

Hospital Management System

Team Members: G8

Aaryan Sharma - 16010123012

Aditey Kshirsagar - 16010123017

Aditya Baheti - 16010123023

Course Name: Object-Oriented Programming Methodology (**OOPM**)

Lab Batch: A1

Course Instructor's Name: Kaustubh Kulkarni

Date: 17 / 10 / 24

Table of Contents

- Introduction – Pg 2
- System Analysis – Pg 3
- System Design – Pg 4
- Implementation – Pg 6
- Conclusion – Pg 10
- Appendices – Pg 11

Introduction

- **Objective**

The primary objective of this **Hospital Management System** software is to create an efficient platform for managing patient records, staff data, and appointments. By implementing a user-friendly interface that supports **CRUD operations** (Create, Read, Update, Delete) for doctors, patients, and staff, the system aims to enhance hospital operations through organized data storage and easy access to critical information.

Key Goals -

- **Patient Records Management:**

Store and manage patient details, including personal information, medical history, and appointment data.

- **Staff Information Management:**

Handle staff details such as doctors, nurses, and administrative personnel, with support for tracking specializations and roles.

- **Appointment Scheduling:**

Enable patients to book appointments with doctors, with the ability for doctors to manage and schedule appointments effectively.

- **CRUD Operations:**

Provide basic operations to create, read, update, and delete records of patients, doctors, and staff members, ensuring data can be managed in real time.

- **Billing and Payments:**

Generate billing details for patients based on consultations and treatments, ensuring a streamlined payment process.

- **Efficient Data Storage:**

Ensure that all patient, doctor, and appointment information is stored efficiently and can be accessed easily when needed for management, reporting, or decision-making.

- **Technology Stack**

Programming Language: Java

IDE Used: VsCode

Libraries: Java Collections Framework

System Analysis

Problem Definition

Our application addresses the real-world challenge of manually managing large volumes of hospital data, which is inefficient and prone to errors. Key issues include inefficient data management, where manual record handling leads to misplaced files and delays; appointment errors, with double bookings and missed appointments due to a lack of automation; slow access to medical records in paper-based systems; billing errors caused by manual calculations; and overall operational inefficiency due to poor coordination between hospital departments.

