



**Wachemo University**

**College of Engineering and Technology**

**Department of Software Engineering**

**Project Work for 3rd year Software Engineering Student**

**Course Title: - Software Engineering Tools and practice**

**Project Title:Online Examination System**

**Group-5 Members**

**Name ID**

**1.Mikias Tadesse ……………………………………………….1501116**

**2. Sisay Kebede …………………………………………………1501279**

**3.Abel Adam ………………………………………………………1500592**

**4.Adisu Moga ……………………………………………………..1500632**

**5.Kaleb Kebede ………………………………………………….1501009**

Summited date:May 15, 2025

Summited to:**Mr. Feyisa K.**

**Table of Contents**

[Acknowledgment 2](#_Toc6075)

[Abstract 3](#_Toc10443)

[CHAPTER ONE 3](#_Toc7443)

[1 .1 Requirement Analysis (Use Cases) 3](#_Toc4733)

[Functional Requirements 4](#_Toc8558)

[Non-Functional Requirements 5](#_Toc13633)

[1.2. Use Case Diagram Components 6](#_Toc19781)

[1.3 . Example of use case model 7](#_Toc2675)

[1.4 Use Case Description/Templates 8](#_Toc1064)

[1.5 Tools and steps to draw Use Case 27](#_Toc126)

[CHAPTER TWO 28](#_Toc6023)

[2.1 High Level Sequence Diagram 28](#_Toc5375)

[2.2 Component of High level Sequence Diagram 29](#_Toc24784)

[2.3 Example of High Level Sequence 30](#_Toc19682)

[2.4 Tools and Steps to Draw High Level Sequence Diagram 33](#_Toc1422)

[CHAPTER THREE 34](#_Toc25232)

[3.1 Low-level (Detail) Design (class design) 34](#_Toc32107)

[3.4 Tools and Steps to Draw class Diagram 36](#_Toc615)

[CHAPTER-FOUR 37](#_Toc28988)

[4.1.Implementation (export class diagram into code and update code and diagram) 37](#_Toc28689)

[4.2 Steps to Generate Code from Class Diagram 37](#_Toc903)

[CHAPTER-FIVE 45](#_Toc887)

[5.1 Change Management (version control using Git) 45](#_Toc22050)

[5.2 Steps and tools that we use in our project to implements git. 45](#_Toc8800)

[Chapter Six 49](#_Toc32217)

[6.1 Unit Test 49](#_Toc17409)

[6.2 Steps and tools used in Unit Test 49](#_Toc23619)

[Chapter -7 52](#_Toc13826)

[7.1 Build (prepare build script for compilation, unit test, jar file creation) 52](#_Toc22821)

[7.2 Steps and tool that we use in our project to implement build 52](#_Toc22219)

# ****Acknowledgment****

We would like to extend our sincere gratitude to all individuals and institutions that supported and guided us throughout the development of this project. First and foremost, we are grateful to **Wachemo University**, particularly the **College of Engineering and Technology** and the **Department of Software Engineering**, for providing us with the platform and resources to carry out this work.

Our deepest thanks go to our instructors and supervisors for their valuable insights, continuous encouragement, and constructive feedback during the course. We also acknowledge the support of departmental staff, whose cooperation during the requirement analysis phase greatly contributed to our understanding of the system's needs.

Lastly, we appreciate the dedication and collaboration of our project team members, whose joint effort and perseverance made this project possible.

# ****Abstract****

This project presents the design and development of an **Online Examination System**, undertaken as part of the Software Engineering Tools and Practice course. The system aims to streamline the examination process in academic institutions by automating exam creation, management, scheduling, participation, and result tracking.

The development process followed a structured approach, beginning with **requirement analysis** using use cases, followed by **design modeling** with UML diagrams such as use case diagrams, high-level sequence diagrams, and class diagrams. The system was then implemented using **Java**, with design-to-code synchronization achieved through **Visual Paradigm**.

Version control was managed using **Git and GitHub**, enabling team collaboration, code tracking, and change management. **JUnit** and **Eclipse IDE** were employed for unit testing, ensuring the correctness of individual modules. Finally, the project included a structured **build process**, resulting in a deployable JAR file.

This project showcases the practical application of software engineering principles, emphasizing modularity, maintainability, scalability, and real-time collaboration in a team-based development environment.

# CHAPTER ONE

# 1 .1 Requirement Analysis (Use Cases)

**Requirement Analysis** is the process of identifying, gathering, analyzing, and documenting the needs and expectations of stakeholders for a new or modified system. It involves understanding, documenting, and managing the needs and expectations of stakeholders.

As 3rd-year Software Engineering students at Wachemo University, we began our Online Examination System project by analyzing system requirements based on stakeholder expectations including students, instructors, and administrators.We conducted our requirement analysis through interviews with department head staff, document analysis, and observation.These methods helped us gather real-world insights into the system’s requirements, user behaviors, and existing manual processes.

## Functional Requirements

1. **Login**: Users (Admin, Instructor, and Student) must authenticate using login credentials to access the system.

2. **Logou**t: Allows users to safely exit their session, ensuring data protection and access control.

3. **Update Password**: Enables all users to change their passwords for enhanced account security.

4.**View Admin Dashboard**: Admin can view a centralized dashboard showing system status and management options.

5. **Assign Instructors**: Admin assigns instructors to specific roles or courses.

6. **Assign Students**: Admin enrolls students into the system or into specific

courses.

7. **Manage Instructors**: Admin adds, updates, or deletes instructor profiles.

8.**Manage Students**: Admin handles student account creation, editing, or deletion.

9**.Add Course**: Admin can create new courses for exams to be associated with.

10.**Manage Courses**: Admin edits or deletes existing course information.

11. **Schedule Exam**: Admin schedules new exams by setting time, date, and assigning to courses.

12. **Manage Schedule**: Admin updates or removes scheduled exams.

13.**All Feedbacks**: Admin reviews all feedback given by students.

14. V**iew Instructor Dashboard**: Instructor accesses a personalized interface with teaching and exam tools.

15.**Create Exam**: Instructor can create new exams for assigned courses.

16. **Edit Exam**: Instructor modifies exam details and questions.

17.**Delete Exam**: Instructor deletes an unwanted or outdated exam.

18**.Manage Exam**: Broad functionality for instructors to control exam creation, editing, and publishing.

19. **Generate Exam Report**: Instructor generates analytical reports on performance.

20. **Feedbacks**: Instructor can view and respond to feedback submitted by students.

21. **View Student Result**: Instructor checks student results for performance evaluation.

22. **Available Exam**: Student views exams currently open for participation.

23. **View Student Dashboard**: Student views a summary of available exams, results, and feedback options.

24. **View Exam Schedule**: Student checks upcoming exam dates and times.

25.**Taken Exams**: Student can view a history of completed exams.

26.**View Result**: Student sees the results of completed exams.

27.**Add Feedback**: Student submits feedback after viewing their exam results.

## Non-Functional Requirements

1. **Security:** The system should ensure only authorized users can access specific features, protecting sensitive data.

2. **Performance**: The application should load and respond quickly, even under peak usage conditions.

3. **Usability**: The user interface should be simple and intuitive, requiring minimal learning.

4.**Reliability**: The system must operate consistently, handling errors gracefully and maintaining uptime.

5. **Scalability:** The system should support more users and courses as the university grows.

6. **Maintainability**: Code and system components should be modular and easy to update or fix.

# 1.2. Use Case Diagram Components

A **use case** is a brief description of how a user (called an **actor**) interacts with a system to achieve a specific goal. It outlines the steps or actions taken by the user and the system to complete a task. Use cases help in understanding **system functionality** from the user's perspective

Is a high-level visual representation of user interactions with a system. Its core components include:

• **Actors:** Represent the external users or systems that interact with the system (e.g., Admin, Instructor, Student).

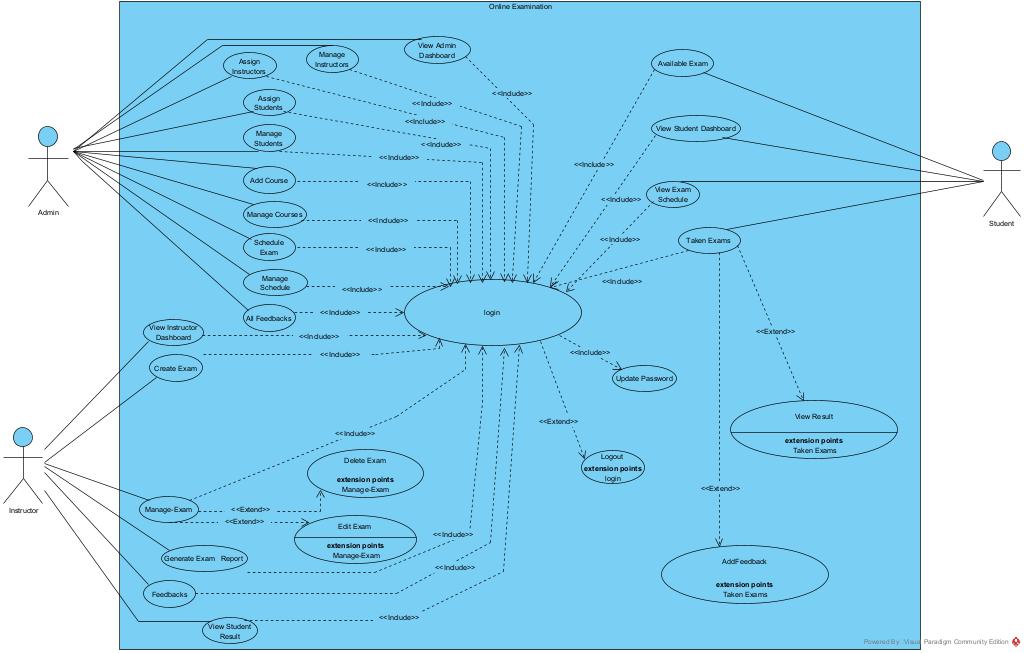
• **Use Cases:** These are the functionalities provided by the system, represented as ovals with actions inside.

• **System Boundary**: A rectangular box that shows the scope of the system; all use cases fall within this boundary.

• **Relationships**: Connections between actors and use cases, or between use cases themselves. Includes:

* **Association**: A straight line showing direct interaction between an actor and a use case.
* **Include**: A dashed arrow showing that a use case always uses another use case as part of its behavior.
* **Extend**: A dashed arrow indicating optional or conditional behavior added to a base use case.
* **Generalization**: An arrow representing inheritance between actors or use cases (e.g., specialized roles).

# 1.3 . Example of use case model



# 1.4 Use Case Description/Templates

Use Case Templates for Online Examination System

UC-1: Login

|  |  |  |
| --- | --- | --- |
| Field | Decription | |
| Use Case ID | UC-01 | |
| Use Case Name | Login | |
| Actor | Admin, Instructor , Student | |
| Summary | Allowa user to log into the system using vaild credentials. | |
| Precondition | The system must be online. | |
| Basic Scenario | Actor Action | System Response |
| Step1.User enters username and password.  Step2.User submits the login form | Step3.system verfies credential  Step4.System redirects the user to the respective dashboard. |
| Alternative Scenario | If the credentials are incorrect, the system displays an error message. | |
| Post Condition | The user is authenticated and logged into the system. | |

|  |  |  |
| --- | --- | --- |
| Field | Decription | |
| Use Case ID | UC-02 | |
| Use Case Name | Logout | |
| Actor | All user(Admin, Instructor , Student) | |
| Summary | Logs the user out of the system securely. | |
| Precondition | The user must be logged in. | |
| Basic Scenario | Actor Action | Sytem Rsponse |
| Step1: The user clicks on the 'Logout’ button. | Step2: The system ends the user session and redirects to the login page. |
| Alternative Scenario | If the session ends unexpectedly, the system redirects the user to the login page upon next access. | |
| Post Condition | User session is terminated. | |

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case Id | UC-03 | |
| Use Case Name | Update Password | |
| Actor | All user(Admin, Instructor, Student) | |
| Summary | Allows users to update their password. | |
| Precondition | The user is logged in. | |
| Basic Scenario | Actor Action | System Rsponse |
| Step1: The user navigates to password update section.  Step3.user enters the new password(and confirms it).  Step4.The user submits the new password. | Step2: The system prompts for current and new password.  Step5.The system validates and updates the password. |
| Alternative Scenario | If the new password is weak, system prompts error. | |
| Post Condition | Password is updated. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-04 | |
| Use Case Name | View Admin Dashboard | |
| Actor | Admin | |
| Summary | Displays information and quick links for Admin. | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | [Actor Action | System Rsponse |
| Step1: Admin logs in | Ste2: The system displays admin dashboard. |
| Alternative Scenario | If dashboard fails to load, system shows error message. | |
| Post Condition | Admin views dashboard. | |

