

The list of core collection interfaces are : just mention the important ones

Important : Collection , Set , Queue , List , Map

Other interface also in the list : SortedSet, SortedMap , Deque, ListIterator etc

**What is the difference between List and Set ?**

Set contain only unique elements while List can contain duplicate elements.

Set is unordered while List is ordered . List maintains the order in which the objects are added .

**Q7 What is the difference between Map and Set ?**

Map object has unique keys each containing some value, while Set contain only unique values.

**Q8 What are the classes implementing List and Set interface ?**

***Class implementing List interface :*** ArrayList , Vector , LinkedList ,

***Class implementing Set interface :*** HashSet , TreeSet

**Q9 What is an iterator ?**

Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection.

**Q10 What is the difference between Iterator and Enumeration ?**

The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.

Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .

**Q11 Which design pattern followed by Iterator ?**

It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.

Enumeration is an example of Iterator design pattern.

**Q12 Which methods you need to override to use any object as key in HashMap ?**

To use any object as key in HashMap , it needs to implement equals() and hashCode() method .

**Q13 What is the difference between Queue and Stack ?**

Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.

Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is insertion or removal of CD from the CD case.

**Q14 How to reverse the List in Collections ?**

There is a built in reverse method in Collections class . reverse(List list) accepts list as parameter.

**Collections.reverse(listobject);**

**Q15 How to convert the array of strings into the list ?**

Arrays class of java.util package contains the method asList() which accepts the array as parameter.

So,

**String[] wordArray = {"Love Yourself" , "Alive is Awesome" , "Be in present"};**

**List wordList = Arrays.asList(wordArray);**

***Intermediate Level (1-3 yrs): Java Collections Interview Questions and Answers***

**Q16 What is the difference between ArrayList and Vector ?**

It is one of the frequently asked collection interview question , the main differences are

Vector is synchronized while ArrayList is not . Vector is slow while ArrayList is fast . Every time when needed, Vector increases the capacity twice of its initial size while ArrayList increases its ArraySize by 50%. find detailed explanation [ArrayList vs Vector](http://javahungry.blogspot.co.uk/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html) .

**Q17 What is the difference between HashMap and Hashtable ?**

It is one of the most popular collections interview question for java developer . Make sure you go through this once before appearing for the interview .

Main differences between HashMap and Hashtable are :

a. HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.

b. HashMap is not synchronized or thread-safe while Hashtable is synchronized or thread-safe .

find detailed explanation here [Hashtable vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html)

**Q18 What is the difference between peek(),poll() and remove() method of the Queue interface ?**

Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().

If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .

peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

**Q19 What is the difference between Iterator and ListIterator.**

Using Iterator we can traverse the list of objects in forward direction . But ListIterator can traverse the collection in both directions that is forward as well as backward.

**Q20 What is the difference between Array and ArrayList in Java ?**

This question checks whether student understand the concept of static and dynamic array. Some main differences between Array and ArrayList are :

a. Array is static in size while ArrayList is dynamic in size.

b. Array can contain primitive data types while ArrayList can not contain primitive data types.

find detailed explanation [ArrayList vs Array in Java](http://javahungry.blogspot.ca/2015/03/difference-between-array-and-arraylist-in-java-example.html)

**Q21 What is the difference between HashSet and TreeSet ?**

Main differences between HashSet and TreeSet are :

a. HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order

b. HashSet can store null object while TreeSet can not store null object.

find detailed explanation here [TreeSet vs HashSet in Java](http://javahungry.blogspot.co.uk/2014/03/difference-between-hashset-and-treeset-similarities-and-example.html)

**Q22 Write java code showing insertion,deletion and retrieval of HashMap object ?**

Do it yourself (DIY) , if found any difficulty or doubts then please mention in the comments.

**Q23 What is the difference between HashMap and ConcurrentHashMap ?**

This is also one of the most popular java collections interview question . Make sure this question is in your to do list before appearing for the interview .

Main differences between HashMap and ConcurrentHashMap are :

a. HashMap is not synchronized while ConcurrentHashMap is synchronized.

b. HashMap can have one null key and any number of null values while ConcurrentHashMap does not allow null keys and null values .

find detailed explanation here [ConcurrentHashMap vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/02/hashmap-vs-concurrenthashmap-java-collections-interview-question.html)

**Q24 Arrange the following in the ascending order (performance):**

**HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap**

Hashtable < Collections.SynchronizedMap < ConcurrentHashMap < HashMap

**Q25 How HashMap works in Java ?**

This is one of the most important question for java developers. HashMap works on the principle of Hashing . Find detailed information here to understand [what is hashing and how hashmap works in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .

**Q26 What is the difference between LinkedList and ArrayList in Java ?**

Main differences between LinkedList and ArrayList are :

a. LinkedList is the doubly linked list implementation of list interface , while , ArrayList is the resizable array implementation of list interface.

b. LinkedList can be traversed in the reverse direction using descendingIterator() method provided by the Java Api developers , while , we need to implement our own method to traverse ArrayList in the reverse direction . find the detailed explanation here [ArrayList vs LinkedList in java](http://javahungry.blogspot.co.uk/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html).

**Q27 What are Comparable and Comparator interfaces ? List the difference between them ?**

We already explained what is comparable and comparator interface in detail along with examples here, [Comparable vs Comparator in Java](http://javahungry.blogspot.com/2013/08/difference-between-comparable-and.html)

**Q28 Why Map interface does not extend the Collection interface in Java Collections Framework ?**

One liner answer : **Map interface is not compatible with the Collection interface.**

Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object . In Collection interface add(Object o) has only one parameter.

