**Advance Java Class Notes Session 2**

Topics:

**LinkedList examples**

**Queue and PriorityQueue with examples**

**Set and HashSet with examples**

**LinkedList:** A LinkedList is a linear data structure that stores a sequence of nodes. Each node contains a data element and a reference to the next node in the sequence. The first node is called the head, and the last node is called the tail. LinkedLists allow for efficient insertion and deletion of elements at any position, unlike arrays which require shifting elements to make room.

**LinkedList Examples**

**1. What is a LinkedList?**

* A LinkedList in Java is a doubly-linked list implementation of the List and Deque interfaces.
* It allows for efficient insertion and deletion at both ends.
* Unlike an ArrayList, LinkedList does not require resizing as it grows dynamically.

**Example:**

***import java.util.LinkedList;***

***public class LinkedListExample {***

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

***LinkedList<String> list = new LinkedList<>();***

***// Adding elements***

***list.add("Java");***

***list.add("Python");***

***list.add("C++");***

***// Adding at first and last positions***

***list.addFirst("HTML");***

***list.addLast("JavaScript");***

***// Display elements***

***System.out.println("LinkedList: " + list);***

***// Remove elements***

***list.removeFirst();***

***list.removeLast();***

***System.out.println("After removing first and last: " + list);***

***// Iterating***

***for (String element : list) {***

***System.out.println("Element: " + element);***

***}***

***}***

***}***

**Queue**:

A Queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. Elements are added to the rear (enqueue) and removed from the front (dequeue) of the queue. Queues are commonly used for processing tasks in the order they were received, such as print spoolers, event handling, and breadth-first search algorithms

**Queue**

* A Queue follows the **First In, First Out (FIFO)** principle.
* Common methods include add(), offer(), poll(), and peek().

***import java.util.LinkedList;***

***import java.util.Queue;***

***public class QueueExample {***

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

***Queue<Integer> queue = new LinkedList<>();***

***// Adding elements***

***queue.add(10);***

***queue.add(20);***

***queue.add(30);***

***// Accessing and removing elements***

***System.out.println("Peek: " + queue.peek()); // 10***

***System.out.println("Poll: " + queue.poll()); // Removes 10***

***System.out.println("Queue after poll: " + queue);***

***}***

***}***

**PriorityQueue**:

A PriorityQueue is a special type of queue where elements are dequeued based on their priority, rather than the order they were enqueued. The element with the highest priority (lowest numerical value) is always at the front of the queue. PriorityQueues are commonly used in algorithms like Dijkstra's shortest path, Prim's minimum spanning tree, and event-driven simulations.

**PriorityQueue**

* A PriorityQueue sorts elements by natural ordering or custom ordering (via Comparator).
* The smallest element has the highest priority by default

***import java.util.PriorityQueue;***

***public class PriorityQueueExample {***

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

***PriorityQueue<Integer> pq = new PriorityQueue<>();***

***// Adding elements***

***pq.add(40);***

***pq.add(10);***

***pq.add(30);***

***pq.add(20);***

***// Accessing elements***

***while (!pq.isEmpty()) {***

***System.out.println("Polled: " + pq.poll());***

***}***

***}***

***}***

***public static void main(String[] args) {  
 // Press Alt+Enter with your caret at the highlighted text to see how  
 // IntelliJ IDEA suggests fixing it.  
  
 LinkedList<String> list = new LinkedList();  
 list.push("apple");  
  
  
 int[] myNumbers = {1,3,6,44,23};  
 int[] newarray = new int[myNumbers.length-1];  
  
 int indexToBeRemoved = 3;  
  
 for (int i = 0; i <myNumbers.length ; i++) {  
 System.out.println(myNumbers[i]);  
  
 if(i == 3){  
 i = i++;  
 }  
 newarray[i] = myNumbers[i];  
  
  
 }***

**Set**:

A Set is an unordered collection of unique elements. Sets do not allow duplicate values and provide constant-time access to check if an element is present in the set. Sets are commonly used for tasks like removing duplicates, finding unique elements, and performing set operations like union, intersection, and difference.

