1) **What is immutable in java? How we create an immutable class?**
Immutable class is a class which once created, it's contents cannot be changed. Immutable objects are the objects whose state cannot be changed once constructed. e.g. String class

To create an immutable class following steps should be followed:

- Create a final class.
- Set the values of properties using constructor only.
- Make the properties of the class final and private
- Do not provide any setters for these properties.
- If the instance fields include references to mutable objects, don't allow those objects to be changed:
- Don't provide methods that modify the mutable objects.
- Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

```
public final class FinalPersonClass {
      private final String name;
      private final int age;

      public FinalPersonClass(final String name, final int age) {
       this.name = name;
       this.age = age;
      }
      public int getAge() {
        return age;
      }
      public String getName() {
       return name;
      }
    }
```

2) **Immutable objects are automatically thread-safe –true/false?**
Ans) True. Since the state of the immutable objects cannot be changed once they are created they are automatically synchronized/thread-safe.

3) Which classes in java are immutable?
All wrapper classes in java.lang are immutable – String, Integer, Boolean, Character, Byte, Short, Long, Float, Double, BigDecimal, BigInteger

4) What are the advantages of immutability?

- Immutable objects are automatically thread-safe, the overhead caused due to use of synchronization is avoided.
- Once created the state of the immutable object cannot be changed so there is no possibility of them getting into an inconsistent state.

- The references to the immutable objects can be easily shared or cached without having to copy or clone them as there state cannot be changed ever after construction.
- The best use of the immutable objects is as the keys of a map.

5) **Can abstract class extends any other class and IF YES then why?**

- Yes you can do it. And it is good practice if your child class adds more functionality. It allows moving toward specification. Your parent class becomes a more general class and child class a more specific one. And you can implement both as per your requirement.
- Public abstract class JspWriter extends writer{
- }

6) **Can abstract class implements any other interface and why?**

Yes, abstract class can implement interface by using implements keyword. Since they are abstract, they don't need to implement all methods. It's good practice to provide an abstract base class, along with an interface to declare Type.

7) Can an abstract class have a constructor?

You would define a constructor in an abstract class if you are in one of these situations:

- you want to perform some initialization (to fields of the abstract class) before the instantiation of a subclass actually takes place
- you have defined final fields in the abstract class but you did not initialize them in the declaration itself; in this case, you MUST have a constructor to initialize these fields

Note that:

- you may define more than one constructor (with different arguments)
- you can (should?) define all your constructors protected (making them public is pointless anyway)
- your subclass constructor(s) can call one constructor of the abstract class; it may even **have to**call it (if there is no no-arg constructor in the abstract class)

In any case, don't forget that if you don't define a constructor, then the compiler will automatically generate one for you (this one is public, has no argument, and does nothing).

8) Can abstract class be final in Java?

No, abstract class cannot be final in Java. Making them final will stop abstract class from being extended, which is the only way to use abstract class. They are also opposite of each other, abstract keyword enforces to extend a class, for using it, on the other hand, final keyword prevents a class from being extended. In real world also, abstract signifies incompleteness, while final is used to demonstrate completeness. Bottom line is, you can not make your class abstract and final in Java, at same time, it's a compile time error.

9) Can abstract class have static methods in Java?

Yes, abstract class can declare and define static methods, nothing prevents from doing that. But, you must follow guidelines for making a method static in Java, as it's not welcomed in a object oriented design, because static methods cannot be overridden in Java. It's very rare, you see static methods inside abstract class, but as I said, if you have very good reason of doing it, then nothing stops you.

10) Can you create instance of abstract class?

No, you cannot create instance of abstract class in Java, they are incomplete. Even though, if your abstract class don't contain any abstract method, you cannot create instance of it. By making a class abstract, you told compiler that, it's incomplete and should not be instantiated. Java compiler will throw error, when a code tries to instantiate abstract class.