The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.

**Q29 When to use ArrayList and when to use LinkedList in application?**

ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .

Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.

**Q30 Write the code for iterating the list in different ways in java ?**

There are two ways to iterate over the list in java :

a. using Iterator

b. using for-each loop

Coding part : Do it yourself (DIY) , in case of any doubts or difficulty please mention in the comments .

***Advance Level (3+ yrs): Java Collections Interview Questions and Answers***

**Q31 How HashSet works internally in java ?**

This is one of the popular interview question . HashSet internally uses HashMap to maintain the uniqueness of elements. We have already discussed in detail [hashset internal working in java](http://javahungry.blogspot.co.uk/2013/08/how-sets-are-implemented-internally-in.html).

**Q32 What is CopyOnWriteArrayList ? How it is different from ArrayList in Java?**

[CopyOnWriteArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread safe variant of ArrayList in which all mutative operations like add , set are implemented by creating a fresh copy of the underlying array.

It guaranteed not to throw ConcurrentModificationException.

It permits all elements including null. It is introduced in jdk 1.5 .

**Q33 How HashMap works in Java ?**

We are repeating this question , as it is one of the most important question for java developer.HashMap works on the principle of Hashing . please find the detailed answer here [hashmap internal working in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .

**Q34 How remove(key) method works in HashMap ?**

This is the new question which is getting popular among java interviewers . We have shared the detailed explanation here [how remove method works internally in java](http://javahungry.blogspot.co.uk/2015/03/how-remove-method-internally-works-in-hashmap-java.html).

**Q35 What is BlockingQueue in Java Collections Framework?**

[BlockingQueue](https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html) implements the java.util.Queue interface . BlockingQueue 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 .

It does not accept null elements.

Blocking queues are primarily designed for the producer-consumer problems.

BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.

This concurrent Collection class was added in jdk 1.5

**Q36 How TreeMap works in Java ?**

TreeMap internally uses Red-Black tree to sort the elements in natural order. Please find the detailed answers here [internal implementation of TreeMap in java](http://javahungry.blogspot.co.uk/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html) .

**Q37 All the questions related to HashSet class can be found here** , [frequently asked HashSet interview questions](http://javahungry.blogspot.co.uk/2014/04/top-10-hashset-java-interview-questions-collection.html)

**Q38 What is the difference between Fail- fast iterator and Fail-safe iterator ?**

This is one of the most popular interview question for the higher experienced java developers .

Main differences between Fail-fast and Fail-safe iterators are :

a. Fail-fast throw ConcurrentModificationException while Fail-safe does not.

b. Fail-fast does not clone the original collection list of objects while Fail-safe creates a copy of the original collection list of objects.

The difference is explained in detail here [fail-safe vs fail-fast iterator in java](http://javahungry.blogspot.co.uk/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html).

**Q39 How ConcurrentHashMap works internally in Java?**

The detailed answer is already explained here [internal implementation of concurrenthashmap](http://javahungry.blogspot.co.uk/2015/02/how-concurrenthashmap-works-in-java-internal-implementation.html)

**Q40 How do you use a custom object as key in Collection classes like HashMap ?**

If one is using the custom object as key then one needs to override equals() and hashCode() method

and one also need to fulfill the contract.

If you want to store the custom object in the SortedCollections like SortedMap then one needs to make sure that equals() method is consistent to the compareTo() method. If inconsistent , then collection will not follow their contracts ,that is , Sets may allow duplicate elements.

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**

In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.

**Q42 Explain the importance of hashCode() and equals() method ? Explain the contract also ?**

HashMap object uses Key object hashCode() method and equals() method to find out the index to put the key-value pair. If we want to get value from the HashMap same both methods are used . Somehow, if both methods are not implemented correctly , it will result in two keys producing the same hashCode() and equals() output. The problem will arise that HashMap will treat both output same instead of different and overwrite the most recent key-value pair with the previous key-value pair.

Similarly all the collection classes that does not allow the duplicate values use hashCode() and equals() method to find the duplicate elements.So it is very important to implement them correctly.

**Contract of hashCode() and equals() method**

a.If object1.equals(object2) , then object1.hashCode() == object2.hashCode() should always be true.

b. If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)

**Q43 What is EnumSet in Java ?**

[EnumSet](http://docs.oracle.com/javase/7/docs/api/java/util/EnumSet.html) is a specialized Set implementation for 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.

The iterator never throws ConcurrentModificationException and is weakly consistent.

Advantage over HashSet:

All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.

It is a part of Java Collections Framework since jdk 1.5.

**Q44 What are concurrentCollectionClasses?**

In jdk1.5 , Java Api developers had introduced new package called java.util.concurrent that have thread-safe collection classes as they allow collections to be modified while iterating . The iterator is fail-safe that is it will not throw ConcurrentModificationException.

Some examples of concurrentCollectionClasses are :

a. CopyOnWriteArrayList

b. ConcurrentHashMap

**Q45 How do you convert a given Collection to SynchronizedCollection ?**

One line code : Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.

**Q46 What is IdentityHashMap ?**

**IdentityHashMap**

[IdentityHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/IdentityHashMap.html) is a class present in java.util package. It implements the Map interface with a hash table , using [reference equality instead of object equality](http://javahungry.blogspot.co.uk/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).

