School Of Computer Science

PROJECT REPORT

Object Oriented Analysis & Design (OOAD)

Hotel Reservation System

Submitted by:

Somya Vats – SAP ID: 500119012

Tanisha Kathpal – SAP ID: 500125283

Tejasvi Hazarika – SAP ID: 500119060

To: Prof. Yogesh Rathia

Associate Professor, School of Computer Science

November 2025

In this project, our group developed a complete Hotel Reservation System using Java. The topic was selected using the SAP ID rule, where we took the last three digits of each member's SAP ID (012, 060, 283), added them to get 355, applied modulo 25, and arrived at topic number 5, which corresponds to a reservation system.

Throughout the report, we designed and implemented the system by following the Rational Unified Process (RUP). All four phases of RUP were applied to explain how the system was analyzed, modelled with UML diagrams, and implemented in both Object-Oriented Java and Procedural Java. The report presents the hotel reservation workflow, including room search, booking, payment, modification, and cancellation.

It also includes the architecture, testing outcomes, comparison of programming approaches, and the learning gained from building a functional reservation management system.

Acknowledgment

*We would like to express our sincere gratitude to **Prof. Yogesh Rathia** for his guidance, support, and valuable insights throughout the development of this project. His encouragement helped us understand the concepts of Object-Oriented Analysis and Design and apply them effectively in building our Hotel Reservation System. We also thank our institution for providing the resources and environment that enabled us to complete this project successfully.*

Abstract

This project presents the design and development of a Hotel Reservation System using the Rational Unified Process (RUP) methodology. The system aims to streamline hotel operations by enabling customers to search rooms, check availability, make bookings, process payments, and manage reservations efficiently. The project is implemented in two approaches: an Object-Oriented (OOP) system using Java classes, encapsulation, inheritance, and polymorphism; and a Procedural version using functions and linear programming structure.

RUP's four phases- Inception, Elaboration, Construction, and Transition, guided the complete development lifecycle. UML diagrams, including the Use Case Diagram, Class Diagram, and Activity Diagram, support the system's analysis and design. The report highlights the system workflow, implementation details, testing results, and a comparative evaluation of procedural versus OOP programming paradigms. The project demonstrates how structured methodologies improve software quality, maintainability, and real-world applicability.

Overview:

Introduction to RUP (Rational Unified Process)

The Rational Unified Process (RUP) is a software development methodology that provides a disciplined approach to assigning tasks and responsibilities within a development team. It ensures that high-quality software is produced in a predictable manner.

RUP divides the software lifecycle into four iterative phases:

1. Inception – understanding *what* needs to be built.
2. Elaboration – designing *how* it will work.
3. Construction – coding and transforming design into a working product.
4. Transition – testing, deployment, and refinement.

Each phase produces concrete outputs such as diagrams, code, and test results.
RUP follows best practices such as iterative development, risk management, continuous testing, and clear documentation.

What is a System?

A **system** is a coordinated set of components that work together to achieve a specific goal.
In software engineering, a system:

- receives **inputs**,

- processes information, and
- generates **outputs**.

A well-designed system reduces human effort, minimizes errors, automates processes, and improves decision-making.

What is a Hotel Reservation System?

A Hotel Reservation System (HRS) is the core software used by hotels to manage their bookings and operations.
It enables:

- Searching for rooms
- Checking real-time availability
- Selecting dates
- Booking and payment
- Generating confirmations and invoices
- Allowing cancellations and modifications
- Hotel staff to manage inventory, pricing, and customer records
- Administrators to manage users and generate reports

It ensures that no two customers are assigned the same room at the same time, maintains pricing consistency, and supports hotel operations through automated workflows.

In simple words, it is the brain of a hotel's operational workflow.

Why RUP is Suitable for a Hotel Reservation System?

Hotel reservation systems are complex, involving:

- multiple users (customers, staff, admin),
- multiple functionalities (booking, payments, pricing),
- multiple modules (availability, notifications, reports), and
- real-time decisions (prevent double-booking).

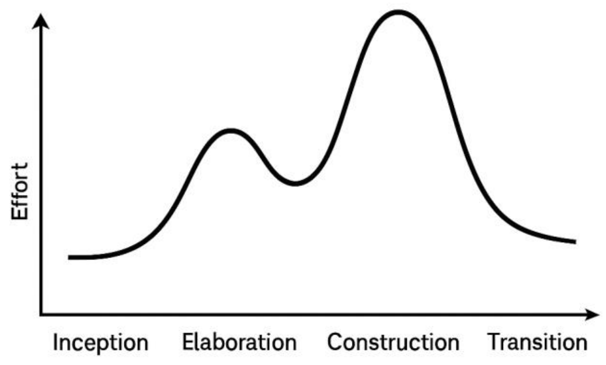
RUP is suitable because:

- **Inception** clearly defines problem, objectives, and users.
- **Elaboration** helps model the system using UML (Use Case, Class, Activity diagrams).
- **Construction** supports coding in both OOP and procedural approaches.
- **Transition** helps refine the working system, test workflows, and suggest improvements.

Using RUP ensures the project is built systematically with clarity and accuracy.

This Hotel Reservation System follows RUP in the following manner:

- **Inception Phase:**
  We identified the problem of manual hotel booking, defined objectives, recognized actors (Customer, Staff, Admin), and created the Use Case Diagram.
- **Elaboration Phase:**
  We modelled the system using UML, identified main classes (Room, Reservation, Customer, Payment), established their relationships, and designed workflows such as the booking process.
- **Construction Phase:**
  The system was implemented using **two different coding styles**—Java OOP and Java Procedural. OOP used concepts like inheritance and polymorphism, while the procedural version used static functions and data structures.
- **Transition Phase:**
  We performed testing, demonstrated booking and cancellation flows, documented results, compared OOP vs Procedural approaches, and proposed future enhancements.



(The detailed R.U.P for the Hotel Reservation System has been explained at the end, after the completion of each phase. This image contains only a sample diagram similar to how the development process of our system turned out)

INCEPTION PHASE

(The what & why)

Problem Statement

Hotel management is often affected by inefficient manual processes such as handwritten registers, phone-based bookings, and human-driven room allocation. These methods lead to frequent issues including double booking of rooms, incorrect price calculations, missing customer records, and delays in managing check-ins/check-outs.

As the demand for online services increases, hotels require a centralized digital system that automates booking operations, maintains real-time room availability, generates invoices, manages cancellations, assists staff, and supports administrators in monitoring hotel performance through reports.

The problem addressed in this project is the absence of a unified software solution that ensures efficient hotel reservation management, prevents conflicts, and supports decision-making through structured data handling.

Objectives

The primary objectives of the Hotel Reservation System are:

1. *Automate hotel booking workflows by allowing customers to search rooms, choose dates, and complete reservations digitally.*
2. *Ensure accuracy and eliminate double booking through real-time availability checks.*
3. *Provide structured access for different roles such as Customer, Staff, and Admin.*
4. *Maintain centralized data for rooms, users, reservations, payments, and reports.*
5. *Support staff operations such as room management, check-ins/ check-outs, and updates to pricing.*
6. *Enable administrators to generate insights such as occupancy rates, overall revenue, and usage statistics.*
7. *Improve user experience through a clear, step-by-step menu-driven interface.*
8. *Enhance reliability with validation, error handling, and consistent workflows.*

Scope of the Project

In Scope

- ❖ Customer registration & login
- ❖ Searching for rooms
- ❖ Viewing room details

- Selecting check-in & check-out dates
- Booking and payment processing
- Generating confirmation receipts
- Cancelling and modifying reservations
- Staff-level room management
- Admin-level user & rate plan management
- Reports (occupancy & revenue)
- Basic notifications (console-based)

Out of Scope

- Web interface / mobile app
- Complex payment gateways (only simulated)
- Integration with actual databases or third-party APIs
- Multi-hotel inventory (single hotel only)

Users / Actors

Actor	**Responsibilities**
Customer	Searches rooms, books reservations, makes payments, modifies/cancels bookings, views past reservations
Staff / Receptionist	Manages room inventory, updates room status, handles check-in & check-out, views reservations for operations
Administrator	Manages user accounts (staff/admin), updates pricing & rate plans, generates occupancy and revenue reports
Payment Gateway (Simulated)	Processes payments and returns transaction status & reference number
Notification Service (Simulated)	Sends booking confirmations and cancellation notifications

Functional Requirements (FR)

Customer:

- FR1: Customer shall be able to register and log in.
- FR2: Customer shall search rooms based on date range and type.
- FR3: Customer shall view room details & price.

- FR4: Customer shall complete a reservation with payment.
- FR5: Customer shall cancel or modify existing reservations.
- FR6: Customer shall receive a confirmation message after booking.

