The university of Azad Jammu and Kashmir, Muzaffarabad

Group no 7

Group Members:

Roll NO	Name
2024-SE- 35	Saqlain Mushtaq
2024-SE-40	Afzaal Ahmed
2024-SE- 06	Muneeba Arshad

Smart Hospital Management System

1. Introduction

The **Smart Hospital Management System (SHMS)** is a **console-based application** built in **C++** using **Object-Oriented Programming (OOP)** concepts.
It automates hospital management tasks such as:

- Managing patients, doctors, staff
- Scheduling appointments
- Generating bills
- Managing pharmacy stock
- Emergency admission
- Viewing diagnostic reports
- Viewing statistics

The project uses **C++ standard libraries** and **Windows console functions** (for color and formatting) to create an attractive interface.

2. Objectives

- Provide a single system to manage all hospital entities (patients, doctors, staff, appointments, pharmacy, billing).
- Demonstrate practical use of OOP concepts: classes, inheritance, polymorphism, encapsulation, and abstraction.
- Show file-based persistence using text files for long-term storage.
- Provide user-friendly, colored console menus.

3. Features Overview

Feature	Description
User Login System	Admin, Receptionist, Doctor, and Patient roles, each with its own menu.
Guest View	Non-logged-in users can view public info (doctors, medicines, appointments).
Admin Functions	Add doctors/staff, view statistics, view lists in tables, manage surgeries.
Receptionist Functions	Register patients, schedule appointments, create bills, add medicines, emergency admission.
Doctor Functions	View appointments, add diagnostic reports, schedule surgeries.
Patient Functions	View personal information, appointments, bills, and diagnostic reports.
Pharmacy Service	Add medicines, list stock, check low stock.
Diagnostics Service	Store and show patient diagnostic reports.
Emergency Admission	Register emergency patients quickly.
Surgery Service	Placeholder for surgery management.
Tables & Color UI	All lists displayed as formatted tables with headings and colors.
File Persistence	All data stored in text files (patients.txt, doctors.txt, appointments.txt, etc.).

4. System Architecture

4.1 Major Classes

- **Person (Abstract Base Class)**
  - Common attributes: id, name, age, gender, contact.
  - Common method: displayInfo() (virtual).
  - Used as base class for Patient, Doctor, and Staff.
- **Patient (Derived from Person)**
  - Adds: medical history, insurance info, national ID.
  - Methods: addHistory(), toCSVRow(), etc.
- **Doctor (Derived from Person)**

    - Adds: specialization, booked slots, consultation fee.

- **Staff (Derived from Person)**
    - Adds: role and username.
- **HospitalService (Abstract Base Class)**
    - Polymorphic base for services: PharmacyService, DiagnosticsService, EmergencyService, SurgeryService.
    - Method: performService() is overridden in each derived class.
- **PharmacyService**
    - Manages stock, expiry dates, issuing and adding medicines.
- **DiagnosticsService**
    - Manages diagnostic reports per patient.
- **EmergencyService & SurgeryService**
    - Demonstrates abstraction and polymorphism; these services can be expanded easily.
- **SHMSDatabase**
    - Central class holding all collections: patients, doctors, staff, appointments, bills, and services.
    - Handles loading and saving all data to files.
- **Bill & Appointment Structures**
    - Represent billing and appointment details.
    - Bill class includes formatted printing with colors.
- **User Struct**
    - Stores login info: username, password, role, and linked ID.

4.2 File Structure

- patients.txt
- doctors.txt
- staff.txt
- appointments.txt
- bills.txt
- medicines.txt
- users.txt

Each text file uses CSV format. The application loads these at startup and saves changes automatically.

5. Object-Oriented Concepts Used

OOP Concept	Where Used
Classes	Person, Patient, Doctor, Staff, SHMSDatabase, PharmacyService, DiagnosticsService, EmergencyService, SurgeryService, Bill, Appointment.
Inheritance	Patient, Doctor, Staff inherit from Person. PharmacyService, DiagnosticsService, EmergencyService, SurgeryService inherit from HospitalService.
Encapsulation	Private data members with public getters and setters in Person, Patient, Doctor, Staff.
Polymorphism	HospitalService::performService() overridden in derived services; Person::displayInfo() overridden in Patient and Doctor.
Abstraction	HospitalService defines a pure virtual interface for services.
Composition	SHMSDatabase contains maps of patients, doctors, staff, appointments, bills, and services.
File Handling	All CRUD operations saved to text files using streams.

6. How OOP Concepts Work in Our Project

1. **Inheritance**:
   We create a general Person class and extend it for specialized types (Patient, Doctor, Staff).
   This reduces code duplication — common attributes stay in the base class.

2. **Polymorphism**:
   HospitalService is an abstract class with a virtual performService() method.
   Each derived service (Pharmacy, Diagnostics, Emergency, Surgery) implements it differently.
   This allows us to call db.getEmergency().performService(); without knowing the concrete class.

3. **Encapsulation**:
   We protect sensitive attributes (like patient info, consultation fee) using private variables and expose them through getter/setter methods.

4. **Abstraction**:
   HospitalService hides the internal details of each service and only exposes the interface.

5. **File Handling**:
   Data persistence is achieved using CSV files.
   Each entity class has toCSVRow() and fromCSV() to convert objects to/from string form.

6. **Formatted Output**:
   Using Windows console functions (SetConsoleTextAttribute) for colors and setw() for column formatting creates a GUI-like console.

7. User Interface Flow

- **Main Menu →**
  - Login
  - Continue as Guest
  - Register Patient

    - Exit
- **Login Menu →**
    - As Admin
    - As Receptionist
    - As Doctor
    - As Patient

Each role sees a dedicated menu with its own functions.

8. Advantages of This System

- **Modular**: New features can be added easily.
- **Reusable Code**: OOP allows reuse of Person class and HospitalService.
- **Persistent**: All data saved between runs.
- **User Friendly**: Colored tables and menus.
- **Scalable**: Can be expanded to web or GUI frameworks in the future.

9. Limitations

- No database server - uses text files.
- Limited to console environment.
- Surgery and emergency modules are basic placeholders.

10. Conclusion

The Smart Hospital Management System demonstrates how **OOP principles** can be applied to build a **realistic, multi-user hospital management application**.
With a console interface, file persistence, and clean menus, this project can serve as a foundation for larger, more advanced systems (like a full GUI or web-based hospital management platform).

Team Collaboration

This project was developed collaboratively by **Muneeba Arshad**, **Saqlain Mushtaq**, and **Afzaal Ahmed**.

- **Muneeba Arshad** initiated the project by developing the basic code structure and implementing the core functionality.
- **Saqlain Mushtaq** refined the project by debugging, improving functionality, and applying extensive formatting to create a clean, user-friendly interface.
- **Afzaal Ahmed** contributed by designing the supporting components, testing the system across multiple use cases, all documentation, like report creating, creating UML diagram and use case table for this project and ensuring that all features worked cohesively.

Through effective teamwork, communication, and division of tasks, the team successfully delivered a well-structured, fully functional **Smart Hospital Management System** that demonstrates the core concepts of object-oriented programming.

11. Future Enhancements

- Replace text files with a real database (MySQL/SQLite).
- Add graphical or web front-end.
- Implement user permission levels.
- Enhance surgery and emergency modules to track real data.

Appendix: Color Codes Used

- Cyan (11): Headings
- Yellow (14): Table Headers
- Green (10): Success Messages and Option Numbers
- Red (12): Errors and Exit Options
- White (7): Default Text