IdentityHashMap is not synchronized.

Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q47 What is WeakHashMap ?**

**WeakHashMap :**

[WeakHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html) is a class present in java.util package similar to IdentityHashMap. It is a Hashtable based implementation of Map interface with weak keys. An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.

It permits null keys and null values.

Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.

Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q48 How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)

Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**Q49 What is UnsupportedOperationException?**

This exception is thrown to indicate that the requested operation is not supported.

Example of UnsupportedOperationException:

In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.

**Q50 Suppose there is an Employee class. We add Employee class objects to the ArrayList. Mention the steps need to be taken , if I want to sort the objects in ArrayList using the employeeId attribute present in Employee class.**

a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)

b. We will now call Collections class sort method and pass the list as argument , that is ,

Collections.sort(empList)

If you want to add more java collections interview questions and answers or in case you have any doubts related to the Java Collections framework , then please mention in the comments.

Frequently asked Java interview questions – Part II

In my previous [post](http://www.javatechblog.com/java/frequently-asked-java-interview-questions-answers/), I have discussed some of the frequently asked [Java interview questions](http://www.javatechblog.com/tag/java-interview-questions/) for an experience level of 2-5 years. In this post we will discuss some more questions and answers.

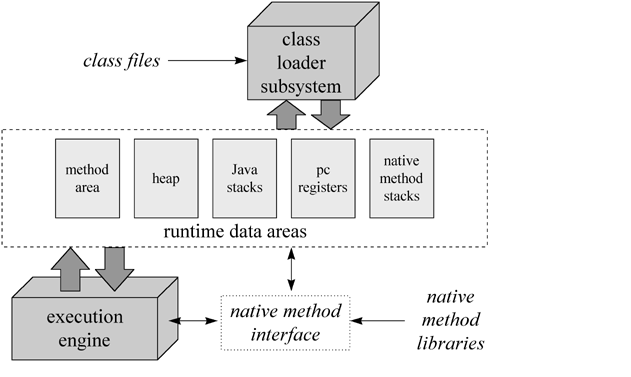
1. Difference between ArrayList and HashSet?

This question may look simple but is one of the important java interview questions. The way you answer is really important.

|  |  |
| --- | --- |
| **ArrayList** | **HashSet** |
| Implements the List interface | Implements the Set interface |
| ArrayList is an ordered collection, which means it stores and displays elements in the order they were added | HashSet is an unordered collection, it doesn’t guarantee any order.  Note: LinkedHashSet maintains the elements in insertion order. |
| Allows duplicate elements | Doesn’t allow duplicate elements |
| Can contain any number of null values | At most can contain one null value. |
| ListIterator can be used to traverse ArrayList in both the directions(forward and backward) | There is no iterator to traverse a HashSet in both directions |
| ArrayList can be accessed by index using the method get (index). | HashSet doesn’t provide get() method |
| ArrayList can be converted to an array by calling the toArray() method | HashSet doesn’t provide method to convert to an array |

2. What is a JVM? What is the difference between JRE and JVM?

A Java virtual machine (JVM) is a virtual machine that can execute Java byte code. It is the code execution component of the Java platform. Oracle’s Java execution environment is termed the Java Runtime Environment, or JRE. Programs intended to run on a JVM must be compiled into Java byte code, a standardized portable binary format which typically comes in the form of .class files (Java class files). Source code is compiled to Java byte code, which is verified, interpreted or JIT-compiled for the native architecture. The Java APIs and JVM together make up the Java Runtime Environment (JRE).



As it name specifies, it is a virtual computer placed on top of your hardware and OS. Whenever a java application is called, a JVM instance is created. The thread from OS creates the JVM instance.

Inside the JVM, the Class loader subsystem loads your compiled java byte code to the memory. The JVM creates the instance of your class object. The object instance is stored in the heap. The non-demon thread executes your methods inside your object instance and finally the non-demon thread is destroyed and the JVM instance is also ended.

3. What is String constant pool? Will the string constant pool be present when the JVM is just started?

Refer my previous post [Strings in Java](http://www.javatechblog.com/java/strings-in-java/) to understand about strings and string constant pool.

4. What is the difference between throw and throws keyword?

This is one of the most important java interview questions. Let see the difference.

|  |  |
| --- | --- |
| **Throw** | **Throws** |
| Sometimes we can create exception object and we can handover that exception object to the JVM explicitly by throw keyword. The main purpose of throw keyword is to handover our created exception object explicitly to the JVM. | The main purpose of throws keyword is to delegate the responsibilities of exception handling to the caller (method). It can be used to propagate checked exceptions. |
| Throw is used within the method body to invoke an exception | Throws is used in the method signature. When a method declares throws clause, the caller should handle this exception otherwise it will show a compile time error. |
| You cannot throw multiple exceptions withthrow keyword. | You can declare multiple exceptions in thethrows clause e.g. public void method () throws SQLException, IOException. |
| throw is followed by an instance of Exception | throws is followed by an exception class name |

5. What is LinkedList in Java? What is the difference between ArrayList and LinkedList?

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Resizable-array implementation of the List interface. | Doubly-linked list implementation of the List and Deque interfaces |
| Frequent insertion/removal of elements can be slower based on the index. This is because the elements have to be shifted in the array. | Frequent insertion/removal of elements is faster. This is because removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed. |
| Search operation is faster in ArrayList because it maintains index based system for its elements. | Search operation is slower because it requires traversing through all the elements for searching an element. |
| Memory consumption is comparatively low | Memory consumption is high because of the additional pointers required for each element to maintain the structure. |
| Supports only List operations | Supports both list and queue operations like poll(), peek() etc. |

Java doc – [LinkedList](https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html), [ArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html).