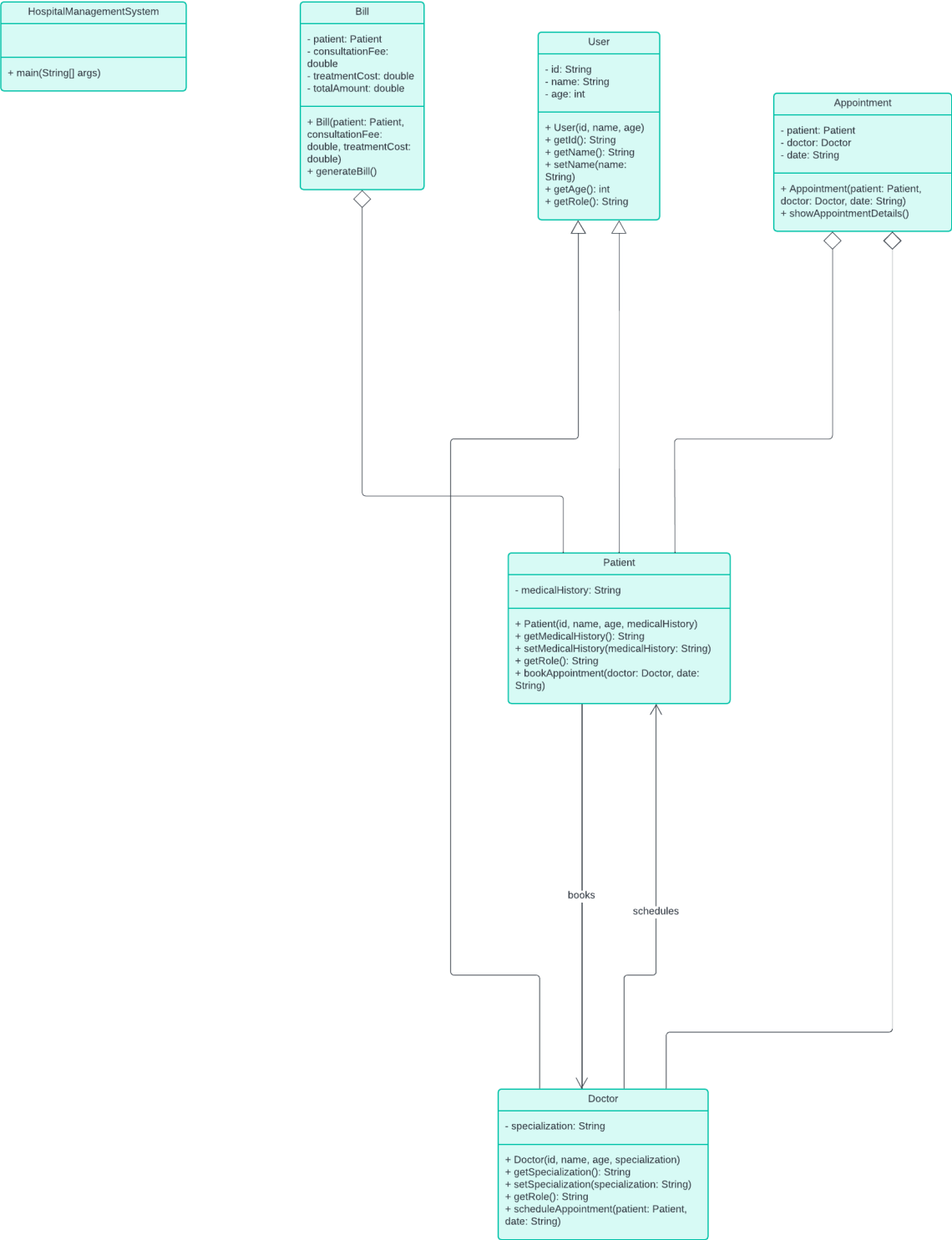
The solution provided by our system includes centralized data storage for secure and easy access to patient and staff information, efficient appointment scheduling that eliminates conflicts and improves patient satisfaction, quick access to medical records for faster diagnosis and care, automated billing that ensures clear and error-free calculations, and improved workflow through better coordination between departments. This digital approach ensures faster access to information, better organization, and enhanced management of hospital operations.

System Design

Class Diagram

1. **User Class:** The base class representing common attributes (id, name, age) and methods for all users in the system.
2. **Doctor Class:** Inherits from User and adds the specialization attribute and related methods.
3. **Patient Class:** Inherits from User and adds medical history and related methods.
4. **Appointment Class:** Represents the relationship between a Patient and a Doctor, along with the appointment date and related methods.
5. **Bill Class:** Manages billing information related to a Patient, including consultation fees and treatment costs.
6. **HospitalManagementSystem Class:** Contains the main method to run the application and interact with users.

The class diagram below illustrates the relationships between the User, Patient, Doctor, and Staff classes. The User class serves as an abstract base class, encapsulating common attributes and methods shared by its subclasses. The Patient and Doctor classes inherit these characteristics, extending functionality specific to each role in the hospital management system. Additionally, the Staff class can be included as another subclass, representing administrative or support personnel in the system.



Implementation

OOP Principles Applied

1. Encapsulation

```
abstract class User {
    private String id;
    private String name;
    private int age;

    // Constructor with validation
    public User(String id, String name, int age) {
        if (id == null || id.trim().isEmpty()) {
            throw new IllegalArgumentException("ID cannot be empty");
        }
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Name cannot be empty");
        }
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative");
        }
        this.id = id;
        this.name = name;
        this.age = age;
    }

    // Getters and setters
    public String getId() { return id; }
    public String getName() { return name; }

    public void setName(String name) {
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Name cannot be empty");
        }
        this.name = name;
    }

    public int getAge() { return age; }
    public abstract String getRole();
}
```

The attributes of the User class (and its subclasses) are declared as private, which prevents direct access from outside the class. This ensures that the attributes can only be accessed and modified through public methods. Similar to the User class, the Doctor and Patient classes have their own private attributes

