Q1. What is Collection in Java?

Answer: 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.

Answer: Collection: Collection is a interface present in java.util.package. It is used to represent a group of individual objects as a single unit. It is similar to the container in the C++ language. The collection is considered as the root interface of the collection framework. It provides several classes and interfaces to represent a group of individual objects as a single unit.
The List , Set , and Queue are the main sub-interfaces of the collection interface. The map interface is also part of the java collection framework, but it doesn't inherit the collection of the interface. The **add ()**, **remove()**, **clear()**, **size()**, and **contains()** are the important methods of the Collection interface.
Declaration:
public interface Collection<E> extends Iterable<E>

Type Parameters: E - the type of elements returned by this iterator.

Collections: Collections is a utility class present in java.util.package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods. These methods provide much-needed convenience to developers, allowing them to effectively work with Collection Framework. For example, It has a method *sort()* to sort the collection elements according to default sorting order, and it has a method *min()*, and *max()* to find the minimum and maximum value respectively in the collection elements.
Declaration:
public class Collections extends Object

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?

Answer:

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: Most of programmer's report that their development speed 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 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.

Answer: 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.

Answer: The **List interface** allows storing the ordered collection. It is a child interface of Collection. It is an ordered collection of objects in which duplicate values are allowed to store. List preserves the insertion order, it allows positional access and insertion of elements.
Declaration:
public abstract interface List extends Collection.

The **Set interface** in the java.util package and extends Collection interface is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the maths set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts to insert the duplicate elements.
Declaration: The Set interface is declared as:
public interface Set extends Collection.

Example:
Input : Add Elements = [1, 2, 3, 1]
Output: Set = [1, 2, 3]
List = [1, 2, 3, 1]

Input : Add Elements = [a, b, d, b]
Output: Set = [a, b, d]

List = [a, b, d, b]

Below is the illustration of Set and List :

```
// Implementation of List and Set in Java
import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        // List declaration
        List<Integer> l = new ArrayList<>();
        l.add(5);
        l.add(6);
        l.add(3);
        l.add(5);
        l.add(4);

        // Set declaration
        Set<Integer> s = new HashSet<>();
        s.add(5);
        s.add(6);
        s.add(3);
        s.add(5);
        s.add(4);

        // printing list
        System.out.println("List = " + l);
        // printing Set
        System.out.println("Set = " + s);
    }
}
```

Output

List = [5, 6, 3, 5, 4]

Set = [3, 4, 5, 6]

Difference between List and Set:

List	Set
1. The List is an indexed sequence.	1. The Set is an non-indexed sequence.

List	Set
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.

Answer: Iterators are used in Collection framework in Java to retrieve elements one by one. It can be applied to any Collection object. By using Iterator, we can perform both read and remove operations. Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List ,Queue ,Deque and also in all implemented classes of Map interface. Iterator is the only cursor available for entire collection framework. Iterator object can be created by calling iterator () method present in Collection interface.

```
// Here "c" is any Collection object. itr is of

// type Iterator interface and refers to "c"

Iterator itr = c.iterator();
```

ListIterator It is only applicable for List collection implemented classes like arraylist , linkedlist etc. It provides bi-directional iteration. ListIterator must be used when we want to enumerate elements of List. This cursor has more functionality(methods) than iterator. ListIterator object can be created by calling listIterator() method present in List interface.

```
// Here "l" is any List object, ltr is of type

// ListIterator interface and refers to "l"

ListIterator ltr = l.listIterator();
```

Differences between Iterator and ListIterator:

1.Iterator can traverse only in forward direction whereas ListIterator traverses both in forward and backward directions.

Example:

```
import java.io.*;
import java.util.*;

class IteratorDemo1 {
    public static void main(String[] args)
    {
        ArrayList<Integer> list
            = new ArrayList<Integer>();

        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);

        // Iterator
        Iterator itr = list.iterator();
```

```
        System.out.println("Iterator:");
        System.out.println("Forward traversal: ");

        while (itr.hasNext())
           System.out.print(itr.next() + " ");

        System.out.println();

        // ListIterator
        ListIterator i = list.listIterator();

        System.out.println("ListIterator:");
        System.out.println("Forward Traversal : ");

        while (i.hasNext())
           System.out.print(i.next() + " ");

        System.out.println();

        System.out.println("Backward Traversal : ");

        while (i.hasPrevious())
           System.out.print(i.previous() + " ");

        System.out.println();
     }
  }
```

Output:

Iterator:

Forward traversal:

1 2 3 4 5

ListIterator:

Forward Traversal :

1 2 3 4 5

Backward Traversal :

5 4 3 2 1

2.ListIterator can help to replace an element whereas Iterator cannot.

Example:

```
import java.util.ArrayList;
import java.util.ListIterator;

public class ListIteratorDemo2 {
   public static void main(String[] args)
   {

      ArrayList<Integer> aList
         = new ArrayList<Integer>();
      aList.add(1);
      aList.add(2);
      aList.add(3);
      aList.add(4);
      aList.add(5);
```

```
        System.out.println("Elements of ArrayList: ");
        for (Integer i : aList) {
            System.out.println(i);
        }
        ListIterator<Integer> l
            = aList.listIterator();
        l.next();
        l.set(80000);

        System.out.println("\nNow the ArrayList"
                           + " elements are: ");
        for (Integer i : aList) {
            System.out.println(i);
        }
    }
     }
```

Output:

Elements of ArrayList:

1

2

3

4

5

Now the ArrayList elements are:

80000

2

3

4

5

OUTPUT Table showing 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)

Iterator	ListIterator
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

Answer: Difference between Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

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?

Answer: In the context of a HashMap, collision occurs when two or more keys in the map are assigned to the same index in the underlying array structure. Each key-value pair in a HashMap is stored in a bucket, and the bucket is determined by applying a hash function to the key. The resulting hash code is used to compute the index of the array where the key-value pair will be stored.

Ideally, each key should have a unique hash code and be assigned to a different index in the array. However, due to various factors, such as the finite size of the array and the nature of the hash function, collisions can occur. When two or more keys produce the same hash code, they are said to have a collision.

HashMaps typically use a technique called chaining to handle collisions. Chaining means that each bucket in the array can store multiple key-value pairs. When a collision occurs, the HashMap appends the new key-value pair to the existing pairs in the same bucket. This creates a linked list or a similar data structure within each bucket, allowing multiple values to be associated with the same index.

When retrieving a value from a HashMap, the key is hashed again to determine the bucket, and then a search is performed within that bucket to find the desired key-value pair. This search involves comparing the keys of the stored pairs with the provided key until a match is found or the end of the linked list is reached.

Efficient collision resolution strategies, such as maintaining an average load factor (the ratio of stored elements to the capacity of the HashMap) and resizing the underlying array when needed, are employed to minimize the likelihood of collisions and maintain good performance characteristics.