**THE NORTHCAP UNIVERSITY**

**Sector 23A, Gurgaon, Haryana, 122017**

**Report on Exam Scheduling System**

**By:**

**Utkarsh Kumar (23CSU405)**

**For:**

**Dr. Anuradha Dhull**

**DATA STRUCTURES**

**November 14, 2024**

**COPYWRITE NOTICE**

This project report, titled "**Exam Scheduling System"**

is protected under copyright law. No part of this report may be

reproduced, distributed, or transmitted in any form or by any

means, including photocopying, recording, or other electronic or

mechanical methods, without the prior written permission of the

author, except in the case of brief quotations embodied in critical

reviews and certain other non-commercial uses permitted by

copyright law.

Unauthorized use of this report may result in legal action.

**© 2024 Utkarsh Kumar. All rights reserved**.

FORWARDING LETTER

To

The Northcap University

November 14, 2024

Subject: Report: **Exam Scheduling System**

Dear Anuradha Ma'am,

I am pleased to submit the report titled "**Exam Scheduler System**" for your kind consideration. The report outlines the design, implementation, and functionality of a Java-based application developed to streamline the scheduling of examinations.

This system effectively addresses key challenges such as room allocation, conflict-free scheduling for instructors, and optimizing resources. It is designed to automate the process, reduce manual errors, and provide a seamless scheduling experience for educational institutions.

The report covers the following aspects:

1. Objectives of the system.
2. Details of system components and their functionalities.
3. Workflow, features, and code execution.
4. Advantages, limitations, and potential future enhancements.

I believe this project demonstrates a practical solution to managing exam schedules and reflects the potential for further development to meet broader institutional requirements. I look forward to your feedback and any suggestions for improvement.

Thank you for your time and attention. Please do not hesitate to contact me at [Your Email Address] or [Your Phone Number] should you have any questions or need additional details.

Sincerely,

Utkarsh Kumar

**Preface**

**The Exam Scheduler System** report is the culmination of a project aimed at addressing the complexities and challenges associated with scheduling exams in academic institutions. With advancements in technology and increasing administrative demands, it has become essential to automate processes to enhance efficiency and accuracy.

**This report provides a detailed account of the design, development, and functionality of a Java-based application tailored to streamline the scheduling of exams.** The system ensures effective room allocation, prevents scheduling conflicts, and optimizes the use of available resources. The project demonstrates how modern programming techniques can solve real-world problems in educational administration.

Through this system, the goal is to reduce the workload on administrative staff, improve coordination, and provide a robust framework that can be scaled and adapted to meet institutional needs. While the current implementation serves as a functional prototype, the report also highlights potential improvements and future extensions that could make the system more versatile and comprehensive.

I hope this report will serve as an informative resource for readers, offering insights into the practical application of programming concepts to build efficient solutions.

I would like to extend my gratitude to **Dr. Anuradha** in completing this project. Their encouragement and feedback have been invaluable in refining this work.

[Your Name]  
[Your Title/Position]  
[Date]

**Acknowledgment**

I would like to express my deepest gratitude to everyone who contributed to the successful completion of this report on the Exam Scheduler System.

**First and foremost, I extend my sincere thanks to Dr Anuradha Dhull** whose invaluable guidance, insightful feedback, and constant encouragement were instrumental throughout this project. Their expertise and support helped me overcome challenges and refine the system to its current form.

I am also grateful to The NorthCap University for providing the resources and environment conducive to learning and development. The knowledge and skills gained during this project have been immensely rewarding.

Additionally, I would like to acknowledge the support of my peers and colleagues, whose collaborative discussions and suggestions played a significant role in shaping this work.

This report is a reflection of all the guidance, efforts, and collaboration I received, and I am truly thankful to everyone who contributed to this journey.

Utkarsh Kumar

14th November 2024

**Table of Contents**

1. **Preface**
2. **Forwarding Letter**
3. **Acknowledgment**
4. **Introduction**4.1 Objectives  
   4.2 Purpose of the System
5. **System Overview**5.1 Components of the Exam Scheduler System  
   5.2 Workflow
6. **Detailed Description of Components**  
   6.1 Course Class  
   6.2 Room Class  
   6.3 Exam Class  
   6.4 ExamScheduler Class
7. **Workflow and Features**  
   7.1 Room Allocation  
   7.2 Conflict Avoidance  
   7.3 Exam Scheduling
8. **Code Walkthrough**  
   8.1 Sample Execution  
   8.2 Output
9. **Advantages of the System**
10. **Limitations of the System**
11. **Future Enhancements**
12. **Conclusion**

**Report: Exam Scheduler System**

**Introduction**

The **Exam Scheduler System** is a Java-based application that manages the scheduling of exams for various courses. The system is designed to optimize the allocation of rooms, ensure there are no scheduling conflicts for instructors, and accommodate class sizes. This report provides an overview of the code, its functionality, and the architecture of the system, along with its benefits and limitations.

