Object Oriented Software Design Group Project
RAD Report

GROUP NUMBER 15	
NAME SURNAME	NO
Muhammed Enes Gündüz	150120038
Mustafa Tolga Akbaba	150120001
Kadir Pekdemir	150121069
Murat Albayrak	150120025
Necati Koçak	150120053
Taha Enes Karaduman	150120530

Brief Description of the Project

In this project, we're embarking on the creation of a robust Java-based course registration system for a university department. Our primary focus is to provide a seamless registration process for both students and advisors, with the potential for future expansion to accommodate roles like department heads or administrators.

We're developing a variety of classes, including Course, Student, Advisor, and more. These classes are not mere data containers; they come equipped with meaningful methods to enforce the department's business logic. We'll use inheritance and encapsulation to improve code flexibility, with clear aggregation relationships between classes to define how they interact.

Additionally, the clean separation of domain logic from the user interface is another key aspect of our design, with a dedicated controller class ensuring a clear boundary between the two.

In essence, this project is all about building an adaptable and secure course registration system that adheres to the department's rules, incorporates object-oriented principles to make the code maintainable and efficient, and ensures smooth academic record management for both students and advisors. It's a comprehensive solution that addresses the core needs of the department and sets the stage for potential future enhancements.

Glossary

- **Student:** A registered student in the department.
- **Advisor:** Academic advisors helping students choose their courses.
- **Course:** An instructional module provided by the department.
- **CourseSection:** Particular example of a semester-long course offered.
- **Grade:** A student's academic assessment for a particular course.
- **Registration:** The course enrollment process.
- **Transcript:** A student's academic performance record.
- **CourseRegistrationSystem:** Control class overseeing communication.
- **Username and Password:** Use data to verify users' identities within the system. To access the system, users and students enter their passwords and usernames.

