of threads:  
- Define a new class that extends the Thread class  
- Define a new class that implements the Runnable interface, and pass an object of that class to a Thread's constructor.   
- An advantage of the second approach is that the new class can be a subclass of any class, not just of the Thread class.  
The Runnable interface has only one method: public void run();Thus, every class (thread) implements the Runnable interface, has to provide logic for run() method

**Q) How can I tell what state a thread is in ?**   
 Prior to Java 5, isAlive() was commonly used to test a threads state. If isAlive() returned false the thread was either new or terminated but there was simply no way to differentiate between the two.   
Starting with the release of Tiger (Java 5) you can now get what state a thread is in by using the getState() method which returns an Enum of Thread.States. A thread can only be in one of the following states at a given point in time.  
  
NEW   
A Fresh thread that has not yet started to execute.  
RUNNABLE  
A thread that is executing in the Java virtual machine.  
BLOCKED  
A thread that is blocked waiting for a monitor lock.  
WAITING  
A thread that is wating to be notified by another thread.  
TIMED\_WAITING  
A thread that is wating to be notified by another thread for a specific amount of time  
TERMINATED  
A thread whos run method has ended.

The following code prints out all thread states.   
**public** **class** ThreadStates{

**public** **static** **void** main(String[] args){

Thread t = **new** Thread();

Thread.State e = t.getState();

Thread.State[] ts = e.*values*();

**for**(**int** i = 0; i < ts.length; i++){

System.***out***.println(ts[i]);

}

}

}

**Q) What is the purpose of finalization?**  
 The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup, before the object gets garbage collected. For example closing an opened database Connection.

**Q) What is Thread priority?**

Every thread has a priority, the higher priority thread gets preference over the lower priority thread by the thread scheduler

Priority- 1-10, default priority = 5

**Q) What is the difference between final, finally and finalize?**  
 final - declare constant  
finally - handles exception  
finalize - helps in garbage collection  
Variables defined in an interface are implicitly final. A final class can't be extended i.e., final class may not be sub classed. This is done for security reasons with basic classes like String and Integer. It also allows the compiler to make some optimizations, and makes thread safety a little easier to achieve. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant). finalize() method is used just before an object is destroyed and garbage collected. Finally, a key word used in exception handling and will be executed whether or not an exception is thrown. For example, closing of open connections is done in the finally method.

**Q) What methods java providing for Thread communications ?**  
 Java provides three methods that threads can use to communicate with each other: wait, notify, and notifyAll. These methods are defined for all Objects (not just Threads). The idea is that a method called by a thread may need to wait for some condition to be satisfied by another thread; in that case, it can call the wait method, which causes its thread to wait until another thread calls notify or notifyAll.

**Q) What is the difference between notify and notify All methods ?**  
 A call to notify causes at most one thread waiting on the same object to be notified (i.e., the object that calls notify must be the same as the object that called wait). A call to notifyAll causes all threads waiting on the same object to be notified. If more than one thread is waiting on that object, there is no way to control which of them is notified by a call to notify (so it is often better to use notifyAll than notify).

**Q) What happens when a thread cannot acquire a lock on an object?**  
 If a thread attempts to execute a synchronized method or synchronized statement and is unable to acquire an object's lock, it enters the waiting state until the lock becomes available.

**Q) What happens when you invoke a thread's interrupt method while it is sleeping or waiting?**  
 When a task's interrupt() method is executed, the task enters the ready state. The next time the task enters the running state, an InterruptedException is thrown.

**Q) What are three ways in which a thread can enter the waiting state?**   
 A thread can enter the waiting state by invoking its sleep() method, by blocking on I/O, by unsuccessfully attempting to acquire an object's lock, or by invoking an object's wait() method. It can also enter the waiting state by invoking its (deprecated) suspend() method.

**Q) What happens when you call Thread.yield()**   
 It caused the currently executing thread to move to the ready state if the scheduler is willing to run any other thread in place of the yielding thread. Yield is a static method of class Thread

**Q) What is the advantage of yielding?**  
 It allows a time consuming thread to permit other threads to execute

**Q) What happens when you call Thread.sleep()**   
 It passes time without doing anything and without using the CPU. A call to sleep method requests the currently executing thread to cease executing for a specified amount of time.

