ANDROID INTERVIEW QUESTION PART 1

1) What are OOPS concepts?

Object ☐ *Any entity that has a state and behavior is known as an object*. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Class ☐ *Collection of objects* is called a class. It is a logical entity.
A class can also be defined as a blueprint from which you can create an individual object. Class doesn’t consume any space.

Inheritance ☐ *When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism ☐ *If one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, a shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.
Another example can be to speak something; for example, a cat speaks meow, a dog barks woof, etc.

Abstraction ☐ *Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don’t know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation ☐ *Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines.

2) What is Method Overloading?

If a class has multiple methods having the same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having the same name of the methods increases the readability of the program.

3) What is Method Overriding?

If a subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

4) Types of inheritance in Java?

On the basis of class, there can be three types of inheritance in Java: *single*, *multilevel*, and *hierarchical*.

In Java programming, multiple and hybrid inheritance is supported through the interface only.

5) Why multiple inheritance is not supported in Java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from the child class object, there will be ambiguity to call the method of the A or B class.

6) What is an Abstract class in Java?

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

7) What is an Abstraction in Java?

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way is it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don’t know the internal processing of message delivery.

There are two ways to achieve abstraction in Java

1. Abstract class (0 to 100%)
2. Interface (100%)

8) What is an Interface in Java?

An ***interface in Java*** *is a blueprint of a class*. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java

interface, not a method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class

9) Why use Java interface?

There are mainly three reasons to use an interface.

1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

10) What are the various access specifiers in Java?

In Java, access specifiers are the keywords which are used to define the access scope of the method, class, or variable.

In Java, there are four access specifiers given below.

Public: The classes, methods, or variables which are defined as public, can be accessed by any class or method.

Protected: The Protected can be accessed by the class of the same package, by the sub-class of this class, or within the same class.

Default: The Default is accessible within the package only. By default, all the classes, methods, and variables are of default scope.

Private: The private class, methods, or variables defined as private can be accessed within the class only.

11) What is a Constructors in Java?

The constructor can be defined as the special type of method that is used to initialise the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

There are below rules defined for the constructor.

1. The constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

There are two types of constructors in Java:

1. **Default constructor:** A constructor is called a "Default Constructor" when it doesn't have any parameter.
2. **Parameterized constructor:** A constructor which has a specific number of parameters is called a parameterized constructor.

12) What is this keyword in Java?

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

13) Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

14) Can there be an abstract method without an abstract class?

No, if there is an abstract method in a class, that class must be abstract.

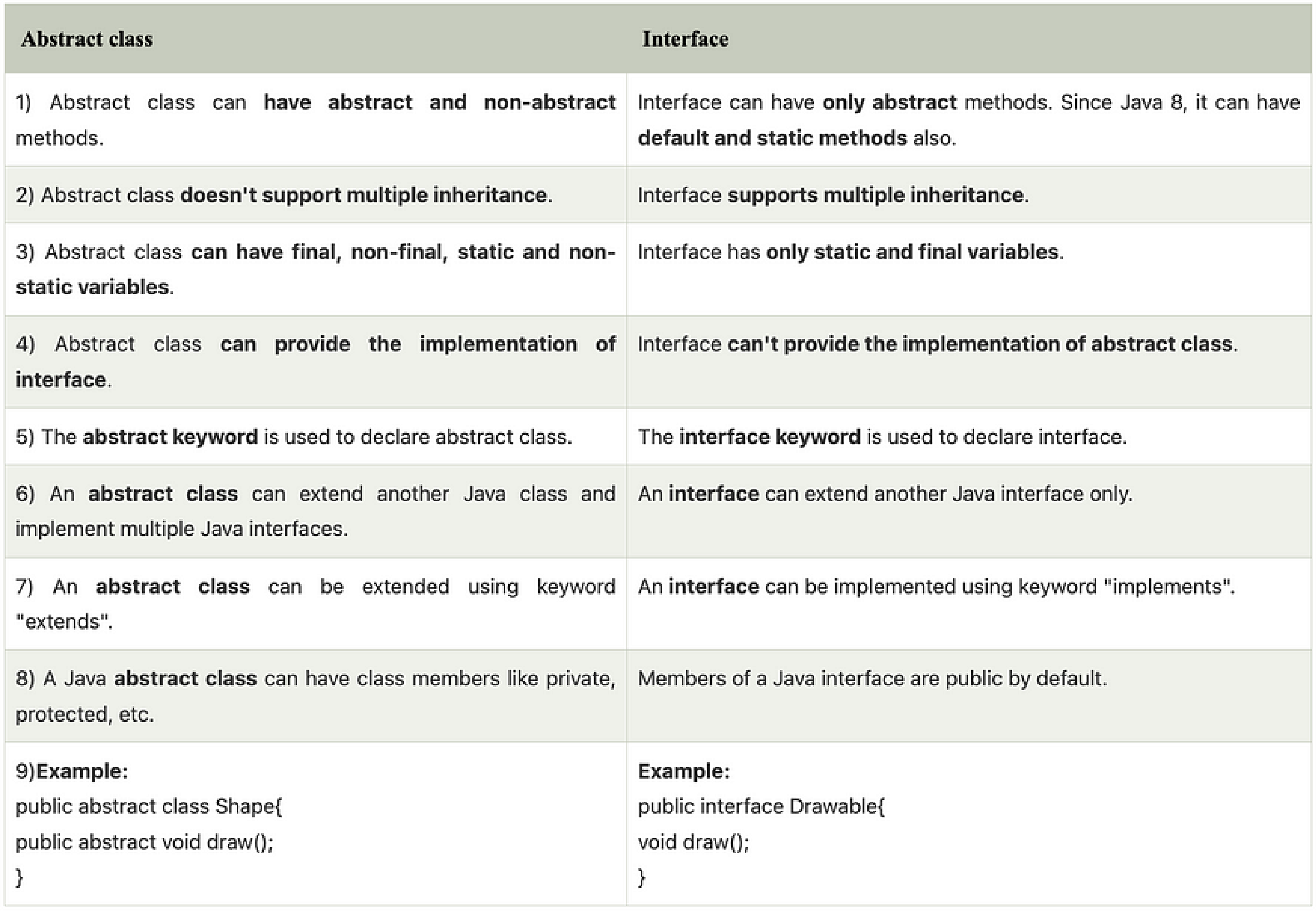
15) Why are the objects immutable in Java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object “sachin”. If one reference variable changes the value of the object, it will be affected by all the reference variables.

16) What is String Pool?

The string pool is the space reserved in the heap memory that can be used to store the strings. The main advantage of using the String pool is whenever we create a string literal; the JVM checks the “string constant pool” first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn’t exist in the pool, a new string instance is created and placed in the pool. Therefore, it saves memory by avoiding duplicacy.

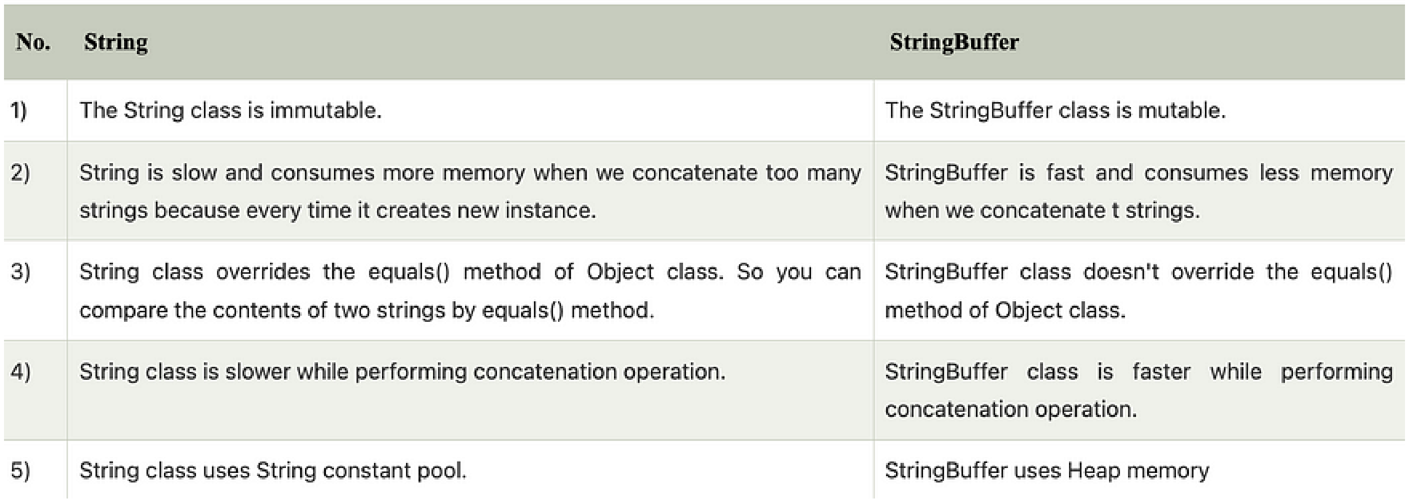
17) What are the differences between abstract class interface?



Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

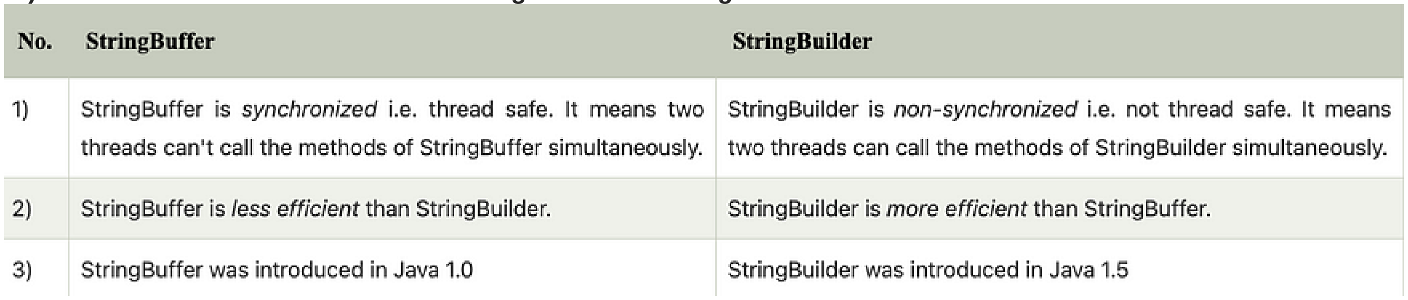
9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

18) What are the differences between String and StringBuffer?



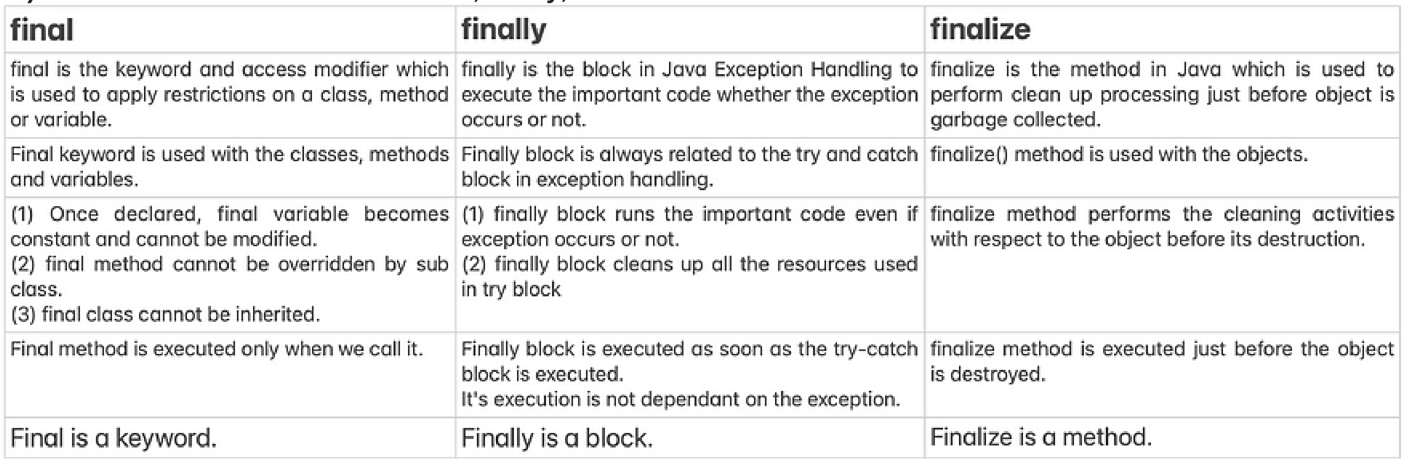
No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

19) What are the differences between StringBuffer and StringBuilder?



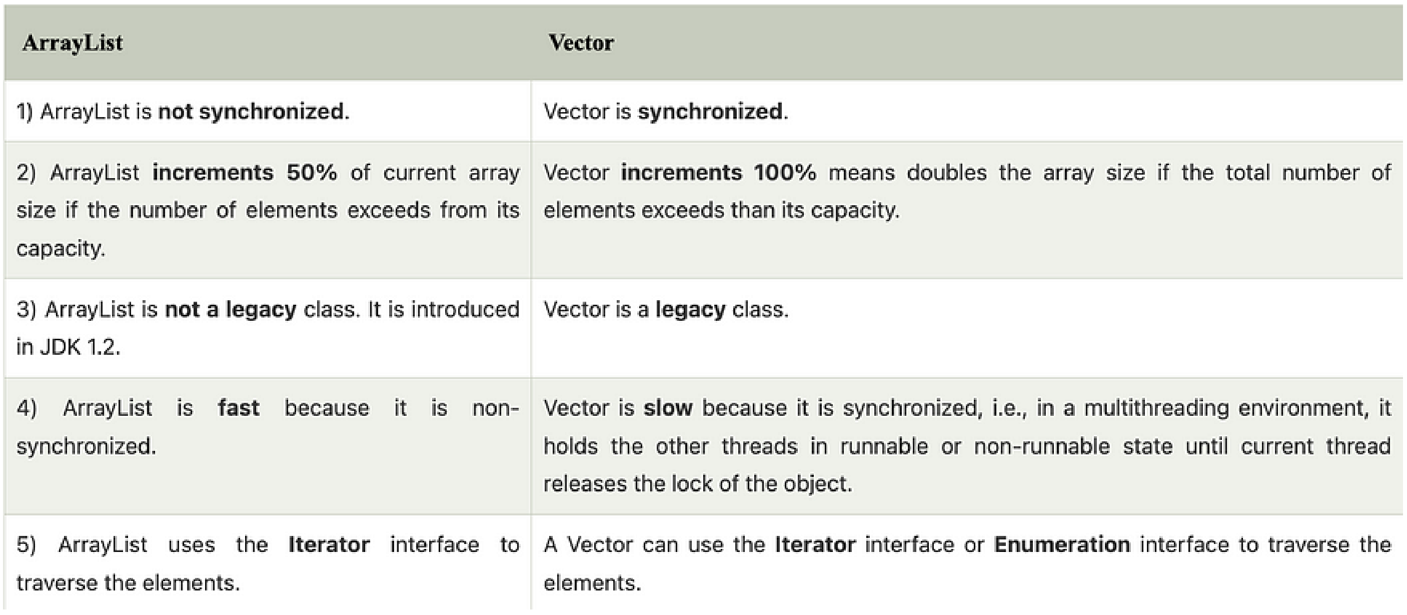
No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

19) What is the difference between final, finally, and finalize?