11) When do you favor abstract class over interface?
Since it's almost impossible to add a new method on a published interface, it's better to use abstract class, when evolution is concern. Abstract class in Java evolves better than interface. Similarly, if you have too many methods inside interface, you are creating pain for all it's implementation; consider providing an abstract class for default implementation. This is the pattern followed in Java collection package, you can see `AbstractList` provides default implementation for `List` interface.

10) Can abstract class contains main method in Java ?
Yes, abstract class can contain main method, it just another static method and you can execute Abstract class with main method, until you don't create any instance.

11) Composition and inheritance difference?

- **Inheritance** brings out **IS-A** relation. **Composition** brings out **HAS-A relation**.
- Inheritance means inheriting something from a parent. For example, you may inherit your mother's eyes or inherit your father's build.
- Inheritance is a relationship between classes; containership is relationship between instances of classes.

Now composition is another thing. A Child can have a toy or a Parent can have a child. So I could do:

```
class Toy
{
  string model ;
};

class Child
{
   Toy transformersToy ;
};
```

So the Child has the transformers toy now.. but does Child inherit the transformersToy.model attribute? No, because it isn't inheriting.

12**) How service () method of Servlet class works fine in multi threaded environment?**
No. Servlets are not Thread safe.
The servlet is allows to access more than one threads at a time.
if u want to make it Servlet as Thread safe ., U can go for
Implement SingleThreadInterface(i) which is a blank Interface there is no methods
or we can go for synchronize methods
We can make whole service method as synchronized by using synchronized keword in front of method
Ex::

public Synchronized class service(ServletRequest request,ServletResponse response)throws ServletException,IOException
or we can put block of the code in the Synchronized block
Ex::

Synchronized(Object)

{

- ----Instructions-----
- }
- I feel that Synchronized block is better than making the whole method
- Synchronized

Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
1.  import javax.servlet.http.*;
2.  import javax.servlet.*;
3.  import java.io.*;
4.  public class DemoServlet extends HttpServlet{
5.  public void doGet(HttpServletRequest req,HttpServletResponse res)
6.  throws ServletException,IOException
7.  {
8.  res.setContentType("text/html");//setting the content type
9.  PrintWriter pw=res.getWriter();//get the stream to write the data
10.
11.  //writing html in the stream
12.  pw.println("<html><body>");
13.  pw.println("Welcome to servlet");
14.  pw.println("</body></html>");
15.
16.  pw.close();//closing the stream
17.  }}
```

13) **How we make collection immutable?**
The unmodifiableCollection() method is used to return an unmodifiable view of the specified collection.
And an attempt to modify the collection will result in an UnsupportedOperationException.

- Following is the declaration for java.util.Collections.unmodifiableCollection() method.
- public static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c)

Example

The following example shows the usage of java.util.Collections.unmodifiableCollection()

```
package com.tutorialspoint;

import java.util.*;

public class CollectionsDemo {
   public static void main(String[] args) {
      // create array list
      List<Character> list = new ArrayList<Character>();

      // populate the list
      list.add('X');
      list.add('Y');

      System.out.println("Initial list: "+ list);

      Collection<Character> immutablelist = Collections.unmodifiableCollection(list)

      // try to modify the list
      immutablelist.add('Z');
   }
}
```

Let us compile and run the above program, this will produce the following result.

```
Initial list: [X, Y]
Exception in thread "main" java.lang.UnsupportedOperationException
```

`static <T> Collection<T>`	**`unmodifiableCollection(Collection<? extends T> c)`** Returns an unmodifiable view of the specified collection.
`static <T> List<T>`	**`unmodifiableList(List<? extends T> list)`** Returns an unmodifiable view of the specified list.
`static <K,V> Map<K,V>`	**`unmodifiableMap(Map<? extends K,? extends V> m)`** Returns an unmodifiable view of the specified map.
`static <T> Set<T>`	**`unmodifiableSet(Set<? extends T> s)`** Returns an unmodifiable view of the specified set.
`static <K,V> SortedMap<K,V>`	**`unmodifiableSortedMap(SortedMap<K,? extends V> m)`** Returns an unmodifiable view of the specified sorted map.
`static <T> SortedSet<T>`	**`unmodifiableSortedSet(SortedSet<T> s)`** Returns an unmodifiable view of the specified sorted set.

14) How we make a object list as sorted?

15) Template design pattern implementation?

16) Difference between factory design pattern and abstract factory design pattern?

17) Can we write overridden method with checked exception for any method which is already defined unchecked exception in super class?

The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method.