Staff:

- FR7: Staff shall add new rooms.
- FR8: Staff shall update room prices or status (Available / Maintenance).
- FR9: Staff shall view active reservations.
- FR10: Staff shall handle check-in/check-out.

Admin:

- FR11: Admin shall manage user accounts (create staff/admin).
- FR12: Admin shall modify rate plans.
- FR13: Admin shall generate occupancy and revenue reports.

Non-Functional Requirements (NFR)

- NFR1: *Usability* → system must provide clear menu navigation.
- NFR2: *Reliability* → ensure no double booking.
- NFR3: *Performance* → search results must display instantly.
- NFR4: *Maintainability* → modular services for different roles.
- NFR5: *Security* → role-based access control.
- NFR6: *Data Integrity* → correct reservation and payment linking.
- NFR7: *Accuracy* → correct price calculation including tax.

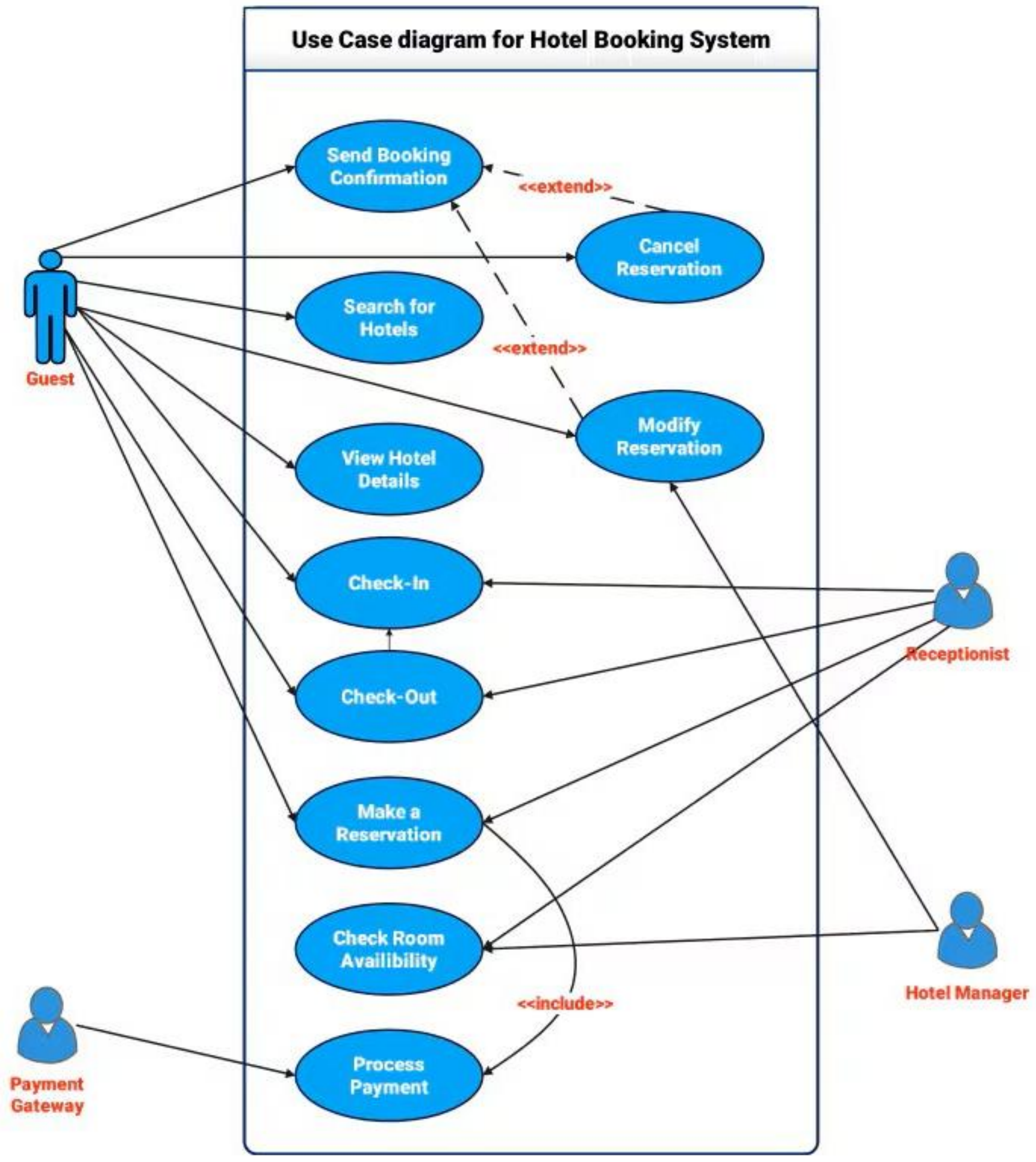
Use Case Overview

Use Cases include:

Login / Register	Search Rooms	View Room Details
Book Room	Make Payment	Modify Reservation
Cancel Reservation	Manage Rooms (Staff)	Manage Users (Admin)
Generate Reports (Admin)	Check-in / Check-out (Staff)	—

The *Use Case Diagram* visually represents the interaction between external actors and the Hotel Reservation System.

- The **Customer** interacts with search, booking, payment, and reservation management features.
- The **Staff** interacts with room management and booking supervision features.
- The **Admin** interacts with user management and reporting use cases.
- External systems such as **Payment Gateway** and **Notification Service** are triggered during reservation creation and confirmation to simulate payment handling and communication.



ELABORATION PHASE

Overview of the Elaboration Phase

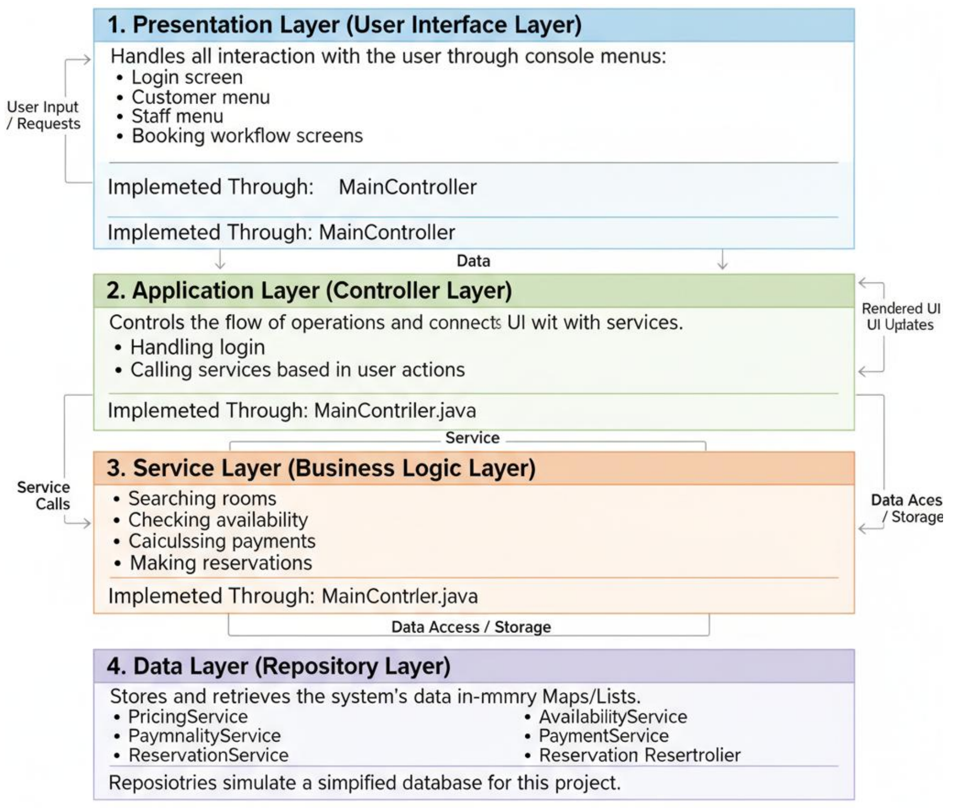
The Elaboration Phase focuses on transforming the requirements identified in the Inception phase into a well-defined system architecture.
This phase ensures that the system's structure, components, and interactions are clearly designed before any coding begins.

During this phase, we performed:

- Identification of the **main** classes used in the system
- Definition of **attributes, methods, and responsibilities**
- Establishing **relationships** between objects
- Designing the complete **Hotel Booking workflow** (Activity Diagram)
- Preparing the **final architecture** for implementation
- Creating UML diagrams (Use Case, Activity, Class Diagram)

The Elaboration phase guarantees that the Construction phase becomes predictable, modular, and easier to implement.