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Description | | |
| Use Case ID | UC -05 | | |
| Use Case Name | Assign Instructors | | |
| Actor | Admin | | |
| Summary | Admin assigns instructors to specific courses. | | |
| Precondition | 1.Admin must be logged in.  2.Courses and instructors must exist | | |
| Basic Scenario | Actor Action | System Response | |
| Step1: Admin selects a instructor.  Step3: Admin selects and assigns an instructor. | Step2.System displays list of courses and instructors  Step4: System confirms and saves the assignment. | |
|  | |  |
| Alternative Scenario | If the instructor is already assigned, show a conflict error. | | |
| Post Condition | Instructor is assigned to course. | | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-06 | |
| Use Case Name | Manage Instructors | |
| Actor | Admin | |
| Summary | Allows admin to view, update, or remove instructors. | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin selects the instructor management option.  Step3: Admin chooses an instructor to edit or delete | Step2.System displays the list of instructors.  Step4: System performs the update or deletion. |
| Alternative Scenario | If no instructor is selected or input is invalid, system shows an error. | |
| Post Condition | Instructor details are updated or removed. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-07 | |
| Use Case Name | Assign Students | |
| Actor | Admin | |
| Summary | Assign students to a course. | |
| Precondition | 1.Admin must be logged in.  2.Courses and students must exist. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin selects ‘Assign Students’.  Step3: Admin assigns students to the course. | Step2: System displays a list of students and course   |  | | --- | | Step4: System saves the assignment. | |
| Alternative Scenario | If selection fails, system prompts with an error message. | |
| Post Condition | Student-course assignment is saved. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-08 | |
| Use Case Name | Manage Students | |
| Actor | Admin | |
| Summary | Admin can edit or delete student assignments. | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step-1 Admin select ‘Manage Student’  Step3.Admin edits or removes assignments. | Step1: System lists current student assignments.  Step2: System updates the dat |
| Alternative Scenario | If invalid data is entered, system displays an error. | |
| Post Condition | Student assignments are updated. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-09 | |
| Use Case Name | Add Course | |
| Actor | Admin | |
| Summary | Admin adds a new course to the platform | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin opens course creation.  Step3: Admin enters course details and submits. | Step2: System displays input form.  Step4: System validates and stores the course. |
| Alternative Scenario | If duplicate course name exists, system rejects it. | |
| Post Condition | Course is added to the system. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-10 | |
| Use Case Name | Manage Courses | |
| Actor | Admin | |
| Summary | Admin edits or deletes existing courses. | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin selects a course to manage  Step3: Admin updates or deletes the course. | Step2: System shows course details.  Step4: System saves changes or removes the course. |
| Alternative Scenario | System shows errors if condition not met or validation fails. | |
| Post Condition | Course data changes are saved in the system | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-11 | |
| Use Case Name | Schedule Exam | |
| Actor | Admin | |
| Summary | Admin schedules an exam for a course. | |
| Precondition | 1.Admin must be logged in.  2.Course and instructor must be assigned | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin opens the ‘Schedule Exam’.  Step3: Admin sets date and time. | Step2: System displays course and time options.  Step4: System validates saves schedule |
| Alternative Scenario | If time overlaps or input is invalid, system shows error | |
| Post Condition | Exam is scheduled. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-11 | |
| Use Case Name | Manage Schedule | |
| Actor | Admin | |
| Summary | Modify or delete exam schedules. | |
| Precondition | 1.Admin must be logged in.  2.Exams must already be scheduled. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin selects scheduled exam  Step3: Admin updates or deletes schedule | Step2: System displays current schedule  Step4: System confirms and saves changes. |
| Alternative Scenario | If exam has already started, system blocks modification. | |
| Post Condition | Exam schedule is updated or removed. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-12 | |
| Use Case Name | View All Feedbacks | |
| Actor | Admin | |
| Summary | View feedback submitted by students and Instructor. | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Admin opens the feedback section | Step2: System displays all collected feedback. |
| Alternative Scenario | If feedback cannot be retrieved, show error message | |
| Post Condition | Admin views all feedback records. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-13 | |
| Use Case Name | Generate Exam Report | |
| Actor | Admin | |
| Summary | Admin generates performance reports for exams. | |
| Precondition | 1.Admin must be logged in.  2.Exam must be completedted | |
| Basic Scenario | Actor Action | System Response |
| Step1. Admin selects ‘Generate exam report’ | Step2: System generates and displays the report. |
| Alternative Scenario | If report fails, system shows error. | |
| Post Condition | Report is available for Admin. | |

|  |  |  |
| --- | --- | --- |
| Field | Dscription | |
| Use Case ID | UC-14 | |
| Use Case Name | View Admin Dashboard | |
| Actor | Admin | |
| Summary | Access Admin Dashboard | |
| Precondition | Admin must be logged in. | |
| Basic Scenario | [Actor Action | System Response |
| Step1: Admin logs in | Step1: System shows admin dashboard with data panels |
| Alternative Scenario | system shows error if dashboard fails to load. | |
| Post Condition | Dashboard is displayed with system overview. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-15 | |
| Use Case Name | Create Exam | |
| Actor | Instructor | |
| Summary | Instructor creates an exam for an assigned course. | |
| Precondition | 1.Instructor must be logged in.  2.The course must be assigned to the instructor. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor selects the course which to create an exam.  Step3: Instructor enters exam details(name ,duration,questions) | Step2: System displays exam creation form.  Step4: System validates and stores the exam |
| Alternative Scenario | If data is incomplete or invalid, system shows error. | |
| Post Condition | Exam is created and stored. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-16 | |
| Use Case Name | Manage Exam | |
| Actor | Instructor | |
| Summary | Instructor updates or deletes an existing exam. | |
| Precondition | 1.Instructor must be logged in.  2.Exam must be previously created. | |
| Basic Scenario | Actor Action | System Response |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-17 | |
| Use Case Name | Delete Exam | |
| Actor | Instructor | |
| Summary | Instructor deletes a selected exam | |
| Precondition | 1.Instructor must be logged in.  2.Exam must not be taken yet. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor chooses an exam  Step3: Instructor confirms deletion. | Step2: System displays exam details.  Step4: System deletes the exam. |
| Alternative Scenario | If exam is already taken, deletion is not allowed. | |
| Post Condition | Exam is removed from system. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-18 | |
| Use Case Name | Edit Exam | |
| Actor | Instructor | |
| Summary | Instructor modifies questions or settings of an exam. | |
| Precondition | Instructor must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor opens an exam  Step2: Instructor modifies details | Step1: System shows editable fields.  Step2: System validates and updates data. |
| Alternative Scenario | If edits are invalid, system shows error. | |
| Post Condition | Exam is updated. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-19 | |
| Use Case Name | Generate Exam Report | |
| Actor | Instructor | |
| Summary | Generate report for a completed exam. | |
| Precondition | 1.Instructor must be logged in.  2.Exam must be completed. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor navigates to the reports section.  Step2: Instructor selects their exam. | Step3: System generates displays the exam reports. |
| Alternative Scenario | If data is missing, report generation fails. | |
| Post Condition | Report is displayed. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-20 | |
| Use Case Name | View Instructor Dashboard | |
| Actor | Instructor | |
| Summary | View summary of courses, exams, and results. | |
| Precondition | Instructor must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor logs in. | Step1: Instructor logs in. |
| Alternative Scenario | If das1board fails, show error message. | |
| Post Condition | Dashboard is displayed. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-21 | |
| Use Case Name | View Student Result | |
| Actor | Instructor | |
| Summary | View results of students for a specific exam. | |
| Precondition | 1.Instructor must be logged in.  2.Exams must be graded. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor selects an exam | Step1: System displays results by student. |
| Alternative Scenario | If exam not graded, results not available. | |
| Post Condition | Results are viewed. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-22 | |
| Use Case Name | Feedbacks | |
| Actor | Instructor | |
| Summary | Instructor views feedback on exams. | |
| Precondition | Instructor must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Instructor selects a completed exam. | Step2: System shows feedback submitted by students. |
| Alternative Scenario | Step2: If no feedback is found, show message. | |
| Post Condition | Instructor reviews feedback. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-23 | |
| Use Case Name | Available Exam | |
| Actor | Student | |
| Summary | Student sees a list of upcoming exams. | |
| Precondition | 1.Student must be logged in.  2.Exam must be scheduled. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Student opens available exams page | |  | | --- | | Step2: System lists all scheduled exams. | |
| Alternative Scenario | If no exams are found, display notice. | |
| Post Condition | Student sees available exams. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-24 | |
| Use Case Name | Taken Exams | |
| Actor | Student | |
| Summary | Take available exam at scheduled time. | |
| Precondition | Student must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1.student navigate to available exams.  Step2.Student selects an available exam.  Step3.Student starts the exam.  Step4.Student answer question. | Step5.system records answers.  Step6.On completion on time expire ,System submits the exam. |
| Alternative Scenario | Technical issue during exam. | |
| Post Condition | Resposes saved and exam status updated. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-25 | |
| Use Case Name | View Result | |
| Actor | Student | |
| Summary | Student views results of completed exams. | |
| Precondition | 1.Student must be logged in.  2.Exam must be graded. | |
| Basic Scenario | Actor Action | System Response |
| Step1.Student navigate to the ‘view result’ section  Step2.Student selects a completed exam | Step3.System shows the result. |
| Alternative Scenario | Result not available | |
| Post Condition | Student sees result. | |

|  |  |  |
| --- | --- | --- |
| Field | Description | |
| Use Case ID | UC-26 | |
| Use Case Name | Add Feedback | |
| Actor | Student | |
| Summary | :Submit feedback on completed exams. | |
| Precondition | 1.Student must be logged in.  2.Exam must be completed. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Student opens feedback form  Step3. Student writes and submits feedback. | Step2: System displays feedback interface  Step4.System saves the feedback. |
| Alternative Scenario | Feedback form fails to submit. | |
| Post Condition | Feedback is recorded. | |

|  |  |  |
| --- | --- | --- |
| Field | Descriptiom | |
| Use Case ID | UC-27 | |
| **Use Case Name** | View Student Dashboard | |
| **Actor** | Student | |
| **Summary** | Student sees upcoming exams, results, and options. | |
| **Precondition** | Student must be logged in. | |
| Basic Scenario | Actor Action | System Response |
| Step1: Student logs in. | Step2: System shows student dashboard |
| **Alternative Scenario** | If dashboard fails to load, show message. | |
| **Post Condition**: | Student sees dashboard with exam info. | |

# 1.5 Tools and steps to draw Use Case

**Tools used**: Viusal Paradigm.

**Steps to Draw Use Case Diagram in Visual Paradigm**

**Step 1**: Open Visual Paradigm

* Launch **Visual Paradigm**.
* Open your project or create a new one.

### Step 2: Create a New Use Case Diagram

### Go to **Diagram** > **New**.

* In the popup:
* Select **UML Diagrams** → **Use Case Diagram**.
* Enter a diagram name (e.g., Online Examination System).
* Click **Create**.

### Step 3: Add System Boundary

### From the **Diagram Toolbar**, select **System** (rectangle with a name at the top).

* Click on the canvas to place it.
* Name it : Online Examination System.

### Step 4: Add Actors

* From the **Diagram Toolbar**, select **Actor** (human stick figure).
* Click on the canvas (outside the system boundary) to place each actor:
* **Admin, Student ,Instructor**

### **Step5**: Add Use Cases

* From the toolbar, select the **Use Case** tool (ellipse shape).
* Click inside the system boundary to add each use case, such as:
* Login,Create Exam,Take Exam,View Results etc.

### Step 6: Connect Actors to Use Cases

* Use the **Association Line** tool (straight line):
* Click on an actor, drag the line to the related use case.
* Repeat for all relationships.

### Step 7: Use <<include>> or <<extend>> if Needed

### To represent dependencies between use cases:

* Use **Include**: for mandatory sub-use cases (e.g., Login → Authenticate User)
* Use **Extend**: for optional behaviors (e.g., Take Exam → Review Instructions)
* Select **Include/Extend** tool from toolbar and connect use cases.

# CHAPTER TWO

# 2.1 High Level Sequence Diagram

The purpose of the **High-Level Sequence Diagram** is to illustrate the overall flow of interaction between major system components and users during an online examination process. It helps visualize how the system behaves in a time-ordered sequence when key actions like logging in, taking the exam, assign instructors, create exam ,view exam schedule are performed.

This diagram provides a **big-picture view** of the major events without diving into low-level details — ideal for understanding the system's core behavior and logic.

# 2.2 Component of High level Sequence Diagram

A **high-level sequence diagram** includes several key components that show **how objects (actors and systems) interact over time** to complete major processes (like taking an online exam).

### ****Actors****: External entities that initiate interaction with the system.

**Examples** :Admin ,Instruuctor and Student

### ****Boundary Lifeline:**** Represents the interface between the **system and external actors** (e.g., users or external sysytem. Handles interaction with users or external systems — receiving input and displaying output..

### ****Control Lifeline:****Represents the **logic** that manages the flow of the interaction. Orchestrates communication between boundary and entity objects, often encapsulating business logic.

### ****Entity Lifeline:****Represents **data or persistent information** in the system. Encapsulates data and logic for managing that data (usually maps to classes or database tables).

### ****Messages****: Arrows that represent communication between lifelines (i.e., method calls or data transfers).

**Types**:**Synchronous** (solid arrow with filled head): Caller waits for a response.

* **Asynchronous** (solid arrow with open head): Caller doesn't wait

### ****Activation Bars****: Thin rectangles on lifelines showing the duration an object is performing an action.

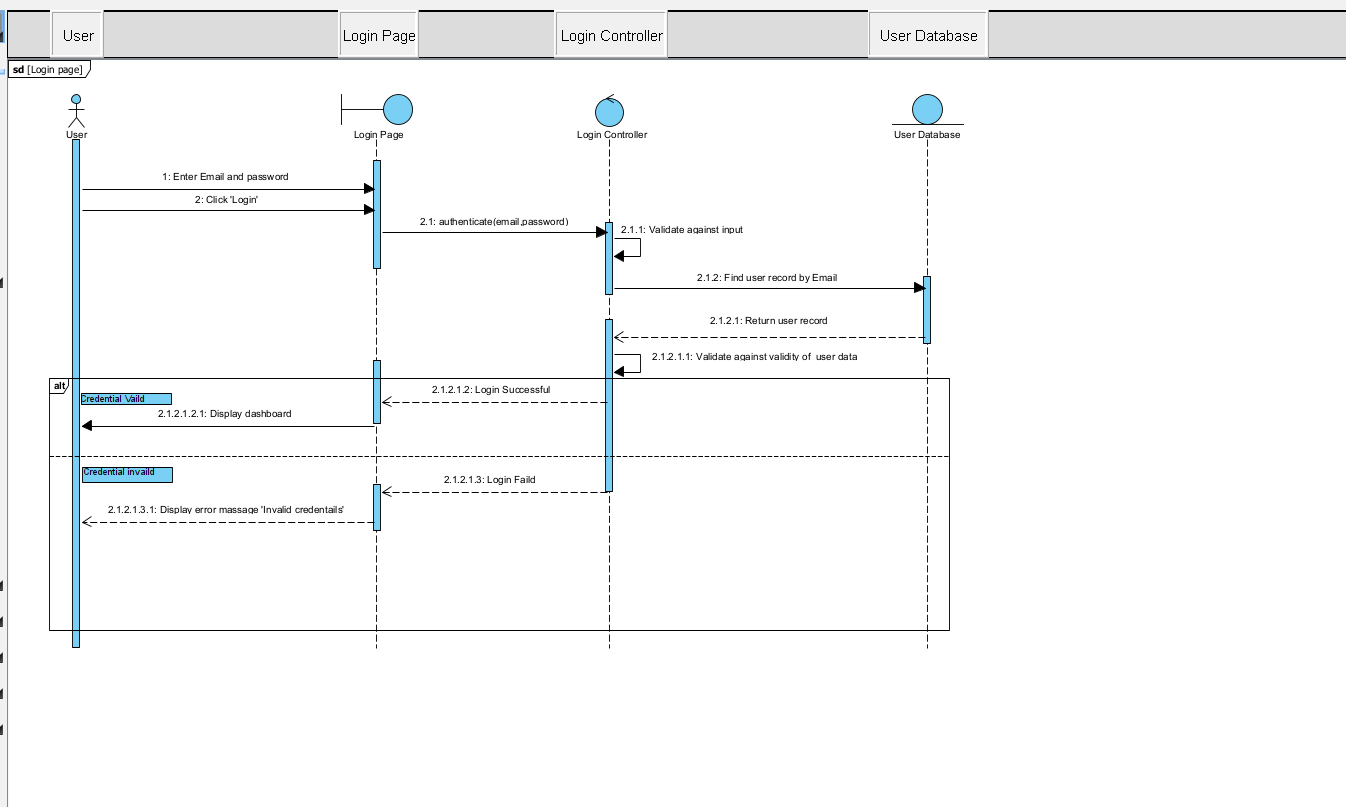
### **Purpose**:Indicate when an object is "active" or processing a message..

