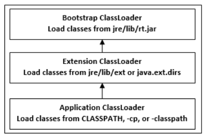
    - 
Bootstrap ClassLoader
Load classes from jre/lib/rt.jar
Extension ClassLoader
Load classes from jre/lib/ext or java.ext.dirs
Application ClassLoader
Load classes from CLASSPATH, -cp, or -classpath

-
- **Java TreeSet class contains unique elements only like HashSet.**
- **Java TreeSet class access and retrieval times are quiet fast.**
- **Java TreeSet class doesn't allow null element.**
- **Java TreeSet class is non synchronized.**
- **Java TreeSet class maintains ascending order.**

- **Multi tasking :- process - chrome , eclipse**
- **Multithreading part of same pr**

```
import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.logging.Level;

import java.util.logging.Logger;

public class ProducerConsumerPattern {

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

```
    //Creating shared object

    BlockingQueue sharedQueue = new LinkedBlockingQueue();



    //Creating Producer and Consumer Thread

    Thread prodThread = new Thread(new Producer(sharedQueue));

    Thread consThread = new Thread(new Consumer(sharedQueue));



    //Starting producer and Consumer thread

    prodThread.start();

    consThread.start();

   }



}
```

```
//Producer Class in java

class Producer implements Runnable {



    private final BlockingQueue sharedQueue;



    public Producer(BlockingQueue sharedQueue) {

        this.sharedQueue = sharedQueue;

    }



    @Override

    public void run() {

        for(int i=0; i<10; i++){

            try {
```

```
            System.out.println("Produced: " + i);

            sharedQueue.put(i);

          } catch (InterruptedException ex) {

            Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);

          }

        }

    }



}



//Consumer Class in Java

class Consumer implements Runnable{



    private final BlockingQueue sharedQueue;
```

```
public Consumer (BlockingQueue sharedQueue) {

    this.sharedQueue = sharedQueue;

}



@Override

public void run() {

    while(true){

        try {

            System.out.println("Consumed: "+ sharedQueue.take());

        } catch (InterruptedException ex) {

            Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);

        }

    }
    }

}
```

Output:

```
Produced: 0

Produced: 1

Consumed: 0

Produced: 2

Consumed: 1

Produced: 3

Consumed: 2

Produced: 4
```

```
Consumed: 3
Produced: 5
Consumed: 4
Produced: 6
Consumed: 5
Produced: 7
Consumed: 6
Produced: 8
Consumed: 7
Produced: 9
Consumed: 8
Consumed: 9
```

You see Producer Thread produced number and Consumer thread consumes it in FIFO order because the blocking queue allows elements to be accessed in FIFO.

That’s all on How to use a Blocking Queue to solve Producer Consumer problem or example of Producer consumer design pattern. I am sure its much better than wait notify example but be prepare with both if you are going for any Java Interview as Interview may ask you both way

Serialization

Well, serialization allows us to convert the state of an object into a byte stream, which then can be saved into a file on the local disk or sent over the network to any other machine. And deserialization allows us to reverse the process, which means reconverting the serialized byte stream to an object again.

Transient - donot serialize

There is no direct way to prevent subclass from serialization in java. One possible way by which a programmer can achieve this is by implementing the writeObject() and readObject() methods in the subclass and needs to throw NotSerializableException from these methods. These methods are executed during serialization and de-serialization respectively. By overriding these methods, we are just implementing our own custom serialization.

Thread multitasking

Java 8 features

Lambada expression

Date

Stream diff method available

forEach

Stream has provided a new method ‘forEach’ to iterate each element of the stream. The following code segment shows how to print 10 random numbers using forEach.

```
Random random = new Random();

random.ints().limit(10).forEach(System.out::println);
```

map

The ‘map’ method is used to map each element to its corresponding result. The following code segment prints unique squares of numbers using map.