**Q) Does the thread method start executing as soon as the sleep time is over**   
 No, after the specified time is over the thread enters into ready state and will only execute when the scheduler allows it to do so.

**Q) What do you mean by thread blocking ?**  
 If a method needs to wait an indeterminable amount of time until some I/O occurrence takes place, then a thread executing that method should graciously step out of the Running state. All java I/O methods behave this way. A thread that has graciously stepped out in this way is said to be blocked.

**Q) What is the purpose of garbage collection in Java, and when is it used?**  
 The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

**Q) Does garbage collection guarantee that a program will not run out of memory?**  
 Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection.

**Q) What is the purpose of the Runtime class?**  
 The purpose of the Runtime class is to provide access to the Java runtime system.  
It returns the runtime information like memory availability.  
\* Runtime.getRuntime().freeMemory() --> Returns JVM Free Memory  
\* Runtime.getRuntime().maxMemory() --> Returns the maximum amount of memory that the JVM will attempt to use. It also helps to run the garbage collector  
\* Runtime.getRuntime().gc().

**Q) What is Garbage Collection and how to call it explicitly?**  
 Garbage collection is one of the most important features of Java. The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used. Garbage collection is also called automatic memory management as JVM automatically removes the unused variables/objects (value is null) from the memory. Every class inherits finalize() method from java.lang.Object, the finalize() method is called by garbage collector when it determines no more references to the object exists. In Java, it is good idea to explicitly assign null into a variable when no more in use. In Java on calling System.gc() and Runtime.gc(), JVM tries to recycle the unused objects, but there is no guarantee when all the objects will garbage collected. Garbage collection is an automatic process and can't be forced. There is no guarantee that Garbage collection will start immediately upon request of System.gc().

**Q) What kind of thread is the Garbage collector thread?**  
 It is a daemon thread.

**Q) Difference between HashMap and HashTable?**  
 The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesnt allow). HashMap does not guarantee that the order of the map will remain constant over time. HashMap is unsynchronized and Hashtable is synchronized.

**Q) How can we access elements of a collection?**   
 We can access the elements of a collection using the following ways:  
1.Every collection object has get(index) method to get the element of the object. This method will return Object.  
2.Collection provide Enumeration or Iterator object so that we can get the objects of a collection one by one.

**Q) What is the Properties class?**  
 The properties class is a subclass of Hashtable that can be read from or written to a stream. It also provides the capability to specify a set of default values to be used.

**Q) What is the difference between the size and capacity of a Vector?**   
 The size is the number of elements actually stored in the vector, while capacity is the maximum number of elements it can store at a given instance of time.

**Q) Can a vector contain heterogenous objects?**  
 Yes a Vector can contain heterogenous objects. Because a Vector stores everything in terms 0f Object.

**Q) What is an enumeration?**  
 An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection.

**Q) Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list ie Vector, ArrayList, LinkedList ?**   
 ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

**Q) How can we use hashset in collection interface?**   
 This class implements the set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the Null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.

**Q) What are different types of collections**  
 A collection has no special order and does not reject duplicates   
A list is ordered and does not reject duplicates   
A set has no special order but rejects duplicates   
A map supports searching on a key field, values of which must be unique.

**Q) What are Checked and Unchecked Exceptions?**  
 A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Making an exception checked forces client programmers to deal with the exception may be thrown. Checked exceptions must be caught at compile time. Example: IOException. Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. Example: ArrayIndexOutOfBoundsException, ArithmaticException . Errors are often irrecoverable conditions.

**Q) What is the difference between error and an exception?**  
 An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you cannot repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.)