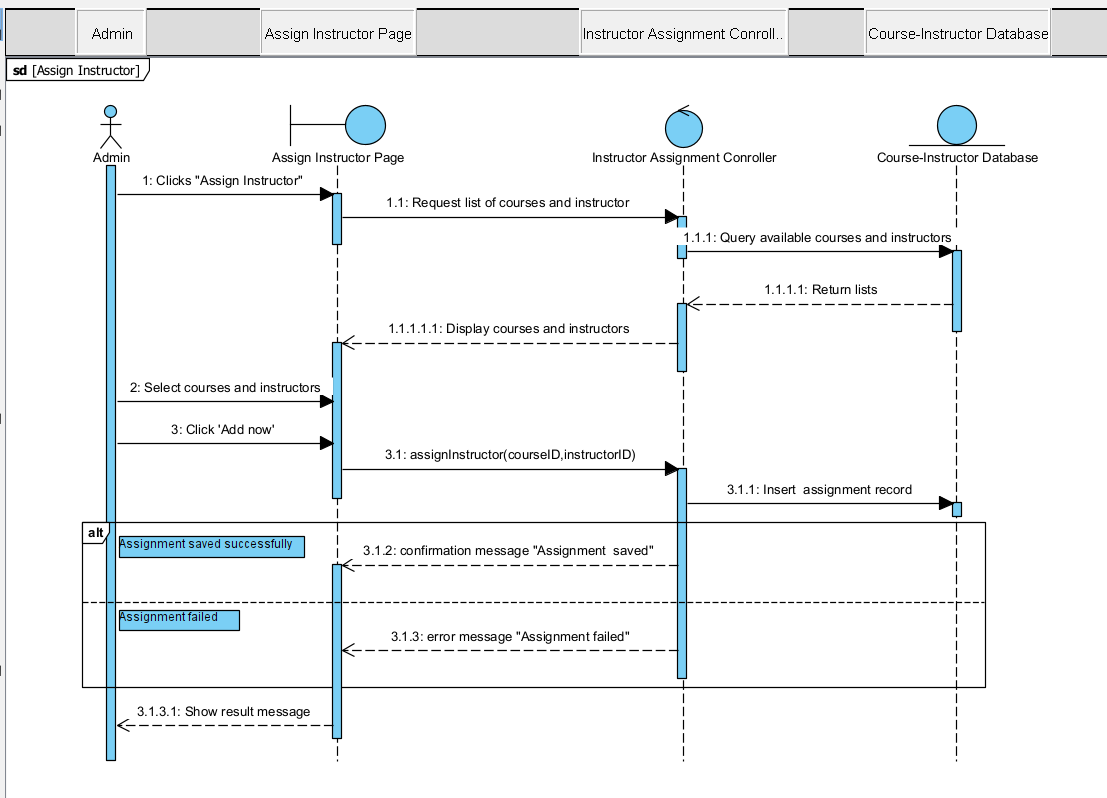
### ****System Components/Objects****: Internal system modules that participate in the process (e.g., Login Form, Auth Controller, UserDB

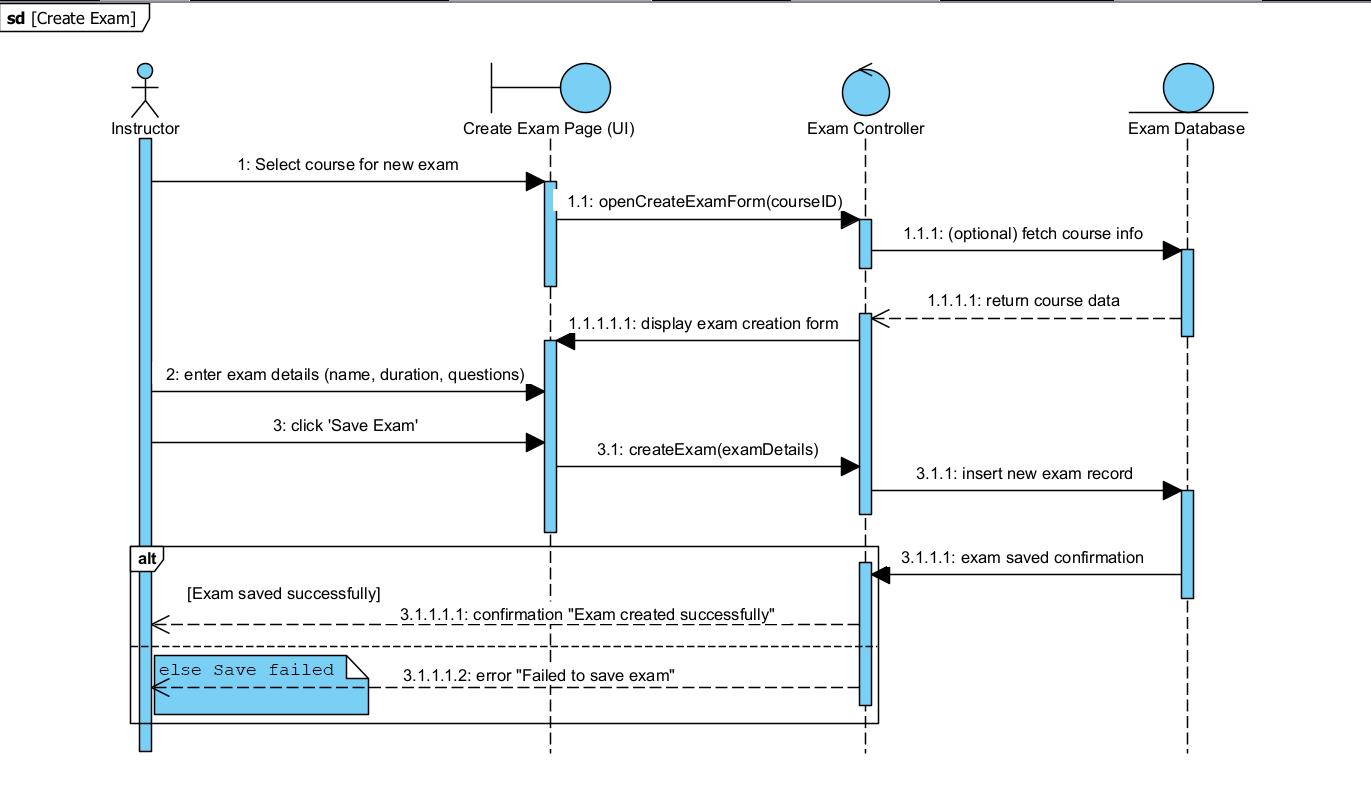
### ****Return Messages****: Dotted arrows going in the opposite direction of a message.

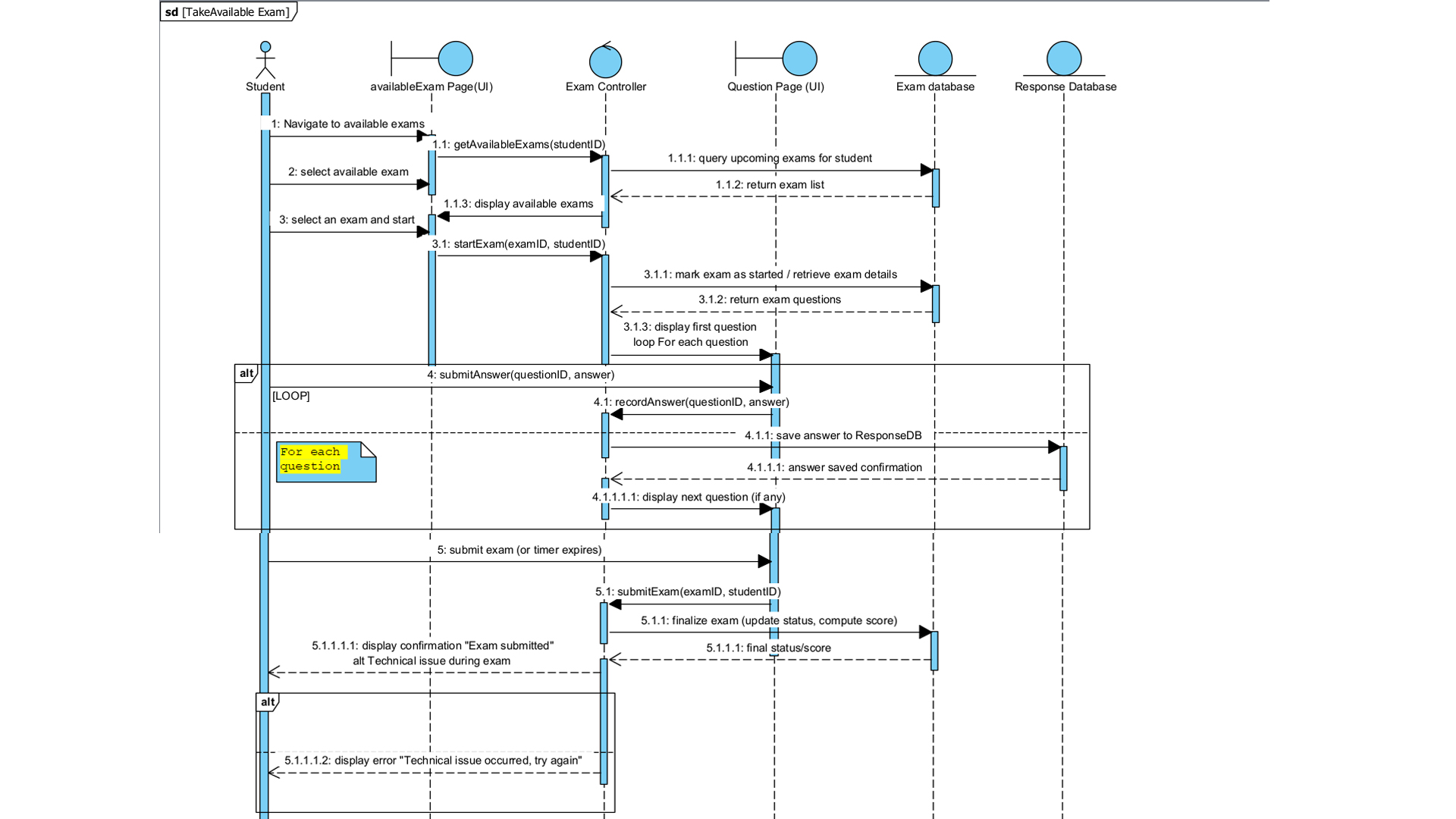
**Use**: To show that a response or result is sent back after processing

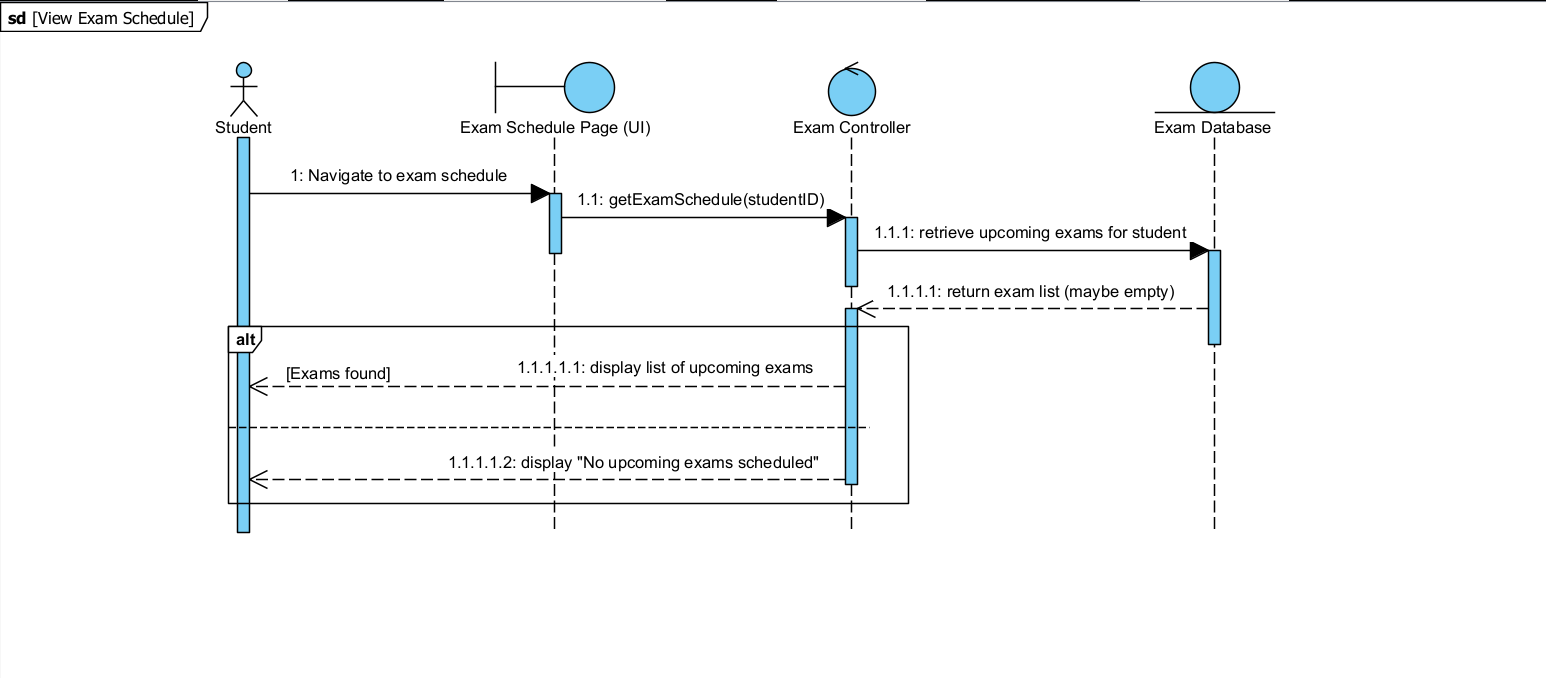
# 2.3 Example of High Level Sequence











# 2.4 Tools and Steps to Draw High Level Sequence Diagram

* **Tools Used : Visual Paradigm**

### ****Step 1:** Open or Create Projec**t****

### Launch **Visual Paradigm**.

### create a new one for the Online Examination System.

### ****Step 2:** Create a New Sequence Diagram:**

* Go to **Diagram** > **New**. In the dialog box:
* Choose **UML Diagrams** → **Sequence Diagram**.