System Architecture Overview


1. Presentation Layer (User Interface Layer)
Handles all interaction with the user through console menus:
• Login screen
• Customer menu
• Staff menu
• Booking workflow screens
Implemeted Through: MainController
Implemeted Through: MainController
User Input / Requests
Data
2. Application Layer (Controller Layer)
Controls the flow of operations and connects UI wit with services.
• Handling login
• Calling services based in user actions
Implemeted Through: MainContriler.java
Rendered UI UI Uplates
Service
Service Calls
3. Service Layer (Business Logic Layer)
• Searching rooms
• Checking availability
• Caiculssing payments
• Making reservations
Implemeted Through: MainContrler.java
Data Aces / Storage
Data Access / Storage
4. Data Layer (Repository Layer)
Stores and retrieves the system's data in-mmry Maps/Lists.
• PricingService
• AvailabilityService
• PaymnalityService
• PaymentService
• ReservationService
• Reservation Resertrolier
Reposiotries simulate a simpified database for this project.

The Hotel Reservation System uses a **Layered (MVC-style) Architecture**, ensuring modularity, separation of concerns, and easy debugging.

Main Classes and Their Responsibilities

Class / Component	**Key Attributes**	**Responsibilities**
User (Abstract)	id, username, password, email, role	Base class for all system users; authentication & access control
Customer (extends User)	Inherits User attributes	Search rooms, create bookings, view reservations
Staff (extends User)	Inherits User attributes	Add rooms, update room details, view all reservations
Room	id, number, type, status, pricePerNight	Represents a hotel room; provides availability & pricing info
Reservation	id, customerId, roomId, checkIn, checkOut, total, status, txnRef	Stores booking details; links user, room, dates, and payment
Payment	id, amount, method, status, txnRef	Represents payment details; confirms booking transactions
PricingService	—	Calculates total booking price with tax
AvailabilityService	—	Checks overlapping bookings; validates room availability
PaymentService	—	Simulates payment processing and generates reference IDs
ReservationService	—	Orchestrates booking workflow; validates data; saves reservations
Repositories (UserRepository, RoomRepository, ReservationRepository)	Internal Maps/Lists	Store and retrieve users, rooms, and reservations (in-memory DB)

Data Structures (Used in OOP Version)

Data	Storage Type	Description
Users	Map<String, User>	stores all user accounts
Rooms	Map<String, Room>	stores all rooms
Reservations	Map<String, Reservation>	stores all bookings

These structures mimic the tables of a real hotel management database system.

Summary of Operations and Methods

Customer Operations

- searchRooms()
- createReservation()
- viewBookings()

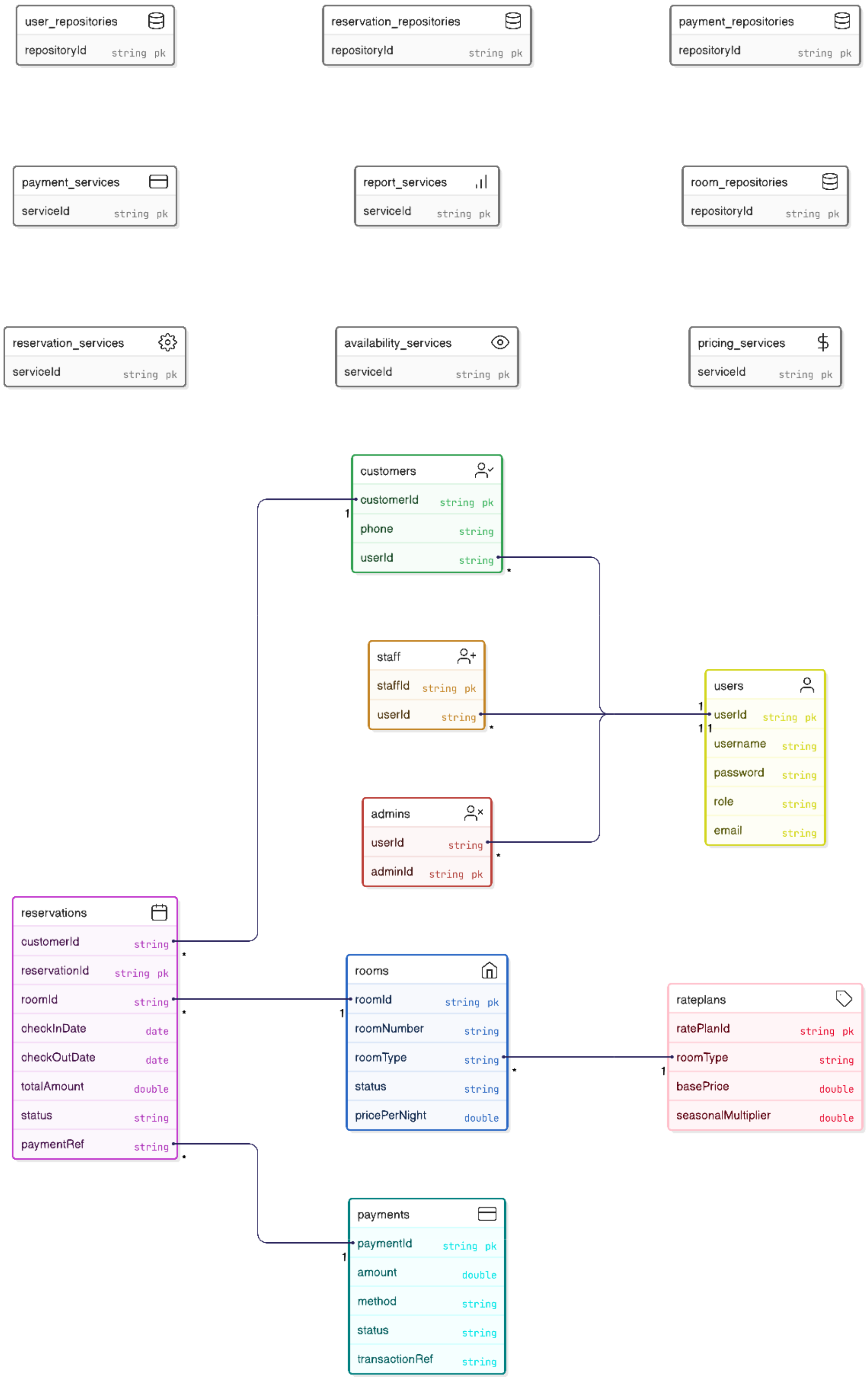
Staff Operations

- addRoom()
- updateRoom()
- viewAllReservations()

System Operations

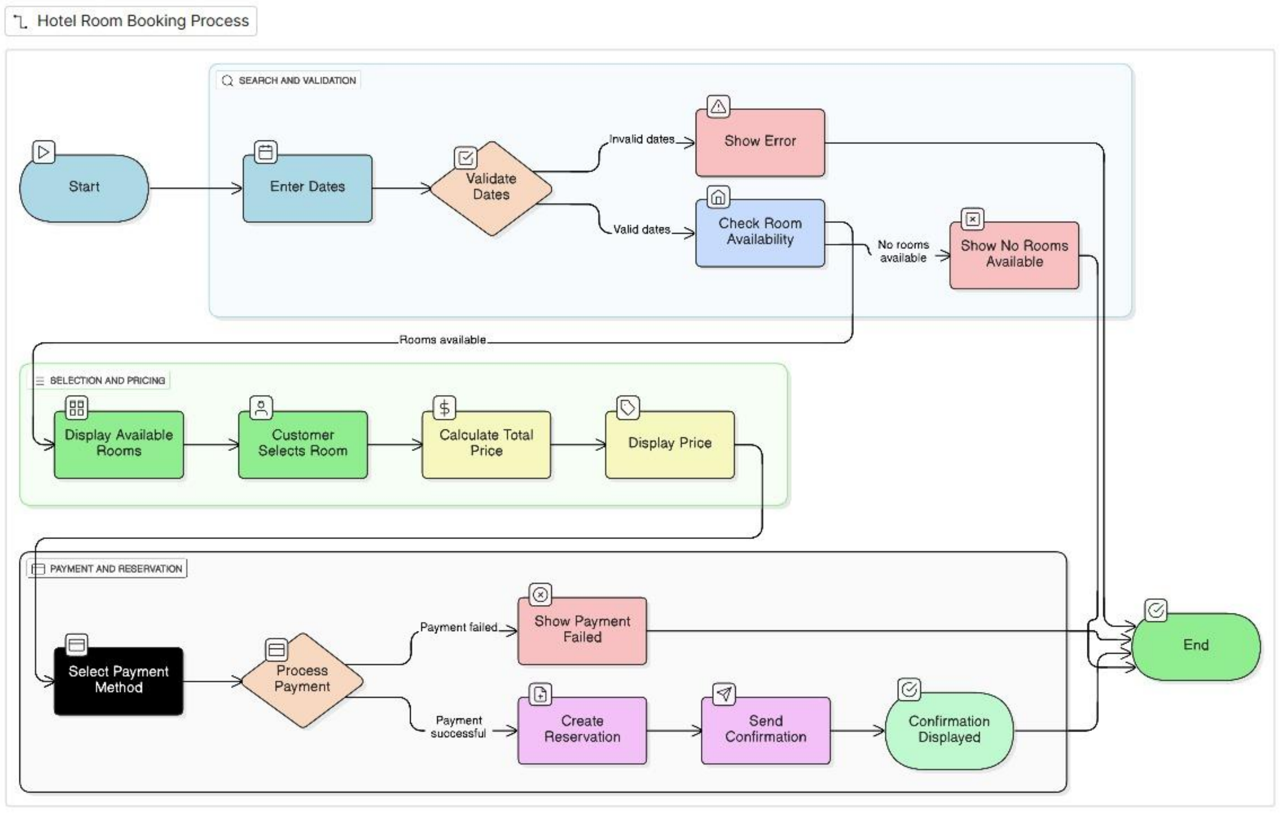
- isAvailable()
- calculatePrice()
- processPayment()
- createReservation()

