UNIVERSITETI® METROPOLITAN TIRANA

FACULTY OF COMPUTER SCIENCES AND IT

Software Engineering Program

Data Structures and Algorithms

Midterm-Project Documentation

Hospital Patient Management System

Worked by: Krisalda Mihali

Accepted by: Prof. Evis Plaku

1. Introduction

The **Hospital Management System** is designed to efficiently manage patients in a hospital. The system allows for the **addition, removal, treatment, and discharge of patients**. It also incorporates functionality to search for patients by their severity, and undo patient discharges. The system leverages a combination of basic data structures such as **stacks, queues, and lists,** while utilizing algorithms **for sorting, searching, and managing patient records**.

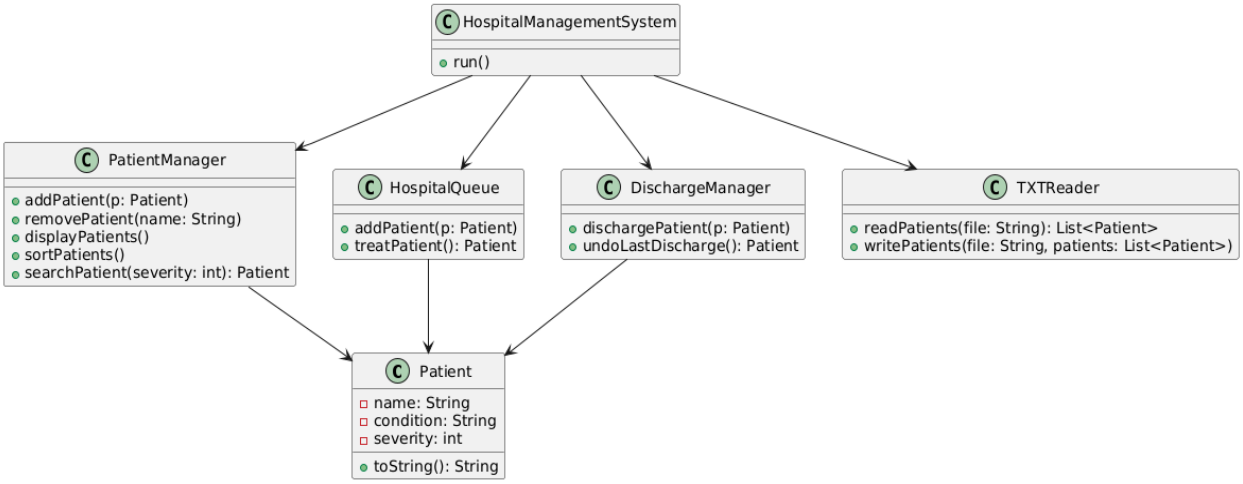

Fig.1. Class Diagram

2. Implementation Overview

The system consists of several classes, each responsible for a specific task:

- **Patient Class:** Represents a patient with properties like name, condition, and severity. This class also includes a method to return a string representation of a patient's details.
- **PatientManager Class:** Handles adding, removing, displaying, and sorting patients. It also provides a binary search for patients based on severity.
- **HospitalQueue Class:** Manages a queue of patients, prioritizing them based on severity using a priority queue (max-heap).
- **DischargeManager Class:** Manages the discharge process of patients and supports undoing the last discharge operation.
- **TXTReader Class:** Handles the reading and writing of patient data to/from text files.
- **HospitalManagementSystem Class:** The main interface of the system, providing users with an interactive menu to perform various operations.

3. Key Features and Functionalities

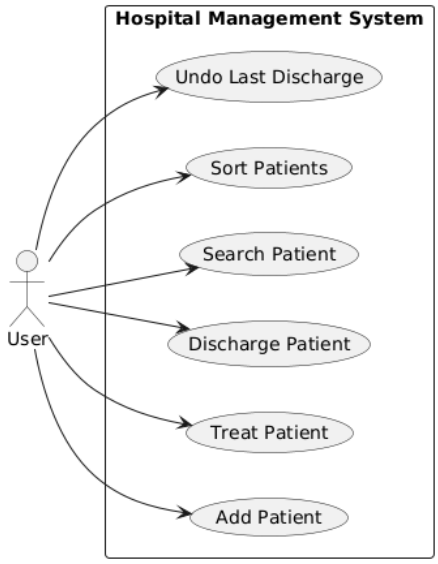

Fig.2. Use case diagram

- **Patient Addition:** New patients are added to both the PatientManager and the HospitalQueue. The system supports multiple attributes for each patient such as name, condition, and severity. Once added, the patients are immediately placed in the queue, sorted by severity, with the highest severity prioritized.
- **Patient Treatment:** The HospitalQueue class is implemented using a priority queue. The patient with the highest severity is treated first, which reflects the real-world priority system in hospitals.
- **Patient Discharge:** Patients are discharged and placed in a stack using the DischargeManager. This allows for the ability to undo the discharge, enabling the re-admission of the last discharged patient.
- **Patient Search and Sort:** The system supports searching for patients based on severity using a binary search algorithm. It also allows sorting patients by severity using a simple selection sort algorithm.
- **Undo Last Discharge:** The system allows undoing the last discharge using a stack. If there are discharged patients, the most recent one is re-admitted into the system.

4. Challenges Encountered

- **Sorting Complexity:** Sorting patients based on severity could become computationally expensive for a large number of patients. Initially, a simple selection sort was used to sort patients in descending order of severity. Although selection sort is easy to implement, it has a time complexity of $O(n^2)$, which might not be efficient for larger datasets.
- **Efficient Search:** Binary search is used to find patients with a given severity, but it requires the list to be sorted first. Ensuring that the list is sorted before performing a binary search added complexity, as the system needs to manage both sorted and unsorted states of patient records.

- **File Handling:** The system required persistent data storage for patient information. Implementing file reading and writing, while ensuring data is saved between sessions, was important for real-world usability.

5. Big-O Complexity Analysis

Operation	Data Structure	Complexity	Notes
Add Patient	LinkedList	O(1)	Simple append operation
Remove Patient	LinkedList	O(n)	Must search list for patient
Sort Patients	LinkedList	$O(n^2)$	Selection sort implementation
Binary Search	LinkedList	O(log n)	After sorting, plus O(k) for multiple matches
Treat Patient	PriorityQueue	O(log n)	Poll operation on heap
Undo Discharge	Stack	O(1)	Simple pop operation
Count Severe Patients	LinkedList	O(n)	Recursive implementation
File I/O	BufferedReader	O(n)	Linear to number of patients

6. Conclusion

The system successfully meets all specified requirements while demonstrating proper use of data structures and algorithms. The implementation makes appropriate tradeoffs between performance and functionality, particularly in handling multiple patients with identical severity levels. The file persistence and undo discharge features add practical utility to the system.

Future enhancements could include:

- More efficient sorting algorithms (e.g., MergeSort)
- Additional search criteria (name, condition)
- Graphical user interface
- Database integration for larger-scale deployment