Classroom Booking System for IIT Tirupati

Object-Oriented Programming (CS203M) Course Project

Developed By:

- **Ch Pranav Tej** (CS24B057)
- **Shivam Purve** (CS24B055)
- *B.Tech 2nd Year, Computer Science & Engineering, IIT Tirupati*

---

1. Abstract & Problem Statement

The Need

As IIT Tirupati expands with new infrastructure, buildings, and academic blocks, the manual management of classroom, lab, and auditorium bookings has become inefficient. The lack of a centralized system leads to:

- Booking conflicts (double booking).
- Lack of transparency regarding room availability.
- Inefficient resource utilization (empty rooms vs. overcrowded study spaces).
- Administrative overhead in approving simple requests.

The Solution

We have developed a robust, dynamic, and automated **Classroom Booking System**. This full-stack application utilizes strict **Object-Oriented Programming (OOP)** principles to model the real-world hierarchy of the campus (Campus -> Building -> Floor -> Room) and the hierarchy of users (Admin -> Faculty -> Student). It features an interactive UI, real-time conflict detection, and a priority-based booking overriding system.

---

2. Technology Stack

The project uses a standard Enterprise Java stack, ensuring reliability and adherence to strict typing and OOP standards.

- **Language:** Java 21 (JDK 17+)
- **Backend Framework:** Spring Boot 3.1.5 (Web, JPA)
- **Database:** H2 Database (In-Memory/File-based persistence) - chosen for portability.
- **Frontend:** HTML5, CSS3, Bootstrap 5, Vanilla JavaScript (ES6).
- **Build Tool:** Maven.
- **Architecture:** MVC (Model-View-Controller) & REST API.

---

3. Object-Oriented Programming Implementation

This project was built specifically to demonstrate the four pillars of OOP. The backend architecture maps real-world entities directly to Java classes.

A. Inheritance

We utilized inheritance to promote code reusability and logical grouping of entities.

- **User Hierarchy:**
  - **Base Class:** `User` (Abstract Class). Contains common fields like `id`, `name`, `email`, `password`, `role`.
  - **Derived Classes:**
    - `Student`: Adds specific fields like `studentId` (Roll No), `branch`, `program`.
    - `Faculty`: Adds `employeeId`, `department`.
    - `Admin`: Adds administrative privileges.
  - *Benefit:* We write login logic once for `User`, and it works for all specific types.
- **Room Hierarchy:**
  - **Base Class:** `Room` (Abstract Class). Contains `capacity`, `name`, `resources`.
  - **Derived Classes:**
    - `Classroom`: Adds boolean flags like `hasSmartBoard`.
    - `Lab`: Adds `labType` (Computer, Hardware, etc.).
  - *Benefit:* The booking system treats everything as a `Room`, but we can store specific details for Labs vs Classrooms.

B. Polymorphism

The system uses polymorphism to handle different behaviors dynamically.

- **Booking Overrides:** The logic to check for booking clashes uses polymorphic checks on the `User` role.
  - Admin > Faculty > Student.
  - The system treats the requester as a generic `User` initially, then checks the specific instance (`instanceof` or Role Enum) to determine priority behavior (e.g., A Faculty can override a Student's booking, but not another Faculty's).
- **Repository Layer:** We use Spring Data JPA's `JpaRepository` , which is a polymorphic interface allowing us to perform CRUD operations on any entity type without rewriting SQL queries.

C. Encapsulation

Data integrity is maintained through strict encapsulation.

- **Private Fields:** All model fields (e.g., `password` , `bookingStatus`) are `private` .
- **Accessors:** Interaction happens strictly through public Getters and Setters.
- **Service Layer Logic:** The controller does not modify database entities directly. It calls the `BookingService` , which encapsulates the business logic (like checking for time conflicts) before modifying the data.

D. Composition & Aggregation

We modeled the physical infrastructure of IIT Tirupati using Composition (Strong "Has-A" relationship).