The Class Diagram illustrates the architecture of the Hotel Reservation System.
It highlights the relationship between the main entities such as User, Customer, Staff, Room, Reservation, and Payment.
Service classes encapsulate the business logic and communicate with repository classes that act as the in-memory database layer.
Inheritance is used through the abstract User class.
Associations exist between Customer and Reservation, Room and Reservation, and Reservation and Payment.
This structure ensures modularity and clear separation of responsibilities, making the system easier to build and maintain.



user_repositories
repositoryId string pk
reservation_repositories
repositoryId string pk
payment_repositories
repositoryId string pk
payment_services
serviceId string pk
report_services
serviceId string pk
room_repositories
repositoryId string pk
reservation_services
serviceId string pk
availability_services
serviceId string pk
pricing_services
serviceId string pk
customers
customerId string pk
phone string
userId string
staff
staffId string pk
userId string
users
userId string pk
username string
password string
role string
email string
admins
userId string
adminId string pk
reservations
customerId string
reservationId string pk
roomId string
checkInDate date
checkOutDate date
totalAmount double
status string
paymentRef string
rooms
roomId string pk
roomNumber string
roomType string
status string
pricePerNight double
rateplans
ratePlanId string pk
roomType string
basePrice double
seasonalMultiplier double
payments
paymentId string pk
amount double
method string
status string
transactionRef string
1
*
1
1 1
*
*
*
1
*
*
1
*
1

The Activity Diagram illustrates the complete booking workflow of the Hotel Reservation System.
The process begins when the customer enters check-in and check-out dates.
The system validates the dates and checks room availability.
If rooms are available, the customer selects a room and the system calculates and displays the total price.
The customer then selects a payment method and initiates payment.
On successful payment, the system creates the reservation and sends a confirmation.
The activity ends when the booking confirmation is displayed to the customer.
This diagram ensures that the entire flow is clearly defined before implementation.



Design Patterns Used

Even though used conceptually, these patterns strengthen the architecture:

- **Layered Architecture (MVC-inspired)**
    - *Encapsulation — private fields, public methods*
    - *Inheritance — Customer & Staff extend User*
    - *Polymorphism — overridden behavior for users*
    - *Service–Repository Pattern — clean separation*
    - *Single Responsibility Principle — each service has exactly one job*

CONSTRUCTION PHASE

(The building stage)

Overview of the Construction Phase

The Construction Phase focuses on transforming the planned system design into a complete working software product.
All the analysis, modeling, and architectural decisions from the Inception and Elaboration phases are now implemented as executable code.

During this phase, we developed the **Hotel Reservation System** using **two separate implementations**:

1. Object-Oriented Implementation (Java OOP)
2. Procedural Implementation (Java - Non-OOP)

Both versions satisfy the same functional requirements but differ in structure, design philosophy, and maintainability.

This phase ensures that the final system functions correctly and aligns with the UML diagrams, classes, actors, and workflows defined earlier.

Technology Used

- **Programming Language:** Java
- **Reason:** Java supports both OOP and procedural styles, is portable, reliable, and widely used in enterprise applications.
- **IDE:** VS Code
- **Execution:** Console-based menu-driven application

Object-Oriented Implementation (OOP Version)

The OOP version follows the class structure and architecture defined in the Elaboration Phase.

Key Principles Used:

✓ **Encapsulation** – All attributes are private, accessed via getters/setters
✓ **Inheritance** – Customer, Staff, and Admin extend the User superclass
✓ **Polymorphism** – Payment processing uses an interface for different payment methods
✓ **Abstraction** – Business logic stored in service classes
✓ **Modular architecture** – Controller → Service → Repository layers

OOP Architecture Summary

1. Model Layer (Entities / Data Classes)

Includes:

- User, Customer, Staff, Admin
- Room, Reservation, Payment, Invoice
- RatePlan, RoomType (enum)

2. Repository Layer

Stores and retrieves data using:

- Maps (for rooms, payments, users)
- Lists (for reservations)

Simulates database behavior.

3. Service Layer

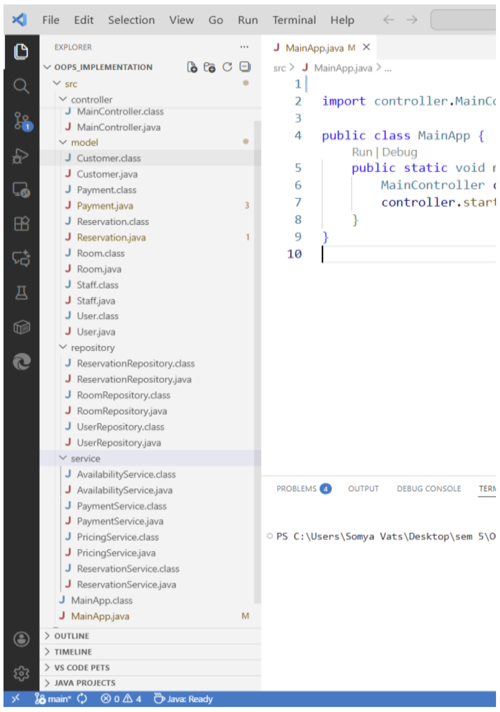
Contains the business logic:

- Authentication
- Availability checking
- Price calculation
- Reservation creation
- Payment simulation
- Reporting

4. Controller Layer

Handles user input & menu navigation:

- CustomerController
- StaffController
- AdminController



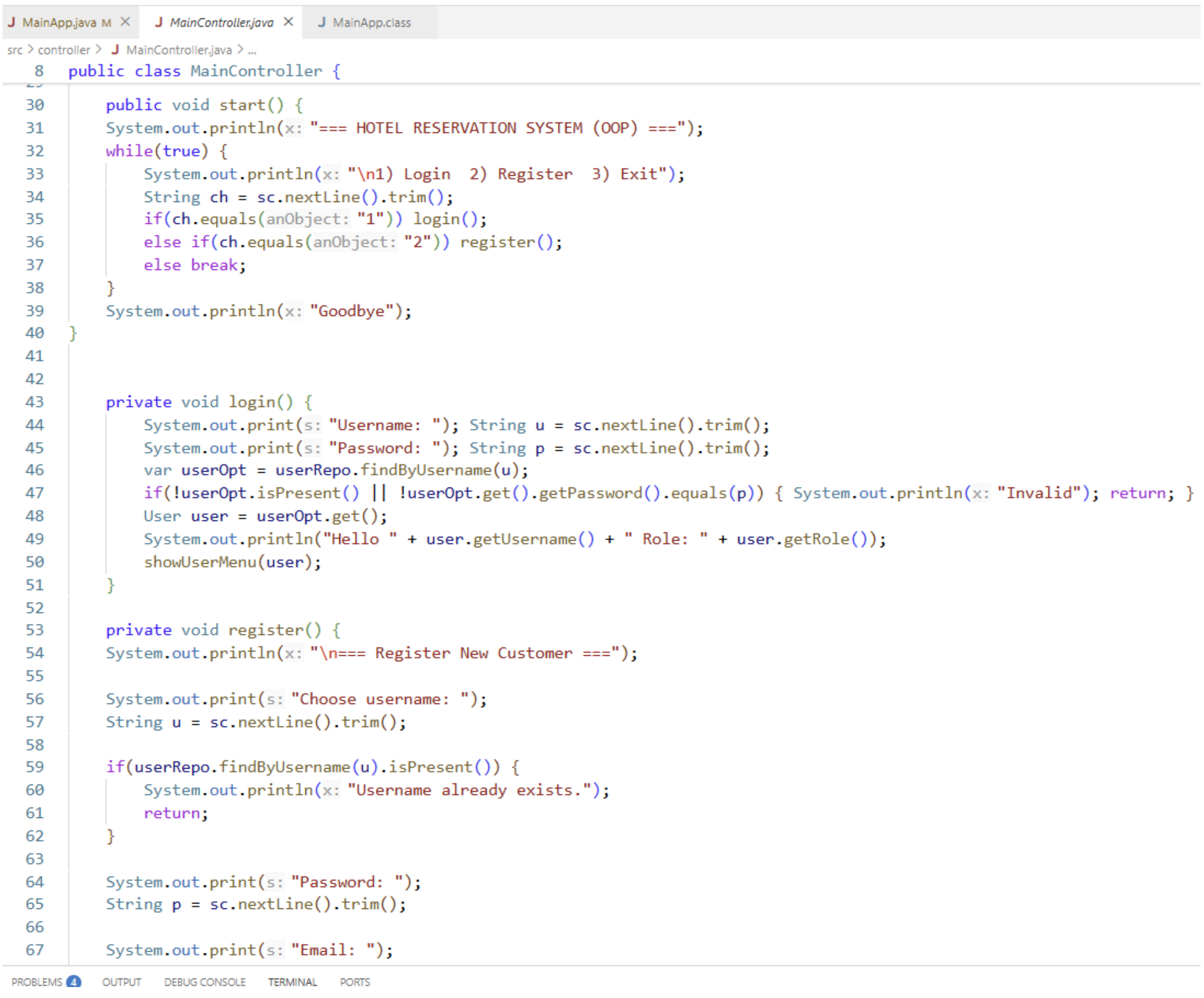
```
OOPS_IMPLEMENTATION
  src
    controller
      MainController.class
      MainController.java
    model
      Customer.class
      Customer.java
      Payment.class
      Payment.java            3
      Reservation.class
      Reservation.java        1
      Room.class
      Room.java
      Staff.class
      Staff.java
      User.class
      User.java
    repository
      ReservationRepository.class
      ReservationRepository.java
      RoomRepository.class
      RoomRepository.java
      UserRepository.class
      UserRepository.java
    service
      AvailabilityService.class
      AvailabilityService.java
      PaymentService.class
      PaymentService.java
      PricingService.class
      PricingService.java
      ReservationService.class
      ReservationService.java
    MainApp.class
    MainApp.java              M
```

```java

import controller.MainC

public class MainApp {
    public static void m
        MainController c
        controller.start
    }
}

```

```
PS C:\Users\Somya Vats\Desktop\sem 5\O
```

Mapping Code to Diagrams

- **Use Case Diagram** → each actor's functionality is implemented as menu-driven console options
- **Class Diagram** → directly converted into Java classes with attributes & methods
- **Activity Diagram (Booking)** → implemented exactly as:
  Search → Select → Calculate Price → Payment → Confirm → Save Reservation



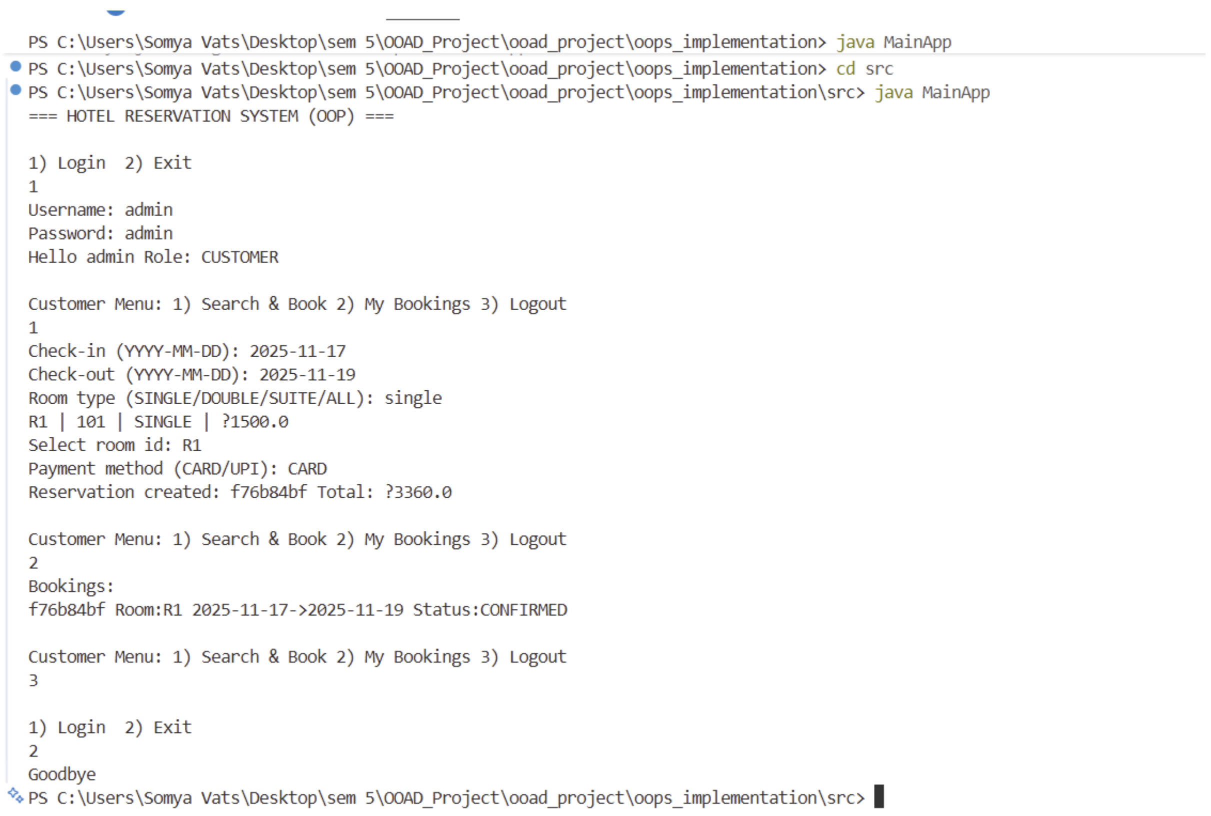
```java
public class MainController {
    public void start() {
    System.out.println(x: "=== HOTEL RESERVATION SYSTEM (OOP) ===");
    while(true) {
        System.out.println(x: "\n1) Login  2) Register  3) Exit");
        String ch = sc.nextLine().trim();
        if(ch.equals(anObject: "1")) login();
        else if(ch.equals(anObject: "2")) register();
        else break;
    }
    System.out.println(x: "Goodbye");
}


    private void login() {
        System.out.print(s: "Username: "); String u = sc.nextLine().trim();
        System.out.print(s: "Password: "); String p = sc.nextLine().trim();
        var userOpt = userRepo.findByUsername(u);
        if(!userOpt.isPresent() || !userOpt.get().getPassword().equals(p)) { System.out.println(x: "Invalid"); return; }
        User user = userOpt.get();
        System.out.println("Hello " + user.getUsername() + " Role: " + user.getRole());
        showUserMenu(user);
    }

    private void register() {
    System.out.println(x: "\n=== Register New Customer ===");

    System.out.print(s: "Choose username: ");
    String u = sc.nextLine().trim();

    if(userRepo.findByUsername(u).isPresent()) {
        System.out.println(x: "Username already exists.");
        return;
    }

    System.out.print(s: "Password: ");
    String p = sc.nextLine().trim();

    System.out.print(s: "Email: ");
```

OOPS Output Samples

View code on Github.

The OOP version generates console-based outputs such as:

- Main menu (Login/Register)
- Role-specific menus (Customer, Staff, Admin)
- Room search results
- Reservation confirmations
- Action logs for cancellations, payments, and check-ins
  These outputs are captured in screenshots during the Transition Phase.



```
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\oops_implementation> java MainApp
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\oops_implementation> cd src
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\oops_implementation\src> java MainApp
=== HOTEL RESERVATION SYSTEM (OOP) ===

1) Login  2) Exit
1
Username: admin
Password: admin
Hello admin Role: CUSTOMER

Customer Menu: 1) Search & Book 2) My Bookings 3) Logout
1
Check-in (YYYY-MM-DD): 2025-11-17
Check-out (YYYY-MM-DD): 2025-11-19
Room type (SINGLE/DOUBLE/SUITE/ALL): single
R1 | 101 | SINGLE | ?1500.0
Select room id: R1
Payment method (CARD/UPI): CARD
Reservation created: f76b84bf Total: ?3360.0

Customer Menu: 1) Search & Book 2) My Bookings 3) Logout
2
Bookings:
f76b84bf Room:R1 2025-11-17->2025-11-19 Status:CONFIRMED

Customer Menu: 1) Search & Book 2) My Bookings 3) Logout
3

1) Login  2) Exit
2
Goodbye
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\oops_implementation\src>
```

Procedural Implementation (Non-OOP Version)

This version is implemented **without classes**.
It uses:

- static methods
- HashMaps
- ArrayLists
- conditional structures (if-else, switch-case)

This demonstrates how the same system can be built without object-oriented features.