```
class A {
  public void foo() throws IOException {..}
}

class B extends A {
  @Override
  public void foo() throws SocketException {..} // allowed

  @Override
  public void foo() throws SQLException {..} // NOT allowed
}
```

`SocketException extends IOException`, but `SQLException` does not.

The overriding method CAN throw any unchecked (runtime) exception, regardless of whether the overridden method declares the exception

```
class Super {
   public void test() {
      System.out.println("Super.test()");
   }
}

class Sub extends Super {
   @Override
   public void test() throws IndexOutOfBoundsException {
      // Method can throw any Unchecked Exception
      System.out.println("Sub.test()");
   }
}

class Sub2 extends Sub {
   @Override
   public void test() throws ArrayIndexOutOfBoundsException {
      // Any Unchecked Exception
      System.out.println("Sub2.test()");
   }
}

class Sub3 extends Sub2 {
   @Override
   public void test() {
      // Any Unchecked Exception or no exception
      System.out.println("Sub3.test()");
   }
}

class Sub4 extends Sub2 {
   @Override
   public void test() throws AssertionError {
      // Unchecked Exception IS-A RuntimeException or IS-A Error
      System.out.println("Sub4.test()");
   }
}
```

18) Class.forName() gives checked or unchecked exception.

The **java.lang.Class.forName(String className)** method returns the Class object associated with the class or interface with the given string name.

Throws checked exception:

```
package com.tutorialspoint;
```

```
import java.lang.*;

public class ClassDemo {

   public static void main(String[] args) {

     try {
        // returns the Class object for the class with the specified name
        Class cls = Class.forName("java.lang.ClassLoader");

        // returns the name and package of the class
        System.out.println("Class found = " + cls.getName());
        System.out.println("Package = " + cls.getPackage());
     }
     catch(ClassNotFoundException ex) {
        System.out.println(ex.toString());
     }
   }
}
```

19) **What is Serialization in Java?**
Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running Java virtual machine; the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc

20) ConcurrentHashMap Example.

```
package com.javapapers.java.collections;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

public class ConcurrentHashMapExample {

   public static void main(String[] args) {

      ConcurrentHashMap concurrentHashMap = new ConcurrentHashMap();

      concurrentHashMap.put("A","Apple");

      concurrentHashMap.put("B","Blackberry");

      for (Map.Entry e : concurrentHashMap.entrySet()) {

         System.out.println(e.getKey() + " = " + e.getValue());