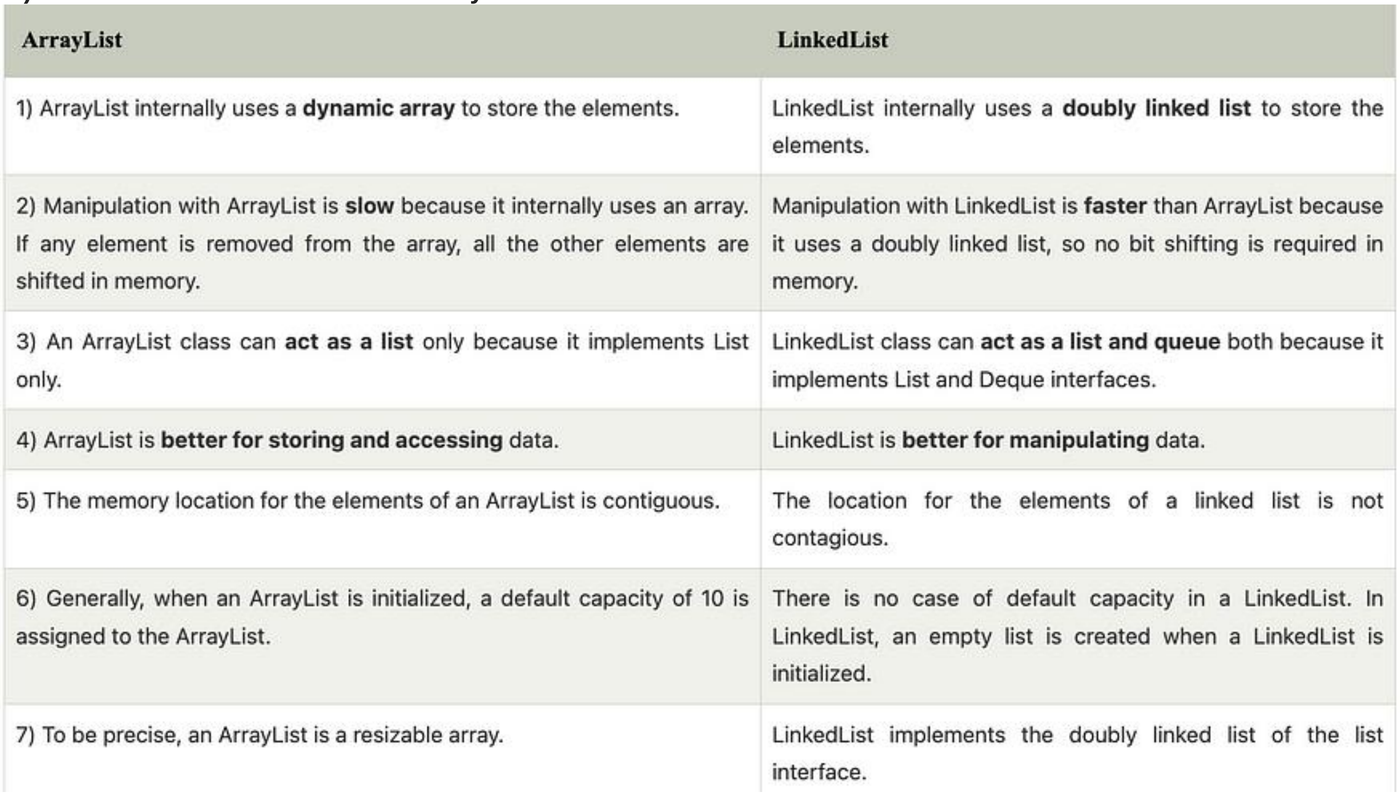
final	finally	finalize
final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.
Final is a keyword.	Finally is a block.	Finalize is a method.

20) What is the difference between ArrayList and Vector?



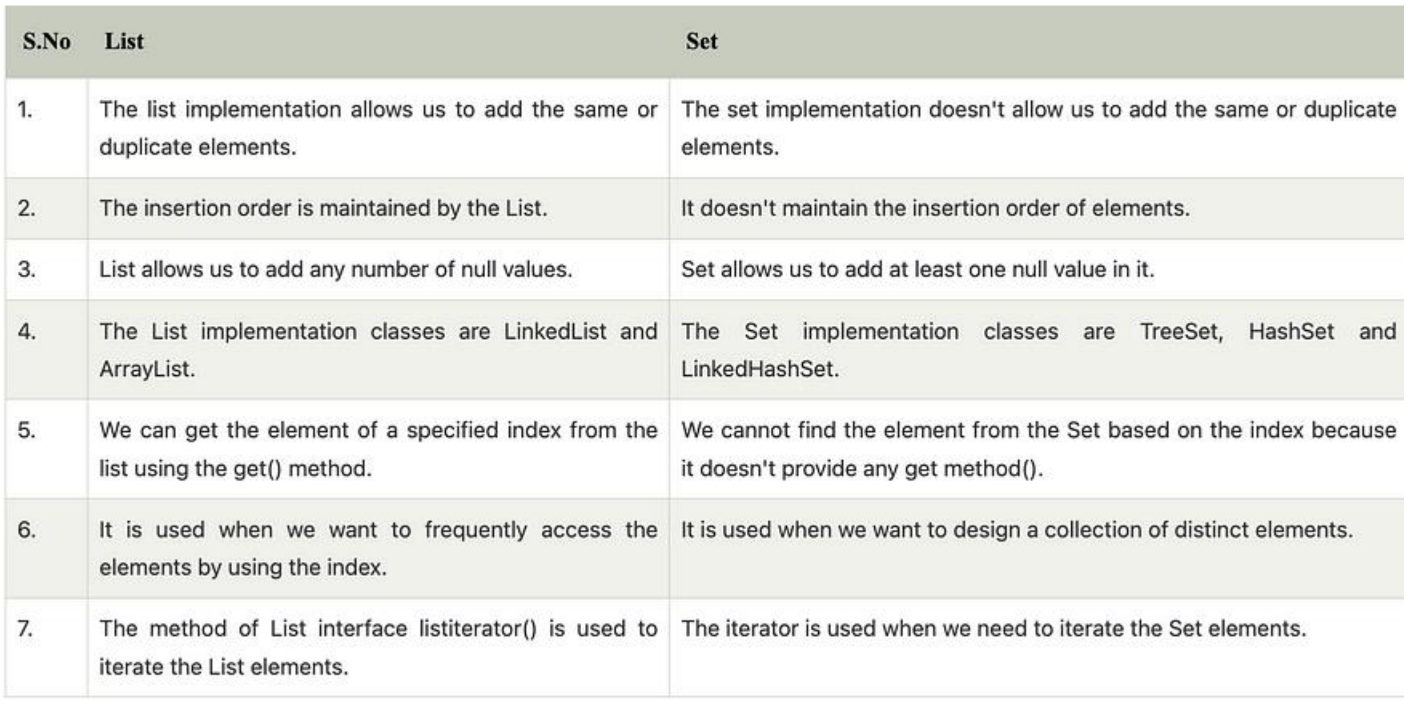
ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.

21) What is the difference between ArrayList and LinkedList?