Procedural Structure Summary

All data stored in static Maps like:

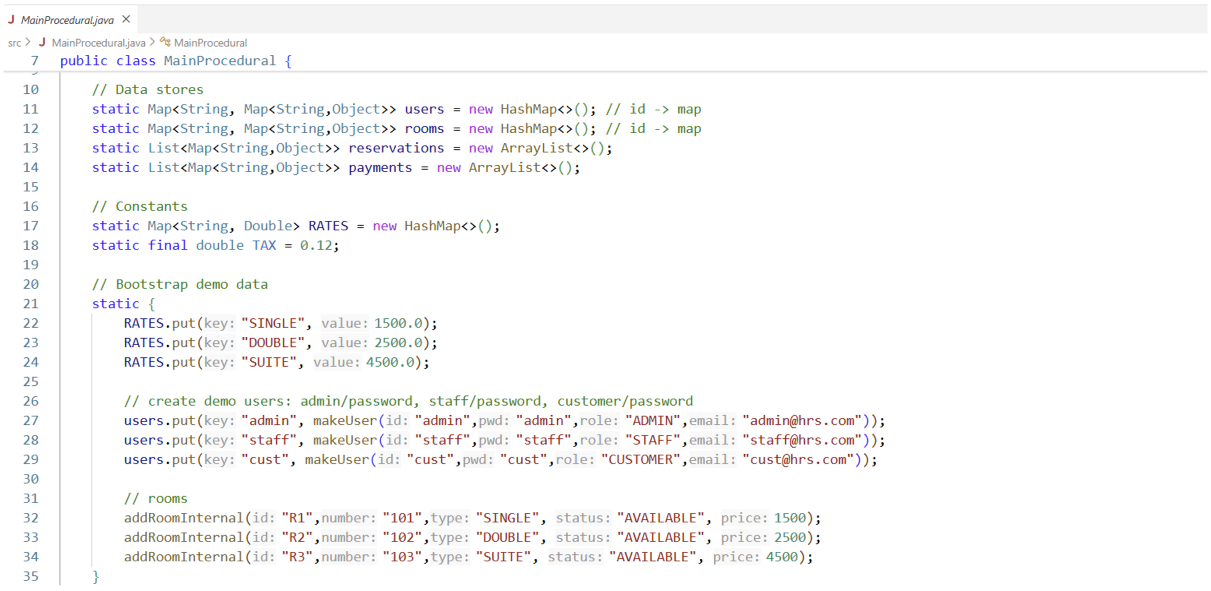
- Map<String, Map<String,Object>> rooms
- List<Map<String,Object>> reservations

All workflows built using static functions:

- login(), searchRooms(), createReservation(),
- calculatePrice(), cancelReservation(), generateReports(), etc.

Menu flow matches the OOP version:

✓ Login
✓ Customer menu
✓ Staff menu
✓ Admin menu
✓ Booking workflow
✓ Payment simulation



```java
public class MainProcedural {

    // Data stores
    static Map<String, Map<String,Object>> users = new HashMap<>(); // id -> map
    static Map<String, Map<String,Object>> rooms = new HashMap<>(); // id -> map
    static List<Map<String,Object>> reservations = new ArrayList<>();
    static List<Map<String,Object>> payments = new ArrayList<>();

    // Constants
    static Map<String, Double> RATES = new HashMap<>();
    static final double TAX = 0.12;

    // Bootstrap demo data
    static {
        RATES.put(key: "SINGLE", value: 1500.0);
        RATES.put(key: "DOUBLE", value: 2500.0);
        RATES.put(key: "SUITE", value: 4500.0);

        // create demo users: admin/password, staff/password, customer/password
        users.put(key: "admin", makeUser(id: "admin",pwd: "admin",role: "ADMIN",email: "admin@hrs.com"));
        users.put(key: "staff", makeUser(id: "staff",pwd: "staff",role: "STAFF",email: "staff@hrs.com"));
        users.put(key: "cust", makeUser(id: "cust",pwd: "cust",role: "CUSTOMER",email: "cust@hrs.com"));

        // rooms
        addRoomInternal(id: "R1",number: "101",type: "SINGLE", status: "AVAILABLE", price: 1500);
        addRoomInternal(id: "R2",number: "102",type: "DOUBLE", status: "AVAILABLE", price: 2500);
        addRoomInternal(id: "R3",number: "103",type: "SUITE", status: "AVAILABLE", price: 4500);
    }
```

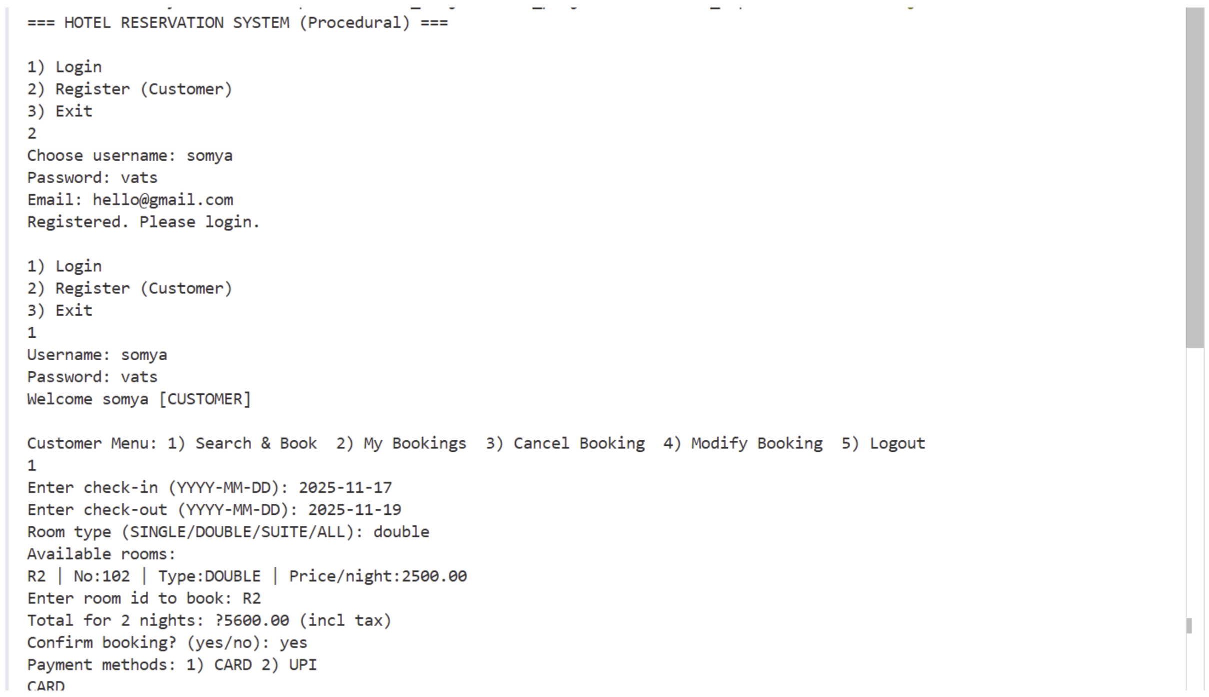
Mapping Procedural Flow to RUP

- Inception: Functional requirements implemented as simple functions
- Elaboration: Use Case & Activity Diagrams inspired the procedural algorithm
- Construction: Procedural code follows the same booking/calculation logic as OOP
- Transition: Outputs tested for correctness

