LinkedList in Java - Detailed Notes

1. Introduction to LinkedList

- Definition: LinkedList is a doubly linked list implementation of the List and Deque interfaces, allowing for efficient insertion and deletion operations.

- Characteristics: It maintains elements in a linear order and is optimized for dynamic operations, particularly where frequent additions or deletions are needed.

- Usage: Ideal for cases where elements need to be added or removed frequently, especially at the beginning or end.

2. Creating a LinkedList

- Declaration: LinkedList<Type> list = new LinkedList<>();

- Example:

```
LinkedList<String> names = new LinkedList<>();

LinkedList<Integer> numbers = new LinkedList<>();
```

3. Adding Elements

- add(element): Adds the specified element at the end.

- addFirst(element): Adds the specified element at the beginning.

- addLast(element): Adds the specified element at the end (similar to add()).

```
names.add("Alice");

names.addFirst("Bob");

names.addLast("Charlie");
```

4. Accessing Elements

- get(index): Returns the element at the specified index.

- getFirst(): Retrieves the first element.

- getLast(): Retrieves the last element.

```
String name = names.get(0);

String firstName = names.getFirst();

String lastName = names.getLast();
```

5. Modifying Elements

- set(index, element): Replaces the element at the specified index with the provided element.

```
names.set(1, "David");
```

6. Removing Elements

- remove(index): Removes the element at the specified index.

- removeFirst(): Removes the first element.

- removeLast(): Removes the last element.

```
names.remove(0);

names.removeFirst();

names.removeLast();
```

7. Size of LinkedList

- size(): Returns the number of elements in the LinkedList.

8. Iterating Over LinkedList

- Using a for loop

- Enhanced for-each loop

- Using Iterator

```
for (int i = 0; i < names.size(); i++) { System.out.println(names.get(i)); }

for (String name : names) { System.out.println(name); }

Iterator<String>    it    =    names.iterator();    while    (it.hasNext())    {
```

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

9. Common Methods in LinkedList

- contains(Object): Checks if the LinkedList contains a specific element.

- indexOf(Object): Returns the index of the first occurrence of the specified element or -1 if not present.

- isEmpty(): Returns true if the LinkedList has no elements.

- clear(): Removes all elements from the LinkedList.

10. LinkedList as a Deque (Queue Operations)

- offer(element): Adds an element at the end (like add()).

- poll(): Retrieves and removes the first element.

- peek(): Retrieves, but does not remove, the first element.

```
names.offer("Eve");

String head = names.poll();

String peeked = names.peek();
```

11. LinkedList vs ArrayList

- Insertion & Deletion: LinkedList is better for frequent insertions and deletions.

- Random Access: ArrayList allows fast random access; LinkedList is slower because of traversal.

- Memory Usage: LinkedList uses more memory as it stores references to the previous and next nodes.

- Use Cases: LinkedList - frequent additions/removals; ArrayList - frequent access by index.

12. Advantages of LinkedList

- Dynamic Size: No need to specify initial size; grows as needed.

- Efficient Insertions/Deletions: Optimized for frequent additions or removals.

- Deque Support: Supports queue operations, suitable for FIFO and LIFO structures.

13. Disadvantages of LinkedList

- Memory Overhead: Uses extra memory for pointers.

- Slower Access Time: No direct access to elements by index; requires traversal.

- Not Thread-Safe: Needs synchronization for concurrent access.

14. FAQs of LinkedList in Java

- How is LinkedList different from ArrayList? LinkedList is better for insertions; ArrayList for access.

- How to add elements? Use methods like add(), addFirst(), or addLast().

- Accessing first or last element? Use getFirst() or getLast().

- Can LinkedList store null values? Yes, it can.

- Is LinkedList synchronized? No, use Collections.synchronizedList(new LinkedList<>()) for safety.