Assignment Questions 6

Q1.What is Collection in Java?

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Q2. Differentiate between Collection and collections in the context of Java.

- Collection is called interface in java whereas Collections is called a utility class in java and both of them can be found in java.util.package.
- Collection is used to represent a single unit with a group of individual objects whereas collections is used to operate on collection with several utility methods.
- Since java 8, collection is an interface with static as well as abstract and default methods whereas collections operate only with static methods.

Q3. What are the advantages of the Collection framework?

The Java Collections Framework provides the following benefits:

- **Reduces programming effort:** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.
- **Increases program speed and quality:** This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.

- **Allows interoperability among unrelated APIs:** The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.
- **Reduces effort to learn and to use new APIs:** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.
- **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
- **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

Q4.Explain the various interfaces used in the Collection framework.

The core collection interfaces within the Java Collection framework are as follows:

- **List:** The List interface extends the Collection interface and represents an ordered collection of elements. Lists allow duplicate elements and maintain the insertion order. Common implementations of List include ArrayList, LinkedList, and Vector.
- **Set:** The Set interface, also an extension of the Collection interface, represents a collection that does not allow duplicate elements. Sets typically do not maintain a specific order of elements. Notable implementations of Set are HashSet, TreeSet, and LinkedHashSet.
- **Queue:** The Queue interface defines a collection that represents a waiting area, where elements are inserted at one end and removed from the other. Queues follow the First-In-First-Out (FIFO) principle. Notable implementations of Queue include LinkedList and PriorityQueue.
- **Deque:** The Deque interface extends the Queue interface and represents a double-ended queue, allowing elements to be inserted and removed from both ends. Deques support operations at both ends, enabling flexibility in

data handling. Common implementations of Deque include ArrayDeque and LinkedList.

- **Map:** The Map interface represents a mapping between unique keys and corresponding values. It does not extend the Collection interface but is an important part of the Java Collection framework. Maps do not allow duplicate keys and are commonly used for key-value pair associations. Notable implementations of Map include HashMap, TreeMap, and LinkedHashMap.

Q5. Differentiate between List and Set in Java.

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.

Q6.What is the Differentiate between Iterator and ListIterator in Java.

Iterator	ListIterator
Can traverse elements present in Collection only in the forward direction.	Can traverse elements present in Collection both in forward and backward directions.
Helps to traverse Map, List and Set.	Can only traverse List and not the other two.
Indexes cannot be obtained by using Iterator.	It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List.
Cannot modify or replace elements present in Collection	We can modify or replace elements with the help of set(E e)
Cannot add elements and it throws ConcurrentModificationException.	Can easily add elements to a collection at any time.
Certain methods of Iterator are next(), remove() and hasNext().	Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e).

Q7.What is the Differentiate 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.

Q8.What is collision in HashMap?

A collision, or more specifically, a hash code collision in a *HashMap*, is a situation where **two or more key objects produce the same final hash value** and hence point to the same bucket location or array index.
This scenario can occur because according to the *equals* and *hashCode* contract, **two unequal objects in Java can have the same hash code**.
It can also happen because of the finite size of the underlying array, that is, before resizing. The smaller this array, the higher the chances of collision.

Q9.Distinguish between a hashmap and a Treemap.

The following table describes the differences 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.

Q10.Define LinkedHashMap in Java

The **LinkedHashMap Class** is just like HashMap with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion, which the LinkedHashMap provides where the elements can be accessed in their insertion order.