Functional and Non-Functional Requirements

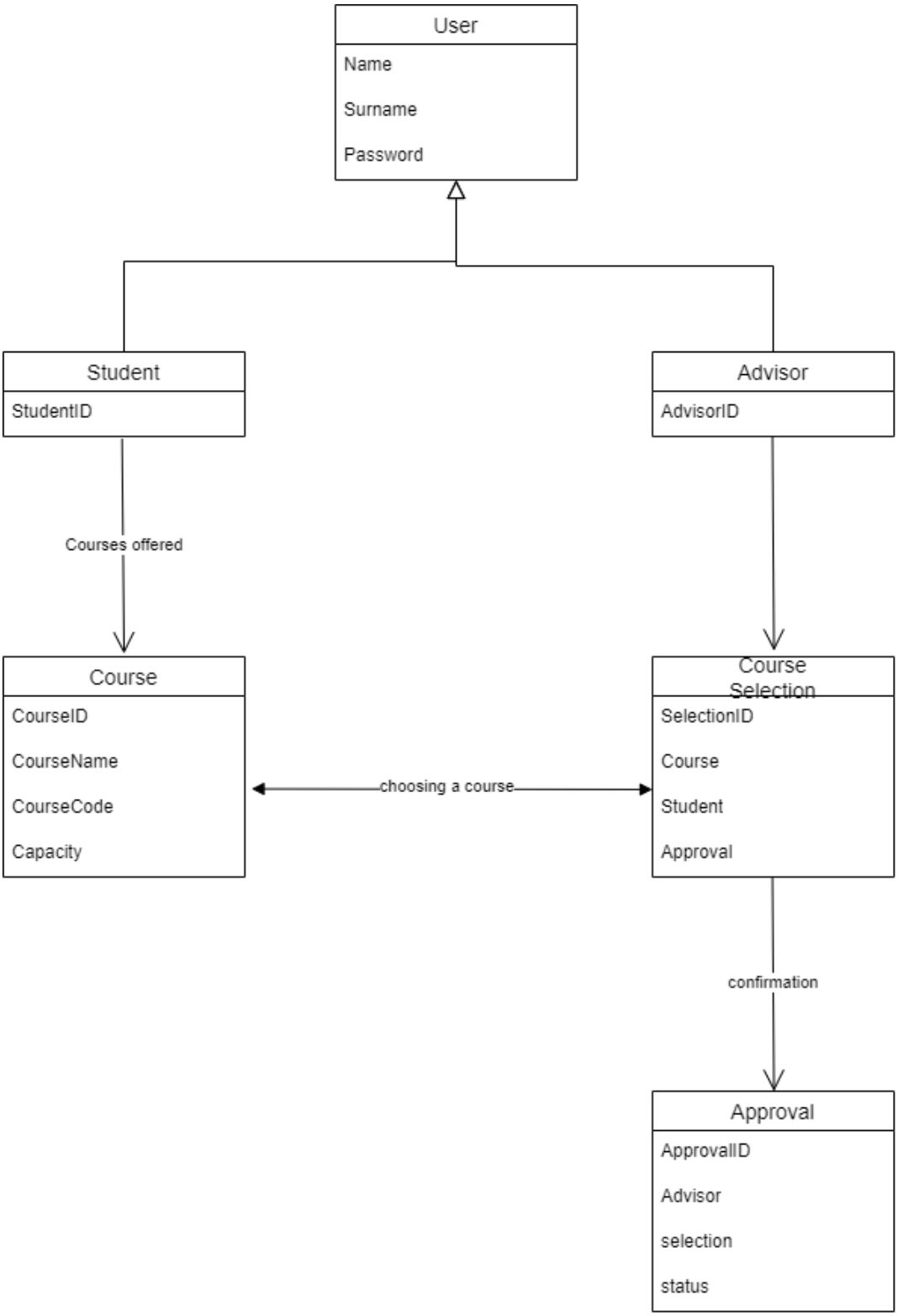
Functional Requirements

1. **User Login**
    - The system should allow users to log in with assigned usernames and passwords.
    - Priority: High
    - Criticality: Very Critical
    - Risks: User authentication errors, password security breaches.
    - Reduce risks: Strong password policies.
2. **User Roles**
    - The system should define different user types (e.g., students and advisors) and manage them with role-based authorizations within the system.
    - Priority: High
    - Criticality: High
    - Risks: Incorrect assignment of user roles or faulty authorizations
    - Reduce risks: Minimizing errors by establishing well-defined user roles and authorizations.
3. **Course Registration**
    - The system should conduct the course registration process in compliance with the department's rules.
    - Priority: High
    - Criticality: High
    - Risks: Violation of course registration rules
    - Reduce risks: Designing and verifying the registration process in accordance with department policies.
4. **Viewing and Registering for Courses**
    - Users should be able to view courses and register for them.
    - Priority: High
    - Criticality: Medium
    - Risks: Users registering for the wrong courses or being inadequately informed.
    - Reduce risks: Minimizing errors by using a user-friendly interface and authentication processes.
5. **User Authorization**
    - The system should perform the necessary processes for user authorization.
    - Priority: High
    - Criticality: Medium
    - Risks: Incorrect users gaining system access.
    - Reduce risks: Ensuring secure operations and complete implementation of authentication processes.
6. **Course Statistics**
    - The system should generate and display statistics for courses to users.
    - Priority: Low
    - Criticality: Medium
    - Risks: Inaccurate or incomplete statistical calculations.
    - Reduce risks: Automating and verifying statistical calculations.

Non-Functional Requirements

1. **Performance:**
    - The performance of the application is crucial, especially to ensure that users can perform operations in the system swiftly. Metrics such as response times, processing speeds, and startup times can be used to measure performance. Techniques like code optimization and database optimization can enhance performance.
2. **Portability:**
    - The application should function seamlessly on different platforms, such as Windows, Linux, macOS. This entails writing platform-independent code, using appropriate tools and frameworks, and avoiding platform-specific code.
3. **Availability:**
    - The availability of the application should be ensured with uninterrupted operation and minimal downtime. Downtime should be maintained at an acceptable level, and backup systems for failovers within a specific time frame may be implemented.
4. **Security:**
    - The application must be protected against unauthorized access and breaches.
5. **Maintainability:**
    - Your code should be sustainable in the long term. Adhere to object-oriented programming (OOP) principles to make your code modular and comprehensible. This enables you to easily expand and reuse your code.

Domain Model



In this model:

- The "Student" and "Advisor" classes represent students and advisors, respectively.

- The "Course" class represents individual courses, and the "Course Selection" class represents the selections made by students for specific courses.
- The "Approval" class represents the approval status of course selections made by students, with the advisor's involvement in the approval process.
- This model allows students to make course selections, advisors to review and approve these selections, and tracks the approval status for each selection.