6. Can we override static method?

Static methods cannot be overridden in the exact sense of the word, but they can hide parent static methods.

In practice it means that the compiler will decide which method to execute at compile time, and not in runtime, as it does with overridden instance methods. That’s what we mean when we say a static method does not have run-time polymorphism. Instance methods can only participate in method overriding.

7. What is a Singleton class? How to create a Singleton class in Java?

A singleton class is a class that can be instantiated once, and only once. This means that there can be only one object of the singleton class at any point of time. This is a fairly unique property, but useful in a wide range of object designs. Creating an implementation of the singleton pattern is fairly straightforward – simply block off access to all constructors, provide a static method for getting an instance of the singleton, and prevent cloning. The class, itself doesn’t allow to create second object is called singleton class.

The Singleton Design Pattern is a Creational type of design pattern which assures to have only a single instance of a class.

Example:1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class A  {  private static A a;  private A()  {  }  public static A getA()  {  if(a==null)  {  a=new A();  }  return a;  }  public static void main(String[] args)  {  A a1=A.getA();  A a2=A.getA();  System.out.println(a1);  System.out.println(a2);  }} |

Example:2

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public class Singleton {  private static final Singleton instance;  static {  instance = new Singleton();  }  public static Singleton getInstance() {  return instance;  }  private Singleton() {  }} |

8. Can static variables be serialized?

Static variables are not serialized.

serialVersionUID is a special static variable used by the serialization and deserialization process, to verify that a local class is compatible with the class used to serialize an object. It’s not just a static variable as others, which are definitely not serialized.

When an object of a class is serialized, the class name and serial version UID are written to the stream of bytes. When it’s deserialized, the JVM checks if the serial version UID read from the stream of bytes is the same as the one of the local class. If they’re not, it doesn’t even try to deserialize the object, because it knows the classes are incompatible.

9. What do you mean by cloning? How to make an object cloneable? Explain deep and shallow copy.

Cloning is nothing but making a copy of a object. In order for the object to be cloneable, the class should implement the marker interface Cloneable. Read my previous post to understand about [shall copy and Deep copy](http://www.javatechblog.com/java/shallow-copy-vs-deep-copy-in-java/).

10. What does transient means or what is a transient variable?

transient is a Java keyword which marks a member variable not to be serialized when it is persisted to streams of bytes. When an object is transferred through the network, the object needs to be ‘serialized’. Serialization converts the object state to serial bytes. Those bytes are sent over the network and the object is recreated from those bytes. Member variables marked by the javatransient keyword are not transferred; they are lost intentionally.

You should serialize variables that you wish to be able to restore that you would otherwise be incapable of restoring programmatically. Similarly, you should ensure that you’re not saving data unnecessarily; if a variable can be easily derived based on processing of other serializable data, you can mark it as transient.

11. Explain Autoboxing and Unboxing.

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, adouble to a Double, and so on. The reverse process is called unboxing (Integer to int etc).

Here is the simplest example of autoboxing and unboxing:

|  |  |
| --- | --- |
| 1  2  3 | Integer i = 2; // autoboxing happens here  if(i>1){ // Unboxing of 'i' happens here  } |

What happens behind the scenes?

In the first statement what compiler does is,

|  |  |
| --- | --- |
| 1 | Integer i = Integer.valueOf(2); |

and in the second statement,

|  |  |
| --- | --- |
| 1 | if(i.intValue()>1){ |

Write a java program to find if two Strings are anagram.

This is one of the most popular interview questions asked in companies like Paypal.

What is an Anagram?

Two Strings are said to be anagrams if they contain same number of characters (excluding space), same set of characters but in different order. Following are some of the anagrams.

• dose – does

• used – dues

• care – race

• silent – listen

Anagram finder in Java using HashMap:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | public class Anagram {  public static void main(String[] args) {  checkIfAnagram("silent", "LISTEN");  checkIfAnagram("Madam Curie", "Radium came");  checkIfAnagram("Care", "Race");  checkIfAnagram("Doll", "Door");  checkIfAnagram("Does", "Dose");  }  private static void checkIfAnagram(String s1, String s2) {  boolean anagram = false;  Map<Character, Integer> letterCount1 = new HashMap<Character, Integer>();  Map<Character, Integer> letterCount2 = new HashMap<Character, Integer>();  // Remove all white spaces from s1 and s2  String s1copy = s1.replaceAll("\\s", "");  String s2copy = s2.replaceAll("\\s", "");  // Check if both strings contain same number of characters excluding white spaces  if (s1copy.length() == s2copy.length()) {  char[] s1CharArray = s1copy.toLowerCase().toCharArray();  char[] s2CharArray = s2copy.toLowerCase().toCharArray();  for (Character c : s1CharArray) {  if (letterCount1.containsKey(c)) {  int count = letterCount1.get(c);  letterCount1.put(c, ++count);  } else {  letterCount1.put(c, 1);  }}  for (Character c : s2CharArray) {  if (letterCount2.containsKey(c)) {  int count = letterCount2.get(c);  letterCount2.put(c, ++count);  } else {  letterCount2.put(c, 1);  }}  for (Character c : s1CharArray) {  if (letterCount2.get(c) != null  && letterCount1.get(c) == letterCount2.get(c)) {  anagram = true;  } else {  anagram = false;  break;  }}}  if (anagram) {  System.out.println(s1 + " and " + s2 + " are anagrams.");  } else {  System.out.println(s1 + " and " + s2 + " are not anagrams.");  }}} |

Output:

silent and LISTEN are anagrams.