```
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
```

```
//get list of unique squares

List<Integer> squaresList = numbers.stream().map( i ->
i*i).distinct().collect(Collectors.toList());
```

filter

The ‘filter’ method is used to eliminate elements based on a criteria. The following code segment prints a count of empty strings using filter.

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");

//get count of empty string

int count = strings.stream().filter(string -> string.isEmpty()).count();
```

limit

The ‘limit’ method is used to reduce the size of the stream. The following code segment shows how to print 10 random numbers using limit.

```
Random random = new Random();

random.ints().limit(10).forEach(System.out::println);
```

sorted

The ‘sorted’ method is used to sort the stream. The following code segment shows how to print 10 random numbers in a sorted order.

```
Random random = new Random();

random.ints().limit(10).sorted().forEach(System.out::println);
```

Parallel Processing

parallelStream is the alternative of stream for parallel processing. Take a look at the following code segment that prints a count of empty strings using parallelStream.

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");

//get count of empty string
```

```
long count = strings.parallelStream().filter(string -> string.isEmpty()).count();
```

It is very easy to switch between sequential and parallel streams.

Collectors

Collectors are used to combine the result of processing on the elements of a stream. Collectors can be used to return a list or a string.

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");

List<String> filtered = strings.stream().filter(string ->
!string.isEmpty()).collect(Collectors.toList());

System.out.println("Filtered List: " + filtered);

String mergedString = strings.stream().filter(string ->
!string.isEmpty()).collect(Collectors.joining(", "));

System.out.println("Merged String: " + mergedString);
```

Statistics

With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.

List numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

IntSummaryStatistics stats = numbers.stream().mapToInt((x) -> x).summaryStatistics();

System.out.println("Highest number in List : " + stats.getMax());

System.out.println("Lowest number in List : " + stats.getMin());

System.out.println("Sum of all numbers :

Difference between parallel and normal stream when we should use which

streams are wrappers around a data source, allowing us to operate with that data source and making bulk processing convenient and fast.

Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes

List<User> sortedList = users.stream().

```
sorted(Comparator.comparingInt(User::getAge))

         .collect(Collectors.toList());
```

- **Abstraction**
    - **Simple things** represent complex things.
    - how to **turn on tv** (do not know how it work)
    - Objects classes variable simple outside complex inside

- **Encapsulation**
    - **Private fields** Keeping fields within private
    - **Access Though public method :-** Provide access through public method
    - **Protective Barrier :-** It’s a protective barrier that keeps the data and code safe within the class itself

There are four main OOP concepts in Java. These are:

- **Abstraction.**
    - **Simple thing** :- Abstraction means using simple things to represent complexity.
    - **How to turn on TV but do not know how it work:-** We all know how to turn the TV on, but we don’t need to know how it works in order to enjoy it.
    - **Simple outside complex inside :-** abstraction means simple things like **objects**, **classes**, and **variables** represent more complex underlying code and data. This is important because it lets you avoid repeating the same work multiple times.
- **Encapsulation.** This is the practice of keeping fields within a class private, then providing access to them via
- public methods. It’s a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.
- **Inheritance.** This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.
- **Polymorphism.** This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. That’s when different meanings are implied by the code itself. The other form is **method overriding**. That’s when the different meanings are implied by the values of the supplied variables. See more on this below.

1. Which two method you need to implement for key Object in HashMap ?
In order to use any object as Key in HashMap, it must implements equals and hashcode method in Java. Read How HashMap works in Java for detailed explanation on how equals and hashcode method is used to put and get object from HashMap.

2. What is immutable object? Can you write immutable object?
Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object result in new object. For example is String is immutable in Java. Mostly Immutable are also final in Java, in order to prevent sub class from overriding methods in Java which can compromise Immutability. You can achieve same functionality by making member as non final but private and not modifying them except in constructor.

3. What is the difference between creating String as new() and literal?
When we create string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

```
String s = new String("Test");
```

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

4. What is difference between StringBuffer and StringBuilder in Java ?

Classic Java questions which some people thing tricky and some consider very easy. StringBuilder in Java is introduced in Java 5 and only difference between both of them is that Stringbuffer methods are synchronized while StringBuilder is non synchronized. See StringBuilder vs StringBuffer for more differences.