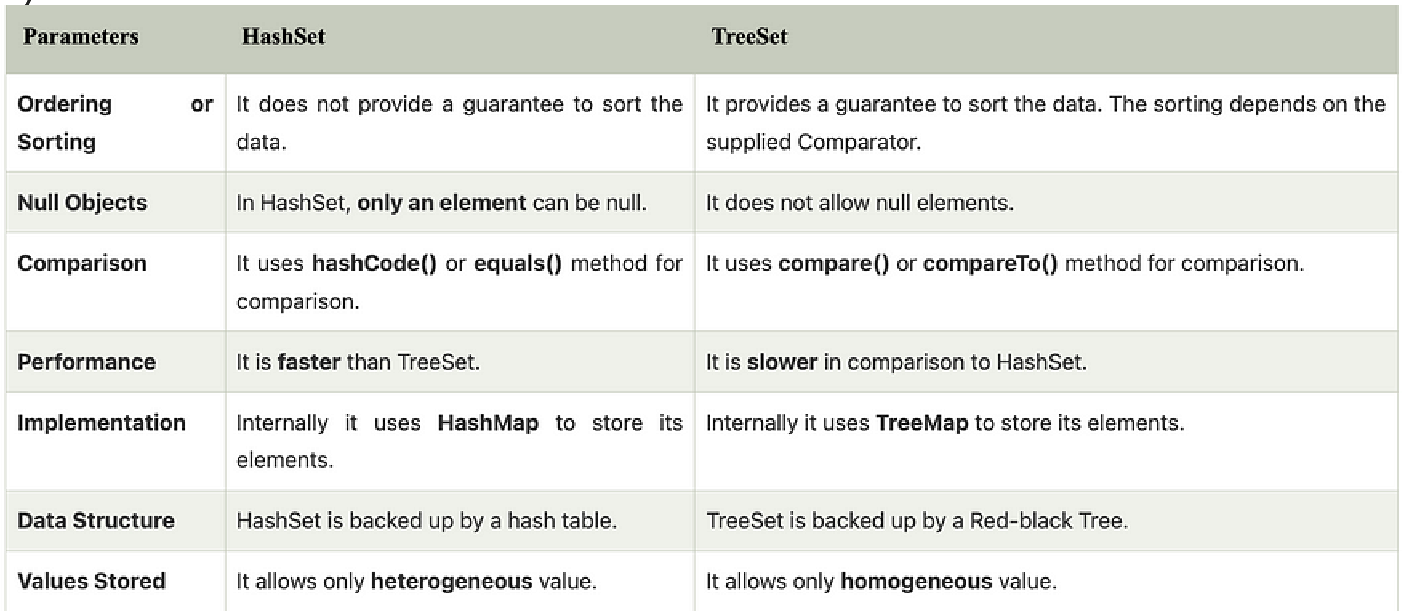
ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

22) What is the difference between List and Set?



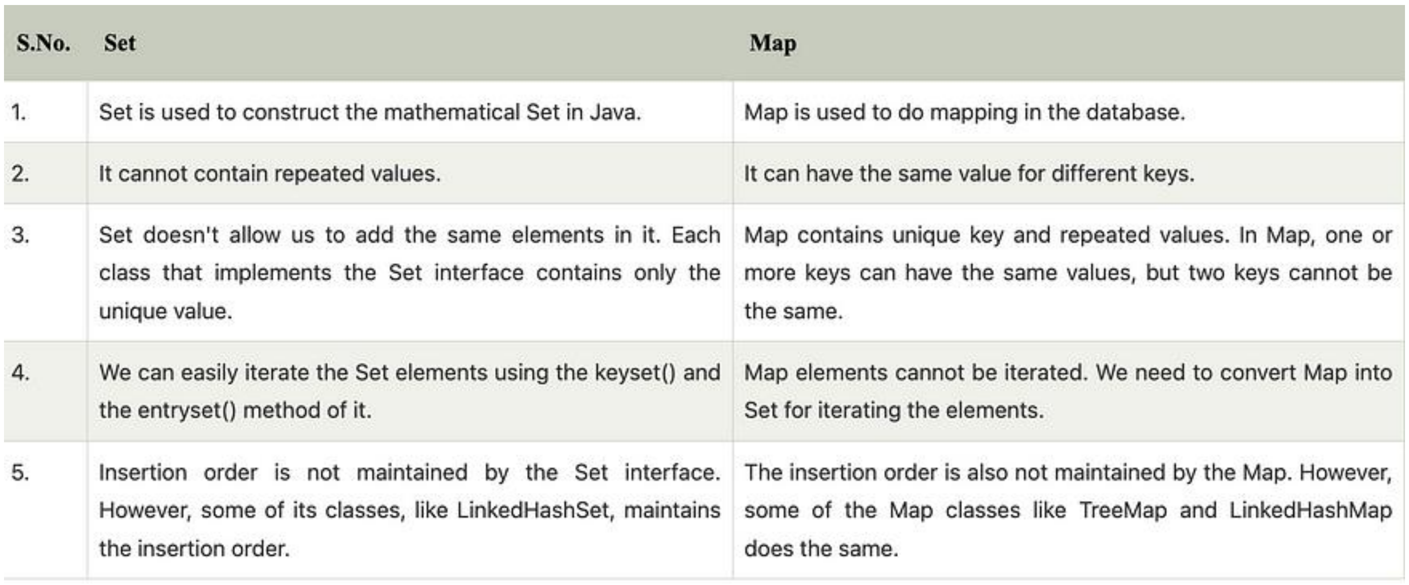
S.No	List	Set
1.	The list implementation allows us to add the same or duplicate elements.	The set implementation doesn't allow us to add the same or duplicate elements.
2.	The insertion order is maintained by the List.	It doesn't maintain the insertion order of elements.
3.	List allows us to add any number of null values.	Set allows us to add at least one null value in it.
4.	The List implementation classes are LinkedList and ArrayList.	The Set implementation classes are TreeSet, HashSet and LinkedHashSet.
5.	We can get the element of a specified index from the list using the get() method.	We cannot find the element from the Set based on the index because it doesn't provide any get method().
6.	It is used when we want to frequently access the elements by using the index.	It is used when we want to design a collection of distinct elements.
7.	The method of List interface listiterator() is used to iterate the List elements.	The iterator is used when we need to iterate the Set elements.

23) What is the difference between HashSet and TreeSet?



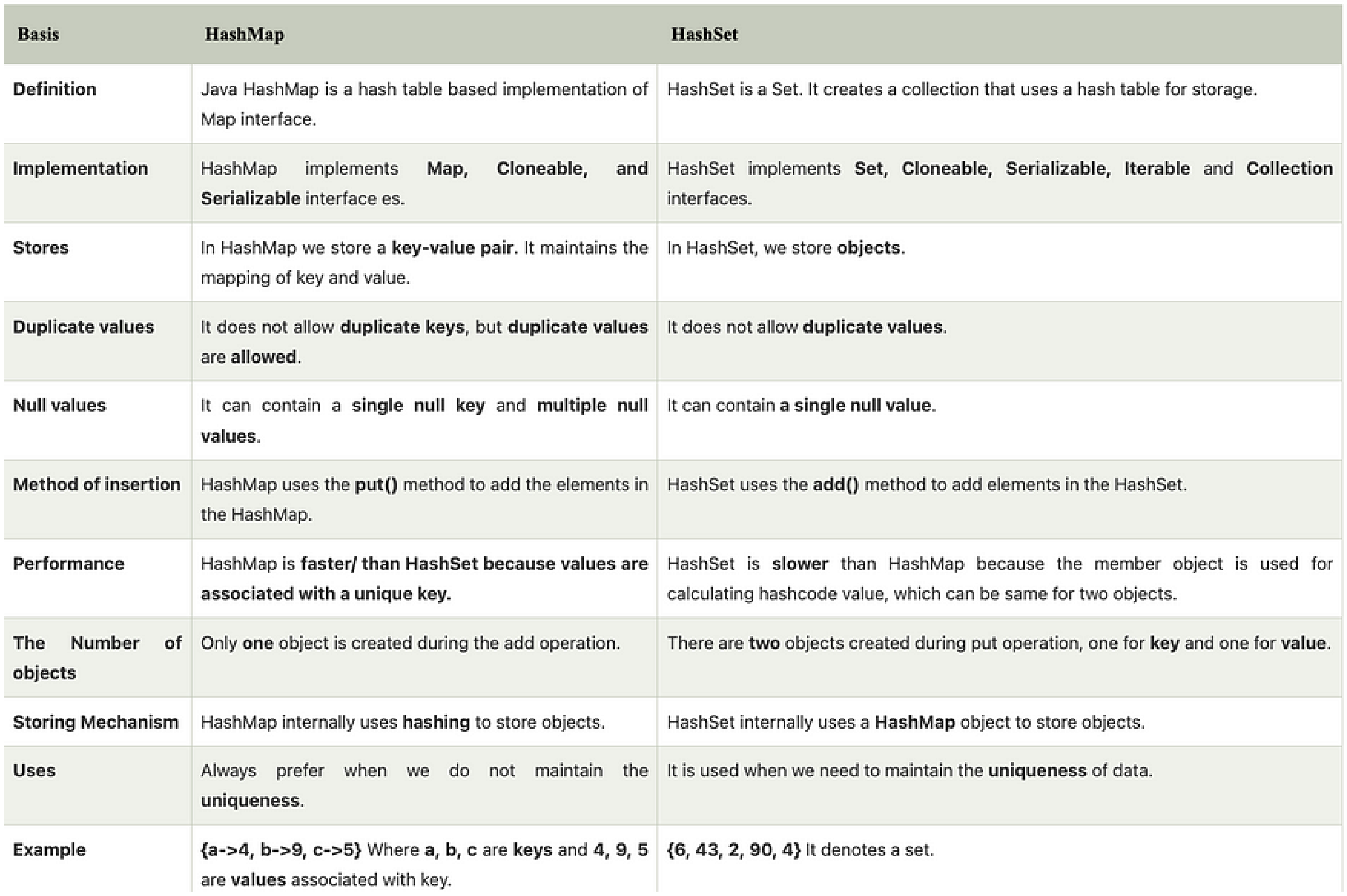
Parameters	HashSet	TreeSet
Ordering or Sorting	It does not provide a guarantee to sort the data.	It provides a guarantee to sort the data. The sorting depends on the supplied Comparator.
Null Objects	In HashSet, **only an element** can be null.	It does not allow null elements.
Comparison	It uses **hashCode()** or **equals()** method for comparison.	It uses **compare()** or **compareTo()** method for comparison.
Performance	It is **faster** than TreeSet.	It is **slower** in comparison to HashSet.
Implementation	Internally it uses **HashMap** to store its elements.	Internally it uses **TreeMap** to store its elements.
Data Structure	HashSet is backed up by a hash table.	TreeSet is backed up by a Red-black Tree.
Values Stored	It allows only **heterogeneous** value.	It allows only **homogeneous** value.

