CMP_SC/INFOTC 3330 - Object-Oriented Programming
Group Assignment #1
Study Room Reservations

Spring 2026

Scenario

The campus library has a small number of study rooms. Students can reserve a room for a one hour time slot. The librarians want a tiny Java program to track reservations and help them manage check ins.

You will design and implement a small object oriented program that models this system. Correctness matters, but **design quality matters just as much**. Your solution should reflect clear responsibilities, good encapsulation, and responsible collaboration between objects.

Required Software

Below have hyperlinks, so just click on them to get to the download pages.

- Eclipse IDE (Latest version: 2025-12)**(Recommended)** or IntelliJ IDEA
- Java SE Development Kit 25.0.2 (JDK25)

Allowed Topics and Constraints

Stay within what we have covered so far:

- Classes, fields, methods, constructors, access modifiers
- Encapsulation and class invariants (fail fast validation)
- Object relationships (association, composition) and collaboration
- Law of Demeter (avoid long chains and knowledge leaks)
- Basic arrays are allowed

Do **NOT** use:

- Java Collections Framework (no `List`, `ArrayList`, etc.)
- Inheritance, interfaces, polymorphism

- Overriding or implementing `equals`
- Static fields for application state
- User input (no scanner required). Hardcode a demonstration in `Main`.

Submission Instructions

- Export your Java project as `.zip` or `.rar` file through the IDE. Don't zip it externally through other programs.
- Submit your exported Java project through Canvas, and make sure all `.java` source files are included in your exported project.
- Your code must compile and run
- While submitting, you must specify which IDE you used. You can write it in a `README` file.
- Only Eclipse (Recommended) or IntelliJ is allowed for the IDE.
- Submit screenshots of your running output. You can add the screenshots into your project, but not where your source files, or packages are. Create a new folder in your project, and you can call it `screeshots` and store them into that folder.

Academic Integrity

Use of AI tools is strictly prohibited for this assignment.
Any use of AI generated code, design, or explanations will be reported to the Academic Integrity Office.

What You Are Building

Your system must support:

- Creating rooms (example: Room 101, capacity 4)
- Creating reservations for a specific room, student name, and time slot
- Check in: mark a reservation as checked in
- Cancel: cancel a reservation
- Simple reporting: print reservations for a room or print all reservations

To avoid `equals`, you will use integer IDs to identify reservations.

Required Classes

You must implement at least these classes:

Room

Represents a study room.

Required data:

- `String roomName` (example: `"Room 101"`)
- `int capacity` (must be positive)

Required behavior:

- Constructor enforces invariants (fail fast)
- Getters as needed (use intentionally)
- A `toString()` that prints a meaningful description (example: `Room 101 (cap 4)`)

Design note: Consider making `Room` immutable.

TimeSlot

Represents a one hour slot.

Required data:

- `int hour` where 0 to 23

Required behavior:

- Constructor validates hour (fail fast)
- `toString()` such as `13:00-14:00`

Design note: This is a great candidate for an immutable class.

Reservation

Represents a reservation for one student in one room at one time slot.

Required data:

- `int id` (must be positive)
- `Room room`
- `String studentName` (not null or blank)
- `TimeSlot timeSlot`
- `boolean canceled`
- `boolean checkedIn`

Required invariants:

- `id > 0`
- `room` and `timeSlot` are not null

- `studentName` is not null or blank
- A reservation cannot be both canceled and checked in at the same time

Required behavior:

- Constructor validates invariants (fail fast)
- `cancel()` marks the reservation canceled if allowed
- `checkIn()` marks the reservation checked in if allowed
- Methods that answer questions such as `isActive()` or `canCheckIn()` are encouraged
- `toString()` prints a useful line including id, student, room, time slot, and status

Open design choice: Decide what should happen if a reservation is checked in after being canceled, or canceled after being checked in. You must pick a rule and enforce it consistently.

ReservationBook

This class stores reservations and owns them. It should **not** coordinate the whole application, it should manage reservation storage and basic queries.

Constraints: You must store reservations using a plain Java array, for example:

- `private Reservation[] reservations;`
- `private int count;`

Required behavior:

- Constructor creates an empty book with a fixed maximum capacity (positive)
- `add(Reservation r)` adds if space exists, otherwise throws an exception
- `findById(int id)` returns the matching reservation or `null` if not found
- `printAll()` prints all stored reservations (one per line)
- `printForRoom(Room room)` prints reservations for that room

Design requirements:

- Fields must be private
- Do not expose the internal array directly
- Keep methods focused. Avoid turning this into a god class.

Open design choice: Decide whether `printForRoom` compares rooms by name, by reference, or by another approach, without using `equals`. Document your choice in a short comment.

ReservationManager

Coordinates the system at a higher level.

Required behavior:

- Stores a `ReservationBook`
- Generates reservation IDs (you can use a private counter field)
- Provides high level operations such as:
  - `createReservation(Room room, String studentName, TimeSlot slot)`
  - `cancelReservation(int id)`
  - `checkInReservation(int id)`

Design requirements:

- Avoid reaching through objects to manipulate internals
- Avoid long chains like `a.getB().getC().doSomething()`
- Delegate behavior to the class that owns the relevant data

Main

Your `Main` must demonstrate functionality with hardcoded data.

Minimum demo requirements:

- Create at least 2 rooms
- Create at least 5 reservations across different rooms and time slots
- Cancel at least 1 reservation
- Check in at least 1 reservation
- Print all reservations
- Print reservations for one specific room

Deliverables

Submit:

- `Room.java`
- `TimeSlot.java`
- `Reservation.java`
- `ReservationBook.java`
- `ReservationManager.java`
- `Main.java`

Grading Rubric (40 points total)

1. Correctness and Required Features (12 points)

- All required classes compile and run (3 points)
- Reservation creation works and stores reservations (3 points)
- Cancel and check in operations work correctly (3 points)
- Printing functions produce meaningful output (3 points)

2. Encapsulation and Invariants (10 points)

- Fields are private, internal state is protected (3 points)
- Constructors validate inputs and enforce invariants (4 points)
- No unnecessary setters or direct state exposure (3 points)

3. Object Relationships and Collaboration (8 points)

- Appropriate relationships: ReservationBook owns reservations, reservations refer to room and timeslot responsibly (3 points)
- Responsibilities are placed in the right class (3 points)
- Follows Law of Demeter, avoids knowledge leaks and long chains (2 points)

4. Design Quality and Code Smell Avoidance (6 points)

- No god class (manager coordinates, book stores, reservation owns its status rules) (2 points)
- Avoids anemic objects: Reservation has meaningful behavior, not just getters (2 points)
- Excessive void method usage and relying on mutable objects. Methods should all be testable. Print methods can be void. (2 points)

5. Readability and Naming (4 points)

- Clear naming and consistent formatting (2 points)
- Well-commented methods, describing the behavior (2 points)

Notes to Students

There are multiple valid designs. You will be graded on how well your design follows object-oriented principles and how consistently you enforce your own rules.