5. Write code to find the First non repeated character in the String ?
Another good Java interview question, This question is mainly asked by Amazon and equivalent companies. See first non repeated character in the string : Amazon interview question

6. What is the difference between ArrayList and Vector ?
This question is mostly used as a start up question in Technical interviews on the topic of Collection framework . Answer is explained in detail here Difference between ArrayList and Vector .

7. How do you handle error condition while writing stored procedure or accessing stored procedure from java?
This is one of the tough Java interview question and its open for all, my friend didn't know the answer so he didn't mind telling me. my take is that stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

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

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

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

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

Singleton in Java is a class with just one instance in whole Java application, for example java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum its very easy. see my article How to create thread-safe Singleton in Java for more details on writing Singleton using enum and double checked locking which is purpose of this Java interview question.

11. Can you write critical section code for singleton?

12. Can you write code for iterating over HashMap in Java 7 and Java 8 ?
Tricky one but he managed to write using while and for loop. You can find the answer here How to iterate or loop over HashMap in Java with Example.

13. When do you override hashcode and equals() ?
Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap.

14. What will be the problem if you don't override hashcode() method ?
You will not be able to recover your object from hash Map if that is used as key in HashMap. See here How HashMap works in Java for detailed explanation.

15. Is it better to synchronize critical section of getInstance() method or whole getInstance() method ?
Answer is critical section because if we lock whole method than every time some one call this method will have to wait even though we are not creating any object)

16. What is the difference when String is gets created using literal or new() operator ?
When we create string with new() its created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap.

17. Does not overriding hashcode() method has any performance implication ?
This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.

18. What’s wrong using HashMap in multithreaded environment? When get() method go to infinite loop ?
Another good question. His answer was during concurrent access and re-sizing.

19. What do you understand by thread-safety ? Why is it required ? And finally, how to achieve thread-safety in Java Applications ?

Java Memory Model defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably see writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.

Let's first discuss about Memory Barrier which are the base for our further discussions. There are two type of memory barrier instructions in JMM - read barriers and write barrier.

A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.
A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.
JMM semantics for synchronized
When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead).
Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)
Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

JMM semantics for Volatile fields
Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

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

```
Data data = null;
volatile boolean flag = false;

Thread A
-------------
data = new Data();
flag = true; <-- writing to volatile will flush data as well as flag to main memory

Thread B
-------------
if(flag==true){ <-- as="" barrier="" data.="" flag="" font="" for="" from="" perform=""
read="" reading="" volatile="" well="" will="">
use data;  <!--- data is guaranteed to visible even though it is not declared volatile because of
the JMM semantics of volatile flag.
}
```

20. What will happen if you call return statement or System.exit on try or catch block ? will finally block execute?
This is a very *popular tricky Java question* and its tricky because many programmer think that finally block always executed. This question challenge that concept by putting return statement

in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put return statement in try block or catch block but finally block won't run if you call System.exit form try or catch.

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

22. What will happen if we put a key object in a HashMap which is already there ?
This tricky Java questions is part of How HashMap works in Java, which is also a popular topic to create confusing and tricky question in Java. well if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys.

23. If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException?
One more tricky Java questions from overloading and overriding concept. Answer is you can very well throw super class of RuntimeException in overridden method but you can not do same if its checked Exception.

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

```
public int compareTo(Object o){
  Employee emp = (Employee) emp;
  return this.id - o.id;
}
```

25. How do you ensure that N thread can access N resources without deadlock
If you are not well versed in writing multi-threading code then this is real tricky question for you. This Java question can be tricky even for experienced and senior programmer, who are not really exposed to deadlock and race conditions. Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

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

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

28. What is the difference between sleep() and wait() method?
sleep() does not release the lock while wait method release the lock. sleep() method is present in java.lang.Thread class while wait() method is present in java.lang.Object class. For more differences
please check difference between sleep and wait method.