- **Campus Structure:**
  - A `Campus` is composed of `Building` s.
  - A `Building` is composed of `Floor` s.
  - A `Floor` is composed of `Room` s.
  - *Implementation:* Deleting a `Building` cascades and deletes all its `Floor` s and `Room` s automatically (Cascading Delete), demonstrating strong composition.

---

4. Installation & Execution Guide

A. Prerequisites

Ensure your local machine has the following installed:

- **Java Development Kit (JDK):** Version 17 or higher (Recommended: JDK 21).
- **Apache Maven:** Version 3.6 or higher.
- **Git:** For version control.

- **Web Browser:** Chrome, Firefox, or Edge.

B. Step-by-Step Setup

1. Clone the Repository

Open your terminal or command prompt and run:

```
git clone [https://github.com/Codebank-Pranav-Tej-Ch-Network/Classroom-Ma
cd Classroom-Management-System-for-IIT-Tirupati
```

2. Clean and Install Dependencies

This command downloads all necessary Spring Boot libraries defined in `pom.xml`.

```
mvn clean install
```

3. Run the Application

Start the embedded Tomcat server:

```
mvn spring-boot:run
```

Wait until you see the "Started BookingApplication" log message.

C. Accessing the System

1. Open your web browser.
2. Navigate to: `http://localhost:8080`
3. You will see the **Login Screen**.

D. First-Time Setup (Important)

Since the database is reset on every fresh install (if the `data` folder is cleared), follow these steps to initialize the hierarchy:

1. **Create Admin Account:**
    - Click "Create an Account".
    - Select Role: **ADMIN**.
    - Secret Key: `iit_admin_2025`
    - Fill in details and Sign Up.

2. **Initialize Infrastructure:**
    - Login as the new Admin.
    - Go to **Manage Infrastructure**.
    - Add a **Building** (e.g., "Academic Block A", 4 Floors).
    - Add **Rooms** to specific floors (ensure you check "Smartboard" or "Lab Type" where applicable).
3. **Create Faculty/Student Accounts:**
    - Faculty Secret Key: `iit_fac_2025`
    - Students: No key required.

---

5. System Architecture & Data Flow

- **Client Layer:** The user interacts with the `index.html` interface. JavaScript fetches data asynchronously using `fetch()`.
- **Controller Layer:** Exposes REST endpoints (e.g., `/api/book`, `/api/login`). It accepts JSON requests and deserializes them into Java Objects.
- **Service Layer:** Contains the core logic:
    - Validating time slots (8 AM - 12 AM).
    - Checking Booking Overlaps.
    - Handling Priority Overrides.
- **Repository Layer:** Interfaces extending `JpaRepository` that interact with the H2 Database.
- **Database:** Stores `Users`, `Buildings`, `Rooms`, and `Bookings`.

---

6. Key Features

1. Dynamic Infrastructure Management (Admin)

- Admins can create Buildings dynamically.
- Floors are auto-generated based on the building height.
- Rooms can be added to specific floors with capacity and resource details.
- **Delete Functionality:** Granular deletion of Rooms, Floors, or entire Buildings.

2. Hierarchy-Based Booking Engine

- **Student:** Can book available slots. Requests go to "Pending" or "Confirmed".

- **Faculty:** Can book slots. If a student has already booked the slot, the Faculty has the **Override** option to take that slot (Student gets bumped).
- **Admin:** Has supreme priority. Can override both Faculty and Students.

3. Interactive UI

- **Split-Screen Login:** Professional UI with Campus imagery and Developer Credits.
- **Campus Map:** A tree-view visualization of every building, floor, and room.
- **Dashboard:** Real-time statistics (Total Rooms, Pending Requests) using "Flashcards".

4. Resource Management

- When booking, users can select a checklist of required resources (Smartboard, Mic, Projector).
- Admins can view these requirements before approving.