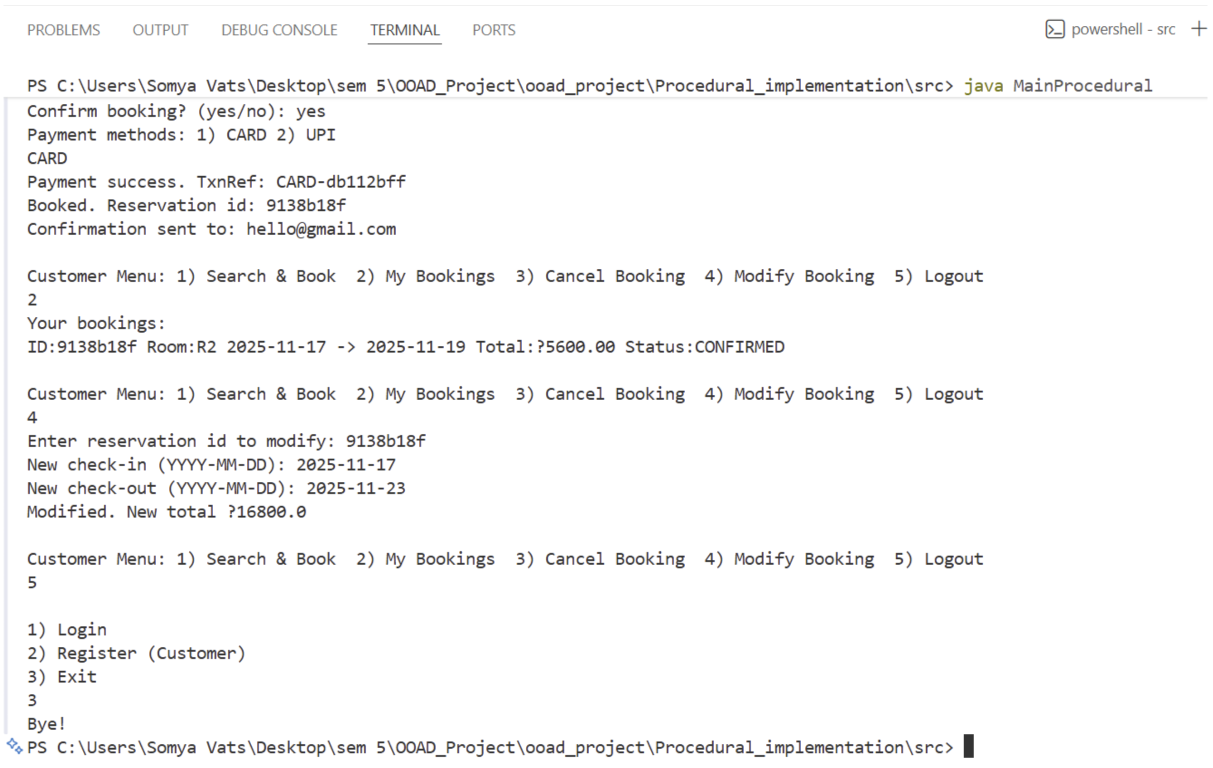
Procedural Output Samples

The procedural version provides menu-based console outputs similar to the OOP version. The screenshots include login attempts, searching rooms, booking confirmations, and cancellation results, ensuring functional consistency despite different programming styles.

View code on Github.



```
=== HOTEL RESERVATION SYSTEM (Procedural) ===

1) Login
2) Register (Customer)
3) Exit
2
Choose username: somya
Password: vats
Email: hello@gmail.com
Registered. Please login.

1) Login
2) Register (Customer)
3) Exit
1
Username: somya
Password: vats
Welcome somya [CUSTOMER]

Customer Menu: 1) Search & Book  2) My Bookings  3) Cancel Booking  4) Modify Booking  5) Logout
1
Enter check-in (YYYY-MM-DD): 2025-11-17
Enter check-out (YYYY-MM-DD): 2025-11-19
Room type (SINGLE/DOUBLE/SUITE/ALL): double
Available rooms:
R2 | No:102 | Type:DOUBLE | Price/night:2500.00
Enter room id to book: R2
Total for 2 nights: ?5600.00 (incl tax)
Confirm booking? (yes/no): yes
Payment methods: 1) CARD 2) UPI
CARD
```



```
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\Procedural_implementation\src> java MainProcedural
Confirm booking? (yes/no): yes
Payment methods: 1) CARD 2) UPI
CARD
Payment success. TxnRef: CARD-db112bff
Booked. Reservation id: 9138b18f
Confirmation sent to: hello@gmail.com

Customer Menu: 1) Search & Book  2) My Bookings  3) Cancel Booking  4) Modify Booking  5) Logout
2
Your bookings:
ID:9138b18f Room:R2 2025-11-17 -> 2025-11-19 Total:?5600.00 Status:CONFIRMED

Customer Menu: 1) Search & Book  2) My Bookings  3) Cancel Booking  4) Modify Booking  5) Logout
4
Enter reservation id to modify: 9138b18f
New check-in (YYYY-MM-DD): 2025-11-17
New check-out (YYYY-MM-DD): 2025-11-23
Modified. New total ?16800.0

Customer Menu: 1) Search & Book  2) My Bookings  3) Cancel Booking  4) Modify Booking  5) Logout
5

1) Login
2) Register (Customer)
3) Exit
3
Bye!
PS C:\Users\Somya Vats\Desktop\sem 5\OOAD_Project\ooad_project\Procedural_implementation\src>
```

Challenges Faced During Construction

This section shows maturity in report-writing:

- Converting UML diagrams into fully functional code
- Implementing availability check to prevent overlapping bookings
- Building two versions (OOP & procedural) with identical features
- Designing layered architecture in OOP version
- Ensuring role-based access control
- Handling date inputs and validations
- Maintaining clean menu-driven UI

Connecting Construction Phase to RUP

The Construction Phase transformed the design artifacts from the Elaboration Phase into executable software. The system was implemented using both Object-Oriented and Procedural paradigms to highlight architectural differences. All functional requirements such as search, booking, payment, modification, and reporting were successfully implemented in this phase.

TRANSITION PHASE

(The finishing & refinement stage)

Overview of the Transition Phase

The Transition Phase focuses on **refining, testing, validating, and delivering** the final software product.
In this stage, the Hotel Reservation System was executed, tested for functional correctness, validated against requirements, and prepared for submission.

This phase also includes evaluation, improvements, documentation, and comparison between the Object-Oriented and Procedural versions of the system.

Testing Approach

The system was tested manually using a set of predefined test cases that simulate real-world hotel operations.
The goal was to ensure that:

- all features behave as expected
- all roles (Customer/Staff/Admin) work correctly
- no double booking occurs
- payment and reservation workflows execute smoothly
- cancellation and modification work logically
- menu navigation is clear and error-free

Testing Technique Used

- Functional Testing
- Boundary Testing (dates, availability)
- Role-based Testing
- Validation Testing (invalid inputs)
- Scenario Testing (real booking flow)

Test Scenarios and Expected Results

These test cases were used for both procedural and OOP versions.

Test Scenario	Steps Performed	Expected Outcome
1. Successful Room Booking	Customer searches → selects room → enters dates → makes payment	Reservation is stored with correct dates, correct price, and valid payment reference
2. Double Booking Prevention	Two customers attempt to book the same room for overlapping dates	First booking succeeds; second booking fails with “Room not available”
3. Reservation Cancellation	Customer selects an existing reservation → cancels it	Reservation status becomes **CANCELLED**; room becomes available again
4. Modify Reservation	Customer changes dates → system checks availability	Reservation updates only if new date range is available
5. Staff Adds New Room	Staff adds room through staff menu	New room appears in customer room search results
6. Admin Generates Reports	Admin opens report menu	Accurate occupancy percentage and revenue totals are displayed

Performance Evaluation

The system performs efficiently for console-based operations:

- Search results appear instantly
- Room and reservation operations execute smoothly
- Procedural version is slightly faster due to its linear structure
- OOP version is more scalable and maintainable

Comparison: OOP vs Procedural Implementation

Aspect	OOP Implementation	Procedural Implementation
Structure	Classes & Objects	Functions & Data Structures
Reusability	High (inheritance/polymorphism)	Low
Maintainability	Easy to update & extend	Difficult for large systems
Modularity	Strong (service layers)	Weak (everything in one place)
Realism	Matches real-world hotel systems	Educational, simple
Scalability	Very high	Limited
Code Length	Longer but cleaner	Shorter but harder to manage

The OOP implementation is far more robust, extensible, and suitable for real-world hotel management systems. Its modular structure aligns perfectly with Object-Oriented Analysis and Design (OOAD) principles and the Rational Unified Process (RUP).

The procedural version is simple and easier to build initially but becomes difficult to maintain and scale as system complexity increases.

Thus, while both implementations satisfy the project's requirements, OOP provides a professional-level design that reflects industry practices.

(A detailed explanation of the comparison is shown below, where we also discussed the advantages of both implementations)

Improvements & Future Scope

Possible enhancements to make the system production-ready:

1. Database Integration (MySQL/PostgreSQL) instead of in-memory lists
2. Graphical User Interface using JavaFX or Swing
3. Online Booking Portal / Mobile App
4. Real Payment Integration (UPI/Card/NetBanking APIs)
5. Dynamic Pricing (demand-based pricing like hotels.com)
6. Email/SMS Notifications using JavaMail/Twilio APIs
7. Multi-hotel support (platform like OYO or Booking.com)
8. User analytics & dashboards
9. Cloud deployment (AWS/GCP/Azure)

These future improvements show strong scope for expansion.