**Objectives**

The Exam Scheduler System aims to:

1. Automate the scheduling process for exams.
2. Optimize room allocation based on capacity and availability.
3. Avoid scheduling conflicts for instructors and students.
4. Provide a streamlined view of the exam schedule.

**System Components**

**1. Course Class**

This class represents a course in the system, containing:

* courseCode: A unique identifier for the course.
* courseName: The name of the course.

**Methods:**

* Getters for courseCode and courseName.

**2. Room Class**

This class models a room available for exams, containing:

* roomId: A unique identifier for the room.
* capacity: The seating capacity of the room.
* isAvailable: A boolean indicating if the room is available.

**Methods:**

* Getters for room details.
* setAvailability: Updates the room's availability status.

**3. Exam Class**

This class represents an exam, containing:

* course: The associated course.
* date: The date and time of the exam.
* duration: The duration of the exam in minutes.
* room: The room allocated for the exam.
* instructor: The instructor supervising the exam.

**Methods:**

* Getters for all properties.

**4. ExamScheduler Class**

The core of the system, responsible for managing exams, rooms, and schedules.

**Key Methods:**

* **addRoom(Room room)**  
  Adds a room to the system.
* **scheduleExam(Course course, Date date, int duration, String instructor, int classSize)**  
  Schedules an exam by allocating a room and ensuring no conflicts.
* **allocateRoom(int classSize, Date date)**  
  Allocates a room based on class size and availability.
* **isInstructorBusy(String instructor, Date date)**  
  Checks if the instructor is available.
* **addInstructorSchedule(String instructor, Date date)**  
  Updates the instructor's schedule.
* **displaySchedule()**  
  Displays the list of scheduled exams.

**Workflow and Features**

**1. Room Allocation**

Rooms are allocated based on:

* Class size: Ensuring the room can accommodate the students.
* Availability: Ensuring the room is not already in use.

**2. Conflict Avoidance**

* **Instructor Conflicts**: Ensures instructors are not assigned to multiple exams at the same time.
* **Student Conflicts**: While not implemented fully, provisions exist to avoid overlapping schedules for students.

**3. Exam Scheduling**

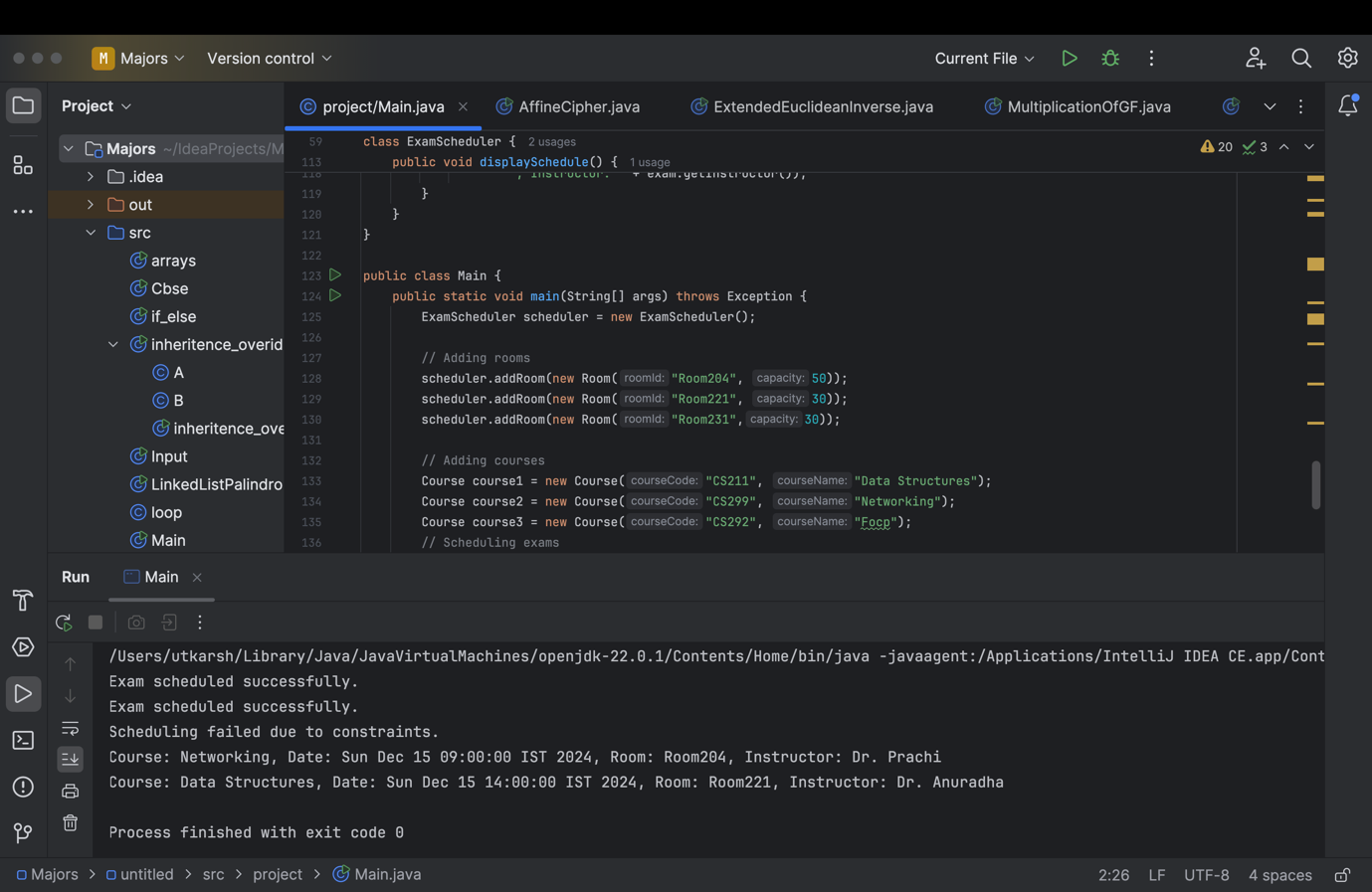
The system schedules exams by:

* Assigning an available room.
* Validating instructor and room availability.
* Logging the schedule for display and management.

**Code Walkthrough**

**Raw code**

**package project;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
import java.util.Date;  
class Course {  
 private String courseCode;  
 private String courseName;  
  
 public Course(String courseCode, String courseName) {  
 this.courseCode = courseCode;  
 this.courseName = courseName;  
 }  
  
 // Getters and Setters  
 public String getCourseCode() { return courseCode; }  
 public String getCourseName() { return courseName; }  
}  
  
class Room {  
 private String roomId;  
 private int capacity;  
 private boolean isAvailable;  
  
 public Room(String roomId, int capacity) {  
 this.roomId = roomId;  
 this.capacity = capacity;  
 this.isAvailable = true;  
 }  
  
 // Getters and Setters  
 public String getRoomId() { return roomId; }  
 public int getCapacity() { return capacity; }  
 public boolean isAvailable() { return isAvailable; }  
 public void setAvailability(boolean available) { isAvailable = available; }  
}  
  
class Exam {  
 private Course course;  
 private Date date;  
 private int duration; // in minutes  
 private Room room;  
 private String instructor;  
  
 public Exam(Course course, Date date, int duration, Room room, String instructor) {  
 this.course = course;  
 this.date = date;  
 this.duration = duration;  
 this.room = room;  
 this.instructor = instructor;  
 }  
  
 // Getters and Setters  
 public Course getCourse() { return course; }  
 public Date getDate() { return date; }  
 public int getDuration() { return duration; }  
 public Room getRoom() { return room; }  
 public String getInstructor() { return instructor; }  
}  
class ExamScheduler {  
 private List<Exam> exams = new ArrayList<>();  
 private List<Room> rooms = new ArrayList<>();  
 private Map<String, List<Date>> studentSchedules = new HashMap<>();  
 private Map<String, List<Date>> instructorSchedules = new HashMap<>();  
  
 // Add room  
 public void addRoom(Room room) {  
 rooms.add(room);  
 }  
  
 // Schedule an exam  
 public boolean scheduleExam(Course course, Date date, int duration, String instructor, int classSize) {  
 Room allocatedRoom = allocateRoom(classSize, date);  
 if (allocatedRoom == null || isInstructorBusy(instructor, date) || hasStudentConflict(course, date)) {  
 System.*out*.println("Scheduling failed due to constraints.");  
 return false;  
 }  
  
 Exam exam = new Exam(course, date, duration, allocatedRoom, instructor);  
 exams.add(exam);  
 allocatedRoom.setAvailability(false);  
 addInstructorSchedule(instructor, date);  
 System.*out*.println("Exam scheduled successfully.");  
 return true;  
 }  
  