Madam Curie and Radium came are anagrams.

Care and Race are anagrams.

Doll and Door are not anagrams.

Does and Dose are anagrams.

Frequently asked Java interview questions – Part III

In my previous [post](http://www.javatechblog.com/java/frequently-asked-java-interview-questions-part-ii/), I have discussed some of the frequently asked [Java interview questions](http://www.javatechblog.com/tag/java-interview-questions/) for an experience level of 2-5 years. In this post we will discuss some more questions and answers.

1. What data structure would you recommend for ordered/sorted data?

[TreeSet](https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) data structure can be used for ordered data.

2. What are the ways to create a Thread in Java?

An application that creates an instance of Thread must provide the code that will run in that thread.There are two different ways to specify which code to run in that Thread:

* Implement the java.lang.Runnable interface and pass an instance of the class implementing it to the Thread constructor.
* Extend the Thread class itself and override its run() method.

3. Difference between HashMap, LinkedHashMap and TreeMap.

Difference between any two structure will be asked. It is important that you understand all the three structures.

|  |  |  |
| --- | --- | --- |
| **HashMap** | **TreeMap** | **LinkedHashMap** |
| Hash table based implementation of the Map interface | A Red-Black tree based NavigableMap implementation. | Hash table and linked list implementation of the Map interface, with predictable iteration order. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. |
| Implements Map interface | Implements Map, NavigableMap,  SortedMap interfaces | Implements Map interface |
| This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. | The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used | The linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (*insertion-order*). |
| Null key/values Allowed | null values are allowed | Null key/values Allowed |

4. What is a daemon thread?

Daemon threads are non-user threads. They are typically used to carry out low-priority tasks that should not take priority over the main task of the program. They can be used to do useful work when all other user threads are blocked. The *garbage collector* is one example of a daemon thread. In one line, main difference between daemon thread and user thread is that as soon as all user thread finish execution java program or JVM terminates itself, JVM doesn’t wait for daemon thread to finish their execution. As soon as last non daemon thread finished, JVM terminates no matter how many Daemon thread exists or running inside JVM.

5. Difference between String, StringBuffer and StringBuilder. Which performs better?

|  |  |  |
| --- | --- | --- |
| **String** | **StringBuffer** | **StringBuilder** |
| String objects are immutable in Java | Mutuble object | Mutuable |
| Because String is final it can be safely shared between multiple threads without any extra synchronization. | StringBuffer is synchronized i.e. Thread safe | StringBuilder is not synchronized. |
| Operation is Fast | Slow | Fast |
| You can use “+” for concatenating two string because “+” operation is internally implemented using either StringBuffer or StringBuilder in Java. | You can convert a StringBuffer into String by its toString() method. | You can convert a StringBuilder into String by its toString() method. |
| Use String if you require immutability. | Use Stringbuffer in java if you need mutable + thread-safety | Use StringBuilder in java if you need mutable without thread-safety |

6. What is a marker interface?

Marker interface in Java is interface with no field or methods or in simple word empty interface in java is called marker interface. Example of market interface is Serializable, Cloneable and Remoteinterface. Looking carefully on marker interface in Java e.g. Serializable, Clonnable and Remote it looks they are used to indicate something to compiler or JVM. So if JVM sees a Class is Serializable it does some special operation on it, similar way if JVM sees one Class is implementing Cloneable it performs some operation to support cloning. Same is true for RMI and Remote interface. In summary marker interface in Java is used to indicate something to compiler, JVM or any other tool but Annotation is better way of doing same thing.

7. Write a program to check if the given number is prime or not (or) write a program to display Prime numbers from 1 to n.

Both the programs are simple. It is important to remember the logic at the time of interview.

Refer my previous [post](http://www.javatechblog.com/java/prime-number-program-in-java/) for answer.

8. Explain final, finally and finalize in Java.

|  |  |  |
| --- | --- | --- |
| **Final** | **Finally** | **Finalize** |
| Final is a keyword in Java used to apply restrictions on variable, method and Class. If you make a variable final, you cannot change its value; it will act like a constant. If you make a method final in Java, you cannot override it in sub class. If you make a class final, it cannot be sub classed. Making a class final automatically makes all its method final. | finally is a block used in exception handling along with try and catch  In Java, for exception handling you at least need either catch or finally block. finally block is guaranteed to be executed whether Exception is thrown or not, this makes it, an ideal place to close system resources e.g. InputStream or OutputStream etc. Closing streams, network connection, database connection in finally block is good coding practice in Java. | finalize() is a method called by Garbage collection thread just before collecting eligible Objects. This is the last chance for object to perform any cleanup but since it is not guaranteed that whether finalize() will be called, its bad practice to keep resource till finalize call. |

9. How do you make an object Serializable? (or) What is a Serializable interface?

To make an object serializable, the class must implement the Serializable interface.

Serializable is a marker interface(no methods or fields) which defines that the object’s state can be serialized or deserialized. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable.

10. Explain about Static keyword in Java.

11. Can we declare a class as private?

A top level class cannot be declared as private. Only public, abstract & final modifiers are permitted for a top level class. However an inner class can be declared as private.