**Q) How to create custom exceptions?**  
 Your class should extend class Exception, or some more specific type thereof.

**Q) What are the different ways to handle exceptions?**  
 There are two ways to handle exceptions,   
1. By wrapping the desired code in a try block followed by a catch block to catch the exceptions.   
2. List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

**Q) What is the basic difference between the 2 approaches to exception handling?**   
1) try catch block and   
2) specifying the candidate exceptions in the throws clause  
When should you use which approach?  
In the first approach as a programmer of the method, you yourself are dealing with the exception. This is fine if you are in a best position to decide should be done in case of an exception. Whereas if it is not the responsibility of the method to deal with its own exceptions, then do not use this approach. In this case use the second approach. In the second approach we are forcing the caller of the method to catch the exceptions that the method is likely to throw. This is often the approach library creator’s use. They list the exception in the throws clause and we must catch them. You will find the same approach throughout the java libraries we use.

**Q) Is it necessary that each try block must be followed by a catch block?**   
 It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

**Q) If I write return at the end of the try block, will the finally block still execute?**  
 Yes even if you write return as the last statement in the try block and no exception occurs, the finally block will execute. The finally block will execute and then the control return.

**Q) How does a try statement determine which catch clause should be used to handle an exception?**  
 When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exceptionis executed. The remaining catch clauses are ignored.

**Q) What is the difference between throw and throws keywords?**   
 The throw keyword denotes a statement that causes an exception to be initiated. It takes the Exception object to be thrown as an argument. The exception will be caught by an enclosing try-catch block or propagated further up the calling hierarchy. The throws keyword is a modifier of a method that denotes that an exception may be thrown by the method. An exception can be rethrown.

**Q) What is the purpose of the finally clause of a try-catch-finally statement?**  
 The finally clause is used to provide the capability to execute code no matter whether or not an exception is thrown or caught. If an exception occurs and there is a return statement in catch block, the finally block is still executed. The finally block will not be executed when the System.exit(0) statement is executed earlier or on system shut down earlier or the memory is used up earlier before the thread goes to finally block.  
try{  
//some statements  
}  
catch{  
//statements when exception is caught  
}  
finally{  
//statements executed whether exception occurs or not  
}

**Q) Does the order of placing catch statements matter in the catch block?**  
 Yes, it does. The FileNoFoundException is inherited from the IOException. So FileNoFoundException is caught before IOException. Exception's subclasses have to be caught first before the General Exception

**Q) How will you handle the checked exceptions**  
 You can provide a try/catch block to handle it. OR Make sure method declaration includes a throws clause that informs the calling method an exception might be thrown from this particular method. When you extend a class and override a method, can this new method throw exceptions other than those that were declared by the original method? No it cannot throw, except for the subclasses of those exceptions.

**Q) What is serialization?**   
 The serialization is a kind of mechanism that makes a class or a bean persistent by having its properties or fields and state information saved and restored to and from storage. That is, it is a mechanism with which you can save the state of an object by converting it to a byte stream, that stream can be file or byte array or stream associated with TCP/IP socket.

**Q) How do I serialize an object to a file?**  
 An object must implement the Serializable or Externalizable interface before it can be written to a stream as an object. The class whose instances are to be serialized should implement an interface Serializable. Then you pass the instance to the ObjectOutputStream which is connected to a fileoutputstream. This will save the object to a file.  
To serialize an object into a stream perform the following actions:   
  
- Open one of the output streams, for exaample FileOutputStream   
- Chain it with the ObjectOutputStream - Call the method writeObject() providingg the instance of a Serializable object as an argument.  
- Close the streams   
For Ex:  
try{  
fOut= new FileOutputStream("c:\\emp.ser");  
out = new ObjectOutputStream(fOut);  
out.writeObject(employee); //serializing   
System.out.println("An employee is serialized into c:\\emp.ser");  
} catch(IOException e){  
e.printStackTrace();   
}

**Q) What is the common usage of serialization?**  
 Whenever an object is to be sent over the network, objects need to be serialized. Moreover if the state of an object is to be saved, objects need to be serialized.

**Q) What is Externalizable Interface?**  
 Externalizable interface is a subclass of Serializable. Java provides Externalizable interface that gives you more control over what is being serialized and it can produce smaller object footprint. ( You can serialize whatever field values you want to serialize)  
This interface defines 2 methods: readExternal(ObjectInput in) and writeExternal(ObjectOuput out) and you have to implement these methods in the class that will be serialized. In these methods you'll have to write code that reads/writes only the values of the attributes you are interested in. Programs that perform serialization and deserialization have to write and read these attributes in the same sequence..