Learning Outcomes

Each team member demonstrated understanding of:

- Applying RUP methodology end-to-end
- Converting real-world problems into UML diagrams
- Designing class structures using OOP principles
- Implementing menu-driven software in Java
- Handling date calculations, price logic, and data validation

- Developing both procedural and OOP versions of the same system
- Evaluating differences between programming paradigms
- Testing workflows and preventing logical conflicts like double-booking

These outcomes reflect practical OOAD skills gained.

The Transition Phase ensured the Hotel Reservation System was fully tested, validated, and refined before final submission. Both the OOP and Procedural versions achieved the intended functionality, and the project outcomes aligned with all requirements defined during the earlier phases of the RUP methodology.

CONCLUSION

Comparison of OOP vs Procedural Approach

This project required the implementation of the Hotel Reservation System in **both OOP and Procedural styles**, allowing a clear understanding of how the same system behaves under different paradigms. The following table summarizes the differences.

Feature / Aspect	**OOP Implementation (Java OOP)**	**Procedural Implementation (Java – Non-OOP)**
Structure	Organized into classes, objects, packages	Organized into functions and data structures
Code Organization	Highly modular (Controller, Service, Repository)	Linear and sequential code in a few files
Data Handling	Encapsulation protects data with private fields	Data stored openly in Maps and Lists
Reusability	High – inheritance, polymorphism allow reusability	Low – repeated code in multiple functions
Maintainability	Easy to update and extend due to separation of concerns	Hard to scale as complexity increases
Scalability	Excellent for large systems (supports real hotel apps)	Poor – becomes messy with many features
Security	Better – access control via objects and roles	Weak – all data is globally accessible
Real-World Mapping	Models real hotel systems perfectly through objects	Educational but unrealistic for production
Testing	Easier – unit tests can target individual classes	Harder – functions depend on shared data
Abstraction Level	High – business logic isolated in service classes	Low – logic scattered in procedural flows
Learning Purpose	Demonstrates full OOP concepts for OOAD	Demonstrates basic logic and flow control
Performance	Slightly more overhead, but optimized for bigger systems	Fast for small programs but unsustainable long-term

The Object-Oriented Programming (OOP) implementation of the Hotel Reservation System differs significantly from the procedural version in terms of structure, design quality, maintainability, and scalability. In the OOP version, the entire system is organized into logical classes such as User, Room, Reservation, Payment, and Service modules, each encapsulating specific data and behavior. This modular structure allows clear separation of concerns, making the system easier to extend, debug, and maintain. Concepts like inheritance (Customer and Staff extending User), polymorphism, and encapsulation ensure reusability and cleaner design, which closely resembles real-world hotel operations. In contrast, the procedural implementation follows a linear flow where functions and global data are used to perform tasks. While simpler to build, it becomes harder to manage as the system grows because logic is scattered and deeply interdependent. There is little reusability, and modifying one part often affects others. The OOP version supports layered architecture (controller → services → repositories), ensuring role-based operations and scalable workflows, whereas the procedural version remains suitable only for small, straightforward applications. Overall, OOP provides a structured, robust, and future-ready solution, while the procedural approach serves as a basic demonstration of system logic without long-term flexibility.

RUP Diagram for Hotel Reservation System

The **Rational Unified Process (RUP)** uses four iterative phases:
Inception, Elaboration, Construction, and Transition.
The “Hump Diagram” visually represents the intensity of activities performed during each phase.

- Inception → LOW effort (just defining problem, actors, scope)
- Elaboration → HIGH effort (UML, class design, architecture decisions)
- Construction → HIGHEST effort (actual coding in OOP + procedural)
- Transition → MODERATE (testing, refining, documentation)

The RUP Hump Diagram illustrates the overall effort distribution across the main phases of the Rational Unified Process. Each phase emphasizes different software engineering activities such as business modeling, requirements gathering, analysis, design, implementation, and testing.

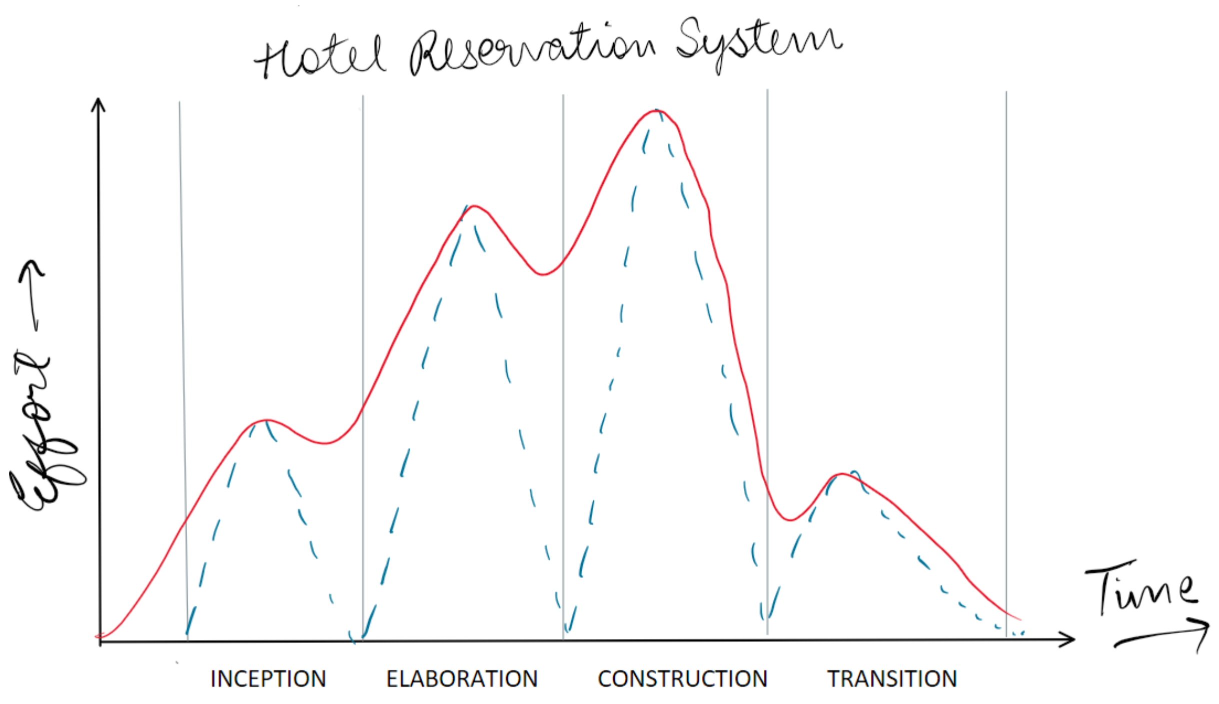
In the **Inception Phase**, the effort is focused on understanding the problem, identifying actors, preparing the problem statement, and defining the system scope. This phase has moderate activity.

During the **Elaboration Phase**, the analytical effort reaches its peak. Here, UML modeling (Use Case Diagram, Class Diagram, Activity Diagram), architecture definition, and risk identification dominate. Design decisions are finalized in this phase.

The **Construction Phase** shows the highest coding and implementation activity. Both the Object-Oriented and Procedural versions of the Hotel Reservation System were developed in this phase. Functional verification, integration of modules, and system-building efforts peak here.

Finally, in the **Transition Phase**, effort shifts toward testing, refinement, bug fixing, performance evaluation, documentation, demonstration, and deployment preparation. User feedback is integrated, and the system is finalized.

This diagram demonstrates how RUP supports iterative development while ensuring systematic progress from requirements to a fully working software system.



The development of the **Hotel Reservation System** successfully demonstrated the complete application of the **Rational Unified Process (RUP)** across its four major phases: Inception, Elaboration, Construction, and Transition.
Through systematic planning, analysis, design, implementation, and testing, a functional and structured software solution was produced.

The **Inception phase** clarified the project scope, actors, and problem domain.
The **Elaboration phase** transformed these requirements into detailed system models using UML diagrams, identifying architectural components and workflows.
During the **Construction phase**, the system was implemented using both **Object-Oriented Java** and **Procedural Java**, showcasing different programming paradigms for the same problem.

Finally, the **Transition phase** validated the system through testing, evaluation, and refinement, ensuring that the solution meets the intended objectives.

The project provided valuable hands-on experience in OOAD, UML modelling, Java development, and software engineering best practices. It also highlighted the importance of structured methodologies such as RUP in delivering reliable and maintainable software systems. Overall, the Hotel Reservation System stands as a robust foundation that can be extended in the future with advanced features like databases, GUIs, online integration, and cloud deployment.

End

Group Members:

1. SOMYA VATS- 500119012
2. TANISHA – 500125283
3. TEJASVI HAZARIKA- 500119060

Batch 1 Data Science