2. Abstraction

```
// Doctor class (inherits from User)
class Doctor extends Users {
    private String specialization;
    public Doctor(String id, String name, int age, String specialization) {
        super(id, name, age);
        this.specialization = specialization;
    }
    public String getSpecialization() { return specialization; }
    public void setSpecialization(String specialization) { this.specialization =
specialization; }
    @Override
    public String getRole() {
        return "Doctor";
    }
    public void scheduleAppointment(Patient patient, String date) {
        System.out.println("Appointment scheduled for patient: " +
patient.getName() + " on " + date);
    }
}

// Patient class (inherits from User)
class Patient extends Users {
    private String medicalHistory;
    public Patient(String id, String name, int age, String medicalHistory) {
        super(id, name, age);
        this.medicalHistory = medicalHistory;
    }
    public String getMedicalHistory() { return medicalHistory; }
    public void setMedicalHistory(String medicalHistory) { this.medicalHistory =
medicalHistory; }
    @Override
    public String getRole() {
        return "Patient";
    }
    public void bookAppointment(Doctor doctor, String date) {
        System.out.println("Appointment booked with Dr. " + doctor.getName() + "
on " + date);
    }
}
```

The subclasses Doctor and Patient provide specific implementations of the getRole() method, demonstrating how abstraction works in conjunction with inheritance.

3. Inheritance

```
@Override
public String getRole() {
    return "Doctor";
}

@Override
public String getRole() {
    return "Patient";
}
```

Both Doctor and Patient provide their implementations for the getRole() method inherited from the User class.

4. Polymorphism

```
abstract class User {
    public abstract String getRole();
}

class Doctor extends User {
    @Override
    public String getRole() {
        return "Doctor";
    }
}

class Patient extends User {
    @Override
    public String getRole() {
        return "Patient";
    }
}
```

Both the Doctor and Patient classes override the getRole() method from the User abstract class.

Exception Handling

```
1. public User(String id, String name, int age) {
    if (id == null || id.trim().isEmpty()) {
        throw new IllegalArgumentException("ID cannot be empty");
    }
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Name cannot be empty");
    }
    if (age < 0) {
        throw new IllegalArgumentException("Age cannot be negative");
    }
    this.id = id;
    this.name = name;
    this.age = age;
}
```

When creating a new User, we validate the input in the constructor. If invalid data is provided an `IllegalArgumentException` is thrown.

```
2. public Bill(Patient patient, double consultationFee, double treatmentCost)
{
    if (patient == null) {
        throw new IllegalArgumentException("Patient cannot be null");
    }
    if (consultationFee < 0) {
        throw new IllegalArgumentException("Consultation fee must be
positive");
    }
    if (treatmentCost < 0) {
        throw new IllegalArgumentException("Treatment cost must be
positive");
    }
}
```

In the Bill class constructor, we perform checks to ensure that the patient is not null and that the consultation fee and treatment cost are non-negative.

Challenges Faced

1. Time Management
2. Data Management
3. Designing the System
4. Integrating Different Components
5. Exception Handling

Conclusion

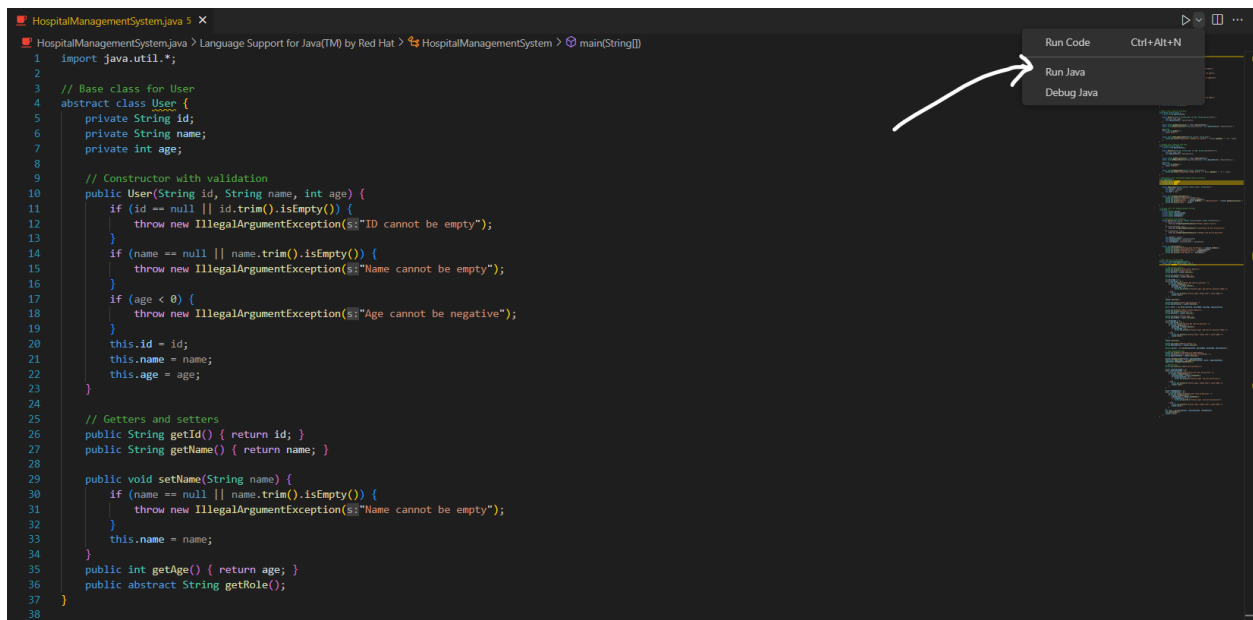
In developing the Hospital Management System, we navigated various challenges that highlighted the intricacies of software implementation. Through careful planning, clear communication, and collaborative effort, we were able to address the complexities of requirements gathering, system design, and user input validation.