**Q) What happens to the static fields of a class during serialization?**  
There are three exceptions in which serialization doesn't necessarily read and write to the stream. These are  
1. Serialization ignores static fields, because they are not part of any particular state.  
2. Base class fields are only handled if the base class itself is serializable.  
3. Transient fields.

**Q) What are the rules of serialization**  
 1. Static fields are not serialized because they are not part of any one particular object   
2. Fields from the base class are handled only if hose are serializable   
3. Transient fields are not serialized

**Q) What is Transient Variable?**   
 A transient variable is a variable that may not be serialized i.e. the value of the variable can't be written to the stream in a Serializable class. If you don't want some field to be serialized, you can mark that field transient or static. In such a case when the class is retrieved from the ObjectStream the value of the variable is null.

**Q) What one should take care of while serializing the object?**  
 One should make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

**Q) What is the difference between Serializable and Externalizable interface? How can you control over the serialization process i.e. how can you customize the seralization process?**   
 When you use Serializable interface, your class is serialized automatically by default. But you can override writeObject() and readObject() two methods to control more complex object serailization process. When you use Externalizable interface, you have a complete control over your class's serialization process. This interface contains two methods namely readExternal and writeExternal. You should implement these methods and write the logic for customizing the serialization process.

**Q) What are the uses of Serialization?**   
 In some types of applications you have to write the code to serialize objects, but in many cases serialization is performed behind the scenes by various server-side containers.  
These are some of the typical uses of serialization:  
\*To persist data for future use.  
\*To send data to a remote computer using such client/server Java technologies as RMI or socket programming.  
\*To "flatten" an object into array of bytes in memory.  
\*To exchange data between applets and servlets.  
\*To store user session in Web applications.   
\*To activate/passivate enterprise java beans.  
\*To send objects between the servers in a cluster.

**Q) What is Volatile Variable ?**  
 Volatile modifier applies to variables only and it tells the compiler that the variable modified by volatile can be changed unexpectedly by other parts of the program.

**Q) How are Observer and Observable used?**  
 Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.

**Q) Explain the usage of Java packages.**  
 A Java package is a naming context for classes and interfaces. A package is used to create a separate name space for groups of classes and interfaces. Packages are also used to organize related classes and interfaces into a single API unit and to control accessibility to these classes and interfaces. For example: The Java API is grouped into libraries of related classes and interfaces; these libraries are known as package.

**Q) If a class is declared without any access modifiers, where may the class be accessed?**  
 A class that is declared without any access modifiers is said to have package or default access. This means that the class can only be accessed by other classes and interfaces that are defined within the same package.

**Q) If a class is located in a package, what do you need to change in the OS environment to be able to use it?**   
 You need to add a directory or a jar file that contains the package directories to the CLASSPATH environment variable. Let's say a class Employee belongs to a package com.xyz.hr; and is located in the file c:\dev\com\xyz\hr\Employee.java. In this case, you'd need to add c:\dev to the variable CLASSPATH. If this class contains the method main(), you could test it from a command prompt window as follows:  
c:\>java com.xyz.hr.Employee

**Q) State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.**  
 public : Public class is visible in other packages, field is visible everywhere (class must be public too)  
private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.  
protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.  
default :What you get by default ie, without any access modifier (ie, public private or protected).It means that it is visible to all within a particular package.

**Q) Can a top level class be private or protected?**   
 No. A top level class cannot be private or protected. It can have either "public" or no modifier. If it does not have a modifier it is supposed to have a default access. If a top level class is declared as private the compiler will complain that the "modifier private is not allowed here". This means that a top level class cannot be private. Same is the case with protected.

