JAVA ASSIGNMENT 12

PART I

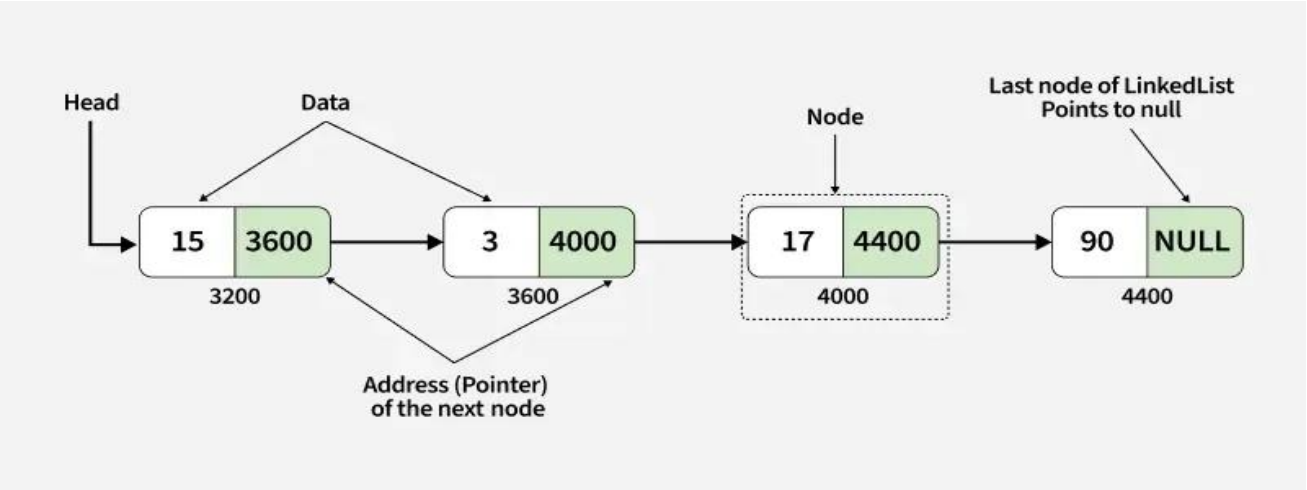
Q. 1) What is LinkedList?

Answer :

linked list is a fundamental data structure in computer science. It mainly allows efficient insertion and deletion operations compared to arrays. Like arrays, it is also used to implement other data structures like stack, queue and deque.

A linked list is a type of linear data structure individual items are not necessarily at contiguous locations. The individual items are called nodes and connected with each other using links.

- A node contains two things first is data and second is a link that connects it with another node.
- The first node is called the head node and we can traverse the whole list using this head and next links.



Key Components and Characteristics

- **Nodes:** The fundamental building blocks containing data and a reference to the next node.
- **Head/Tail:** The list starts at the "head" node. The last node ("tail") points to NULL, indicating the end of the list

.

- **Memory Efficiency:** They use dynamic memory allocation, meaning the size can change at runtime.
- **Access:** Accessing an element is slow (

time complexity) because it requires sequential traversal from the head.

- **Types:** Singly Linked List (one-way), Doubly Linked List (two-way), and Circular Linked List.

Advantages and Use Cases

- **Efficient Operations:** Inserting or removing nodes is fast, as it only requires updating pointers, not shifting elements.
- **Applications:** Used to implement data structures like queues, stacks, and hash maps.
- **Common Scenarios:** Ideal for applications with frequent insertions/deletions, such as playlist management.

Q. 2) Difference between ArrayList and LinkedList.

Answer :

ArrayList and LinkedList are two popular implementations of the List interface in Java. Both store elements in insertion order and allow duplicate values, but they differ in their internal data structure and performance.

What is an ArrayList?

ArrayList is a dynamic array-based implementation of the List interface. It internally uses a resizable array to store elements.

Explanation

- Elements are stored in contiguous memory locations
- Accessing elements using index is fast
- When the internal array becomes full, a new larger array is created and elements are copied

What is LinkedList ?

Linkedlist is a doubly linked list-based implementation of the List and Deque interfaces. Each element is stored as a separate node.

