Q. 1)What is LinkedList?

- Data (actual value)
- Address (reference) of the next node
- In doubly linked list → also reference to previous node
- Java's `LinkedList` is doubly linked.
- Implements **List**, **Deque**, and **Queue**
- Maintains insertion order
- Allows duplicate elements
- Allows null values
- Dynamic size (no fixed capacity)
- Better for frequent insertions and deletions
- You frequently add/remove elements
- You don't need fast random access
- You need queue or deque operations

Q .2) Difference between ArrayList and LinkedList.

ArrayList

- Based on dynamic array
- Internally uses resizable array
- Elements stored sequentially
- Uses contiguous memory locations
- May allocate extra capacity
- Resizing creates new array and copies elements
- Less memory overhead
- Stores only data

LinkedList

- Based on doubly linked list
- Each element stored as a node
- Node contains data + previous + next reference
- Uses non-contiguous memory
- Memory allocated per node
- No resizing concept
- Only link adjustments required
- Efficient for frequent modifications
- Slower iteration
- Must traverse node by node

Q .3) What is a HashSet?

- HashSet is a class in the Java Collection Framework.
- It implements the Set interface.

- It is available in the `java.util` package.
- It uses a hash table internally for storage.
- Does not allow duplicate elements
- Allows only one null value
- Does not maintain insertion order
- Unordered collection
- Not synchronized (not thread-safe by default
- Internally backed by a HashMap
- Stores elements as keys in HashMap

Q. 4) What is LinkedHashSet?

- LinkedHashSet is a class in the Java Collection Framework.
- It implements the Set interface.
- It is available in the `java.util` package.
- It extends HashSet.
- Does not allow duplicate elements
- Allows only one null value
- Maintains insertion order
- Not synchronized (not thread-safe by default)
- Combines features of HashSet and LinkedList

Q. 5) What is TreeSet?

- TreeSet is a class in the Java Collection Framework.
- It implements the Set interface.
- It is available in the `java.util` package.
- It stores elements in sorted order.
- Does not allow duplicate elements
- Does not allow null values (throws NullPointerException)
- Maintains sorted (ascending) order by default
- Not synchronized (not thread-safe by default)
- Elements must be Comparable or a Comparator must be provided

Q. 6) Write a program to reverse a LinkedList.

```
import java.util.Collections;
import java.util.LinkedList;
import java.util.Scanner;

public class ReverseLink {

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

```
        Scanner sc=new Scanner(System.in);
        LinkedList<Integer> li=new LinkedList<>();

        System.out.println("Enter  n value");
        int n=sc.nextInt();

        for (int i=0;i<n;i++){
            li.add(sc.nextInt());
        }
        System.out.println("Original List: "+li);

        Collections.reverse(li);

        System.out.println("Reversed List: "+li);
    }
}
```

Output

Enter n value

5

10

20

30

40

50

Original List: [10, 20, 30, 40, 50]

Reversed List: [50, 40, 30, 20, 10]

Q .7) Write a program to iterate ArrayList using Iterator.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class IteratorArray{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ArrayList<String> list = new ArrayList<>();

        System.out.print("Enter number of elements: ");
```

```
        int n = sc.nextInt();
        sc.nextLine();  // consume newline

        System.out.println("Enter " + n + " elements:");

        for (int i = 0; i < n; i++) {
            list.add(sc.nextLine());
        }

        System.out.println("Elements using Iterator:");

        Iterator<String> itr = list.iterator();

        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

Output:-

Enter number of elements: 5

Enter 5 elements:

1

2

3

4

5

Elements using Iterator:

1

2

3

4

5

Q. 8) Write a program to sort an ArrayList of integers.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<>();

        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();

        System.out.println("Enter " + n + " integers:");

        for (int i = 0; i < n; i++) {
            list.add(sc.nextInt());
        }

        System.out.println("Before Sorting: " + list);

        Collections.sort(list);

        System.out.println("After Sorting : " + list);
    }
}
```

Output:-

Enter number of elements: 5

Enter 5 integers:

1

4

3

6

2

Before Sorting: [1, 4, 3, 6, 2]

After Sorting : [1, 2, 3, 4, 6]

Q. 9) Write a program to remove duplicates from ArrayList.

```
import java.util.ArrayList;
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<>();

        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();

        System.out.println("Enter " + n + " integers:");

        for (int i = 0; i < n; i++) {
            list.add(sc.nextInt());
        }

        System.out.println("Original List: " + list);

        ArrayList<Integer> uniqueList = new ArrayList<>();

        for (Integer num : list) {
            if (!uniqueList.contains(num)) {
                uniqueList.add(num);
            }
        }

        System.out.println("After Removing Duplicates: " + uniqueList);
    }
}
```

Output:-

Enter number of elements: 5

Enter 5 integers:

5

4

3

4

3

Original List: [5, 4, 3, 4, 3]

After Removing Duplicates: [5, 4, 3]