**Q) What are different types of inner classes ?**  
 Inner classes nest within other classes. A normal class is a direct member of a package. Inner classes, which became available with Java 1.1, are four types   
\* Static member classes   
\* Member classes   
\* Local classes   
\* Anonymous classes   
  
Static member classes - a static member class is a static member of a class. Like any other static method, a static member class has access to all static methods of the parent, or top-level, class.   
Member Classes - a member class is also defined as a member of a class. Unlike the static variety, the member class is instance specific and has access to any and all methods and members, even the parent's this reference.   
Local Classes - Local Classes declared within a block of code and these classes are visible only within the block.   
Anonymous Classes - These type of classes does not have any name and its like a local class.

**Q) Can inner classes be static**  
 Yes inner classes can be static, but they cannot access the non static data of the outer classes, though they can access the static data

**Q) What are the rules of anonymous class**   
 The class is instantiated and declared in the same place The declaration and instantiation takes the form new Xxxx () {// body} Where Xxxx is an interface name. An anonymous class cannot have a constructor. Since you do not specify a name for the class, you cannot use that name to specify a constructor

**Q) What is a Java package and how is it used?**  
 A Java package is a naming context for classes and interfaces. A package is used to create a separate name space for groups of classes and interfaces. Packages are also used to organize related classes and interfaces into a single API unit and to control accessibility to these classes and interfaces.

**Q) Can a private method of a superclass be declared within a subclass?**   
 Yes. A private field or method or inner class belongs to its declared class and hides from its subclasses. There is no way for private stuff to have a runtime overloading or overriding (polymorphism) features.

**Q) If the method to be overridden has access type protected, can subclass have the access type as private**  
 No, it must have access type as protected or public, since an overriding method must not be less accessible than the method it overrides

**Q) What if the main method is declared as private?**   
 The program compiles properly but at runtime it will give "Main method not public." message.

**Q) Can we use this() and super() both in a constructor?**

Ans: NO

**Association**

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

http://javapapers.com/wp-content/uploads/2010/06/association.jpg

**Example:** A Student and a Faculty are having an association.

**Aggregation**

Aggregation is a special case of association. It is a directional association between objects. When an object ‘has-a’ another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a “Has-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/aggregation.jpg

**Composition**

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

http://javapapers.com/wp-content/uploads/2010/06/composition.jpg

**Example:** A class contains students. A student cannot exist without a class. There exists composition between class and students.

**Abstraction**

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

**Example:** A wire frame model of a car.

**Generalization**

Generalization uses a “is-a” relationship from a specialization to the generalization class. Common structure and behaviour are used from the specializtion to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a “Is-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/generalization.jpg

**Example:** Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

**Realization**

Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

http://javapapers.com/wp-content/uploads/2010/06/realization.jpg

**Example:** A particular model of a car ‘GTB Fiorano’ that implements the blueprint of a car realizes the abstraction.

**Dependency**

Change in structure or behaviour of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.

http://javapapers.com/wp-content/uploads/2010/06/dependency.jpg

**Example:** Relationship between shape and circle is dependency.

**Q) What is Volatile Variable ?**  
 Volatile modifier applies to variables only and it tells the compiler that the variable modified by volatile can be changed unexpectedly by other parts of the program.

## Java (JVM) Memory Model

[](https://cdn.journaldev.com/wp-content/uploads/2014/05/Java-Memory-Model.png)

As you can see in the above image, JVM memory is divided into separate parts. At broad level, JVM Heap memory is physically divided into two parts – **Young Generation** and **Old Generation**.

## Memory Management in Java – Young Generation

The young generation is the place where all the new objects are created. When the young generation is filled, garbage collection is performed. This garbage collection is called **Minor GC**. Young Generation is divided into three parts – **Eden Memory** and two **Survivor Memory** spaces.

Important Points about Young Generation Spaces:

* Most of the newly created objects are located in the Eden memory space.
* When Eden space is filled with objects, Minor GC is performed and all the survivor objects are moved to one of the survivor spaces.
* Minor GC also checks the survivor objects and move them to the other survivor space. So at a time, one of the survivor space is always empty.
* Objects that are survived after many cycles of GC, are moved to the Old generation memory space. Usually, it’s done by setting a threshold for the age of the young generation objects before they become eligible to promote to Old generation.