Is Java platform independent?

- Yes. Java is a platform independent language.
- We can write java code on one platform and run it on another platform.
- e.g. we can write and compile the code on windows and can run it on Linux or any other supported platform. This is one of the main features of java.

What is javac ?

- It produces the java byte code from *.java file.
- It is the intermediate representation of your source code that contains instructions.

What is class?

- Class is nothing but a template that describes the data and behavior associated with instances of that class

What is the base class of all classes?

`java.lang.Object`

What is Unicode?

- Java uses Unicode to represent the characters.
- Unicode defines a fully international character set that can represent all of the characters found in human languages.

What is Type casting in Java?

- To create a conversion between two incompatible types,
- we must use a cast.
- There are two types of casting in java: automatic casting (done automatically) and explicit casting (done by programmer).

Abstract class?

An abstract class is a class which can't be instantiated (we cannot create the object of abstract class), we can only extend such classes. It provides the generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. We can achieve partial abstraction using abstract classes, to achieve full abstraction we use interfaces.

Q) What is Interface in java?

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Read more about interface here.

Q) What is the difference between abstract class and interface?

1) abstract class can have abstract and non-abstract methods. An interface can only have abstract methods.

2) An abstract class can have static methods but an interface cannot have static methods.

3) abstract class can have constructors but an interface cannot have constructors.

What is Collection ? What is a Collections Framework ? What are the benefits of Java Collections Framework ?

Collection : A collection (also called as container) is an object that groups multiple elements into a single unit.

Collections Framework : Collections framework provides unified architecture for manipulating and representing collections.

Benefits of Collections Framework :

1. Improves program quality and speed
2. Increases the chances of reusability of software
3. Decreases programming effort.

Root Element in collection Hierarchy
Collection interface

What is the difference between Collection and Collections ?
Collection is an interface while Collections is a java class

Which collection classes are synchronized or thread-safe ?
Stack, Properties , Vector and Hashtable

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 .

What is the difference between Map and Set ?
Map object has unique keys each containing some value, while Set contain only unique values.

Class implementing List interface : ArrayList , Vector , LinkedList ,
Class implementing Set interface : HashSet , TreeSet

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

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 .

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

Convert array of string String[] wordArray = {"Love Yourself" , "Alive is Awesome" , "Be in present"};
List wordList = Arrays.asList(wordArray);

Difference between ArrayList and Vector
Synchronized :- Vector synchronized
ArrayList NotSync
Speed :- Vector slow
Arraylist Fast