      }

   }

}
```

General question

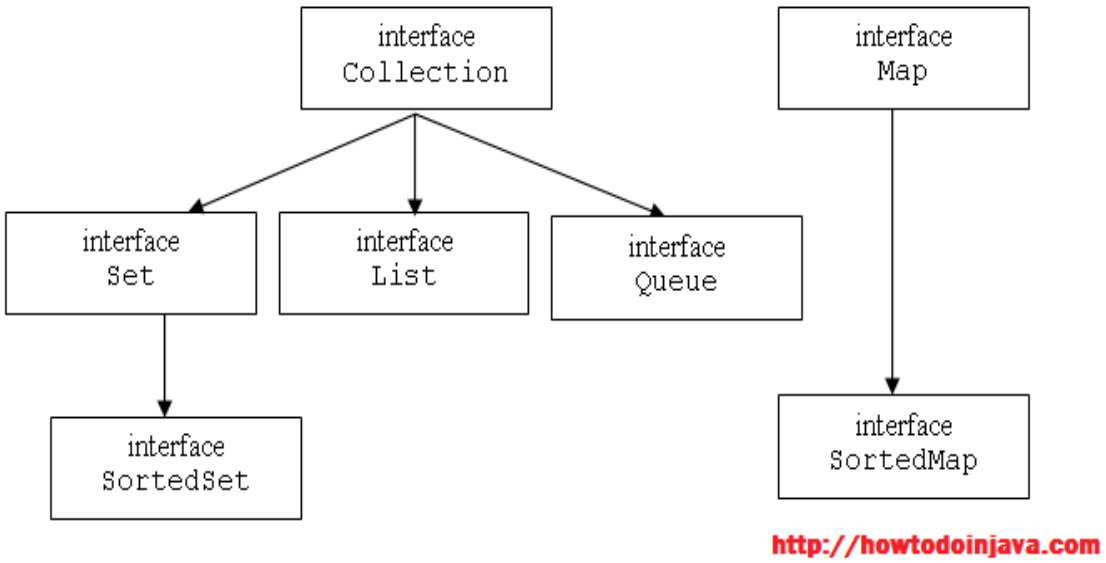
1) What is the Java Collection framework? List down its advantages?

By definition, a collection is **an object that represents a group of objects**. Like in set theory, a set is group of elements. Easy enough !!
Prior to JDK 1.2, JDK has some utility classes such as Vector and HashTable, but there was no concept of Collection framework. Later from JDK 1.2 onwards, JDK felt the need of having a consistent support for reusable data structures. Finally, the collections framework was designed and developed primarily by Joshua Bloch, and was **introduced in JDK 1.2**.

Its most **noticeable advantages** can be listed as:

- Reduced programming effort due to ready to use code
- Increased performance because of high-performance implementations of data structures and algorithms
- Provides interoperability between unrelated APIs by establishing a common language to pass collections back and forth
- Easy to learn APIs by learning only some top level interfaces and supported operations

2) Explain Collection's hierarchy?



Java Collection Hierarchy

As shown in above image, collection framework has one interface at top i.e. **Collection**. It is **extended by Set, List and Queue interfaces**. Then there are loads of other classes in these 3 branches which we will learn in following questions.
Remember the signature of Collection interface. It will help you in many question.

```
1  public interface Collection extends Iterable {
2  //method definitions
3  }
```

Framework also consist of Map interface, which is part of collection framework. but it does not extend Collection interface. We will see the reason in 4th question in this question bank.

3) Why Collection interface does not extend Cloneable and Serializable interface?

Well, simplest answer is "**there is no need to do it**". Extending an interface simply means that you are creating a subtype of interface, in other words a more specialized behavior and Collection interface is not expected to do what Cloneable and Serializable interfaces do.

Another reason is that not everybody will have a reason to have Cloneable collection because if it has very large data, then every **unnecessary clone operation will consume a big memory**. Beginners might use it without knowing the consequences.
Another reason is that **Cloneable and Serializable are very specialized behavior** and so should be implemented only when required. For example, many concrete classes in collection implement these interfaces. So if you want this feature. use these collection classes otherwise use their alternative classes.

4) Why Map interface does not extend Collection interface?

A good answer to this interview question is "**because they are incompatible**". Collection has a method add(Object o). Map can not have such method because it need key-value pair. There are other reasons also such as Map supports keySet, valueSet etc. Collection classes does not have such views.

Due to such big differences, Collection interface was not used in Map interface, and it was build in separate hierarchy.

List interface related

5) Why we use List interface? What are main classes implementing List interface?

A java list is a **"ordered" collection of elements**. This ordering is a **zero based index**. It does not care about duplicates. Apart from methods defined in Collection interface, it does **have its own methods** also which are largely to manipulate the collection **based on index location of element**. These methods can be grouped as search, get, iteration and range view. All above operations support index locations.
The main classes implementing List interface are: **Stack, Vector, ArrayList and LinkedList**. Read more about them in java documentation.

6) How to convert an array of String to arraylist?

This is more of a programmatic question which is seen at beginner level. The intent is to check the knowledge of applicant in Collection utility classes. For now, lets learn that there are two utility classes in Collection framework which are mostly seen in interviews i.e. **Collections and Arrays**.
Collections class provides some static functions to perform specific operations on collection types. And Arrays provide utility functions to be performed on array types.