Explanation

- Each element is stored as a node. Every node contains:
- Data
- Reference to previous node
- Reference to next node
- No shifting of elements during insertion or deletion

ArrayList vs LinkedList :

Feature	ArrayList	LinkedList
Definition	A resizable array implementation of the List interface	A doubly-linked list implementation of the List interface
Data Storage	Stores elements in contiguous memory locations	Stores elements as nodes linked using pointers
Access Time	Fast random access using index (O(1))	Slower random access (O(n))

Feature	ArrayList	LinkedList
Insertion/Deletion	Slower in middle or beginning (O(n)), faster at end	Faster for insertion/deletion anywhere (O(1) if position known)
Memory Usage	Less memory overhead	More memory due to storing pointers
Iteration	Faster iteration using index-based loop	Faster iteration using iterator
Use Case	Best for frequent access and rare modifications	Best for frequent insertions and deletions

Q. 3) What is a HashSet?

Answer :

HashSet in Java implements the **set interface** of the Collections Framework. It is used to store the unique elements, and it doesn't maintain any specific order of elements.

- HashSet does not allow duplicate elements.
- Uses **HashMap** internally which is an implementation of hash table data structure.
- Also implements Serializable and Cloneable interfaces.
- HashSet is not thread-safe. To make it thread-safe, synchronization is needed externally.

{

Set :

Set interface is a part of the Java Collection Framework, located in the java.util package. It represents a collection of unique elements, meaning it does not allow duplicate values.

- The set interface does not allow duplicate elements.
- It can contain at most one null value except TreeSet implementation which does not allow null.
- The set interface provides efficient search, insertion, and deletion operations.

HashMap :

A HashMap is a part of Java's Collection Framework and implements the Map interface. It stores elements in key-value pairs, where, Keys are unique. and Values can be duplicated.

- Internally uses Hashing, hence allows efficient key-based retrieval, insertion, and removal with an average of O(1) time.
- HashMap is not thread-safe, to make it synchronized, use Collections.synchronizedMap().
- Insertion order is not preserved in HashMap. To preserve the insertion order, LinkedHashMap is used and to maintain sorted order, TreeMap is used.

HashMap Declaration :

public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable

}

Capacity of HashSet

Capacity refers to the number of buckets in the hash table. The default capacity of a HashSet is 16 and the load factor is 0.75.

When the number of elements exceeds the capacity automatically increases (resizes) to maintain performance.

new capacity = old capacity × 2

Load Factor

Load Factor is a measure that controls how full the HashSet can get before resizing. Default Load Factor = 0.75. If the number of elements exceeds the threshold, the capacity is doubled.

Threshold = capacity × load factor

Q. 4) What is LinkedHashSet?

Answer :

LinkedHashSet in Java implements the Set interface of the Collections Framework.

- It combines the functionalities of a HashSet with a doubly-linked list to maintain the insertion order of elements.
- LinkedHashSet stores unique elements only and allows a single null.
- Implements Set, Cloneable and Serializable interfaces.

Constructors of LinkedHashSet :

1. LinkedHashSet()

This constructor is used to create an empty LinkedHashSet with the default capacity i.e. 16 and load factor 0.75.

LinkedHashSet<E> hs = new LinkedHashSet<E>();

2. LinkedHashSet(Collection C)

Used in initializing the LinkedHashSet with the elements of the collection C.

LinkedHashSet<E> hs = new LinkedHashSet<E>(Collection c);

3. LinkedHashSet(int initialCapacity)

Used to initialize the size of the LinkedHashSet with the integer mentioned in the parameter.

LinkedHashSet<E> hs = new LinkedHashSet<E>((int initialCapacity);

4. LinkedHashSet(int capacity, float fillRatio)

Creates an empty LinkedHashSet with specified capacity and load factor.

LinkedHashSet<E> hs = new LinkedHashSet<E>(int capacity, float loadFactor);

Q. 5) What is TreeSet?

Answer :

A TreeSet in Java is a collection that stores unique elements in a sorted order, using a self-balancing binary search tree (specifically, a Red-Black tree) for storage. It is part of the java.util package and implements the NavigableSet interface.

Key Characteristics

- **Sorted Order**: Elements are automatically sorted, either by their natural ordering (for classes like String or Integer that implement the Comparable interface) or by a custom Comparator provided during creation.
- **No Duplicates**: Like all Set implementations, it does not allow duplicate elements. Attempting to add a duplicate simply results in the add() method returning false and the set remaining unchanged.
- **No Nulls**: TreeSet does not allow null values because it needs to compare elements to maintain order, and null cannot be compared with other objects.
- **Underlying Data Structure**: It internally uses a Red-Black tree to maintain balance and order. This ensures efficient operations.
- **Performance**: Most operations, such as add(), remove(), and contains(), have a time complexity of O(log N), where N is the number of elements in the set.
- **Not Synchronized**: It is not thread-safe by default. For use in multi-threaded environments, it must be synchronized using Collections.synchronizedSet().