The principles of Object-Oriented Programming - abstraction, encapsulation, inheritance, and polymorphism; played a crucial role in structuring our code effectively, promoting code reusability, and ensuring a robust design. By implementing a well-defined architecture and utilizing design patterns, we created a scalable and maintainable system.

Moreover, rigorous testing and documentation practices helped us ensure the reliability of the application while enhancing its usability for both medical staff and patients. Throughout the process, we learned the importance of adaptability and continuous improvement, which are essential in any software development project.

In conclusion, the Hospital Management System not only aims to streamline hospital operations and improve patient care but also serves as a practical example of applying theoretical knowledge to real-world challenges in software engineering. The experience gained during this project will undoubtedly contribute to our growth.

Appendices



The screenshot shows an IDE window titled 'HospitalManagementSystem.java 5'. The code defines an abstract class 'User' with attributes 'id', 'name', and 'age'. It includes a constructor with validation for non-empty 'id' and 'name', and a non-negative 'age'. There are also getters and setters for these attributes. A right-click context menu is open over the code, showing options: 'Run Code' (with a keyboard shortcut 'Ctrl+Alt+N'), 'Run Java', and 'Debug Java'. A white arrow points from the 'Run Java' option to the left.

```
1 import java.util.*;
2
3 // Base class for User
4 abstract class User {
5     private String id;
6     private String name;
7     private int age;
8
9     // Constructor with validation
10    public User(String id, String name, int age) {
11        if (id == null || id.trim().isEmpty()) {
12            throw new IllegalArgumentException("ID cannot be empty");
13        }
14        if (name == null || name.trim().isEmpty()) {
15            throw new IllegalArgumentException("Name cannot be empty");
16        }
17        if (age < 0) {
18            throw new IllegalArgumentException("Age cannot be negative");
19        }
20        this.id = id;
21        this.name = name;
22        this.age = age;
23    }
24
25    // Getters and setters
26    public String getId() { return id; }
27    public String getName() { return name; }
28
29    public void setName(String name) {
30        if (name == null || name.trim().isEmpty()) {
31            throw new IllegalArgumentException("Name cannot be empty");
32        }
33        this.name = name;
34    }
35    public int getAge() { return age; }
36    public abstract String getRole();
37 }
38
```

After Running the Java code, enter the details as prompted

Example Output-

```
Enter Doctor Details
Doctor ID: 002
Doctor Name: Aman
Doctor Age (must be positive): 34
Doctor Specialization: Mbbs

Enter Patient Details
Patient ID: 214
Patient Name: Aditya
Patient Age (must be positive): 17
Medical History: None

Schedule an Appointment
Enter Appointment Date (YYYY-MM-DD): 2024-10-25
Appointment booked with Dr. Aman on 2024-10-25
Appointment Details:
Patient: Aditya
Doctor: Aman (Specialization: Mbbs)
Date: 2024-10-25

Enter Billing Details
Consultation Fee (must be positive): 1000
Treatment Cost (must be positive): 200
Generating Bill for Patient: Aditya
Consultation Fee: $1000.0
Treatment Cost: $200.0
Total Amount: $1200.0
```