```
1  //String array
2  String[] words = {&amp;quot;ace&amp;quot;, &amp;quot;boom&amp;quot;, &amp;quot;crew&amp;quot;, &amp;quot;dog&amp;quot;,
3  &amp;quot;eon&amp;quot;};
4  //Use Arrays utility class
5  List wordList = Arrays.asList(words);
   //Now you can iterate over the list
```

Please not that this function is not specific to String class, it will return List of element of any type, of which the array is. e.g.

```
1  //String array
2  Integer[] nums = {1,2,3,4};
3  //Use Arrays utility class
4  List numsList = Arrays.asList(nums);
```

7) How to reverse the list?

This question is just like above to test your knowledge of **Collections** utility class. Use it **reverse()** method to reverse the list.

```
1   Collections.reverse(list);
```

Set interface related

8) Why we use Set interface? What are main classes implementing Set interface?

It **models the mathematical set in set theory**. Set interface is like List interface but with some differences. First, it is **not ordered collection**. So no ordering is preserved while adding or removing elements. The main feature it does provide is "**uniqueness of elements**". It does not support duplicate elements.
Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.
Based on above reasons, it **does not have operations based on indexes of elements like List**. It only has methods which are inherited by Collection interface.
Main classes implementing Set interface are : **EnumSet, HashSet, LinkedHashSet, TreeSet**. Read more on related java documentation.

9) How HashSet store elements?

You must know that HashMap store key-value pairs, with one condition i.e. keys will be unique. HashSet uses Map's this feature to ensure uniqueness of elements. In HashSet class, a map declaration is as below:

```
1   private transient HashMap<E,Object> map;
2
3   //This is added as value for each key
4   private static final Object PRESENT = new Object();
```

So **when you store a element in HashSet, it stores the element as key in map and "PRESENT" object as value**. (See declaration above).

```
1   public boolean add(E e) {
2   return map.put(e, PRESENT)==null;
3   }
```

I will highly suggest you to read this post: **How HashMap works in java?** This post will help you in answering all the HashMap related questions very easily.

10) Can a null element added to a TreeSet or HashSet?

As you see, There is no null check in add() method in previous question. And HashMap also allows one null key, so **one "null" is allowed in HashSet**.
TreeSet uses the same concept as HashSet for internal logic, but uses NavigableMap for storing the elements.

```
1   private transient NavigableMap<E,Object> m;
2
3   // Dummy value to associate with an Object in the backing Map
4   private static final Object PRESENT = new Object();
```

NavigableMap is subtype of SortedMap which does not allow null keys. So essentially, **TreeSet also does not support null keys**. It will throw NullPointerException if you try to add null element in TreeSet.

Map interface related

11) Why we use Map interface? What are main classes implementing Map interface?

Map interface is a special type of collection which is **used to store key-value pairs**. It does not extend Collection interface for this reason. This interface provides methods to add, remove, search or iterate over various views of Map.
Main classes implementing Map interface are: **HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.**

12) What are IdentityHashMap and WeakHashMap?

IdentityHashMap is similar to HashMap except that **it uses reference equality when comparing elements**. IdentityHashMap class is not a widely used Map implementation. While this class implements the Map interface, it intentionally violates Map's general contract, which mandates the use of the equals() method when comparing objects. IdentityHashMap is designed for use only in the rare cases wherein reference-equality semantics are required.
WeakHashMap is an implementation of the Map interface **that stores only weak references to its keys**. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. Once such a key is discarded it can never be recreated, so it is impossible to do a look-up of that key in a WeakHashMap at some later time and be surprised that its entry has been removed.

13) Explain ConcurrentHashMap? How it works?

Taking from java docs:
A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do not entail locking, and there is not any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.
Read more about how **concurrent hashmap works and related interview questions**.

14) How hashmap works?

The **most important question** which is most likely to be seen in every level of job interviews. You must be very clear on this topic., not only because it is most asked question but also it will open up your mind in further questions related to collection APIs.
Answer to this question is very large and you should read it my post: **How HashMap works?** For now, lets remember that HashMap works **on principle of Hashing**. A map by definition is : "An object that maps keys to values". To store such structure, **it uses an inner class Entry**:

```
1  static class Entry implements Map.Entry
2  {
3  final K key;
4  V value;
5  Entry next;
6  final int hash;
7  ...//More code goes here
8  }
```

Here key and value variables are used to store key-value pairs. Whole entry object is stored in an array.