Hope you find these questions useful. We will discuss more questions in the upcoming posts. If you have any doubts post it in the comments section.

Method overloading and overriding in Java

In this post we will discuss about method overloading and overriding in Java. These two are important concepts and the difference between them is one of the frequently asked [Java interview questions](http://www.javatechblog.com/tag/java-interview-questions/). Let us explore them with example.

Method overloading

Method overloading is nothing but having two or more methods in the same class with the same name but different number/type of arguments. Method overloading is resolved using static binding at compile time. You can also overload a constructor in Java. To overload a method, you must either need to change number of argument, type of argument or order of argument if they are of different types. Since return type is not part of method signature simply changing return type will not be considered as method overloading. Static methods can also be overloaded in Java. Method overloading is an example of compile time polymorphism (static polymorphism).

**Overloading Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | public class MethodOverloading {  public int add(int a, int b) {  System.out.println("Two int parameter add method called");  return a + b;  }  public int add(int a, int b, int c) {  System.out.println("Three int parameter add method called");  return a + b + c;  }  public double add(double a, double b) {  System.out.println("Two double parameter add method called");  return a + b;  }  public static void main(String[] args) {  MethodOverloading mo = new MethodOverloading();  mo.add(1, 2);  mo.add(1, 2, 3);  mo.add(2d, 5d);  }  }  Output:  Two int parameter add method called  Three int parameter add method called  Two double parameter add method called |

Above is a simple program which shows overloading of add() method. The method implementation to bind to the method call is resolved at the compile time itself.

Method Overriding

Method overriding is nothing but having two methods with the same name, same arguments, but different implementations. Method overriding can be done in sub class only. So, to override a method, you need to create a child class which extends parent class and create a method in child class with same name and same arguments. The return type can be same type as that of method in parent class or it can be a sub type of it. The method implementation to bind to the method call is resolved at run-time based on object using dynamic binding in Java. Method overriding is an example of run-time polymorphism (dynamic polymorphism).

Overriding example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | public class Vehicle {  public void drive() {  System.out.println("Driving vehicle ...");  }  }  class Car extends Vehicle {  @Override  public void drive() {  System.out.println("Driving Car ...");  }  }  class Truck extends Vehicle {  @Override  public void drive() {  System.out.println("Driving truck...");  }  public void load() {  System.out.println("Loading truck...");  }  }    public class RunTimePolymorphismDemo {  public static void main(String[] args) {  Vehicle vehicle = new Vehicle();  vehicle.drive();  Vehicle carVehicle = new Car();  carVehicle.drive();  Vehicle truckVehicle = new Truck();  truckVehicle.drive();  // Compile time error  // truckVehicle.load();  Truck truck = new Truck();  truck.load();  }  }  Output:  Driving vehicle ...  Driving Car ...  Driving truck...  Loading truck... |

Had the run-time polymorphism not kicked in, the output would have been: “Driving vehicle …” for all the three invocations of drive() method. You can also see that truckVehicle.load() results in a compile time error. So what’s happening in the above code?

Any object declaration and instantiation has 2 parts in it: The type of the reference and the type of the object created. For example in Vehicle carVehicle = new Car() the reference type is Vehicle and the object created is of type Car. Such an assignment is only possible when the object created type is a subclass of the reference type i.e in cases where inheritance is used.

Each object reference can be used to invoke methods and the methods which can be invoked is decided based on the reference type. And this is decided during the compile time. But the implementation to be invoked is decided based on the type of the object created. In the above example: carVehicle.drive() compiles because the drive() method is part of the Vehicle class and gives “Driving car…” as the output because the method is overridden by the Car class. On similar lines: truckVehicle.load() gives compile time error because the method load() is not part of the Vehicle class, but is defined only in the Truck class. But the truck.load() compiles because the reference type is Truck class and the compiler can resolve the load() method.

Difference between Method overloading and overriding in Java

|  |  |
| --- | --- |
| **Method overloading** | **Method overriding** |
| Method overloading is nothing but having two or more methods in the same class with the same name but different number/type of arguments. | Method overriding means having two methods with the same name, same arguments, but different implementations. |
| Overloading happens at compile-time so the binding of overloaded method call to its definition happens at compile-time. This is called static binding | Binding of overridden method call to its definition happens at runtime. This is called dynamic binding |
| Static methods can be overloaded which means a class can have more than one static method of same name. | Static methods cannot be overridden, it can be[hidden](https://docs.oracle.com/javase/tutorial/java/IandI/override.html). |
| Overloading is being done in the same class | Overriding happens between a parent and a child class |
| This is also called as compile time polymorphism | This is also called as run time polymorphism |
| private and final methods can be overloaded. | A child class cannot override the private/final methods of its base class. |
| Return type of method does not matter in case of method overloading. It can be same or different. | Return type should be the same or subtype as that of the return type of overridden method. |
| Overloaded method can declare any exception in throws clause as per your need | While overriding a method it can only throw exception declared by the overridden method or any subclass of it, means if overridden method throws IOExcpetion then overriding method can throw sub classes of IOExcpetion e.g.FileNotFoundException but not wider exception e.g. Exception. |

Frequently asked Java interview questions for experience of 2 to 5 years – Part I

Here we will see some of the frequently asked [Java interview questions](http://www.javatechblog.com/tag/java-interview-questions/) for an experience level of 2-5 years. I will also provide answers for these questions. I have faced all these questions in my interviews. It is important that you master the answer for these questions before attending any Java interview. Since there are more number of questions, I will discuss it in series of posts.