---

7. Project Directory Structure

```
ClassroomBookingSystem/
├── pom.xml                        # Maven Dependencies
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/iit/booking/
│   │   │       ├── BookingApplication.java    # Main Entry Point
│   │   │       ├── controller/
│   │   │       │   └── APIController.java     # REST Endpoints
│   │   │       ├── model/
│   │   │       │   ├── User.java              # Abstract Base Class
│   │   │       │   ├── Student.java           # Child Class
│   │   │       │   ├── Faculty.java           # Child Class
│   │   │       │   ├── Admin.java             # Child Class
│   │   │       │   ├── Building.java          # Infrastructure
│   │   │       │   ├── Floor.java
│   │   │       │   ├── Room.java              # Abstract Base Class
│   │   │       │   ├── Classroom.java
│   │   │       │   ├── Lab.java
│   │   │       │   └── Booking.java           # Booking Entity
│   │   │       ├── model/enums/
│   │   │       │   ├── UserType.java
│   │   │       │   └── BookingStatus.java
│   │   │       ├── repo/                      # Data Access Layer
```

```
│   │   │       │   ├── UserRepository.java
│   │   │       │   ├── BookingRepository.java
│   │   │       │   ├── RoomRepository.java
│   │   │       │   ├── FloorRepository.java
│   │   │       │   └── BuildingRepository.java
│   │   │       └── service/
│   │   │           └── BookingService.java    # Business Logic
│   │   └── resources/
│   │       ├── application.properties         # DB Config
│   │       └── static/
│   │           └── index.html                 # Single Page Application
```

---

8. Authentication & Security

The system implements a custom **Role-Based Access Control (RBAC)** mechanism to ensure data integrity and prevent unauthorized access.

A. Registration Logic

We utilize a "Secret Key" validation system to prevent unauthorized users from registering as Faculty or Administrators.

- **Students:** Open registration. Requires valid Roll Number, Branch, and Program.
- **Faculty:** Requires the secret key `iit_fac_2025`.
- **Administrators:** Requires the secret key `iit_admin_2025`.

B. Profile Immutability

To ensure the authenticity of user data, the system employs **Field Immutability** in the OOP model.

- Once registered, critical identifiers like **Roll Number**, **Employee ID**, **Branch**, and **Program** are **locked**.
- Users can only update their **Name** and **Password** via the Profile interface. This prevents identity spoofing.

C. Booking Ownership & Cancellation

- **Ownership Check:** The backend explicitly verifies the `session_user_id` against the `booking_owner_id` before allowing a cancellation.

- **Hierarchy Enforcement:** The logic prevents lower-tier users from overriding higher-tier users (e.g., A Faculty cannot override an Admin).

---

9. Limitations & Future Scope

While the current system offers a robust solution for campus management, there are areas identified for future scaling and improvement.

Limitations

- **Database Persistence:** Currently, the system uses **H2 Database** (File-based). While excellent for portability and development, it is not designed for high-concurrency production environments.
- **Session Management:** The application currently relies on client-side state management.
- **Notification System:** Currently, users must check the dashboard to see if their booking was overridden. There is no active email or SMS push notification system.

Future Scope

- **Email Integration:** Integrating `JavaMailSender` to trigger automatic emails when a booking request is approved/rejected or overridden.
- **Calendar Sync:** Integration with Google Calendar to automatically add confirmed bookings to the user's personal calendar.
- **Mobile Application:** Developing a React Native or Flutter mobile app to allow students to scan QR codes outside classrooms to check immediate availability.
- **Analytics Dashboard:** Providing heatmaps showing which rooms are most frequently used to aid energy conservation planning.

---

10. Conclusion

The **Classroom Booking System for IIT Tirupati** successfully addresses the logistical challenges of managing campus infrastructure. By strictly adhering to **Object-Oriented Programming principles**—such as Inheritance for user roles, Polymorphism for booking conflicts, and Composition for building structure—the codebase remains modular, readable, and easy to extend.

This project not only streamlines the administrative process of room allocation but also ensures fair usage of resources through its transparency and priority-based logic. It stands as a comprehensive solution ready for further scaling as the institute grows.

Course: CS203M (Object Oriented Programming)

Institute: IIT Tirupati