```
/**
* The table, re-sized as necessary. Length MUST Always be a power of two.
*/
transient Entry[] table;
```

The index of array is calculated on basis on hashcode of Key object. Read more of linked topic.

15) How to design a good key for hashmap?

Another good question usually followed up after answering how hashmap works. Well, the most important constraint is**you must be able to fetch the value object back in future**. Otherwise, there is no use of having such a data structure. If you understand the working of hashmap, you will find it largely depends on hashCode() and equals() method of Key objects.
So a good key object **must provide same hashCode() again and again**, no matter how many times it is fetched. Similarly, same keys **must return true when compare with equals() method and different keys must return false**.
For this reason, **immutable classes are considered best candidate for HashMap keys**.
Read more : **How to design a good key for HashMap?**

16) What are different Collection views provided by Map interface?

Map interface provides 3 views of key-values pairs stored in it:

- key set view
- value set view
- entry set view

All the views can be navigated using iterators.

17) When to use HashMap or TreeMap?

HashMap is well known class and all of us know that. So, I will leave this part by saying that it is used to store key-value pairs and allows to perform many operations on such collection of pairs.
TreeMap is special form of HashMap. **It maintains the ordering of keys** which is missing in HashMap class. This ordering is **by default "natural ordering"**. The default ordering can be override by providing an instance of Comparator class, whose compare method will be used to maintain ordering of keys.
Please note that **all keys inserted into the map must implement the Comparable interface** (this is necessary to decide the ordering). Furthermore, all such keys must be mutually comparable: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

Tell the difference questions

18) Difference between Set and List?

The most noticeable differences are :

- Set is unordered collection where List is ordered collection based on zero based index.
- List allow duplicate elements but Set does not allow duplicates.
- List does not prevent inserting null elements (as many you like), but Set will allow only one null element.

19) Difference between List and Map?

Perhaps most easy question. **List is collection of elements where as map is collection of key-value pairs**. There is actually lots of differences which originate from first statement. They have **separate top level interface, separate set of generic methods, different supported methods and different views of collection**.
I will take much time hear as answer to this question is enough as first difference only.

20) Difference between HashMap and HashTable?

There are several differences between HashMap and Hashtable in Java:

- Hashtable is synchronized, whereas HashMap is not.
- Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
- The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not.

21) Difference between Vector and ArrayList?

Lets note down the differences:

- All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.
- Vector is a Legacy class added in first release of JDK. ArrayList was part of JDK 1.2, when collection framework was introduced in java.
- By default, Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it is re-sized.

22) Difference between Iterator and Enumeration?

Iterators differ from enumerations in three ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.
- Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.
- Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc.

23) Difference between HashMap and HashSet?

HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements. That's it. No need to describe further.

24) Difference between Iterator and ListIterator?

There are three Differences are there:

- We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.
- By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.
- ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

25) Difference between TreeSet and SortedSet?

SortedSet is an interface which TreeSet implements. That' it !!

26) Difference between ArrayList and LinkedList?

- LinkedList store elements within a doubly-linked list data structure. ArrayList store elements within a dynamically resizing array.
- LinkedList allows for constant-time insertions or removals, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but grabbing an element in the middle takes time proportional to the size of the list. ArrayLists, on the other hand, allow random access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap.
- LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.

More questions

27) How to make a collection read only?

Use following methods:

- Collections.unmodifiableList(list);
- Collections.unmodifiableSet(set);
- Collections.unmodifiableMap(map);

These methods takes collection parameter and return a new read-only collection with same elements as in original collection.

28) How to make a collection thread safe?

Use below methods:
Collections.synchronizedList(list);
Collections.synchronizedSet(set);
Collections.synchronizedMap(map);

Above methods take collection as parameter and return same type of collection which are synchronized and thread safe.