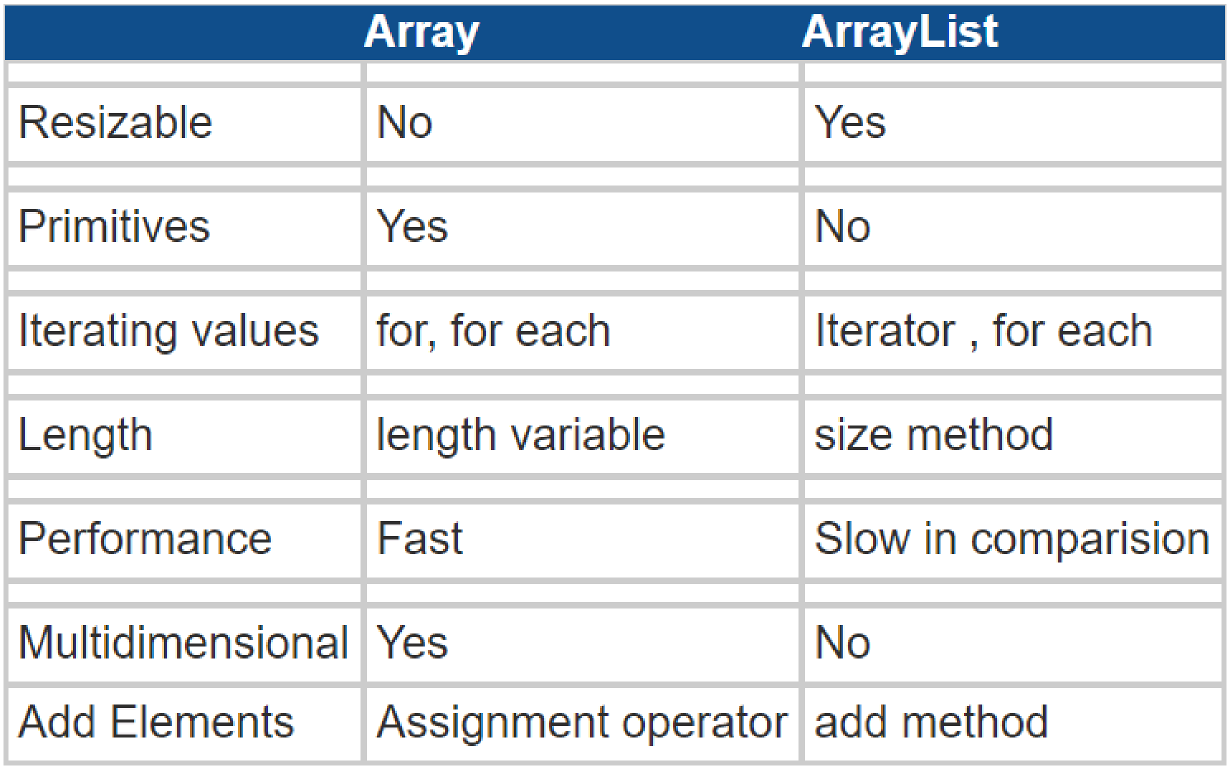
Difference HashMap & HashSet
Null :- HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.
Synchronized :- HashMap synchronized and Thread Safe
HashSet NotSync and Not Thread safe

Iterating the values: Hashmap object values are iterated by using iterator .
HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.

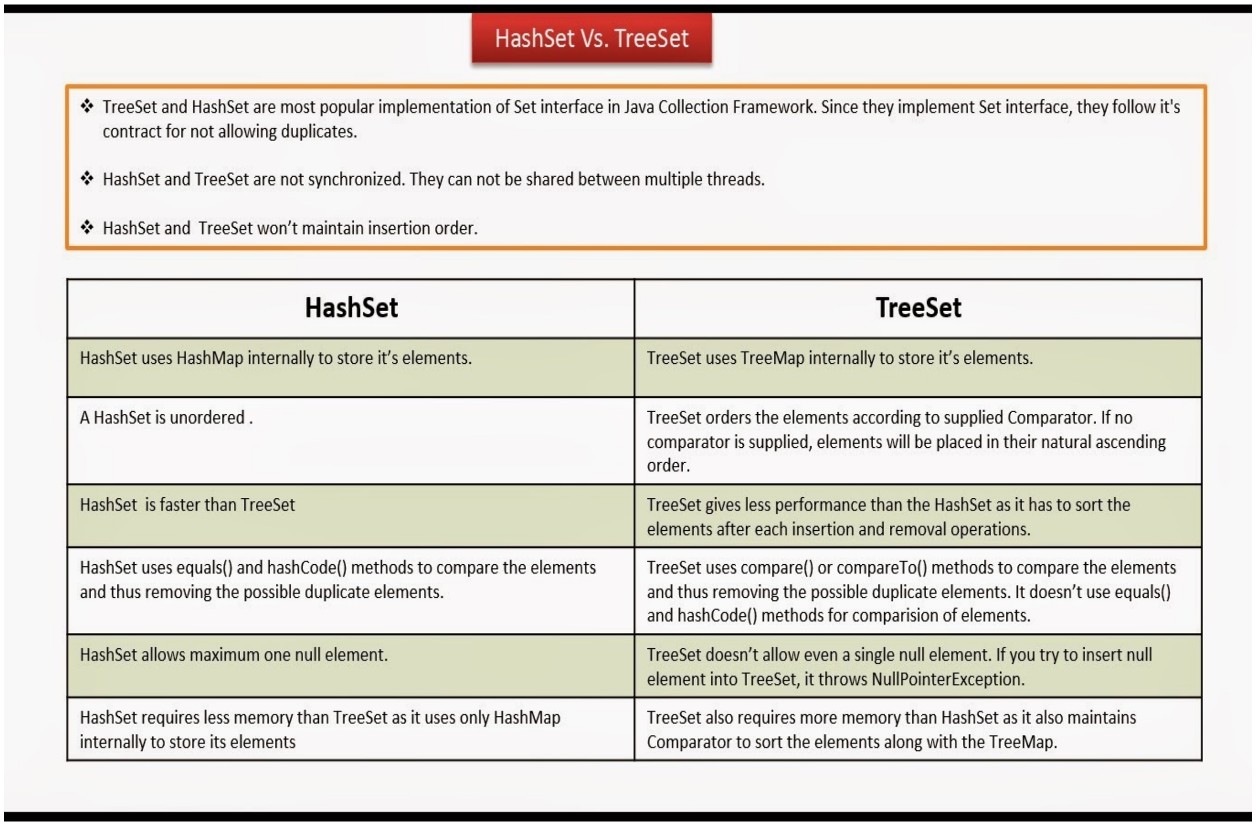
Performance : Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized

Queue :- Peek() , Poll() and remove()
Both poll() and remove() method is used to remove head object of the Queue.
Return Type
queque empty :-
Poll() ==> Null
remove() ==> 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.



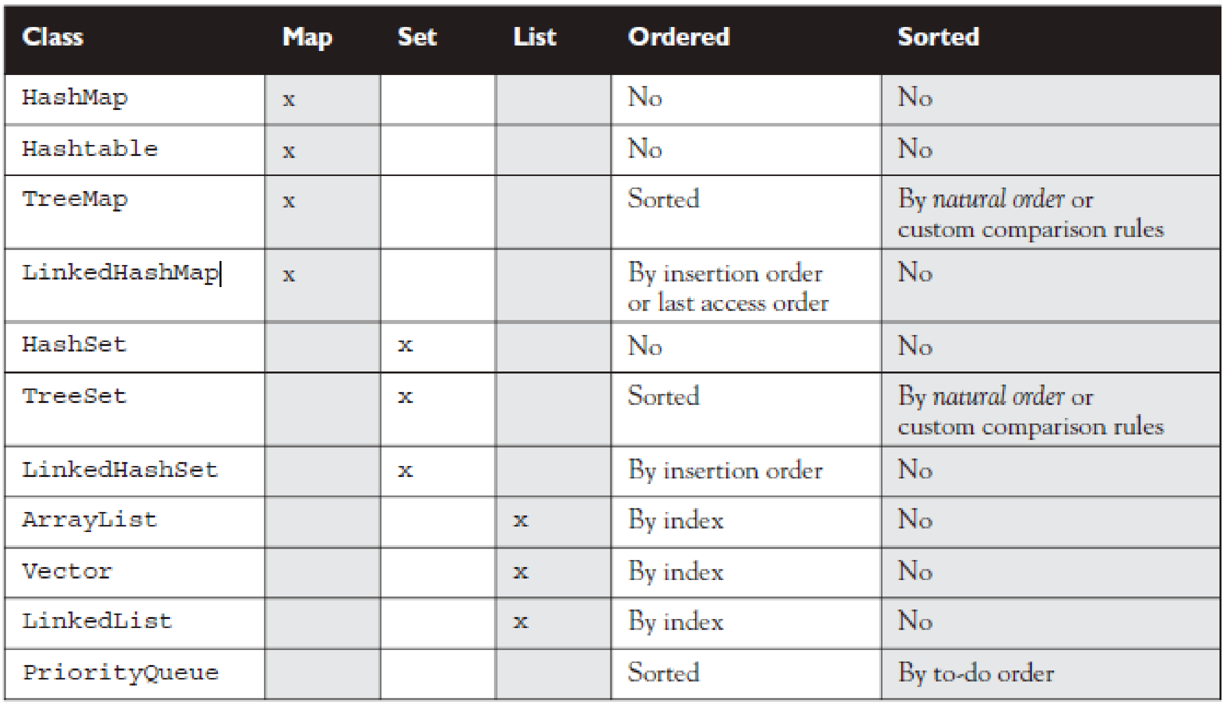
	Array	ArrayList
Resizable	No	Yes
Primitives	Yes	No
Iterating values	for, for each	Iterator , for each
Length	length variable	size method
Performance	Fast	Slow in comparision
Multidimensional	Yes	No
Add Elements	Assignment operator	add method