24) What is the difference between Set and Map?



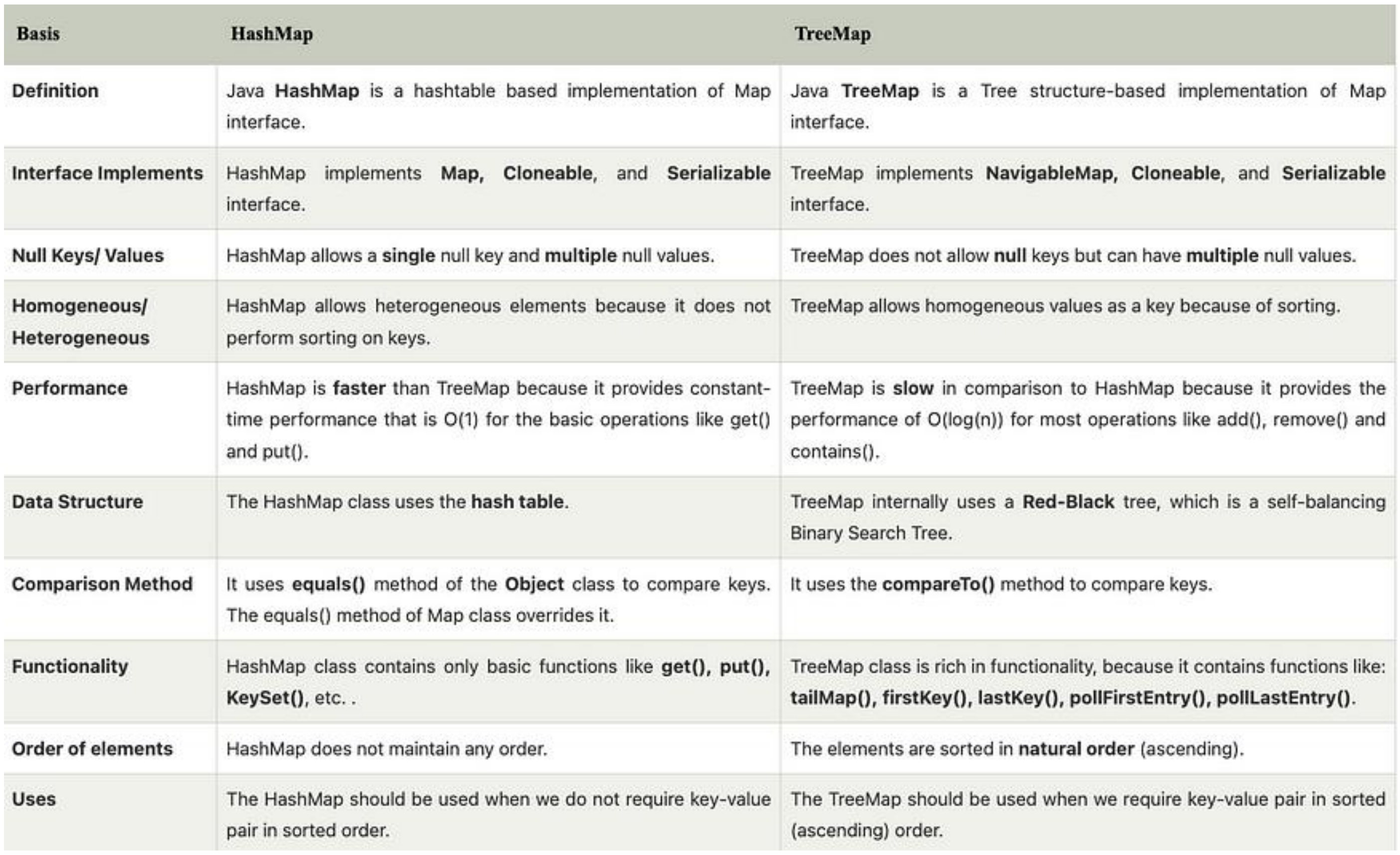
S.No.	Set	Map
1.	Set is used to construct the mathematical Set in Java.	Map is used to do mapping in the database.
2.	It cannot contain repeated values.	It can have the same value for different keys.
3.	Set doesn't allow us to add the same elements in it. Each class that implements the Set interface contains only the unique value.	Map contains unique key and repeated values. In Map, one or more keys can have the same values, but two keys cannot be the same.
4.	We can easily iterate the Set elements using the keyset() and the entryset() method of it.	Map elements cannot be iterated. We need to convert Map into Set for iterating the elements.
5.	Insertion order is not maintained by the Set interface. However, some of its classes, like LinkedHashSet, maintains the insertion order.	The insertion order is also not maintained by the Map. However, some of the Map classes like TreeMap and LinkedHashMap does the same.

25) What is the difference between HashSet and HashMap?



