**Week 1: Introduction to Data Structures and Java Collections**

In this lab, we will explore essential data structures in Java: ArrayList, LinkedList, HashMap, and PriorityQueue. These data structures will be demonstrated through an online ticket booking system. We will also discuss Big-O notation, which is a way to analyze the efficiency of algorithms, and learn how to apply these concepts in real-world scenarios.

**Types of Data Structures:**

**ArrayList:**

An ArrayList is a resizable array implementation of the List interface. Unlike arrays, ArrayLists can dynamically adjust their size when elements are added or removed. They offer constant time access (O(1)) to elements by index, making them ideal for random access.

**Use case**: Managing a list of available seats in an online booking system, where users can quickly view or update seat availability.

**Example:**

|  |
| --- |
| ArrayList<String> seats = new ArrayList<>();  seats.add("Seat 1");  seats.add("Seat 2"); |

You can dynamically add, remove, or modify seats in the ArrayList.

**LinkedList:**

A LinkedList is a doubly linked list implementation of the List interface. Each element (node) contains a reference to both the previous and next elements. LinkedLists are efficient for insertion and deletion operations (O(1)) compared to ArrayLists, especially when adding or removing elements at the beginning or middle of the list.

**Use case:** Managing a queue of customers, such as those waiting for their turn in an online booking system.

**Example:**

|  |
| --- |
| LinkedList<String> queue = new LinkedList<>();  queue.add("Customer 1");  queue.add("Customer 2");  String servedCustomer = queue.poll(); // Serve the first customer |

The poll() method removes the first element, simulating a customer being served.

**HashMap:**

A HashMap is a key-value pair collection that allows fast lookups, insertions, and deletions with an average time complexity of O(1) for these operations. It uses a hash function to map keys to values, making it ideal for situations where quick data retrieval is required.

**Use case:** Storing customer ticket details, where the customer’s ID can be used to quickly access their booking information.

**Example:**

|  |
| --- |
| HashMap<String, String> tickets = new HashMap<>();  tickets.put("C123", "VIP Ticket");  tickets.put("C124", "Regular Ticket"); |

Here, the customer ID serves as the key, and the ticket type is the value.

**PriorityQueue:**

A PriorityQueue is a special type of queue where elements are processed based on their priority rather than the order they were added. The elements are ordered using their natural ordering or a comparator. The element with the highest priority is dequeued first.

**Use case:** Serving VIP customers first in an online booking system. The PriorityQueue ensures that customers with higher priority are served before others.

**Example:**

|  |
| --- |
| PriorityQueue<String> vipQueue = new PriorityQueue<>();  vipQueue.add("VIP Customer 1");  vipQueue.add("VIP Customer 2");  String vipCustomer = vipQueue.poll(); // Serve the highest priority customer |

The poll() method removes the customer with the highest priority.

**Big-O Notation:**

Big-O notation is a way to express the time complexity of an algorithm, i.e., how the running time or space requirements grow as the input size increases. It helps analyze and compare the efficiency of different algorithms. Common Big-O complexities include:

O(1): Constant time – The operation takes the same time regardless of the size of the input.

O(n): Linear time – The operation time grows linearly with the size of the input.

O(log n): Logarithmic time – The operation time grows logarithmically with the size of the input.

O(n^2): Quadratic time – The operation time grows quadratically as the input size increases.

For example, accessing an element in an ArrayList is an O(1) operation, while searching for an element in an ArrayList can take O(n) time if the list is unsorted.

**Applications in Online Ticket Booking System:**

**Dynamic Seating with ArrayList:**

In an online booking system, we can use an ArrayList to manage the seats. For example, when a customer books a seat, it can be added to the ArrayList. When a seat is cancelled, it can be removed.

**Queue Management with LinkedList:**

A LinkedList is perfect for managing the customer service queue. The first customer to enter the queue will be the first to be served, simulating the real-world FIFO (First-In-First-Out) principle.

**Ticket Information Management with HashMap:**

A HashMap allows us to efficiently store and retrieve customer booking details based on their unique customer ID. It ensures that we can quickly access or update ticket information without searching through a list.

**Prioritizing VIP Customers with PriorityQueue:**

PriorityQueue helps manage customer priority based on ticket type (e.g., VIP customers are served first). This ensures that priority customers are always given precedence.

**Practice Lab Tasks**

**Task 1**: Implement a simple ticket booking system using an ArrayList to manage seat reservations. Allow users to add, remove, and view reserved seats.

**Task 2**: Use a LinkedList to simulate a customer service queue. Implement enqueue and dequeue operations, and display the current queue.

**Task 3**: Create a HashMap to store and retrieve customer ticket information. Use customer IDs as keys and ticket types as values. Implement functionality to update or delete a booking.

**Task 4**: Implement a PriorityQueue to prioritize customers based on ticket type (VIP vs Regular). Add customers to the queue, then serve them based on priority.