 // Check room availability  
 private Room allocateRoom(int classSize, Date date) {  
 for (Room room : rooms) {  
 if (room.getCapacity() >= classSize && room.isAvailable()) {  
 return room;  
 }  
 }  
 return null;  
 }  
  
 // Check instructor's schedule  
 private boolean isInstructorBusy(String instructor, Date date) {  
 return instructorSchedules.getOrDefault(instructor, new ArrayList<>()).contains(date);  
 }  
  
 // Check for student conflicts  
 private boolean hasStudentConflict(Course course, Date date) {  
 return false;  
 }  
  
 // Add instructor schedule  
 private void addInstructorSchedule(String instructor, Date date) {  
 instructorSchedules.putIfAbsent(instructor, new ArrayList<>());  
 instructorSchedules.get(instructor).add(date);  
 }  
  
 // Display the schedule  
 public void displaySchedule() {  
 for (Exam exam : exams) {  
 System.*out*.println("Course: " + exam.getCourse().getCourseName() +  
 ", Date: " + exam.getDate() +  
 ", Room: " + exam.getRoom().getRoomId() +  
 ", Instructor: " + exam.getInstructor());  
 }  
 }  
}  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 ExamScheduler scheduler = new ExamScheduler();  
  
 // Adding rooms  
 scheduler.addRoom(new Room("Room204", 50));  
 scheduler.addRoom(new Room("Room221", 30));  
 scheduler.addRoom(new Room("Room231",30));  
  
 // Adding courses  
 Course course1 = new Course("CS211", "Data Structures");  
 Course course2 = new Course("CS299", "Networking");  
 Course course3 = new Course("CS292", "Focp");  
 // Scheduling exams  
 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");  
 scheduler.scheduleExam(course2, sdf.parse("2024-12-15 09:00"), 120, "Dr. Prachi", 40);  
 scheduler.scheduleExam(course1, sdf.parse("2024-12-15 14:00"), 90, "Dr. Anuradha", 20);  
 scheduler.scheduleExam(course3, sdf.parse("2024-12-15 09:00"), 120, "Dr. Garima", 40);  
  
 // Display schedule  
 scheduler.displaySchedule();  
 }  
}**

****

**Sample Execution**

The main method initializes the system:

1. Rooms are added:
2. scheduler.addRoom(new Room("Room204", 50));
3. scheduler.addRoom(new Room("Room221", 30));
4. scheduler.addRoom(new Room("Room231",30));
5. Courses are created:
6. Course course1 = new Course("CS211", "Data Structures");
7. Course course2 = new Course("CS299", "Networking");
8. Course course3 = new Course("CS292", "Focp");
9. Exams are scheduled:
10. scheduler.scheduleExam(course2, sdf.parse("2024-12-15 09:00"), 120, "Dr. Prachi", 40);
11. scheduler.scheduleExam(course1, sdf.parse("2024-12-15 14:00"), 90, "Dr. Anuradha", 20);
12. scheduler.scheduleExam(course3, sdf.parse("2024-12-15 09:00"), 120, "Dr. Garima", 40);
13. Schedule is displayed:
14. scheduler.displaySchedule();

**Output**

The system generates a schedule:

Course: Networking, Date: Sun Dec 15 09:00:00 IST 2024, Room: Room204, Instructor: Dr. Prachi

Course: Data Structures, Date: Sun Dec 15 14:00:00 IST 2024, Room: Room221, Instructor: Dr. Anuradha

**Advantages**

1. **Automation**: Reduces manual intervention and errors.
2. **Scalability**: Can handle multiple courses, rooms, and exams.
3. **Conflict Management**: Ensures proper scheduling without overlapping resources.

**Limitations**

1. **Student Conflict Checking**: Not yet implemented.
2. **Room Utilization**: Rooms are marked unavailable after allocation but are not reassigned even when unused.
3. **Instructor Availability**: Limited to checking exact match on date and time; does not consider overlapping durations.

**Future Enhancements**

1. **Student Conflict Resolution**: Implementing detailed checks for overlapping schedules.
2. **Room Optimization**: Allow partial utilization of rooms for smaller classes.
3. **Enhanced Scheduling Algorithms**: Incorporating AI/ML techniques for more efficient scheduling.

**Conclusion**

The **Exam Scheduler System** effectively automates the scheduling process while avoiding conflicts for instructors and optimizing room usage. With further development, it can evolve into a robust solution for educational institutions.