HashSet Vs. TreeSet

- ❖ TreeSet and HashSet are most popular implementation of Set interface in Java Collection Framework. Since they implement Set interface, they follow it's contract for not allowing duplicates.
- ❖ HashSet and TreeSet are not synchronized. They can not be shared between multiple threads.
- ❖ HashSet and TreeSet won't maintain insertion order.

HashSet	TreeSet
HashSet uses HashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.
A HashSet is unordered .	TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order.
HashSet is faster than TreeSet	TreeSet gives less performance than the HashSet as it has to sort the elements after each insertion and removal operations.
HashSet uses equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements.	TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn't use equals() and hashCode() methods for comparision of elements.
HashSet allows maximum one null element.	TreeSet doesn't allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException.
HashSet requires less memory than TreeSet as it uses only HashMap internally to store its elements	TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap.



Class	Map	Set	List	Ordered	Sorted
HashMap	x			No	No
Hashtable	x			No	No
TreeMap	x			Sorted	By *natural order* or custom comparison rules
LinkedHashMap	x			By insertion order or last access order	No
HashSet		x		No	No
TreeSet		x		Sorted	By *natural order* or custom comparison rules
LinkedHashSet		x		By insertion order	No
ArrayList			x	By index	No
Vector			x	By index	No
LinkedList			x	By index	No
PriorityQueue				Sorted	By to-do order

How to make Map Ordered

- LinkedHashMap

Iteration order for above implementations:

- HashSet - undefined
- HashMap - undefined
- LinkedHashSet - insertion order
- LinkedHashMap - insertion order of keys (by default), or 'access order'
- ArrayList - insertion order
- LinkedList - insertion order
- TreeSet - ascending order, according to Comparable / Comparator
- TreeMap - ascending order of keys, according to Comparable / Comparator

- Collections.synchronizedList(new ArrayList<YourClassNameHere>())

Principal features of non-primary implementations:

- HashMap has slightly better performance than LinkedHashMap, but its iteration order is undefined
- HashSet has slightly better performance than LinkedHashSet, but its iteration order is undefined
- TreeSet is ordered and sorted, but slower
- TreeMap is ordered and sorted, but slower
- LinkedList has fast adding to the start of the list, and fast deletion from the interior via iteration

How HashMap works in Java?

- HashMap stores key-value pair in Map.Entry static nested class implementation.
- HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.
- When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair
- The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.
- When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it's value. Below image will explain these detail clearly.
- The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is 16 and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry,
- if map size is greater than threshold, HashMap rehashes the contents of map into a new array with a larger capacity. The capacity is always power of 2,
- so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it's good idea to initialize the HashMap with correct capacity and load factor.

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

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

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

- If `o1.equals(o2)`, then `o1.hashCode() == o2.hashCode()`should always be `true`.
- If `o1.hashCode() == o2.hashCode` is true, it doesn't mean that `o1.equals(o2)` will be `true`.

https://www.journaldev.com/1330/java-collections-interview-questions-and-answers