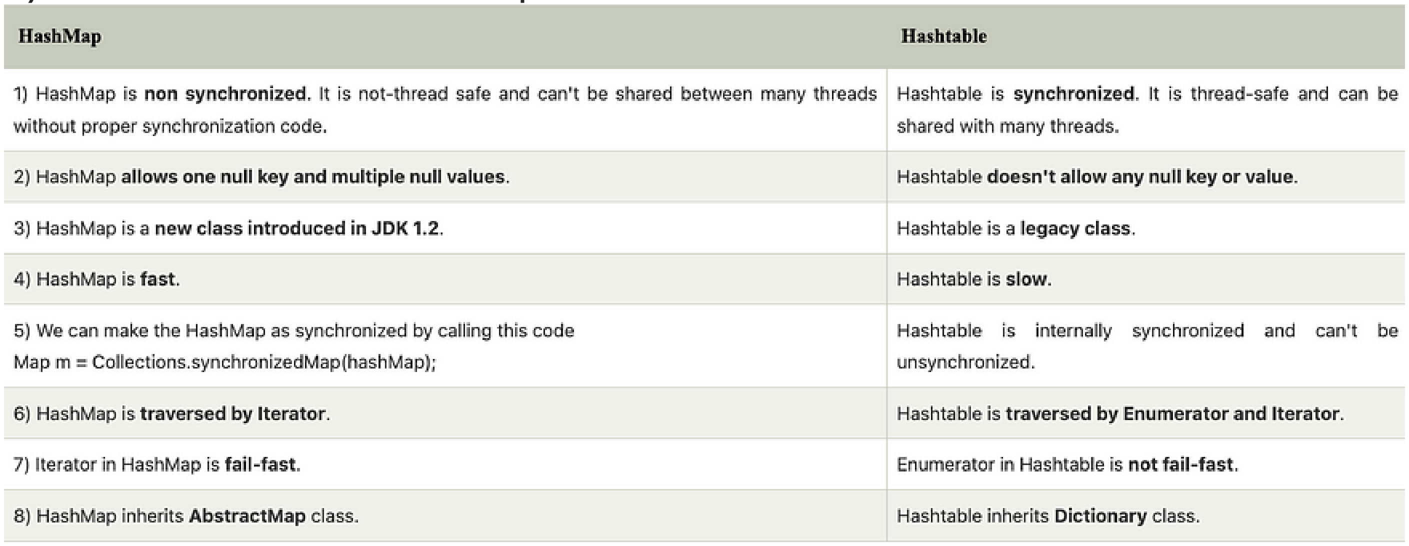
Basis	HashMap	HashSet
Definition	Java HashMap is a hash table based implementation of Map interface.	HashSet is a Set. It creates a collection that uses a hash table for storage.
Implementation	HashMap implements **Map, Cloneable, and Serializable** interface es.	HashSet implements **Set, Cloneable, Serializable, Iterable** and **Collection** interfaces.
Stores	In HashMap we store a **key-value pair.** It maintains the mapping of key and value.	In HashSet, we store **objects.**
Duplicate values	It does not allow **duplicate keys**, but **duplicate values** are **allowed**.	It does not allow **duplicate values.**
Null values	It can contain a **single null key** and **multiple null values.**	It can contain **a single null value.**
Method of insertion	HashMap uses the **put()** method to add the elements in the HashMap.	HashSet uses the **add()** method to add elements in the HashSet.
Performance	HashMap is **faster/ than HashSet because values are associated with a unique key.**	HashSet is **slower** than HashMap because the member object is used for calculating hashcode value, which can be same for two objects.
The Number of objects	Only **one** object is created during the add operation.	There are **two** objects created during put operation, one for **key** and one for **value**.
Storing Mechanism	HashMap internally uses **hashing** to store objects.	HashSet internally uses a **HashMap** object to store objects.
Uses	Always prefer when we do not maintain the **uniqueness**.	It is used when we need to maintain the **uniqueness** of data.
Example	**{a->4, b->9, c->5}** Where **a, b, c** are **keys** and **4, 9, 5** are **values** associated with key.	**{6, 43, 2, 90, 4}** It denotes a set.

26) What is the difference between HashMap and TreeMap?



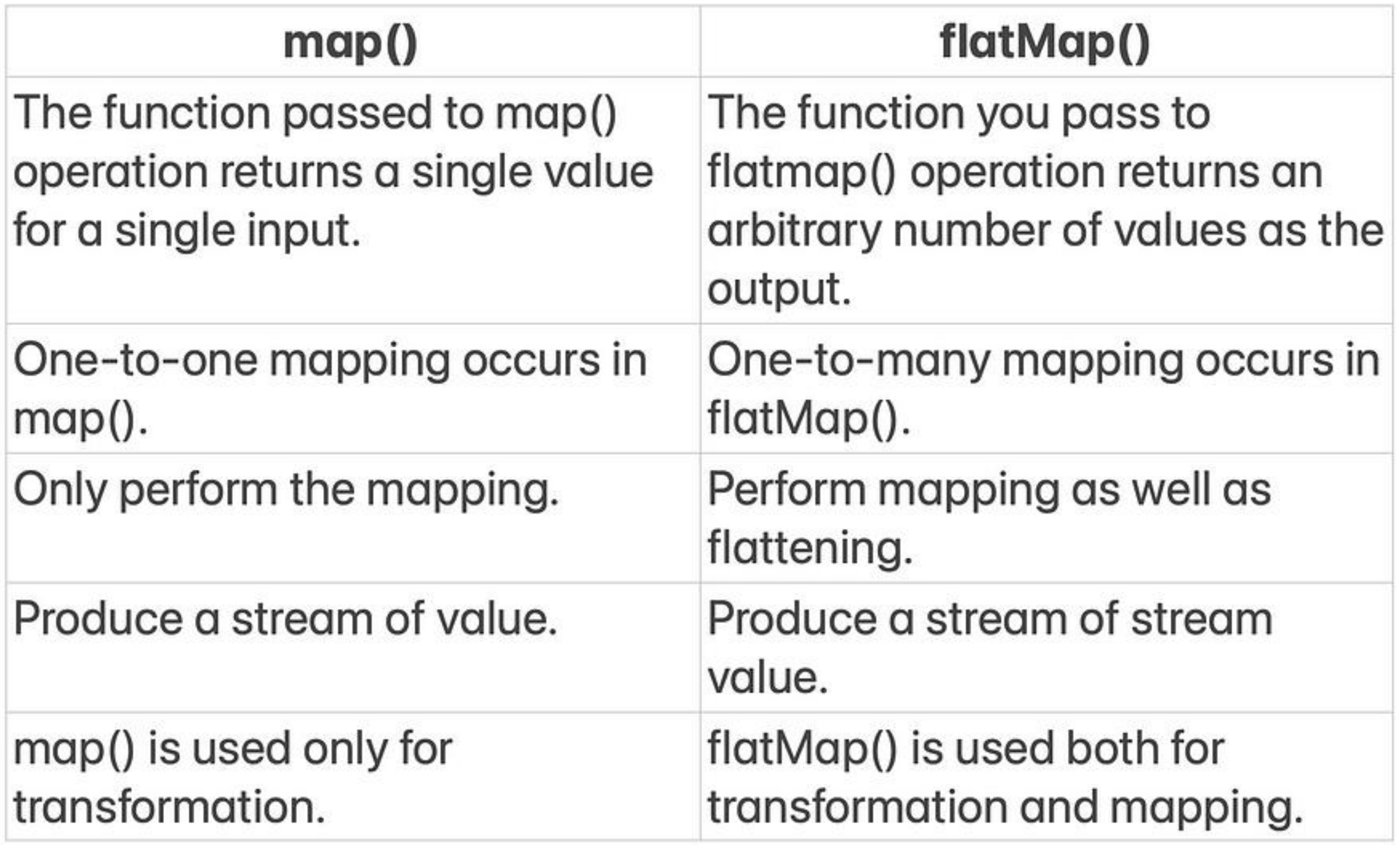
Basis	HashMap	TreeMap
Definition	Java **HashMap** is a hashtable based implementation of Map interface.	Java **TreeMap** is a Tree structure-based implementation of Map interface.
Interface Implements	HashMap implements **Map, Cloneable**, and **Serializable** interface.	TreeMap implements **NavigableMap, Cloneable**, and **Serializable** interface.
Null Keys/ Values	HashMap allows a **single** null key and **multiple** null values.	TreeMap does not allow **null** keys but can have **multiple** null values.
Homogeneous/ Heterogeneous	HashMap allows heterogeneous elements because it does not perform sorting on keys.	TreeMap allows homogeneous values as a key because of sorting.
Performance	HashMap is **faster** than TreeMap because it provides constant-time performance that is O(1) for the basic operations like get() and put().	TreeMap is **slow** in comparison to HashMap because it provides the performance of O(log(n)) for most operations like add(), remove() and contains().
Data Structure	The HashMap class uses the **hash table**.	TreeMap internally uses a **Red-Black** tree, which is a self-balancing Binary Search Tree.
Comparison Method	It uses **equals()** method of the **Object** class to compare keys. The equals() method of Map class overrides it.	It uses the **compareTo()** method to compare keys.
Functionality	HashMap class contains only basic functions like **get(), put(), KeySet()**, etc. .	TreeMap class is rich in functionality, because it contains functions like: **tailMap(), firstKey(), lastKey(), pollFirstEntry(), pollLastEntry()**.
Order of elements	HashMap does not maintain any order.	The elements are sorted in **natural order** (ascending).
Uses	The HashMap should be used when we do not require key-value pair in sorted order.	The TreeMap should be used when we require key-value pair in sorted (ascending) order.

