**Group Members:**

**Huzaifa Habib 65920**

**Syed Bilal Ali 65923  
Sana Riaz 65902**

**DSA LAB: Mon 2:45-5:45**

**LAB 8 OEL PROJECT:**

**Project Proposal: Inventory Management System**

**Problem Statement:** Efficient management of inventory is a critical task in any warehouse or retail environment. The system needs to allow for quick item lookup, optimized sorting, and efficient handling of priority items to prevent bottlenecks in operations. Manual management methods are prone to delays and inaccuracies, which can lead to lost revenue and operational inefficiencies.

**Proposed Application:** An inventory management system will be developed to support item lookup, sorting by priority (e.g., perishables), and efficient addition and removal of items. The application will ensure fast data retrieval and manipulation, thus enhancing operational efficiency.

**Chosen Data Structure:** The primary data structure for this application will be a **Priority Queue implemented using a Heap**. This choice is justified as follows:

* **Efficiency:** A heap allows for O(log n) time complexity for insertion and deletion, which is optimal for managing a dynamic set of items where priority changes frequently.
* **Comparison with Other Data Structures:**
  + Arrays: Insertion and deletion in unsorted arrays have O(n) complexity, making them less efficient.
  + Linked Lists: Searching for the highest-priority item in an unsorted linked list is O(n), and even in a sorted linked list, insertion takes O(n).
  + Doubly Linked Lists: Similar to linked lists, operations are less efficient compared to heaps.
  + Stacks and Queues: These structures do not support prioritization.

**Algorithmic Challenges:**

1. Maintaining the heap property during insertions and deletions.
2. Efficiently handling updates to item priorities.

**Plan to Address Challenges:**

* Implement a min-heap to prioritize items with the smallest values (e.g., expiration dates).
* Use heapify operations to maintain heap properties efficiently during updates.

**Class Diagram:**

|  |
| --- |
| InventoryItem |
| - id: int  - name: String  - priority: int |
| + getId(): int  + getName(): String  + getPriority():int |

|  |
| --- |
| PriorityQueue<T> |
| - heap: List<T> |
| + insert(item: T): void  + deleteMin(): T  + peekMin(): T  + heapify(): void |

|  |
| --- |
| InventoryManagement |
| - pq: PriorityQueue<Item> |
| + addItem(): void  + removeItem(): void  + viewNextItem(): Item  + sortItems(): List<Item> |

**CODE:**

import java.util.\*;

class InventoryItem {

private int id;

private String name;

private int priority; // Lower value = higher priority

public InventoryItem(int id, String name, int priority) {

this.id = id;

this.name = name;

this.priority = priority;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public int getPriority() {

return priority;

}

*@Override*

public String toString() {

return "Item ID: " + id + ", Name: " + name + ", Priority: " + priority;

}

}

class PriorityQueue<T extends InventoryItem> {

private List<T> heap = new ArrayList<>();

public void insert(T item) {

heap.add(item);

int current = heap.size() - 1;

while (current > 0) {

int parent = (current - 1) / 2;

if (heap.get(current).getPriority() >= heap.get(parent).getPriority()) {

break;

}

Collections.*swap*(heap, current, parent);

current = parent;

}

}

public T deleteMin() {

if (heap.isEmpty()) {

throw new NoSuchElementException("Heap is empty");

}

T min = heap.get(0);

heap.set(0, heap.remove(heap.size() - 1));

heapify(0);

return min;

}

public T peekMin() {

if (heap.isEmpty()) {

throw new NoSuchElementException("Heap is empty");

}

return heap.get(0);

}

private void heapify(int index) {

int smallest = index;

int left = 2 \* index + 1;

int right = 2 \* index + 2;

if (left < heap.size() && heap.get(left).getPriority() < heap.get(smallest).getPriority()) {

smallest = left;

}

if (right < heap.size() && heap.get(right).getPriority() < heap.get(smallest).getPriority()) {

smallest = right;

}

if (smallest != index) {

Collections.*swap*(heap, index, smallest);

heapify(smallest);

}

}

}

public class InventoryManagement {

private PriorityQueue<InventoryItem> pq = new PriorityQueue<>();

public void addItem(int id, String name, int priority) {

pq.insert(new InventoryItem(id, name, priority));

}

public void removeItem() {

System.***out***.println("Removed: " + pq.deleteMin());

}

public void viewNextItem() {

System.***out***.println("Next item: " + pq.peekMin());

}

public static void main(String[] args) {

InventoryManagement im = new InventoryManagement();

im.addItem(1, "Apples", 5);

im.addItem(2, "Bananas", 2);

im.addItem(3, "Cherries", 8);

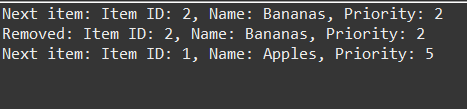
im.viewNextItem();

im.removeItem();

im.viewNextItem();

}

}

**OUTPUT:  
**

**Reflection Report:**

1. **Challenges:**
   * Implementing the heapify operation correctly.
   * Ensuring the priority queue could handle dynamic updates efficiently.
2. **Data Structure Choice:**
   * The heap was chosen due to its optimal performance for insertions and deletions, which are frequent in inventory systems.
3. **Learnings:**
   * Understanding the importance of maintaining heap properties for efficiency.
   * Balancing clarity and optimization in code.

By designing and implementing this inventory management system, we gained practical experience in leveraging data structures and algorithms to solve real-world problems effectively.

Github Link: <https://github.com/HuzaifaHabib1/OEL-Inventory-Management-System-.git>