1. What are the OOP concepts in Java?

From this question, the interviewer will test your understanding of basics of Java. Java is an Object oriented programming language. In order to call Java as an OOP language, it should support OOP principles. Hence it is important to know what are the OOP concepts and how it is implemented in Java. There are four main OOP principles in Java. They are,

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

You can read more about these principles here,

[Object-Oriented Programming Concepts](https://docs.oracle.com/javase/tutorial/java/concepts/)

[OOPs concepts in Java](http://beginnersbook.com/2013/04/oops-concepts/)

2. Difference between an interface and an abstract class

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 inside an interface are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

Note: Java 8 introduces a new concept of default method implementation in interfaces. More on this here, [default methods](https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html).

3. Difference between ArrayList and vector

ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add (), get (int i) is surrounded with a synchronized block and thus making Vector class thread-safe.

Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.

4. Difference between HashMap and HashTable

This is an important question asked in many of the Java interviews.

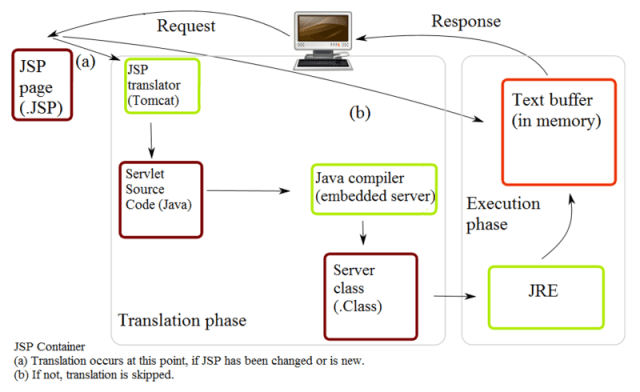
|  |  |
| --- | --- |
| **HashMap** | **HashTable** |
| HashMap is not thread-safe. Since HashMap is not synchronized it performs better than Hashtable. Java 5 introducesConcurrentHashMap which is an alternative of Hashtable and provides better scalability than Hashtable in Java. | Methods in HashTable are synchronized. So it is thread-safe |
| Allows one null key and any number of null values. | Does not permit null key and null values. |
| Iterator can be used to iterate keys and values of HashMap | Iterator or enumerator can be used for iteration |
| Iterator in the HashMap is a fail-fast iterator and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator’s own remove() method. | The enumerator for the Hashtable is not fail-fast. |
| HashMap is much faster than Hashtable | Because of thread-safety and synchronization Hashtable is much slower than HashMap |

5.What are checked and unchecked exceptions? Give an example?

Refer to my previous post [Exception handling in Java](http://www.javatechblog.com/java/exception-handling-java/).

6. Explain the life cycle of a Servlet.

This is an important question to know your understanding of Servlets.

Consider the below diagram which involves typical use case for servlet life cycle.

The following is a typical user scenario involving life cycle methods of servlet.

1. Assume that a user requests to visit a URL.
2. ->The browser then generates an HTTP request for this URL.
3. ->This request is then sent to the appropriate server.
4. The HTTP request is received by the web server and forwarded to the servlet container.
5. ->The container maps this request to a particular servlet.
6. ->The servlet is dynamically retrieved and loaded into the address space of the container.
7. The container invokes the init() method of the servlet.
8. ->This method is invoked only when the servlet is first loaded into memory.
9. ->It is possible to pass initialization parameters to the servlet so that it may configure itself.
10. The container invokes the service () method of the servlet.
11. ->This method is called to process the HTTP request.
12. ->The servlet may read data that has been provided in the HTTP request.
13. ->The servlet may also formulate an HTTP response for the client.
14. The servlet remains in the container’s address space and is available to process any other HTTP requests received from clients.
15. ->The service () method is called for each HTTP request.
16. The container may, at some point, decide to unload the servlet from its memory.
17. ->The algorithms by which this decision is made are specific to each container.
18. The container calls the servlet’s destroy () method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
19. The memory allocated for the servlet and its objects can then be garbage collected.

7. What is the difference between RequestDispatcher.Forward() and response.SendRedirect().

This is one more question to assess your knowledge in servlets.

|  |  |
| --- | --- |
| **Forward()** | **SendRediret()** |
| When we use forward method, request is transfer to other resource within the same server for further processing. | In case of sendRedirect, request is transfer to another resource to different domain or different server for further processing. |
| In case of forward, Web container handle all process internally and client or browser is not involved. | When you use SendRedirect, container transfers the request to client or browser so URL given inside the sendRedirect method is visible as a new request to the client. |
| When forward is called on requestdispatherobject we pass request and response object so our old request object is present on new resource which is going to process our request | In case of SendRedirect call old request and response object is lost because it’s treated as new request by the browser. |
| Visually we are not able to see the forwarded address, it is hidden. | In address bar we are able to see the new redirected address it is transparent. |
| Using forward () method is faster then send redirect. | SendRedirect is slower because one extra round trip is required because completely new request is created and old request object is lost. Two browser requests are required. |
| When we redirect using forward and we want to use same data in new resource we can use request.setAttribute () as we have request object available. | But in sendRedirect if we want to use we have to store the data in session or pass along with the URL. |
| Example:  When we redirect using forward and we want to use same data in new resource we can use request.setAttribute () as we have request object available. | Example:  Any kind of online payment when we use, merchant site will redirect us to net banking site which is completely new request, it process our request and again redirect to merchant site. |

8.Explain compile time and runtime polymorphism in java. Give example.