Use Cases

Use Case 1: User Authentication

Use Case Name: User Authentication

Purpose: To allow users to log into the system.

Actors:

- User (Student, Advisor,)

Preconditions:

- The user must have a valid username and password assigned.

Main Flow:

1. User requests access to the system.
2. The system prompts the user for their username and password.
3. User enters the correct username and password.
4. The system validates the entered username and password.
5. If authentication is successful, the user logs into the system.

Alternative Flows:

- *Step 4:* If the entered username or password is incorrect, the system notifies the user of the incorrect login information and prompts the user to enter the correct details.

Use Case 2: Course Registration

Use Case Name: Course Registration

Purpose: To enable students to register for courses.

Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The user must have a valid student ID.
- Courses must have available and empty slots.

Main Flow:

1. User requests course registration.
2. The system displays a list of available courses and their vacancies.
3. User selects the desired course to enroll in.
4. The system checks the vacancy status of the selected course.
5. If there are available slots, the system registers the user for the course and updates the vacancy.
6. The system notifies the user of the successful registration.

Alternative Flows:

- *Step 5:* If the course is full, the system informs the user about the full capacity and allows them to choose another course.

Use Case 3: Transcript Display

Use Case Name: Transcript Display

Purpose: To allow students to view their transcripts.

Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The user must have a valid student ID.
- The transcript file belonging to the student (e.g., "150119055.json") must be readable.

Main Flow:

1. The system reads the student's transcript file.
2. The user views the existing course and grade information.
3. The system provides options for the user to add or modify course information if needed.

4. The system displays the updated transcript information to the user.

Alternative Flows:

- *Step 3:* The user can add or modify course information as needed. The system can offer verification after each update. If the user declines verification, the system preserves the previous transcript information, canceling the operation.

Use Case 4: Advisor Approval of Course Selection

Use Case Name: Advisor Approval of Course Selection

Purpose: To enable advisors to review and approve/deny students' requested course selections.

Actors:

- User (Student, Advisor,)

Preconditions:

• The student must have successfully logged into the system.

• The student must have completed the course registration process.

• The advisor must have successfully logged into the system.

Main Flow:

1. The system notifies the advisor that there are pending course selection requests from students.
2. The advisor reviews the list of course selection requests.
3. The advisor selects a student's request to view detailed information about the courses selected.
4. The advisor evaluates the selected courses and decides whether to approve or deny the student's course selection.
5. If approved, the system updates the student's course registration status, and the student is notified of the approval.
6. If denied, the system notifies the student of the denial and may provide reasons for the decision.

Alternative Flows:

• Step 4: If the advisor needs more information, they can request additional details from the student before making a decision.

• Step 5: If denied, the advisor may provide comments or suggestions to guide the student in making appropriate course selections.

Use Case 5: Advisor Monitoring of Student Progress

Use Case Name: Advisor Monitoring of Student Progress

Purpose: To allow advisors to monitor the academic progress of their assigned students.

Actors:

- User (Student, Advisor)

Preconditions:

• The advisor must have successfully logged into the system.

Main Flow:

1. The advisor selects the option to view the list of assigned students.
2. The system displays a list of students assigned to the advisor.
3. The advisor selects a specific student to view their academic information.
4. The system presents the academic history, including completed courses, current courses, and grades for the selected student.
5. The advisor may provide feedback or guidance based on the student's academic performance.

Alternative Flows:

• Step 3: The advisor can choose to filter students based on specific criteria, such as those at risk of academic probation or those excelling in their studies.

• Step 5: The advisor can schedule a meeting with the student to discuss academic performance and set goals for improvement if necessary.

System Sequence Diagram (SSD)

Use Case Name: Course Registration

Purpose: To enable students to register for courses.

Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The user must have a valid student ID.
- Courses must have available and empty slots.

Main Flow:

1. User requests course registration.
2. The system displays a list of available courses and their vacancies.
3. User selects the desired course to enroll in.
4. The system checks the vacancy status of the selected course.
5. If there are available slots, the system registers the user for the course and updates the vacancy.
6. The system notifies the user of the successful registration.

Alternative Flows:

- *Step 5:* If the course is full, the system informs the user about the full capacity and allows them to choose another course.