27) What is the difference between HashMap and Hashtable?



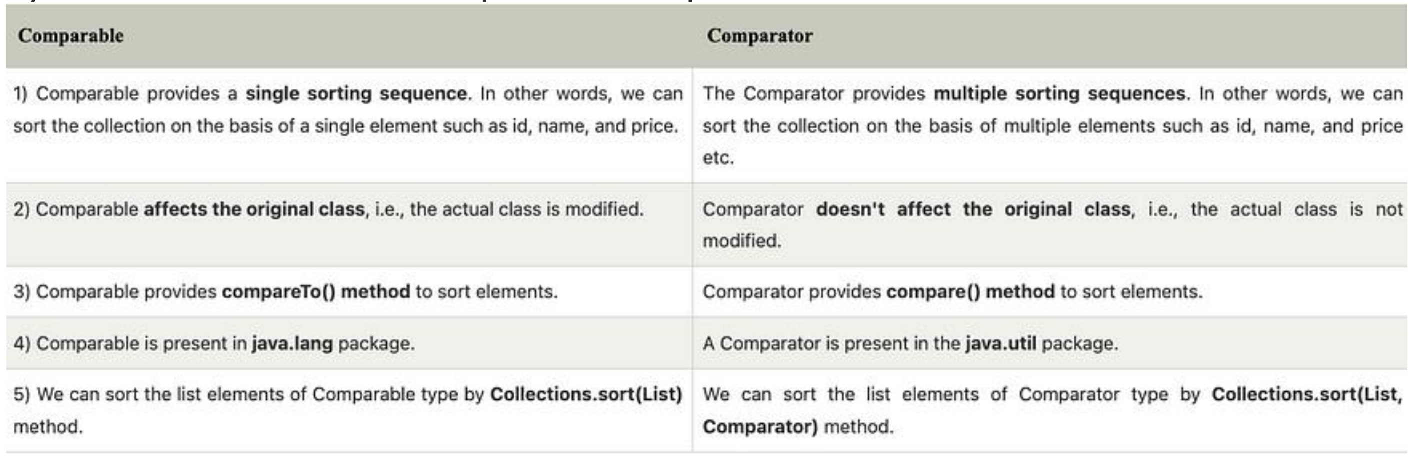
HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

28) What is the difference between Map and FlatMap?



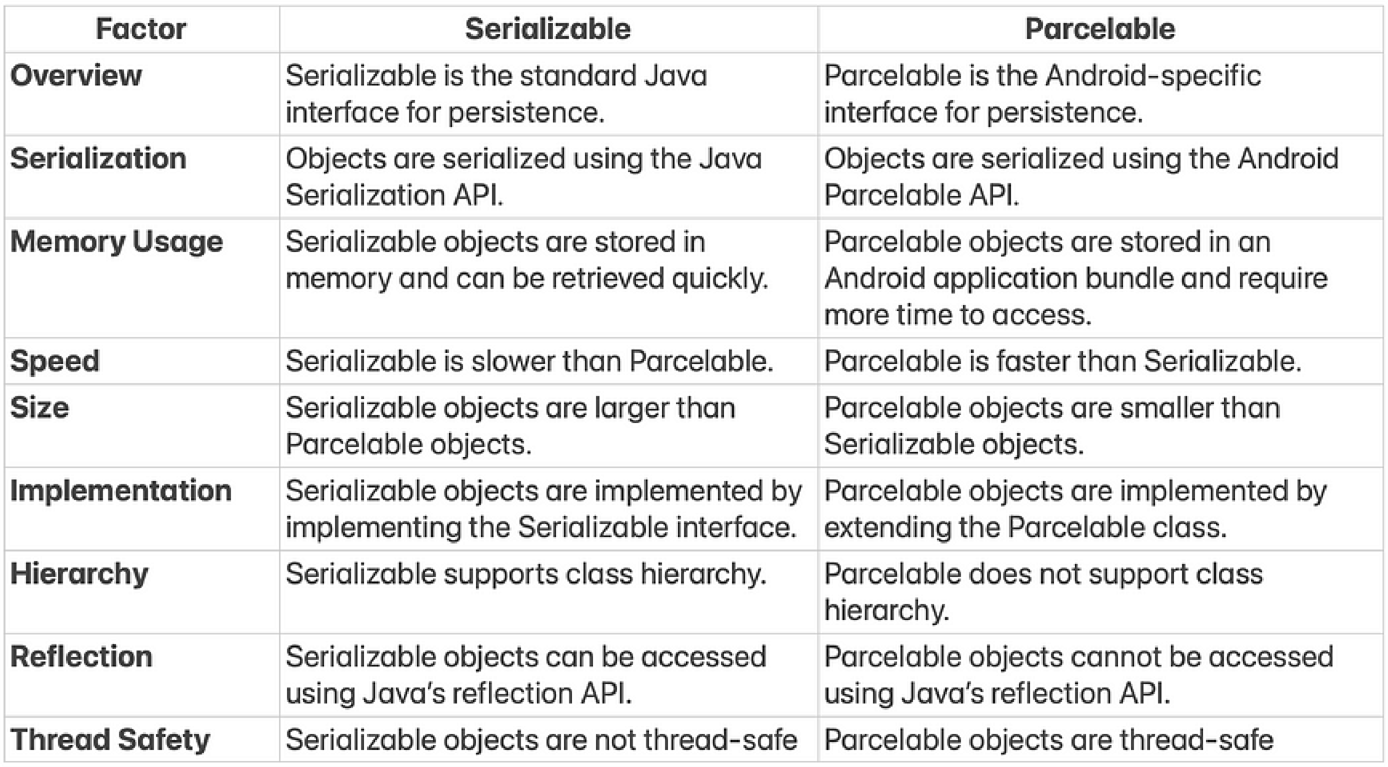
map()	flatMap()
The function passed to map() operation returns a single value for a single input.	The function you pass to flatmap() operation returns an arbitrary number of values as the output.
One-to-one mapping occurs in map().	One-to-many mapping occurs in flatMap().
Only perform the mapping.	Perform mapping as well as flattening.
Produce a stream of value.	Produce a stream of stream value.
map() is used only for transformation.	flatMap() is used both for transformation and mapping.

29) What is the difference between Comparable and Comparator?



Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

30) What is the difference between Serializable and Parcelable?



Factor	Serializable	Parcelable
Overview	Serializable is the standard Java interface for persistence.	Parcelable is the Android-specific interface for persistence.
Serialization	Objects are serialized using the Java Serialization API.	Objects are serialized using the Android Parcelable API.
Memory Usage	Serializable objects are stored in memory and can be retrieved quickly.	Parcelable objects are stored in an Android application bundle and require more time to access.
Speed	Serializable is slower than Parcelable.	Parcelable is faster than Serializable.
Size	Serializable objects are larger than Parcelable objects.	Parcelable objects are smaller than Serializable objects.
Implementation	Serializable objects are implemented by implementing the Serializable interface.	Parcelable objects are implemented by extending the Parcelable class.
Hierarchy	Serializable supports class hierarchy.	Parcelable does not support class hierarchy.
Reflection	Serializable objects can be accessed using Java's reflection API.	Parcelable objects cannot be accessed using Java's reflection API.
Thread Safety	Serializable objects are not thread-safe	Parcelable objects are thread-safe

31) What are anonymous classes?

An anonymous class is just what its name implies — it has no name. It combines the class declaration and the creation of an instance of the class in one step.

Example:

```
MyButton.setOnClickListener(new Button.OnClickListener
{
@override
public void onClick(View view){
//some code
}});
```

32) What is the reflection?

Reflection is the process of examining or modifying the runtime behavior of a class at runtime. The java.lang.Class class provides various methods that can be used to get metadata, examine and change the runtime behavior of a class. The java.lang and java.lang.reflect packages provide classes for Java reflection. It is used in:

- IDE (Integrated Development Environment), e.g., Eclipse, MyEclipse, NetBeans.
- Debugger

- Test Tools, etc.