-Name it (e.g.login ,Assign instructors ,Create exam ,Take available exam ,View exam schedule).

-Click **Create**.

### ****Step 3:** Add Actors and Lifelines**

* **From the left toolbar**, drag and drop
* **Actor** to represent the Student or other users
* **Lifeline** to represent system components

### ****Step 4:** Add Messages**

### Select the ****Message** tool** (solid arrow).

* **Add Return Messages** (dashed arrows) for responses:

**Step 5: Add Activation Bars**

****Step 6:****Use 'Combined Fragments' (alt/loop) to represent conditions and loops.

### ****Step 8**: Save or Export**

### Click ****File > Save Project****.

* Optionally, export the diagram as an image or PDF:
* Go to **File > Export > as Image**

# CHAPTER THREE

# **3.1** **Low-level (Detail) Design (class design)**

The **low-level class design** focuses on the internal structure of the system, identifying the key **classes**, their **attributes**, **methods**, and **relationships**. It is used to guide the implementation phase and ensure maintainability, scalability, and proper separation of concerns.

**3.2 Components of Class Diagram**

A **Class Diagram** is a static structure diagram used in UML to represent the **classes**, their **attributes**, **methods**, and **relationships** within a system.

### ****Main Components of a Class Diagram:****

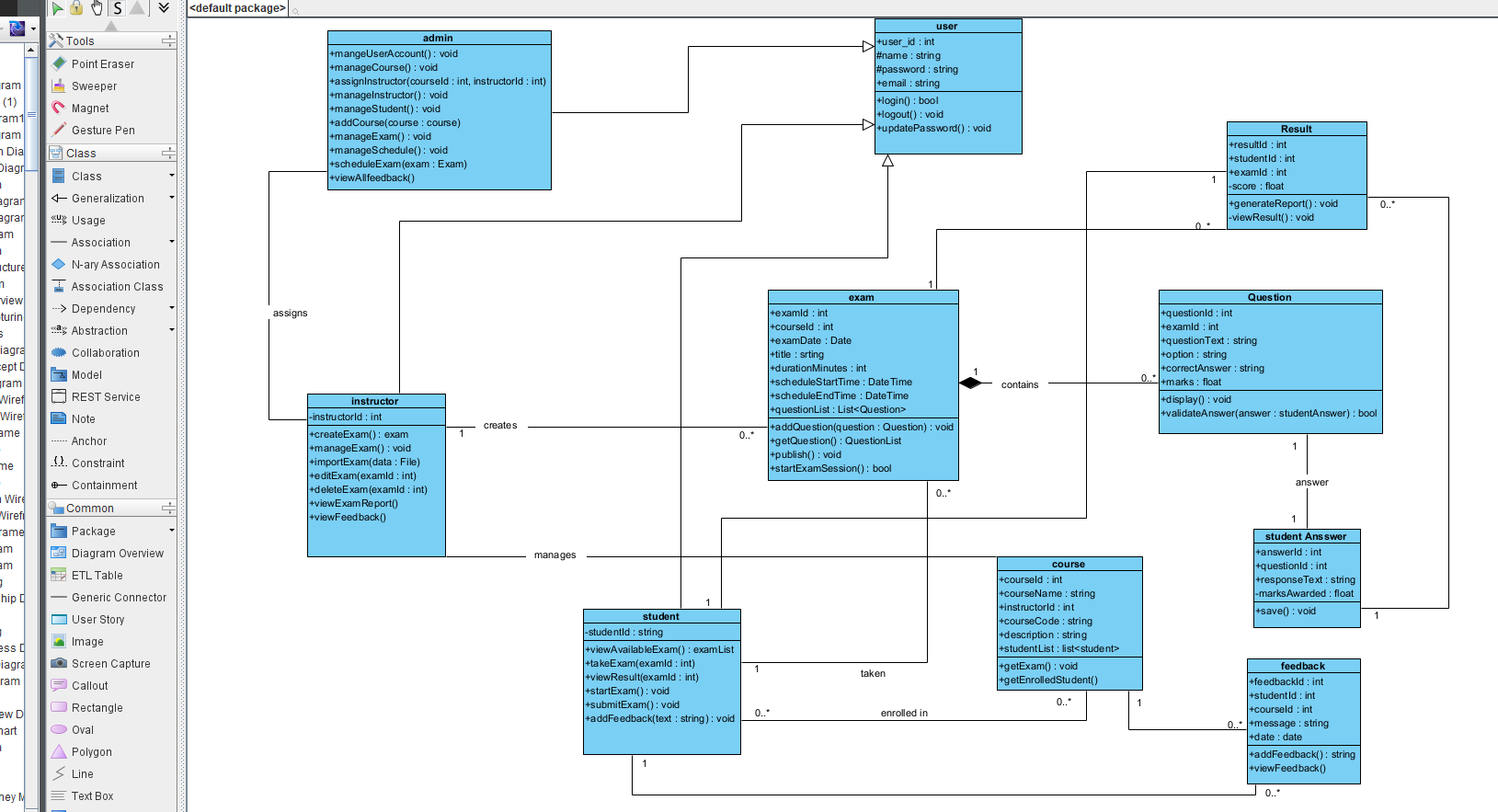
### ****Class:****Represents an object in the system.

* Contains **attributes (fields)** and **methods (operations)**.
* **Attributes:**Represent the data or properties of a class.
* **Methods (Operations):**Define the behavior of the class.
* **Association:**Shows a relationship between two classes.

* **Multiplicity:**Defines how many instances of a class relate to another.
* **Generalization (Inheritance):**Shows that one class inherits from another.
* **Aggregation:**A "whole-part" relationship where parts can exist independently.
* **Composition:**A strong form of aggregation where parts cannot exist without the whole.
* **Dependency:**Indicates that a class uses another class temporarily.

**3.3 Example of Class Diagram**

Below is a  **class diagram** that shows the key classes, their attributes, methods, and relationships that created using **Visual Paradigm**.

****

# 3.4 Tools and Steps to Draw class Diagram

**Tools Used** : Visual Paradigm

### ****Steps to Draw Class Diagram in Visual Paradigm****

* **Open Visual Paradigm**
* Launch the software and create a new one.
* **Create New Class Diagram**
* Go to **Diagram** → **New** → Select **UML Class Diagram**.
* Enter a name (e.g., Class Diagram – Online Examination System) and click **Create**.
* **Add Classes**
* From the **UML palette**, drag and drop the **Class** symbol onto the canvas.
* Double-click on the class to rename (e.g., Student, Exam, Result).
* **Add Attributes**
* Inside each class box, click **+ Attribute**.
* Add variables such as studentID: int, title: String, etc.
* **Add Methods (Operations)**
* Click **+ Operation** to add methods like login().
* **Define Relationships**
* Use connectors to draw:
* **Inheritance (Generalization)** → for parent-child classes.
* **Association** → for links between objects (e.g., Student ↔ Exam).
* **Aggregation/Composition** → for part-whole relationships (e.g., Exam → Question).
* **Add Multiplicity (Optional)**
* Click on the relationship line to set **multiplicity** (e.g., 1..\*, 0..1).
* **Save or Export**
* Save your project.
* Export the diagram as an image .

# **CHAPTER-FOUR**

# 4.1.Implementation (export class diagram into code and update code and diagram)

In the development of the **Online Examination System**, the implementation phase began by transforming the system’s class diagram—created in **Visual Paradigm**—into actual code using a programming language( java).

****Export Class Diagram into Code****

To automatically generate source code (e.g., Java) from a UML class diagram to speed up development and ensure design consistency.

### ****Update Code and Diagram (Synchronization)****

### To keep the **class diagram and source code synchronized** during updates or development changes.

### If ****change code**** (e.g., add methods), you can **reverse-engineer** the updates into the diagram.

* If **change the diagram** (e.g., rename class, add attribute), you can **regenerate** code to reflect changes.

# 4.2 Steps to Generate Code from Class Diagram

The following steps outline how to automatically generate source code (e.g., in Java) from the class diagram designed in **Visual Paradigm** for the **Online Examination System**.

### ****Step-by-Step Process:****

### ****Step-1**:**

### Launch **Visual Paradigm** and open your **Online Examination System** project that contains the completed **Class Diagram**.

### ****Step-2**:Navigate to Code Engineering**

### From the top menu, go to:Tools > Code > Generate Code…

### **Step-3**:**Choose Target Language**

* In the **Code Generation window**, select the programming language (e.g., **Java**).