Constructors of TreeSet :

In order to create a TreeSet, we need to create an object of the TreeSet class. The TreeSet class consists of various constructors which allow the possible creation of the TreeSet. The following are the constructors available in this class:

1. TreeSet()

Creates an empty TreeSet that sorts elements in their natural order

TreeSet ts = new TreeSet();

2. TreeSet(Comparator)

This constructor is used to build an empty TreeSet object in which elements will need an external specification of the sorting order.

TreeSet ts = new TreeSet(Comparator comp);

3. TreeSet(Collection)

Creates a TreeSet containing all elements from the given collection, stored in their natural sorted order. It is useful for converting a collection to a TreeSet.

TreeSet t = new TreeSet(Collection col);

4. TreeSet(SortedSet)

Creates a TreeSet containing the same elements and order as the specified SortedSet.

TreeSet t = new TreeSet(SortedSet s);

PART II

Q. 1) Write a program to reverse a LinkedList.

Answer :

```
import java.util.Collections;
import java.util.LinkedList;
public class ReverseLinkedList {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		LinkedList<Integer> list = new LinkedList<Integer>();
		list.add(10);
		list.add(20);
		list.add(30);
		list.add(40);
		list.add(50);
		System.out.println("Collection list : "+list);
		Collections.reverse(list);
		System.out.println("After reverese a list : "+list);
	}
```

```
}
```

Output :

```
Collection list : [10, 20, 30, 40, 50]

After reverese a list : [50, 40, 30, 20, 10]
```

Q. 2) Write a program to iterate ArrayList using Iterator.

Answer :

```
import java.util.ArrayList;

import java.util.Iterator;

public class IterateArrayList {

	public static void main(String[] args) {

		// TODO Auto-generated method stub


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


		list.add(12);

		list.add(24);

		list.add(36);

		list.add(48);

		list.add(60);

		list.add(72);

		list.add(84);

		list.add(96);

		list.add(108);

		list.add(120);


		System.out.println("ArrayList : "+list);
```

```
            System.out.println("ArrayList iterating using iterator : ");

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

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

Output :

ArrayList : [12, 24, 36, 48, 60, 72, 84, 96, 108, 120]

ArrayList iterating using iterator :

12 24 36 48 60 72 84 96 108 120

Q. 3) Write a program to sort an ArrayList of integers

Answer :

```
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class SortArrayList {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        ArrayList<Integer> list = new ArrayList<Integer>();
```

```
        list.add(10);
        list.add(25);
        list.add(13);
        list.add(8);
        list.add(30);
        list.add(22);
        list.add(15);

        System.out.println("ArrayList before sorting : "+list);

        Collections.sort(list);

        System.out.println("ArrayList after sorting : "+list);

    }

}
```

Output :

ArrayList before sorting : [10, 25, 13, 8, 30, 22, 15]

ArrayList after sorting : [8, 10, 13, 15, 22, 25, 30]

Q. 4) Write a program to remove duplicates from ArrayList.

Answer :

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

```
public class RemoveDuplicates {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

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

		list.add(5);
		list.add(13);
		list.add(7);
		list.add(5);
		list.add(9);
		list.add(13);
		list.add(22);
		list.add(53);
		list.add(7);

		System.out.println("ArrayList before remove duplicates : "+list);
		System.out.println("ArrayList after remove duplicates : ");

		LinkedHashSet<Integer> set = new LinkedHashSet<Integer>(list);
		list = new ArrayList<Integer>(set);

		System.out.println(list);
	}

}
```

Output :

ArrayList before remove duplicates : [5, 13, 7, 5, 9, 13, 22, 53, 7]

ArrayList after remove duplicates :

[5, 13, 7, 9, 22, 53]

Q. 5) Write a program to merge two ArrayLists

Answer :

```
import java.util.ArrayList;

public class MergeTwoArrayList {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		ArrayList<String> list1 = new ArrayList<String>();

		list1.add("Vijay");
		list1.add("Dinanath");

		ArrayList<String> list2 = new ArrayList<String>();

		list2.add("Ram");
		list2.add("Laxman");

		list1.addAll(list2);

		System.out.println(list1);

	}
```

}

Output :

[Vijay, Dinanath, Ram, Laxman]