Let us consider the following Vehicle, Car and Truck class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class Vehicle{  public void drive(){  System.out.println("Driving vehicle ...");  }  }  class Car extends Vehicle{  @Override  public void drive(){  System.out.println("Driving car...");  }  }  class Truck extends Vehicle{  @Override  public void drive(){  System.out.println("Driving truck...");  }  public void load(){  System.out.println("Loading truck...");  }  } |

A Vehicle can be driven, so is a Car and Truck. But in addition to this a Truck can also be loaded with goods. Let us create instances of these classes and drive() them and try to also load() the truck.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class RunTimePolymorphismDemo {  public static void main(String[] args) {  Vehicle vehicle = new Vehicle();  vehicle.drive();  Vehicle carVehicle = new Car();  carVehicle.drive();  Vehicle truckVehicle = new Truck();  truckVehicle.drive();  //Compile time error  //truckVehicle.load();  Truck truck = new Truck();  truck.load();  }  } |

And the output is:

1.Driving vehicle …

2.Driving car…

3.Driving truck…

4.Loading truck…

Had the runtime polymorphism not kicked in, the output would have been: Driving vehicle … for all the three invocations of drive() method. You can also see that truckVehicle.load() results in a compile time error. So what’s happening in the above code?

Any object declaration and instantiation has 2 parts in it – The type of the reference and the type of the object created. For example in Vehicle carVehicle = new Car() the reference type is Vehicle and the object created is of type Car. Such an assignment is only possible when the object created type is a subclass of the reference type i.e in cases where inheritance is used.

Each object reference can be used to invoke methods and the methods which can be invoked is decided based on the reference type. And this is decided during the compile time. But the implementation to be invoked is decided based on the type of the object created. In the above example: carVehicle.drive() compiles because the drive() method is part of the Vehicle class and gives “Driving car…” as the output because the method is overridden by the Car class.

On similar lines: truckVehicle.load() gives compile time error because the method load() is not part of the Vehicle class, but is defined only in the Truck class. But the truck.load() compiles because the reference type is Truck class and the compiler can resolve the load() method.

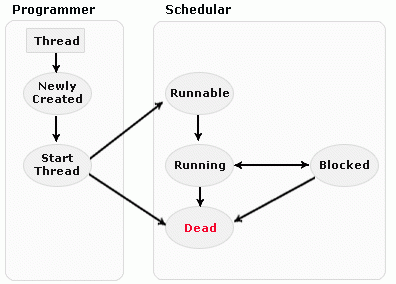
To summarise:

* The method binding happens at the compile time i.e which methods can be invoked on a given reference type is decided at the compile time.
* The selection of the method’s implementation to execute happens at the run time i.e which implementation of the method to be executed i.e the super class version or one of the subclass’s version is decided at the run time and this is what leads to the runtime polymorphism.

9. Explain the life cycle of a Thread.

When you are programming with threads, understanding the life cycle of thread is very valuable. While a thread is alive, it is in one of several states. By invoking start() method, it doesn’t mean that the thread has access to CPU and start executing straight away. Several factors determine how it will proceed.

Different states of a thread are:



10. Difference between equals () and hashCode() method and when do you override it and explain the practical use of overriding it.

1) What is the difference between ArrayList and Vector?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

2) What is the difference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because a lot of shifting is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |

3) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | Iterator traverses the elements in forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | Iterator can be used in List, Set and Queue. | ListIterator can be used in List only. |

4) What is the difference between Iterator and Enumeration?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |

5) What is the difference between List and Set?

List can contain duplicate elements whereas Set contains only unique elements.

6) What is the difference between HashSet and TreeSet?

HashSet maintains **no order** whereas TreeSet maintains **ascending order**.

7) What is the difference between Set and Map?

Set contains values only whereas Map contains key and values both.

8) What is the difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key,value). HashSet can be iterated but HashMap need to convert into Set to be iterated.

9) What is the difference between HashMap and TreeMap?

HashMap maintains **no order** but TreeMap maintains **ascending order**.

10) What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |

11) What is the difference between Collection and Collections?

Collection is an interface whereas Collections is a class. Collection interface provides normal functionality of data structure to List, Set and Queue. But, Collections class is to sort and synchronize collection elements.

12) What is the difference between Comparable and Comparator?

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | Comparator provides multiple sort of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | it is found in java.util package. |
| 4) | If we implement Comparable interface, actual class is modified. | Actual class is not modified. |

13) What is the advantage of Properties file?

If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.

14) What does the hashCode() method?

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same.

But, it is possible that two hash code numbers can have different or same keys.

15) Why we override equals() method?

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

16) How to synchronize List, Set and Map elements?

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

|  |
| --- |
| public static List synchronizedList(List l){} |
| public static Set synchronizedSet(Set s){} |
| public static SortedSet synchronizedSortedSet(SortedSet s){} |
| public static Map synchronizedMap(Map m){} |
| public static SortedMap synchronizedSortedMap(SortedMap m){} |

17) What is the advantage of generic collection?

If we use generic class, we don't need typecasting. It is typesafe and checked at compile time.

18) What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.

19) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

20) What is the default size of load factor in hashing based collection?

The default size of load factor is **0.75**. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.