### Step-4: **Select Classes to Export**

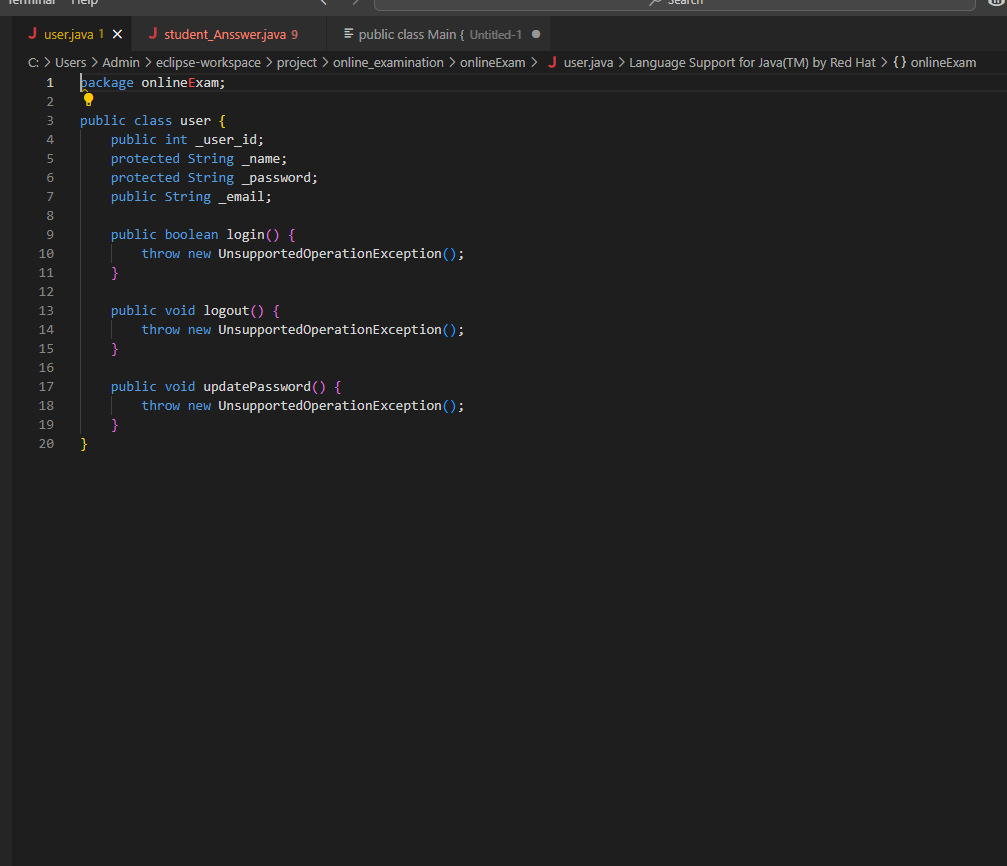
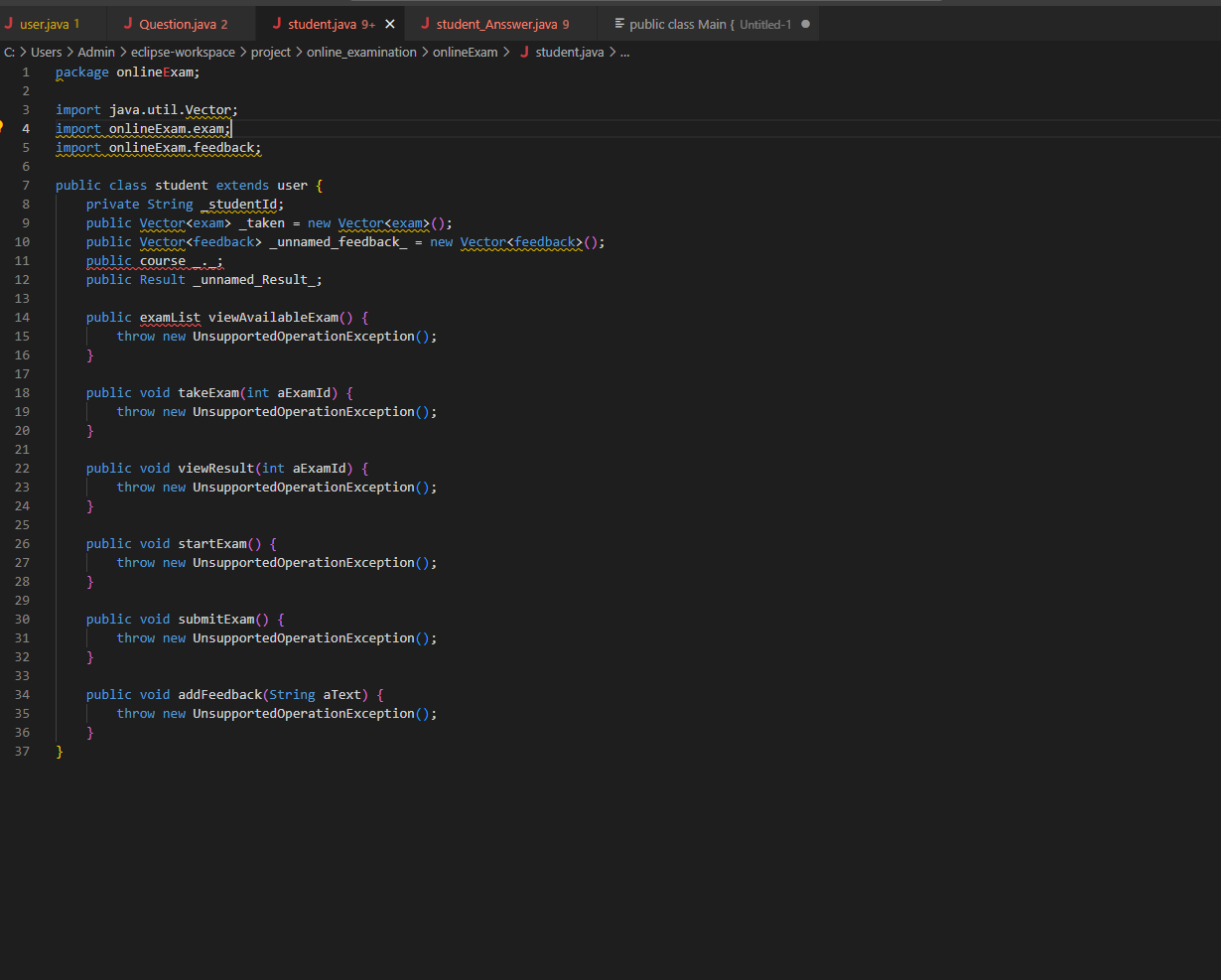
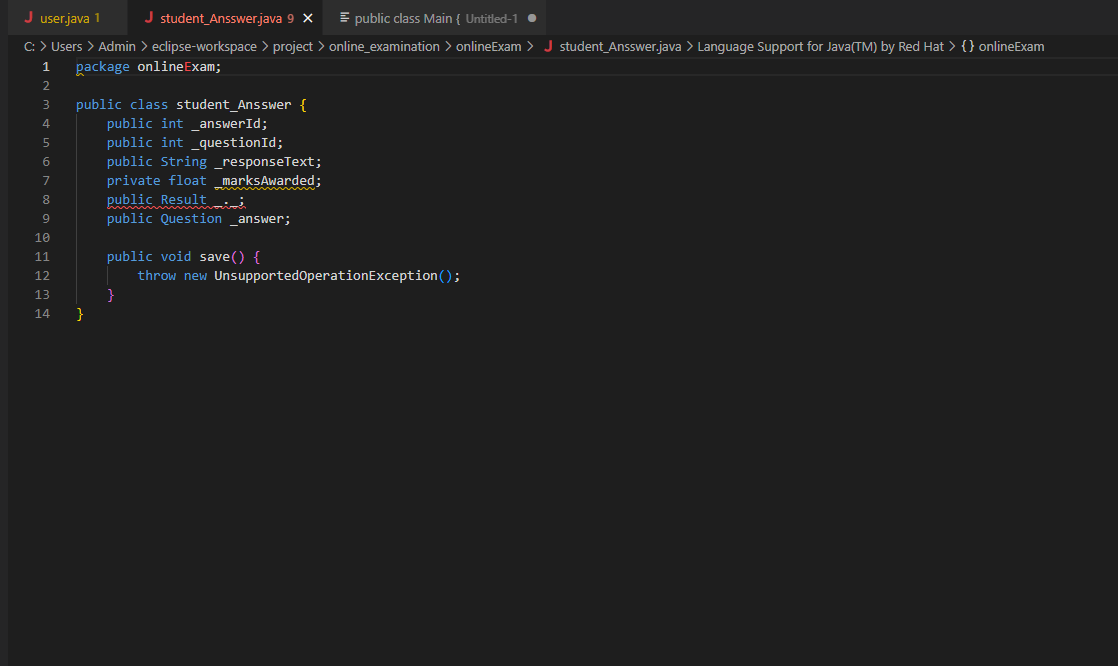
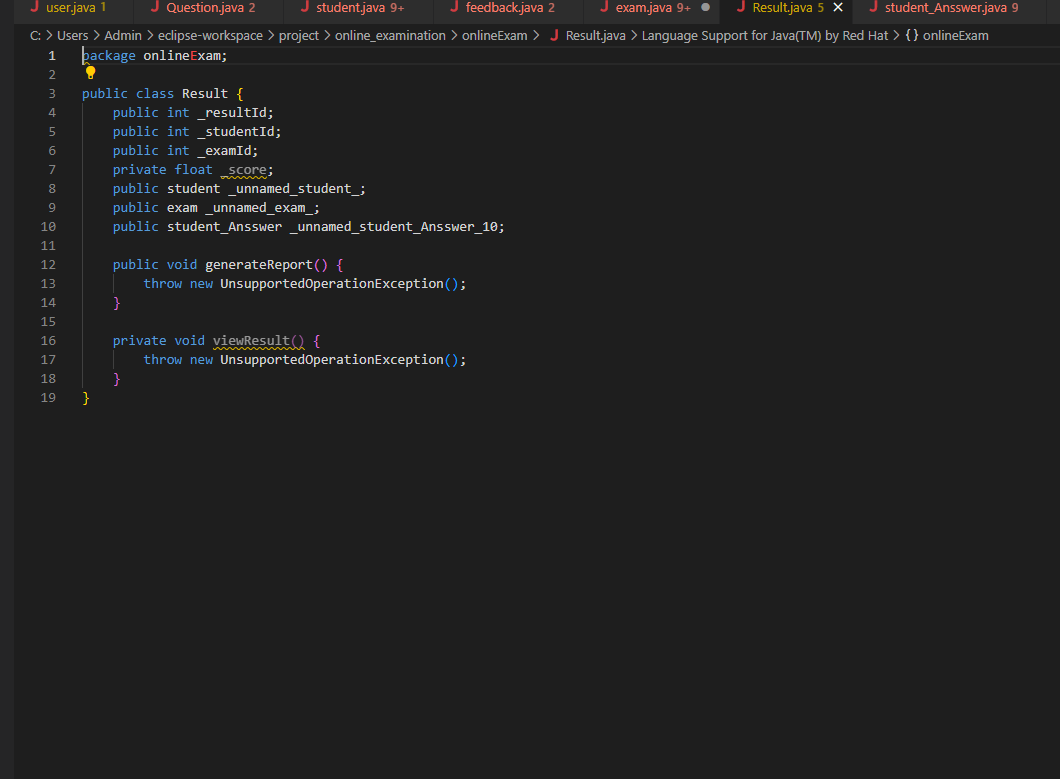
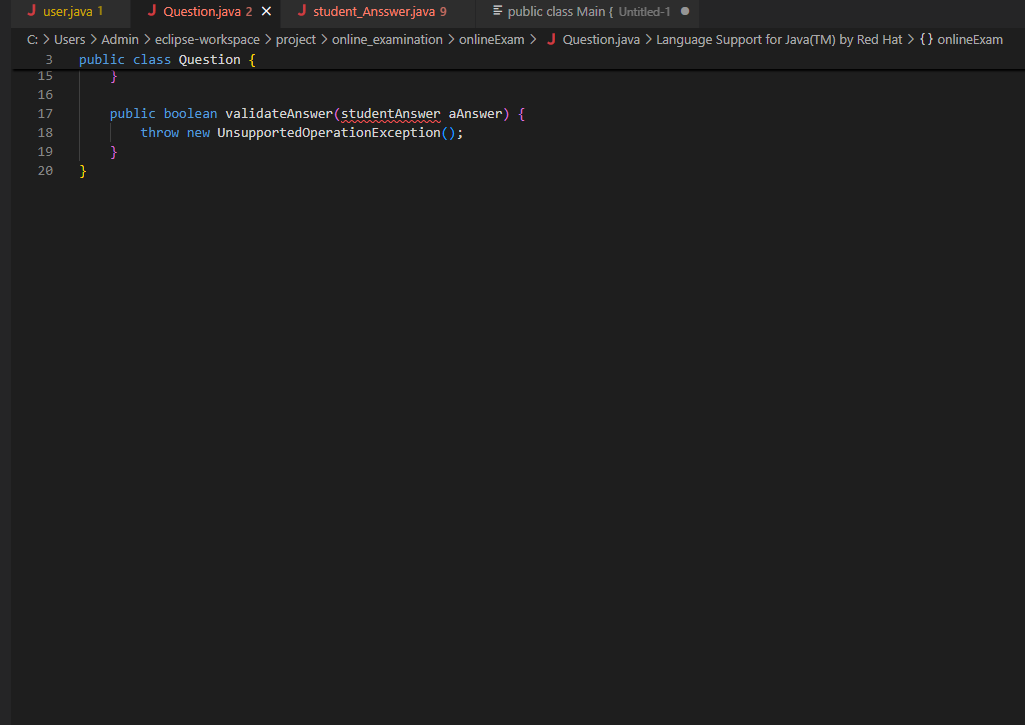
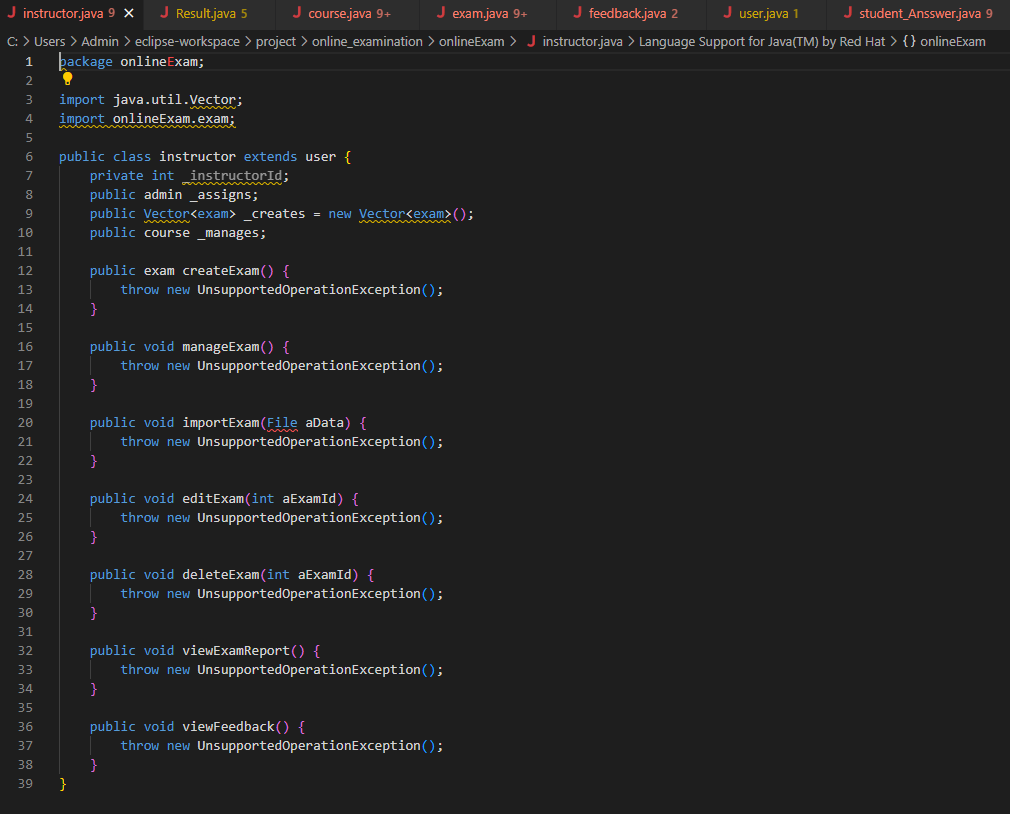
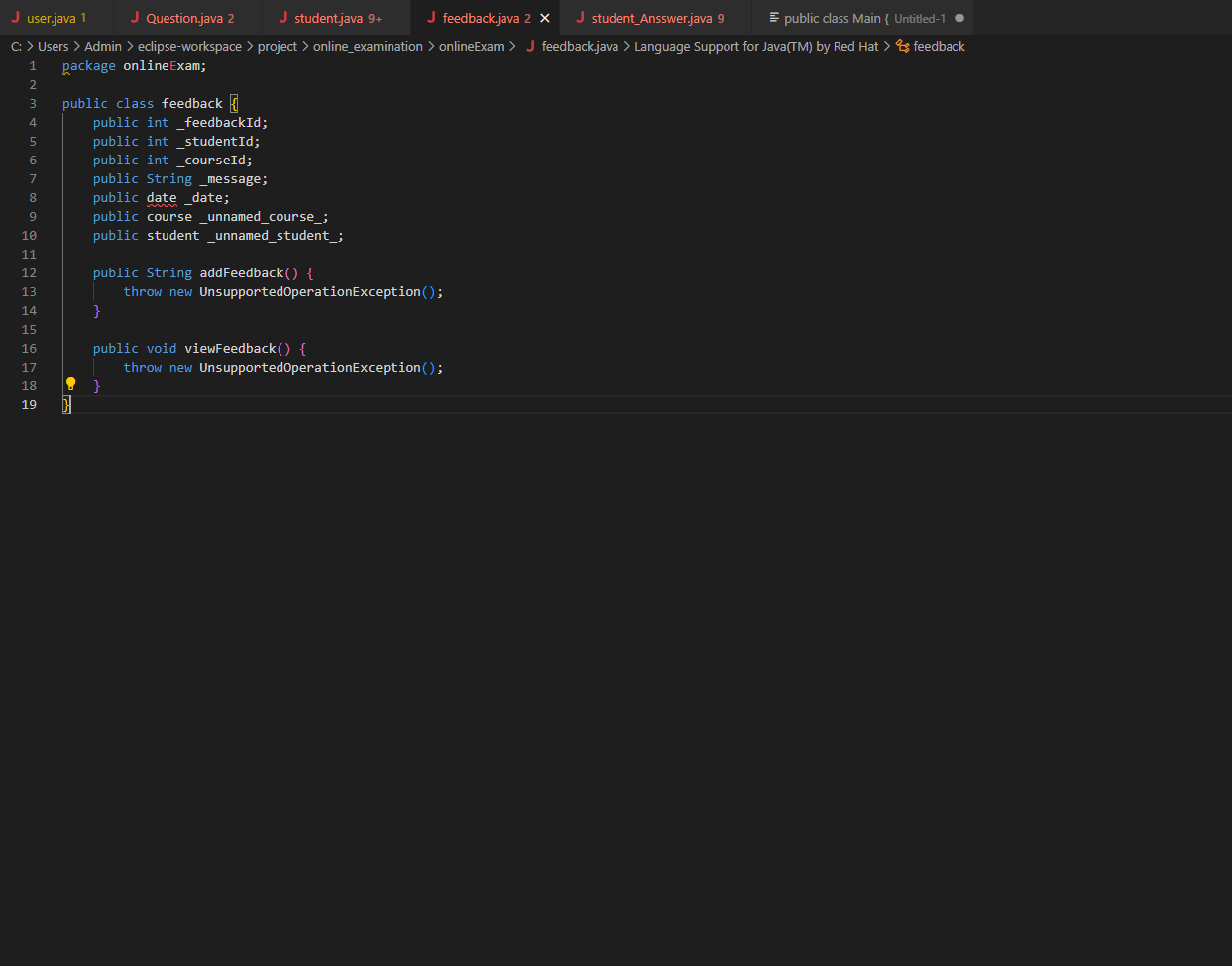
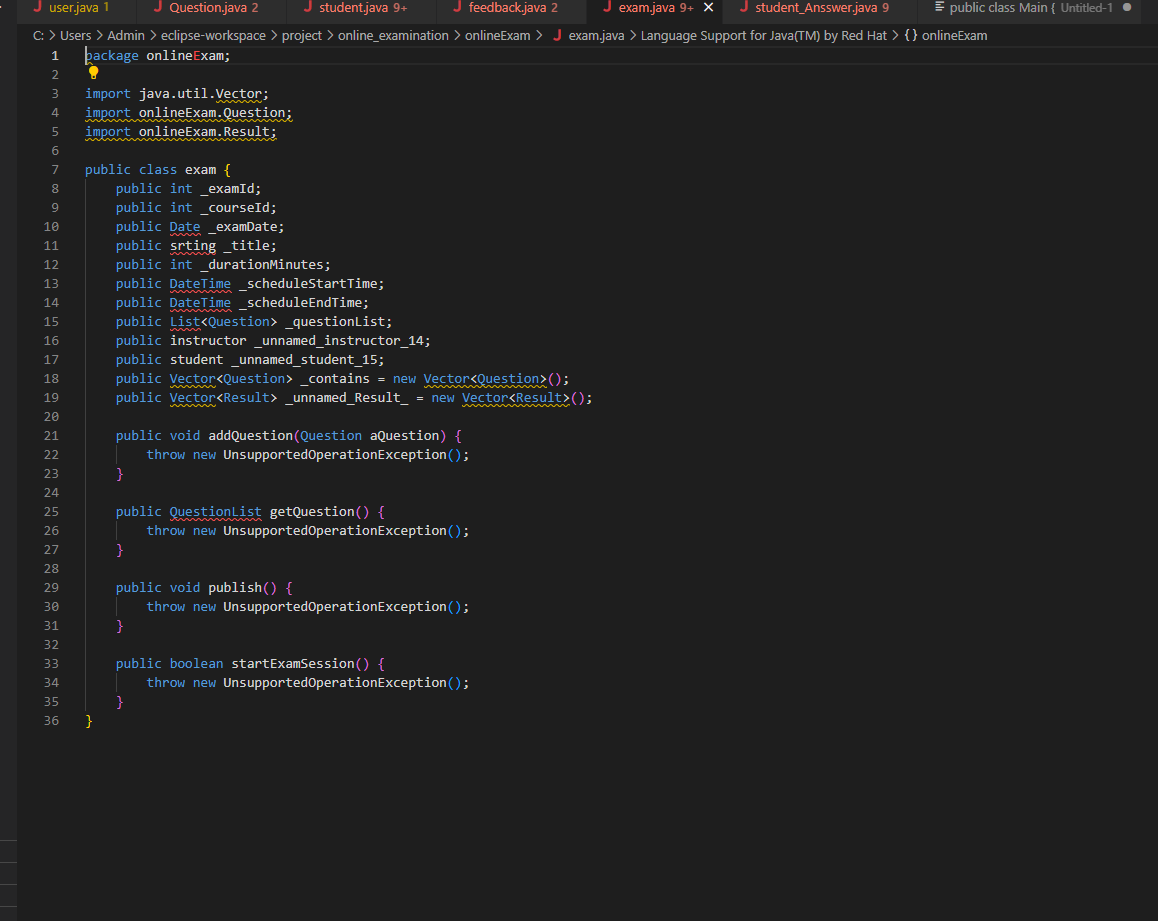
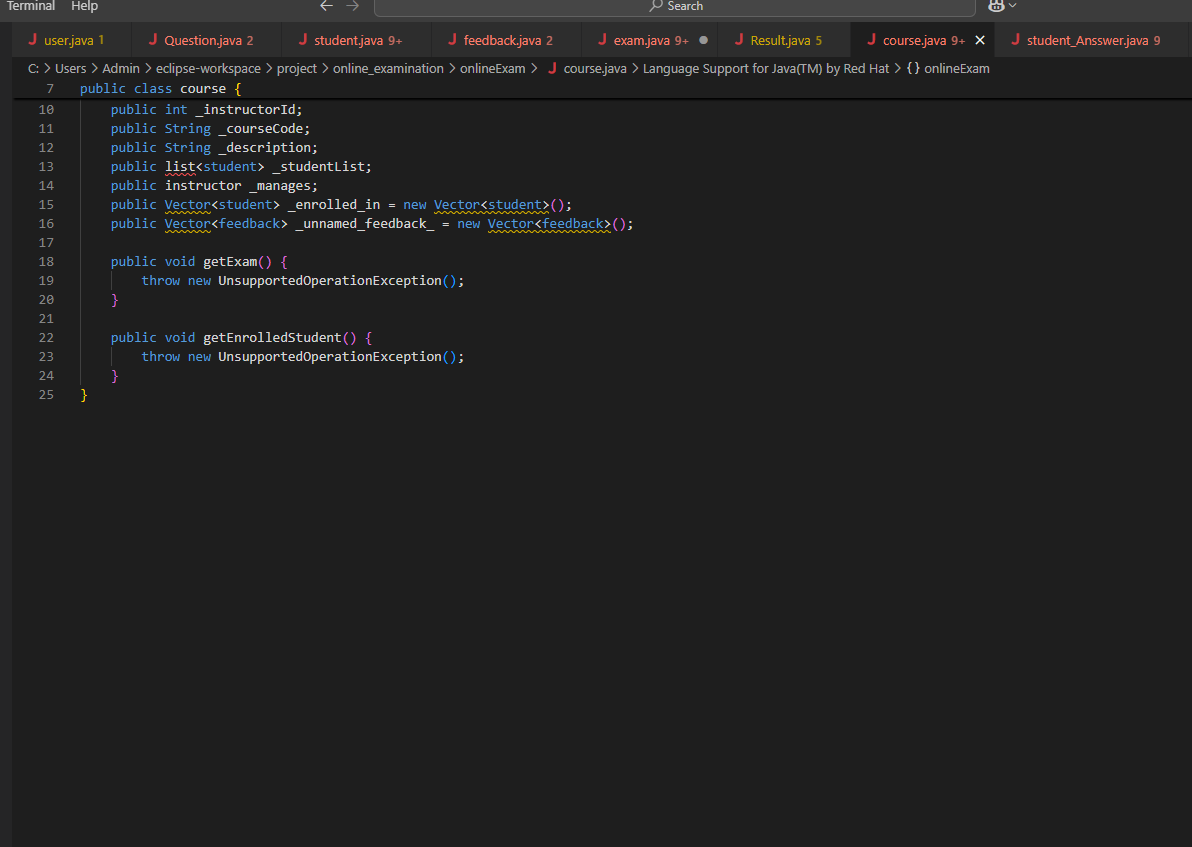
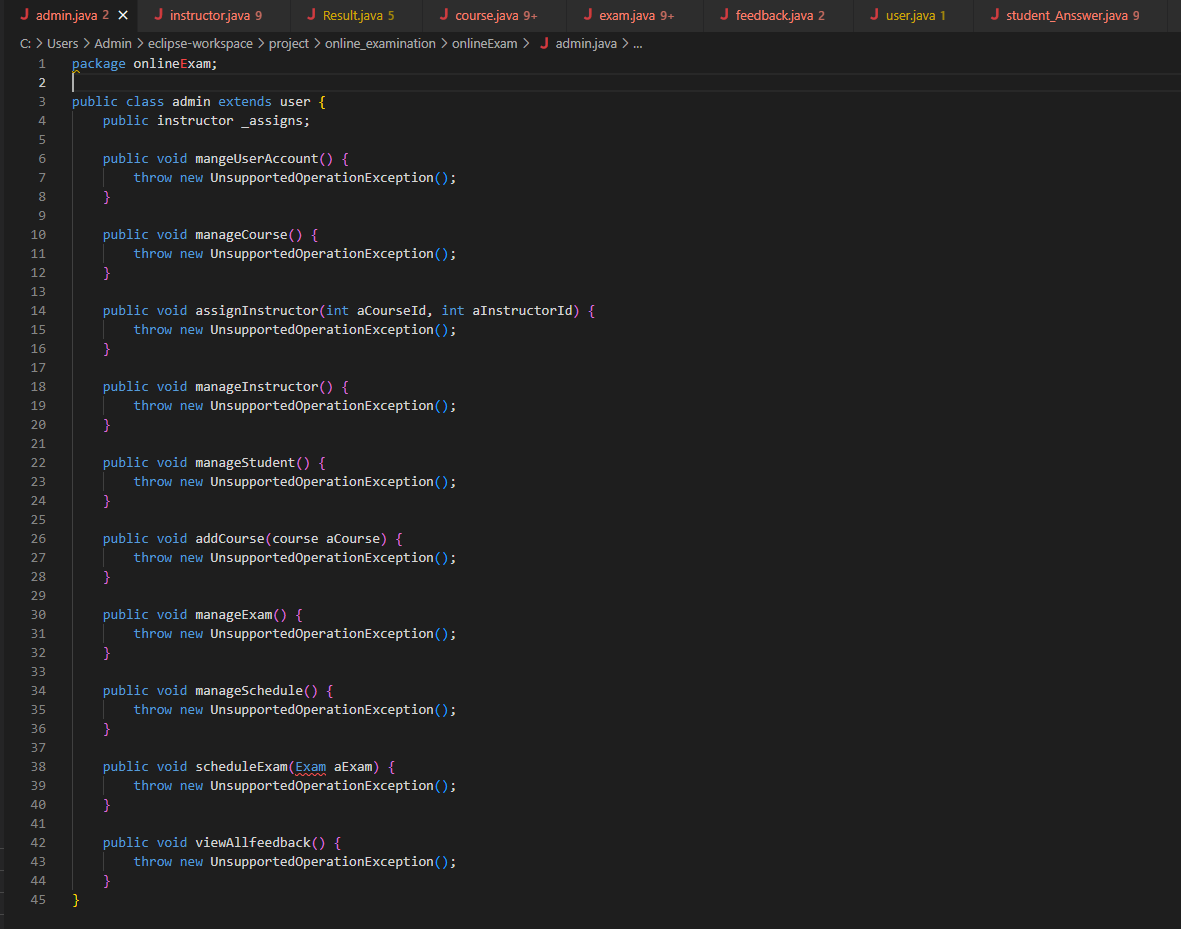
* Choose the **classes** (e.g., Student, Exam, Result, etc.) from class diagram that to generate code .