**Set**

* A Set is a collection that does not allow duplicate elements.
* Implementations include HashSet, TreeSet, and LinkedHashSet.

***import java.util.HashSet;***

***import java.util.Set;***

***public class SetExample {***

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

***Set<String> set = new HashSet<>();***

***// Adding elements***

***set.add("Java");***

***set.add("Python");***

***set.add("Java"); // Duplicate will not be added***

***set.add("C++");***

***System.out.println("Set: " + set);***

***}***

***}***

**HashSet**:

A HashSet is an implementation of the Set interface that uses a hash table as the underlying data structure. HashSets provide constant-time average performance for the basic operations of adding, removing, and checking membership of elements. The hash table allows for efficient storage and retrieval of elements, as the hash function distributes the elements evenly across the table. HashSets are commonly used in scenarios where the order of elements is not important, but fast lookup and uniqueness of elements are required.

**HashSet**

* A HashSet is a Set that uses a hash table for storing elements.
* Order of elements is not guaranteed.

*import java.util.HashSet;*

*public class HashSetExample {*

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

*HashSet<Integer> hashSet = new HashSet<>();*

*// Adding elements*

*hashSet.add(5);*

*hashSet.add(2);*

*hashSet.add(8);*

*hashSet.add(2); // Duplicate won't be added*

*// Iterating*

*for (Integer num : hashSet) {*

*System.out.println("Number: " + num);*

*}*

*}*

*}*

Interview Questions:

1. What is the difference between ArrayList and LinkedList?

Key differences:

- ArrayList uses dynamic array; LinkedList uses doubly linked list

- ArrayList: Better for random access O(1)

- LinkedList: Better for insertions/deletions O(1)

- ArrayList uses contiguous memory

- LinkedList uses scattered memory with references

1. What's the difference between Stack and Queue?

   Stack is LIFO (Last-In-First-Out)

   Queue is FIFO (First-In-First-Out)

1. When would you use PriorityQueue over regular Queue?

 Use PriorityQueue when:

- Need elements processed based on priority

- Implementing algorithms like Dijkstra's

- Task scheduling with priorities

1. What's the difference between HashSet and TreeSet?

 HashSet:

- Unordered collection

- O(1) operations

- Uses equals() and hashCode()

TreeSet:

- Ordered collection

- O(log n) operations

- Uses compareTo()

1. How does HashSet handle collisions?

          HashSet uses HashMap internally:

- Uses chaining (linked lists)

- If collision occurs, elements form a linked list

- After Java 8, uses balanced trees if list size > 8

1. Time Complexity Analysis

What's the time complexity for:

- LinkedList: add/remove at beginning O(1), access O(n)

- Queue: enqueue/dequeue O(1)

* PriorityQueue: add/remove O(log n)
* HashSet: add/remove/contains O(1) average

1. Space Complexity

What's the space complexity for:

- LinkedList: O(n)

- Queue: O(n)

- PriorityQueue: O(n)

- HashSet: O(n)

1. What operations can be performed on a Queue?

Main operations: -

enqueue (add)

- dequeue (remove)

- peek (view front element)

- isEmpty

- size

1. When to use LinkedList vs ArrayList?

 Use LinkedList when:

- Frequent insertions/deletions

- Don't need random access

Use ArrayList when:

- Frequent random access

- Mostly reading operations

1. Why use Queue interface instead of concrete class?

 Programming to interface:

- More flexible code

- Can easily switch implementations

- Better for testing

1. Can null be added to these structures?

- LinkedList: Yes

- Queue: Implementation dependent

- PriorityQueue: No

- HashSet: Yes (one null only)

1. Common errors to watch out for?

- Null pointer exceptions in LinkedList

- Empty queue/PriorityQueue exceptions

- ConcurrentModificationException

- Proper equals/hashCode for custom objects in HashSet

1. What is a singly linked list vs doubly linked list?

Singly: Each node points only to next node

Doubly: Each node points to both next and previous nodes

1. Real-world Applications

- LinkedList: Undo operations in software

- Queue: Print spooler, BFS algorithms

- PriorityQueue: Task schedulers, Dijkstra's algorithm

- HashSet: Removing duplicates, maintaining unique values