# **Assignment 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 vs Collections:**

|  |  |
| --- | --- |
| **Collection** | **Collections** |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods |

Q3. What are the advantages of the Collection framework?

### **Main benefits of Collections Framework in Java**

**I. Reusability:** Java Collections Framework provides common classes and utility methods than can be used with different types of collections. This promotes the reusability of the code. A developer does not have to re-invent the wheel by writing the same method again.

**II. Quality:** Using Java Collection Framework improves the program quality, since the code is already tested and used by thousands of developers.

**III. Speed:** mostprogrammers report that their development speed has increased since they can focus on core logic and use the generic collections provided by Java framework.

**IV. Maintenance:** Since most of the Java Collections framework code is open source and API documents is widely available, it is easy to maintain the code written with the help of Java Collections framework. One developer can easily pick the code of the previous developer.

**V. 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.

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

The Java Collection framework offers a comprehensive set of interfaces and class implementations that facilitate efficient and meaningful data manipulation

While Java does not provide a direct implementation of the Collection framework, the Collection Interface itself is implemented by two important classes: List and Set.

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.

**Difference between List and Set:**

|  |  |
| --- | --- |
| **List** | **Set** |
| 1. The List is an indexed sequence. | 1. The Set is an non-indexed sequence. |
| 2. List allows duplicate elements | 2. Set doesn’t allow duplicate elements. |
| 3. Elements by their position can be accessed. | 3. Position access to elements is not allowed. |
| 4. Multiple null elements can be stored. | 4. Null element can store only once. |
| 5. List implementations are ArrayList, LinkedList, Vector, Stack | 5. Set implementations are HashSet, LinkedHashSet. |

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

An **Iterator** is an interface in Java, and we can traverse the elements of a list in a **forward direction** whereas a **ListIterator** is an interface that extends the **Iterator** interface and we can traverse the elements in **both forward and backward directions.**

**Difference between Iterator and ListIterator**

|  |  |
| --- | --- |
| **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 and Comparator both are interfaces and can be used to sort collection elements.

|  |  |
| --- | --- |
| **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?

The HashMap key contains a hashcode, and a equals() method. Whenever we insert a new entry to the Map, it checks for the hashcode. It parses through the entire pool of objects, searching for similarity of the hashcode using the equals() method.

If any entry is existent, the new value will then replace the primarily existing value. Otherwise, it will simply create a whole new key-value pair.

Collision happens when multiple keys hash to the same bucket or, say when two or more objects have the same hashcode but are different. When two keys get hashed to the same value, a linked list is formed at the bucket location, where all the information is stored as an entry of the map, which contains the key-value pair.

Accessing any object could turn out to be cumbersome if the entries are present inside the lists. These linked lists were converted to binary trees from Java 8 version.

HashMap handles collision cases very efficiently using a concept known as chaining, which suggests storing the values in a linked list or a binary tree as indicated by the conversion of methodology from Java 8.

Q9. Distinguish between a hashmap and a Treemap.

## HashMap

HashMap implements **Map<K, V>, Cloneable** and **Serializable** interface. It extends **AbstractMap<K, V>** class. It belongs to **java.util** package.

* HashMap contains value based on the key.
* It may have a single null key and multiple null values.
* HashMap does not maintain order while iterating.
* It contains unique elements.
* It works on the principle of hashing.

## TreeMap

TreeMap class extends **AbstractMap<K, V>** class and implements **NavigableMap<K, V >**, **Cloneable**, and **Serializable** interface. TreeMap is an example of a **SortedMap**. It is implemented by the Red-Black tree, which means that the order of the keys is sorted.

* TreeMap also contains value based on the key.
* TreeMap is sorted by keys.
* It contains unique elements.
* It cannot have a null key but have multiple null values.
* Keys are in ascending order.
* It stores the object in the tree structure.

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.

**Important Features of a LinkedHashMap are listed as follows:**

* A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is non-synchronized.
* It is the same as HashMap with an additional feature that it maintains insertion order. For example, when we run the code with a HashMap, we get a different order of elements.

**Declaration:**

public class LinkedHashMap<K,​V> extends HashMap<K,​V> implements Map<K,​V>

* **K** – The type of the keys in the map.
* **V** – The type of values mapped in the map.

### How LinkedHashMap Work Internally?

A LinkedHashMap is an extension of the **HashMap** class and it implements the **Map** interface. Therefore, the class is declared as:

public class LinkedHashMap   
extends HashMap   
implements Map

The data is stored in the form of nodes. The implementation of the LinkedHashMap is very similar to a doubly-linked list. Therefore, each node of the LinkedHashMap is represented as:

* **Hash:** All the input keys are converted into a hash which is a shorter form of the key so that the search and insertion are faster.
* **Key:** Since this class extends HashMap, the data is stored in the form of a key-value pair. Therefore, this parameter is the key to the data.
* **Value:** For every key, there is a value associated with it. This parameter stores the value of the keys. Due to generics, this value can be of any form.
* **Next:** Since the LinkedHashMap stores the insertion order, this contains the address to the next node of the LinkedHashMap.
* **Previous:** This parameter contains the address to the previous node of the LinkedHashMap.