### ****Step-5:** Set Output Location**

* Specify the **folder** where the generated source code should be saved on your computer.

### Step-6: **Generate Code**

* Click on the **Generate** button.
* Visual Paradigm will create:
* Class files with proper names
* Attributes with data types
* Method stubs (method headers without logic)



# CHAPTER-FIVE

# 5.1 Change Management (version control using Git)

In our Online Examination System project, we implemented **Change Management** using **Git** and **GitHub**. This allowed us to efficiently track code updates, collaborate as a team, and maintain a stable and organized codebase throughout development.

**Git** is a **Distributed Version Control System (DVCS)** that helps teams:

* **Track changes** Git **records every change** made to our files over time ,Record who made each change and why, using clear commit messages ,Review changes using git log and compare versions using git diff, helping us debug and improve the code efficiently.
* **Collaborate** effectively with teammates without overwriting each other's work
* **Avoid code conflicts** Work on separate **branches** to isolate changes Use git merge to safely integrate our changes into the main branch.
* Roll back to previous versions if errors occur

We used **GitHub** as a **remote repository** to:

* **Host** our Git repository online
* **Collaborate** in real time with features like Pull Requests and Issue Tracking

# 5.2 Steps and tools that we use in our project to implements git.

In our **Online Examination System** project, we integrated **Git** and **GitHub** into our development workflow to ensure smooth collaboration, effective version control, and proper project tracking. Below are the tools and steps we followed to implement this system:

### ****Tools Used:****

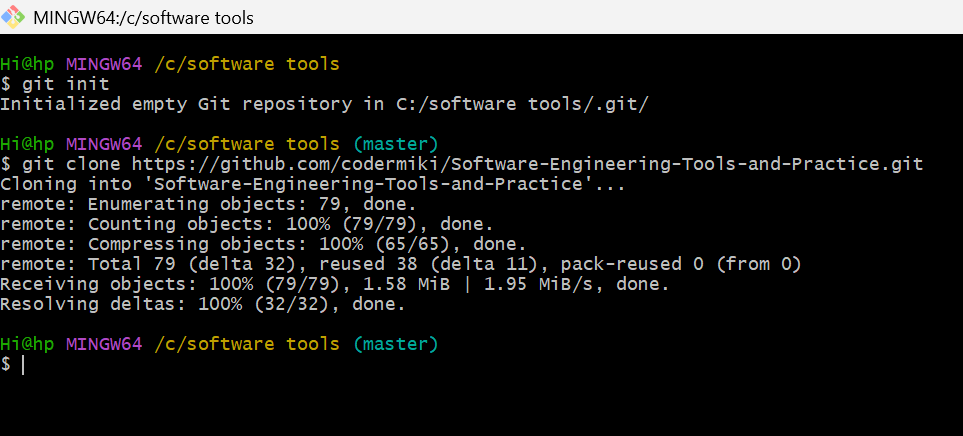
### ****Git****: Installed on each developer’s local machine to manage version control and track changes in the project files.

* **GitHub**: Used as the remote repository to host our project online, enabling team collaboration, version backup, and centralized access.
* **Git Bash**: A command-line tool used for executing Git commands and 1` sdertinteracting with the local repository.

### ****Steps Followed****

**Step-1.Initialize Git in the Project Directory**

* We used the git init command to initialize a Git repository in our local project folder.
* This enabled Git to start tracking changes in the codebase.



**Step-2.Create a Remote Repository on GitHub**

* A GitHub repository was created at:  
  <https://github.com/codermiki/Software-Engineering-Tools-and-Practice.git>

**Step-3.Link Local Project to GitHub Repository**

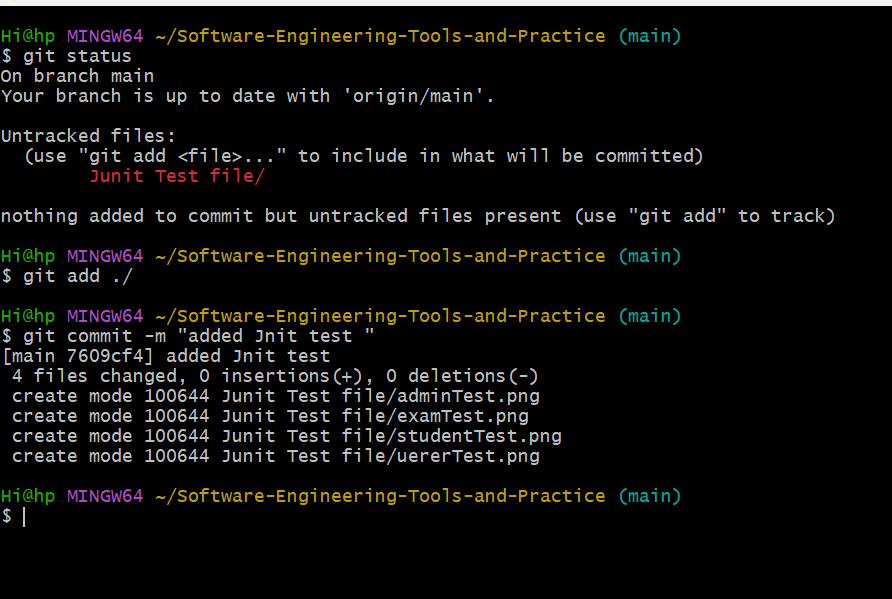
* We connected our local repository to the remote one using the command:
* git remote add origin

**Step-4.Add and Commit Files**

* We used the following commands to stage and commit our code:

-git add .