33) What is a singleton class in Java?

A singleton class is a class that can not be instantiated more than once. To make a class singleton, we either make its constructor private or use the static getInstance method

To create a singleton class, a class must implement the following properties:

- Create a private constructor of the class to restrict object creation outside of the class.
- Create a private attribute of the class type that refers to a single object.
- Create a public static method that allows us to create and access the object we created. Inside the method, we will create a condition that restricts us from creating more than one object.

```
class SingletonExample {

   // private field that refers to the object
   private static SingletonExample singleObject;

   private SingletonExample() {
      // constructor of the SingletonExample class
   }

   public static SingletonExample getInstance() {
      // write code that allows us to create only one object
      // access the object as per our need
   }
}
```

34) Difference between == and .equals() method in java?

== operators for reference comparison (address comparison)

.equals() method for content comparison.

** In simple words, == checks if both objects point to the same memory location whereas .**equals()** evaluates to the comparison of values in the objects.*

35) What is Java Threads?

A thread in Java is the direction or path that is taken while a program is being executed.

The Concept Of Multitasking

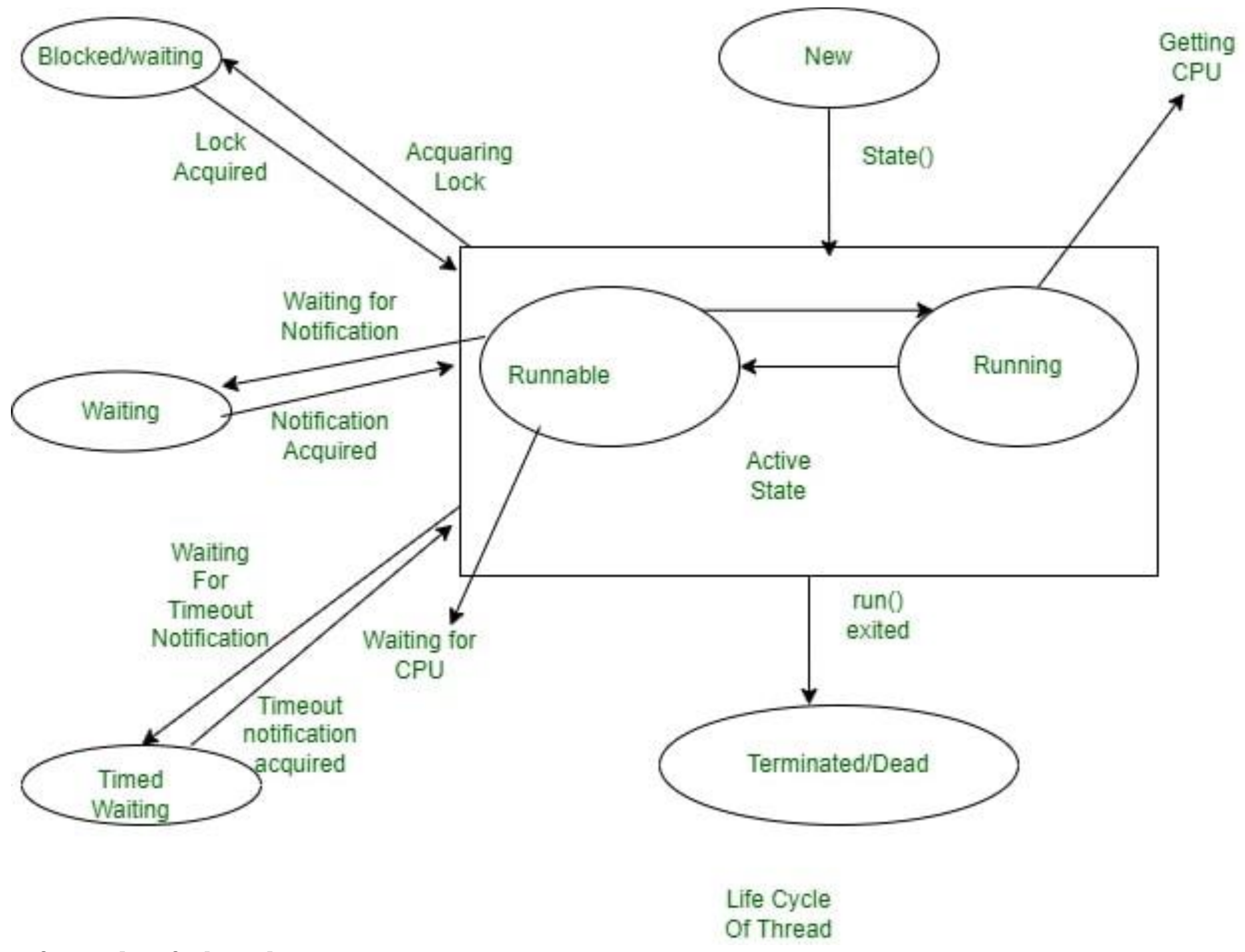
To help users Operating System accommodates users with the privilege of multitasking, where users can perform multiple actions simultaneously on the machine. This Multitasking can be enabled in two ways:

1. **Process-Based Multitasking:** In this type of Multitasking, processes are heavyweight and each process was allocated by a separate memory area. And as the process is heavyweight the cost of communication between processes is high and it takes a long time for switching between processes as it involves actions such as loading, saving in registers, updating maps, lists, etc.
2. **Thread-Based Multitasking:** Threads are provided with lightweight nature and share the same address space, and the cost of communication between threads is also low.

Life Cycle Of Thread

There are different states a Thread transfers into during its lifetime, let us know about those states in the following lines: in its lifetime, a thread undergoes the following states, namely:

1. New State
2. Active State
3. Waiting/Blocked State
4. Timed Waiting State
5. Terminated State



Life Cycle of Threads

1. New State

By default, a Thread will be in a new state, in this state, code has not yet been run and the execution process is not yet initiated.

2. Active State

A Thread that is a new state by default gets transferred to an Active state when it invokes the start() method, his Active state contains two sub-states namely:

- **Runnable State:** In This State, The Thread is ready to run at any given time and it's the job of the Thread Scheduler to provide the thread time for the runnable state preserved threads. A program that has obtained Multithreading shares slices of time intervals which are shared between threads hence, these threads run for some short span of time and wait in the runnable state to get their schedules slice of a time interval.

- **Running State:** When The Thread Receives CPU allocated by Thread Scheduler, it transfers from the "Runnable" state to the "Running" state. and after the expiry of its given time slice session, it again moves back to the "Runnable" state and waits for its next time slice.

3. Waiting/Blocked State

If a Thread is inactive but on a temporary time, then either it is waiting or blocked state, for example, if there are two threads, T1 and T2 where T1 needs to communicate to the camera and the other thread T2 already using a camera to scan then T1 waits until T2 Thread completes its work, at this state T1 is parked in waiting for the state, and in another scenario, the user called two Threads T2 and T3 with the same functionality and both had same time slice given by Thread Scheduler then both Threads T1, T2 is in a blocked state. When there are multiple threads parked in a Blocked/Waiting state Thread Scheduler clears Queue by rejecting unwanted Threads and allocating CPU on a priority basis.

4. Timed Waiting State

Sometimes the longer duration of waiting for threads causes starvation, if we take an example like there are two threads T1, T2 waiting for the CPU and T1 is undergoing a Critical Coding operation, and if it does not exist the CPU until its operation gets executed then T2 will be exposed to longer waiting with undetermined certainty, In order to avoid this starvation situation, we had Timed Waiting for the state to avoid that kind of scenario as in Timed Waiting, each thread has a time period for which sleep() method is invoked and after the time expires the Threads starts executing its task.

5. Terminated State

A thread will be in Terminated State, due to the below reasons:

- Termination is achieved by a Thread when it finishes its task Normally.
- Sometimes Threads may be terminated due to unusual events like segmentation faults, exceptions…etc. and such kind of Termination can be called Abnormal Termination.
- A terminated Thread means it is dead and no longer available.