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

The sole purpose of an Iterator is to enumerate through a collection. All collections contain the add() method to serve your purpose. There would be no point in adding to an Iterator because the **collection may or may not be ordered**. And**add() method can not have same implementation for ordered and unordered collections**.

30) What are different ways to iterate over a list?

You can iterate over a list using following ways:

- Iterator loop
- For loop
- For loop (Advance)
- While loop

Read more : http://www.mkyong.com/java/how-do-loop-iterate-a-list-in-java/

31) What do you understand by iterator fail-fast property?

- **Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun**. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.
- Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

32) What is difference between fail-fast and fail-safe?

You have understood fail-fast in previous question. **Fail-safe iterators** are just opposite to fail-fast. **They never fail if you modify the underlying collection on which they are iterating**, because they work on clone of Collection instead of original collection and that's why they are called as fail-safe iterator.

Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException.

33) How to avoid ConcurrentModificationException while iterating a collection?

You should first try to **find another alternative iterator which are fail-safe**. For example if you are using List and you can use ListIterator. If it is legacy collection, you can use enumeration.

If above options are not possible then you can use one of three changes:

- If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes. It is the recommended approach.
- You can convert the list to an array and then iterate on the array.
- You can lock the list while iterating by putting it in a synchronized block.

Please note that last two approaches will cause a performance hit.

34) What is UnsupportedOperationException?

This exception is thrown **on invoked methods which are not supported by actual collection type**.
For example, if you make a read-only list list using "Collections.unmodifiableList(list)" and then call add() or remove() method, what should happen. It should clearly throw UnsupportedOperationException.

35) Which collection classes provide random access of it's elements?

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it's elements.

36) What is BlockingQueue?

A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.
BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.
Read the example usage of blocking queue in post : **How to use blocking queue?**

37) What is Queue and Stack, list down their differences?

A collection designed for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.
Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.
Stack is also a form of Queue but one difference, it is LIFO (last-in-first-out).
Whatever the ordering used, the head of the queue is that element which would be removed by a call to remove() or poll(). Also note that Stack and Vector are both synchronized.
Usage: Use a queue if you want to process a stream of incoming items in the order that they are received.Good for work lists and handling requests.
Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

38) What is Comparable and Comparator interface?

In java. all collection which have feature of automatic sorting, uses compare methods to ensure the correct sorting of elements. For example classes which use sorting are TreeSet, TreeMap etc.
To sort the data elements a class needs to implement Comparator or Comparable interface. That's why all Wrapper classes like Integer,Double and String class implements Comparable interface.
Comparable helps in preserving default natural sorting, whereas Comparator helps in sorting the elements in some special required sorting pattern. The instance of comparator if passed usually as collection's constructor argument in supporting collections.

39) What are Collections and Arrays classes?

Collections and Arrays classes are special utility classes to support collection framework core classes. They provide utility functions to get read-only/ synchronized collections, sort the collection on various ways etc.

Arrays also helps array of objects to convert in collection objects. Arrays also have some functions which helps in copying or working in part of array objects.

40) Recommended resources

Well it is not interview question.. :-). This is only for fun. But you should really read my blog for more posts on collection framework knowledge.
I hope these java collection interview questions will help in in your next interview. Further, I will suggest you to read more on above questions apart from this post. A more knowledge will only help you.

41) Fail fast and fail safe iterator on collections:

Recap : Difference between Fail Fast Iterator and Fail Safe Iterator

	Fail Fast Iterator	Fail Safe Iterator
Throw ConcurrentModification Exception	Yes	No
Clone object	No	Yes
Memory Overhead	No	Yes
Examples	HashMap,Vector,ArrayList,HashSet	CopyOnWriteArrayList, ConcurrentHashMap

21) Write code for implementation of comparable and comparator?

19) How do we add element into the iterator loop by add method where the element added in serialization which method need to implemented?