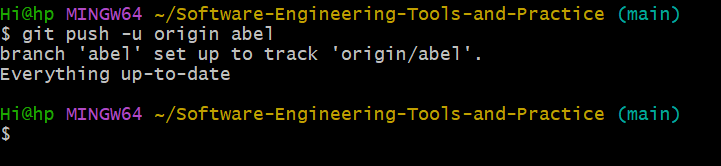
-git commit -m "Initial commit with core modules"



**Step-5.Push Code to GitHub**

* To upload local commits to GitHub, we used:

-git push -u origin main



**Step-6.Pull Requests and Code Reviews**

* Each team member worked on separate branches and created **Pull Requests** for merging into the main branch.
* We reviewed code before merging, discussed possible improvements, and ensured code quality and consistency.

**Step-7.Conflict Resolution**

* In cases where multiple contributors modified the same file, Git identified **merge conflicts**.
* We manually resolved these conflicts, tested the changes, and then committed the resolved versions.

# Chapter Six

# 6.1 Unit Test

**Unit Testing** is a software testing technique where **individual components or functions** of a program are tested **in isolation** to verify that each part works correctly. The goal is to identify and fix bugs **early in development** before the code is integrated with other modules.

In our **Online Examination System** project, we performed **unit testing** to verify that each module—such as login, exam submission, timer, and result calculation—worked correctly on its own.

**Unit testing helped us**:

* Identify bugs early
* Ensure modules functioned as expected
* Maintain code quality

# 6.2 Steps and tools used in Unit Test

To ensure the correctness of individual components in our **Online Examination System**, we followed a structured **unit testing process** using industry-standard tools. These tools helped us plan, write, and run tests effectively during development.

### ****Tools Used****

### ****Eclipse IDE****: Used for writing and executing Java code and unit tests.

* **JUnit 5**: Java testing framework used to create and run unit test cases.
* **Visual Paradigm**: Used to design **class diagrams** to understand the structure before writing test cases.

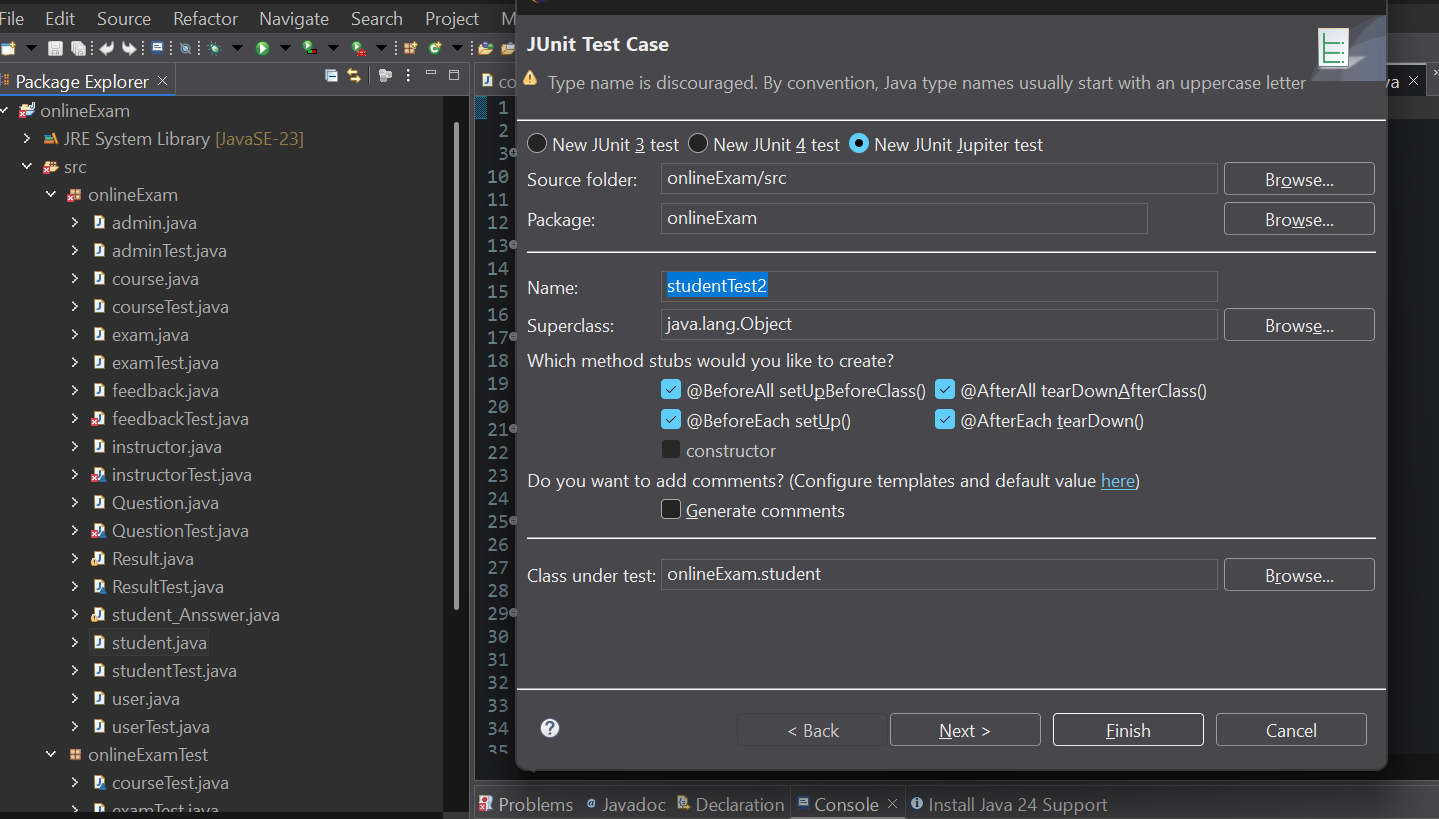
### ****Steps Followed:****

### ****1.Design Class Diagrams****

* Created class diagrams in Visual Paradigm to identify methods and attributes for testing.

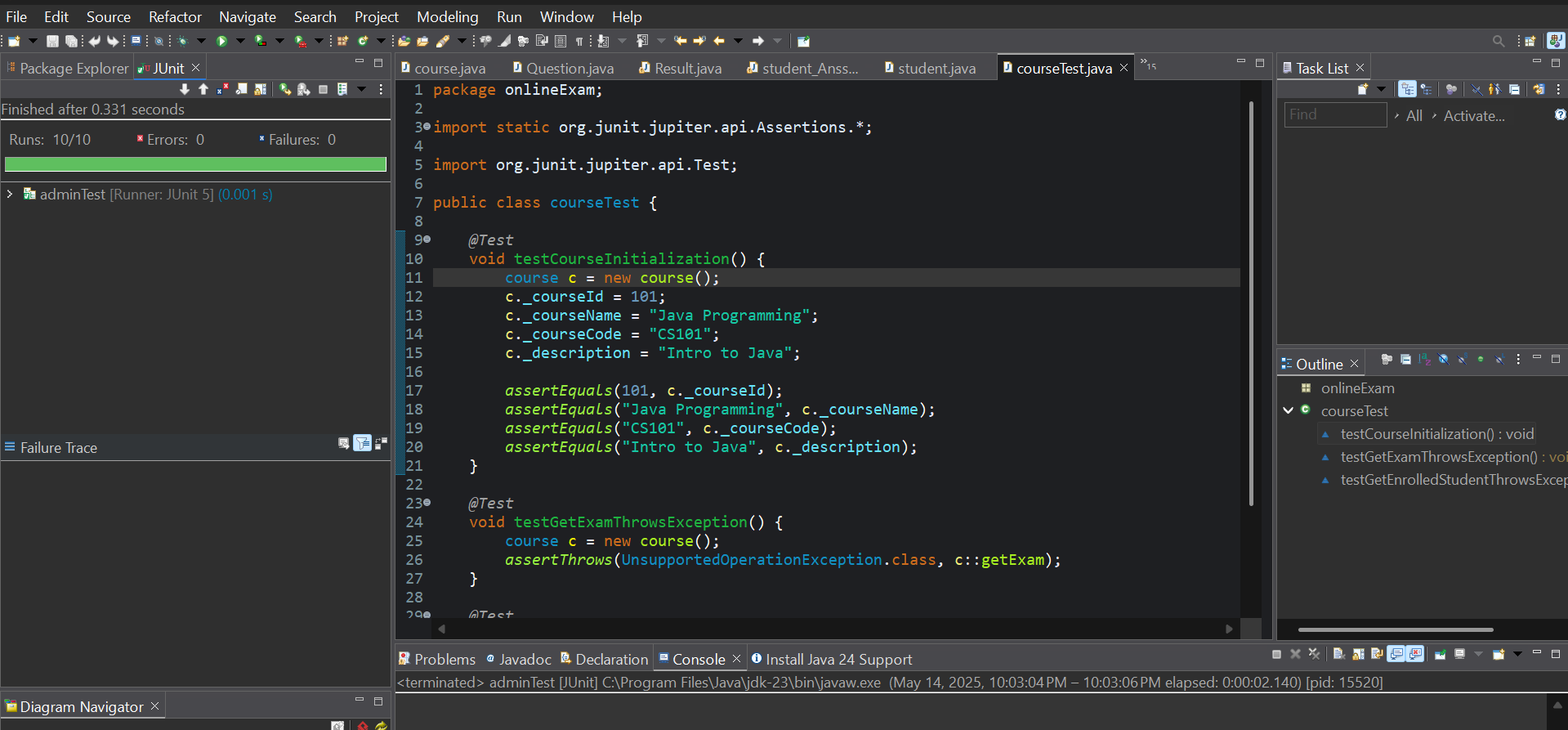
**2.Write Test Cases using JUnit 5**

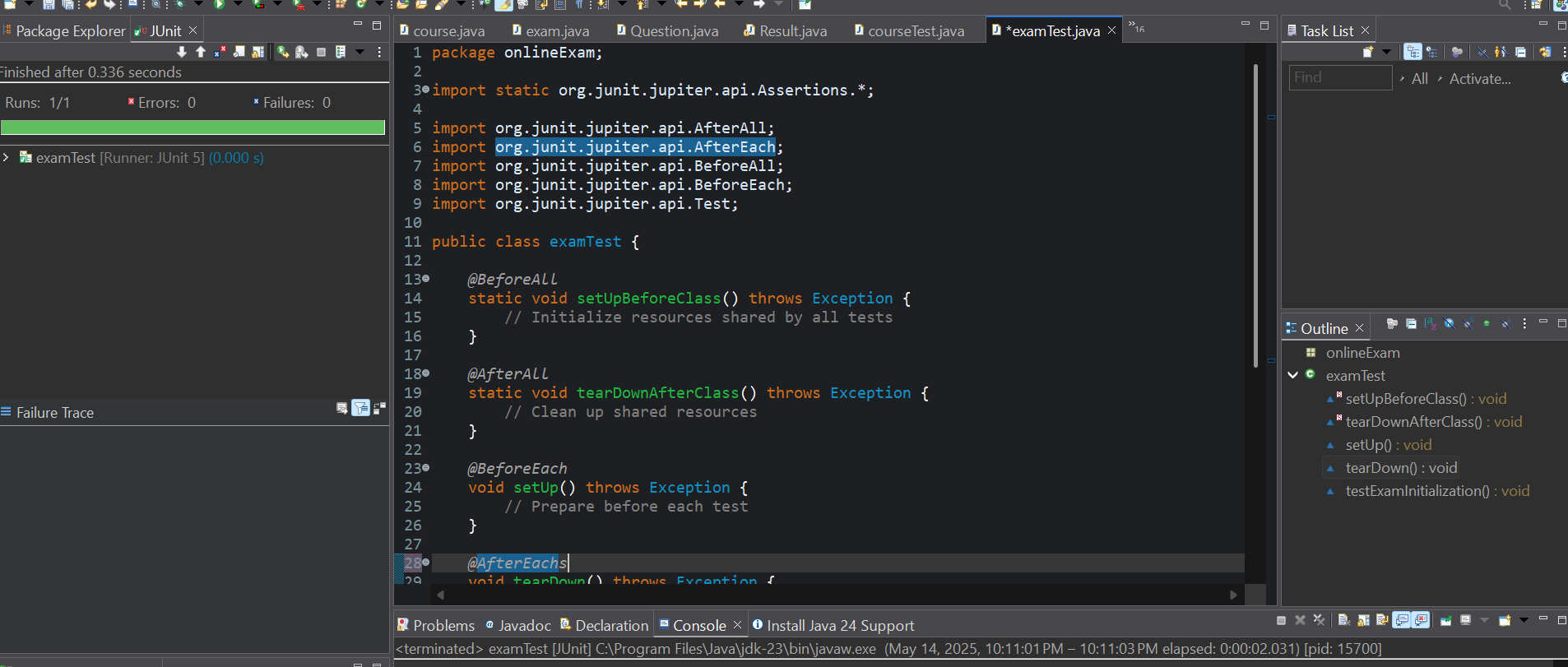
* In Eclipse, we wrote test classes and methods to test individual functions such as login validation, result calculation, and timer logic.

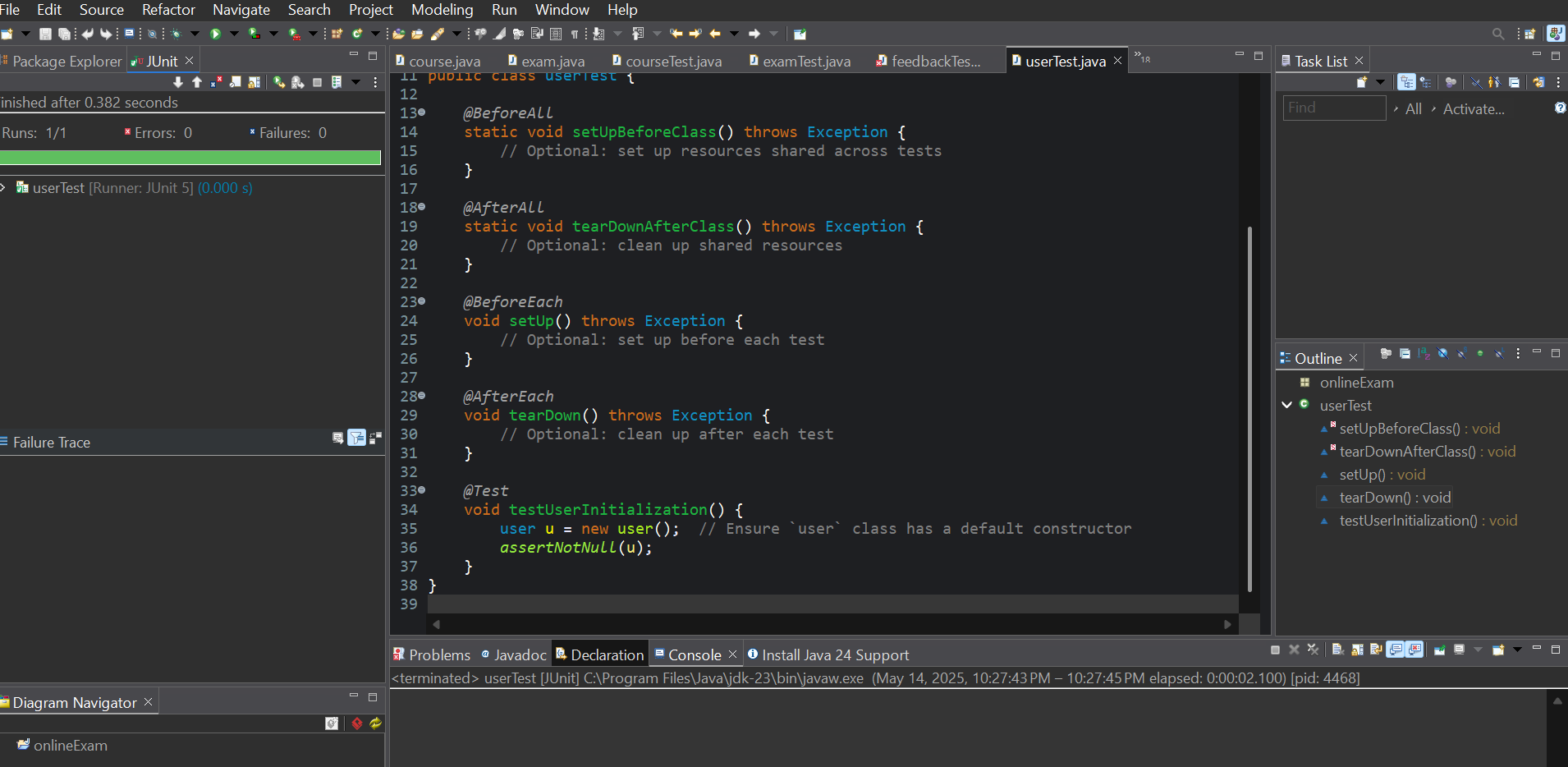


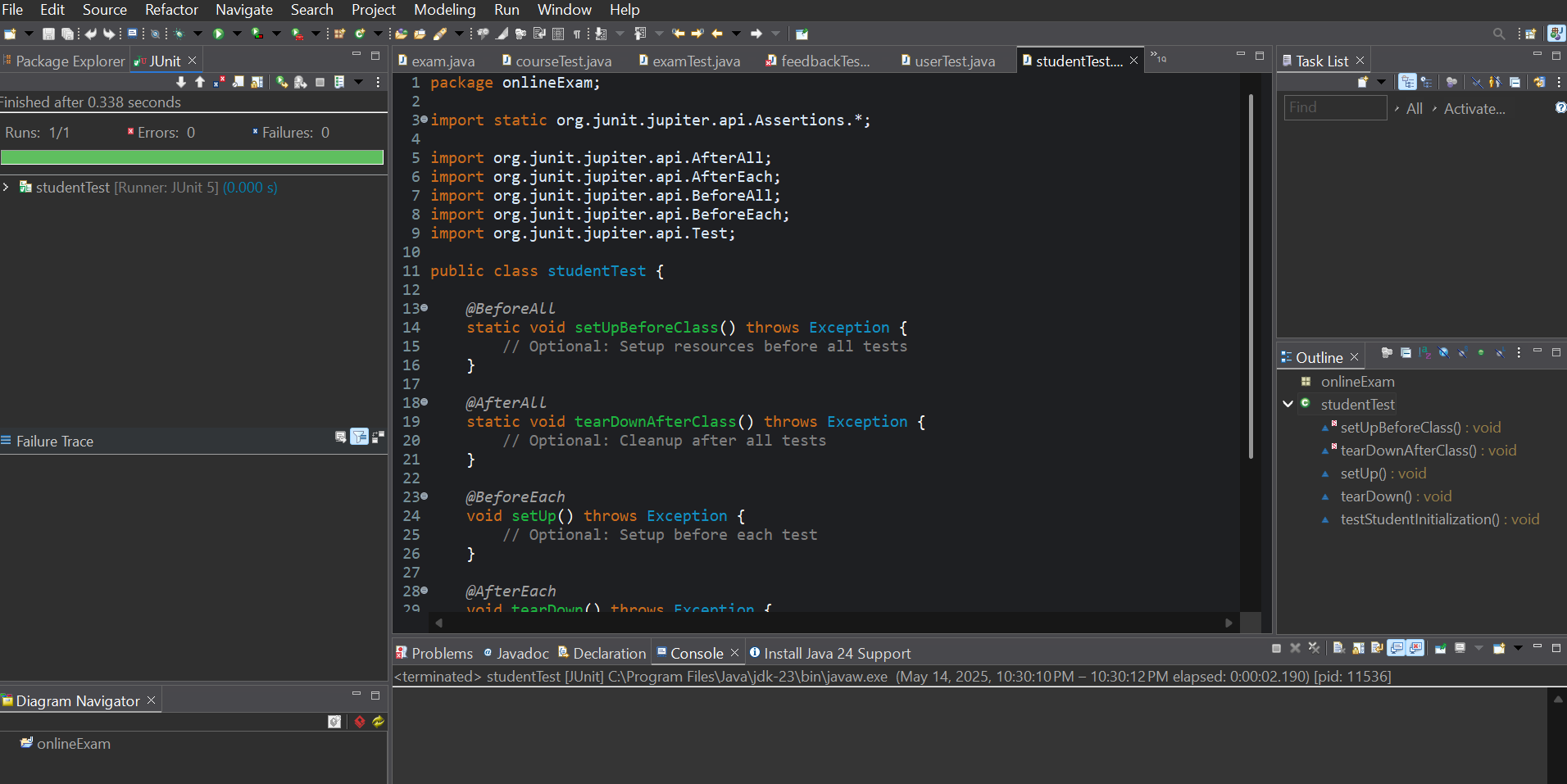
**3.Run Unit Tests in Eclipse**

* Executed tests directly in Eclipse and checked results in the JUnit panel.









**4.Fix Bugs Based on Test Results**

* If a test failed, we debugged the function, fixed the issue, and re-tested it.

**5.Repeat Testing During Development**

* Unit tests were run frequently to ensure that newly added features didn’t break existing functionality.

# Chapter -7

# 7.1 Build (prepare build script for compilation, unit test, jar file creation)

In our **Online Examination System** project, we followed a structured build process to compile the code, run unit tests, and generate a deployable **JAR (Java Archive) file**. This helped ensure that the application could be consistently built and tested across different systems.

The **build process** in software development is the series of steps used to **convert source code into a working software application**. It includes compiling code, running tests, and packaging the application so it can be run or deployed.

# 7.2 Steps and tool that we use in our project to implement build

#### ****Tools Used:****

* **Eclipse IDE:**Used for writing, compiling, and managing the Java project. Integrated with build tools and testing frameworks.
* **JUnit** 5:Java unit testing framework used to run automated tests during the build process.
* **Apache Ant / Maven** (choose the one you actually used): Used to define and automate the build process via scripts (build.xml for Ant or pom.xml for Maven).

-Handled tasks such as compilation, testing, and JAR creation.

**Steps Followed:**

1.Organize Project Files:

* Source code placed in src/ directory.
* Test code placed in test/ directory.
* External libraries added to lib/ folder if needed.

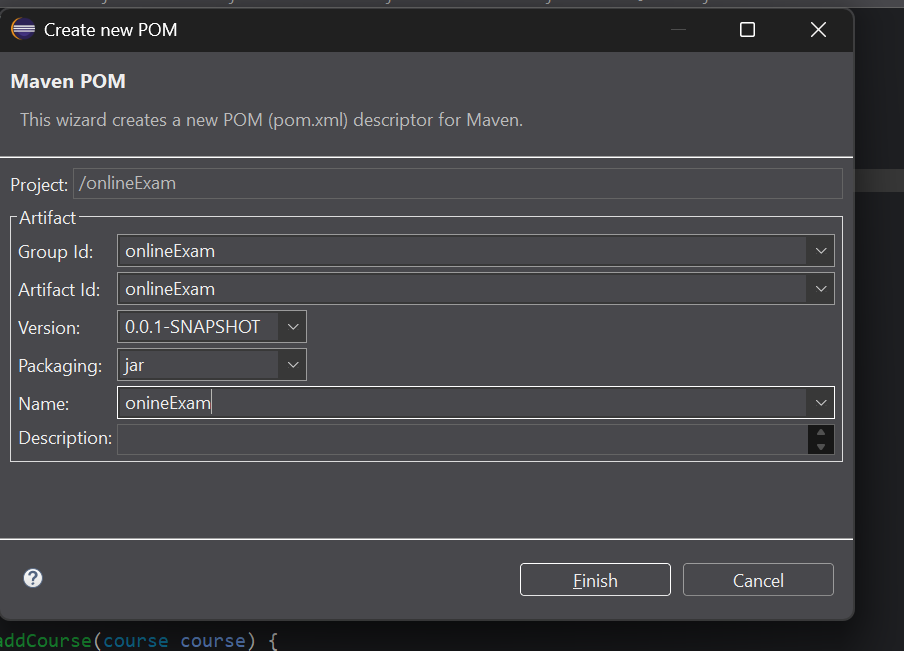
**2.Write Unit Tests:**

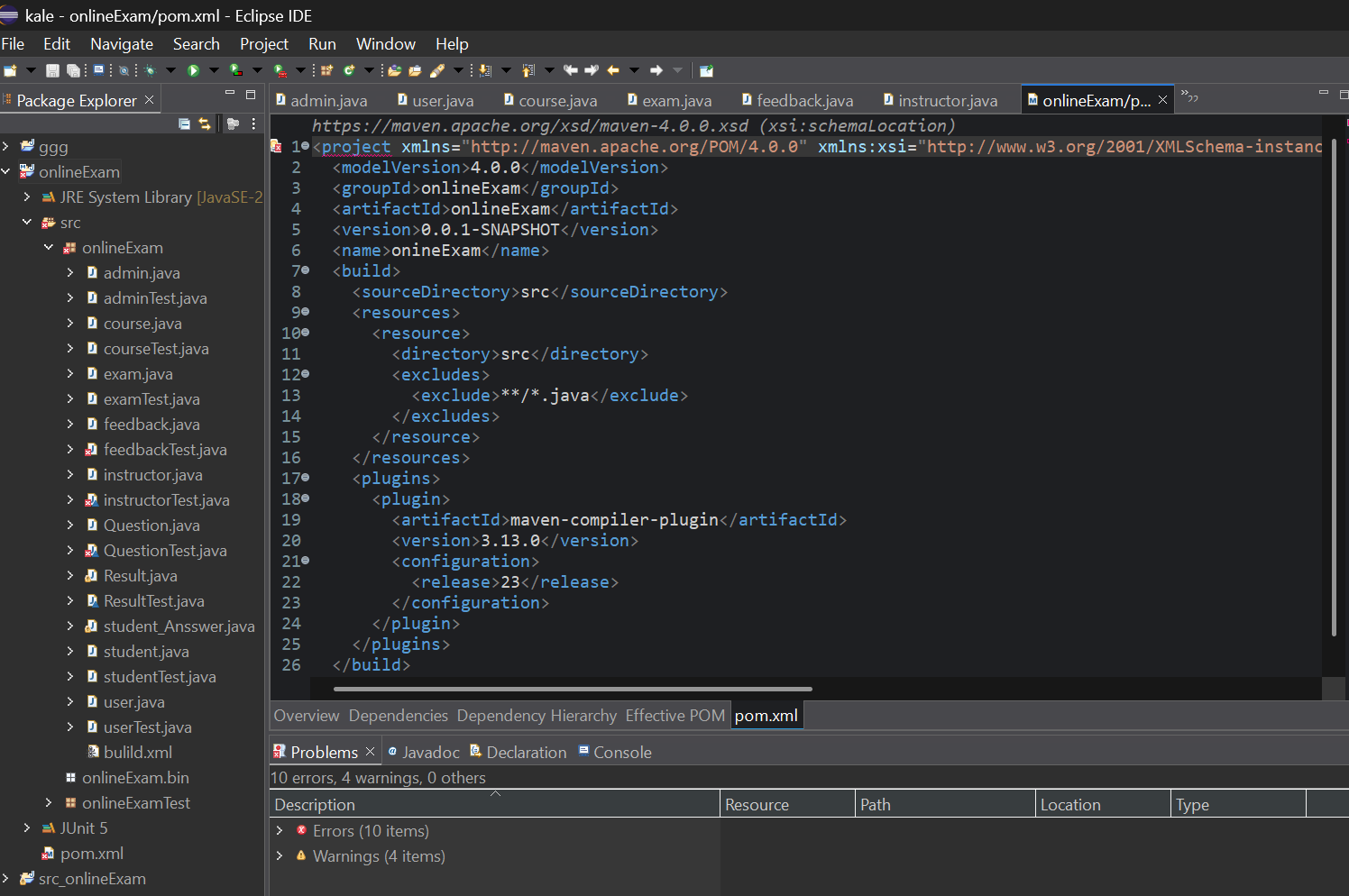
* Created JUnit test cases for key modules (e.g., login, result calculation, exam submission).
* Ensured individual components function correctly before full integration.

**3.Prepare Build Script**

* Wrote a build script (build.xml for Ant or pom.xml for Maven) with steps for:

-Cleaning old build files ,Compiling source code ,Running unit tests ,Generating JAR file





**4.Run Build Process**

* Used the build tool from Eclipse to execute the build:
* Verified successful compilation and test execution.

**5.Generate Executable (JAR) File**

* Packaged compiled classes into a .jar file.
* Ensured it includes all necessary files and dependencies for deployment.

