EGE UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

OBJECT ORIENTED ANALYSIS AND DESIGN

HOMEWORK-2

PREPARED BY

Team NO: 31

05210000261-Bahrihan Torpil

05210000238-Mehmet Ali Avcı

05210000260-Kutlu Çağan Akın

1. Defining the Boundaries of the Course Registration Context

Use Cases:

1. **Course Enrollment**:
   The process where a student enrolls in a course falls within this context. The required information includes:
   - Student ID.
   - Course ID.
   - Checking the availability of sufficient capacity for enrollment.
2. **Drop a Course**:
   The process of canceling a course registration includes:
   - Displaying the list of registered courses.
   - Selecting a course and confirming the drop action.
3. **Out-of-Scope Use Cases**:
   - **Course Scheduling**: This belongs to the **Faculty Management** context.
   - **Grade Management**: This falls under the responsibility of the **Transcript Context** or a similar context.

Justification:

The boundaries of this context are defined to exclusively encompass operations related to course registration. Other contexts have distinct responsibilities and are therefore addressed separately.

2. Apply Tactical Design

Entities

1. **Student** (Root entity of the StudentAggregate)
   - **Attributes**:
     - StudentID: A unique identifier for the student.
     - Name: The student's name.
     - Email: The student's email address.

        - Enrollments: A list of the student's course enrollments.
    - **Behavior**:
        - RegisterForCourse(courseOfferingID, StudentID): Initiates a new course registration.
        - DropCourse(courseOfferingID, StudentID): Removes an existing course registration.
        - GetRegisteredCourses(StudentID): List<Course>: Returns a list of all courses the student is currently enrolled in.

2. **Enrollment** (Belongs to both the StudentAggregate and the CourseOfferingAggregate)
    - **Attributes**:
        - EnrollmentID: A unique identifier for the enrollment.
        - StudentID: The ID of the associated student.
        - CourseOfferingID: The ID of the associated course offering.
        - EnrollmentStatus: The enrollment status ("Active", "Dropped").
        - Grade: The grade assigned to the student.
    - **Behavior**:
        - MarkAsDropped(): Updates the enrollment status to "Dropped".
        - AssignGrade(grade: String): Updates the student's grade.
3. **CourseOffering** (Root entity of the CourseOfferingAggregate)
    - **Attributes**:
        - CourseOfferingID: A unique identifier for the course offering.
        - CourseID: The ID of the associated course.
        - Semester: The semester in which the course is offered.
        - AvailableSeats: The number of seats available for registration.
    - **Behavior**:
        - DecreaseSeats(): Decreases the available seats count.
        - IncreaseSeats(): Increases the available seats count.
4. **Course** (Root entity of the CourseAggregate)
    - **Attributes**:

    - CourseID: A unique identifier for the course.
    - Title: The course title.
    - Credits: The number of credits for the course.
    - Department: The department offering the course.
  - **Behavior**:
    - GetDetails(): Returns detailed information about the course.
5. **Faculty**
  - **Attributes**:
    - Name: The name of the faculty.
    - ID: A unique identifier for the faculty.

Aggregates

1. **StudentAggregate**
   - **Root Entity**: Student.
   - **Boundaries**:
     - Encompasses the Student entity and its related Enrollments.
   - **Consistency Rules**:
     - A student can only be registered for a course once, ensured by the uniqueness of Enrollment.
2. **CourseOfferingAggregate**
   - **Root Entity**: CourseOffering.
   - **Boundaries**:
     - Includes CourseOffering and its associated Enrollments.
   - **Consistency Rules**:
     - The number of students registered must not exceed the course offering's maximum capacity.
3. **CourseAggregate**
   - **Root Entity**: Course.
   - **Boundaries**:
     - Only includes the Course entity.

  - **Consistency Rules**:
    - The course information must remain consistent with its associated course offerings.

Value Objects

1. **EnrollmentStatus**
   - **Attributes**:
     - Status: A string representing the enrollment status (e.g., "Active", "Dropped").
     - Timestamp: The timestamp indicating when the status was last updated.
   - **Behavior**:
     - IsDropped(): Boolean: Returns true if the status is "Dropped".
     - UpdateStatus(newStatus: String): EnrollmentStatus: Creates a new EnrollmentStatus object with the updated status and timestamp.
2. **CourseDetails**
   - **Attributes**:
     - Location: The location where the course will take place.
     - Schedule: Days and times of the course.
     - Syllabus: A document or description detailing the course content.
   - **Behavior**:
     - FormatDetails(): String: Returns a formatted string containing the course title, credits, and department.

Domain Services

1-DropService

Behavior:

1. **viewRegisteredCourses(studentID: String):**
   - Retrieves the list of courses the student is currently enrolled in.

    - Provides the ability to view registered courses before proceeding with the drop operation.

2. **removeCourse(studentID: String, courseOfferingID: String):**
    - Handles the removal of the specified course for the given student.
    - Retrieves the Enrollment object for the student and course offering.
    - Updates the Enrollment status to "Dropped".
    - Increases the available seats in the CourseOffering.
3. **removeStudent(courseOfferingID: String, studentID: String):**
    - Removes the student from the course offering by updating the course's list of enrolled students.
4. **updateEnrollment(courseID: String, studentID: String):**
    - Updates the enrollment record for the student, marking the course as dropped and ensuring that related dependencies (e.g., grades, records) are updated.

Responsibilities:

1. **Manage the Course Dropping Process:**
    - Coordinates the entire process of dropping a course for a student by interacting with the necessary aggregates and entities.
2. **Handle Data Consistency Across Aggregates:**
    - Ensures consistent updates between the **StudentAggregate** and **CourseOfferingAggregate** during the course removal process.
    - Updates enrollment records in the **Enrollment** entity and maintains the integrity of available seats in the **CourseOffering** entity.
3. **Enable Detailed Control Over Course Operations:**
    - Provides distinct methods (removeCourse, removeStudent, updateEnrollment) to manage specific aspects of the course dropping process, ensuring modularity and ease of testing.

2- RegistrationService

Behavior:

- RegisterStudentForCourse(student: Student, courseOffering: CourseOffering): Handles the registration process for a course.
  - Checks if the student is already enrolled in the course.
  - Ensures that the course offering has available seats.
  - Creates a new Enrollment for the student and course offering.
  - Decreases the available seats in the CourseOffering.

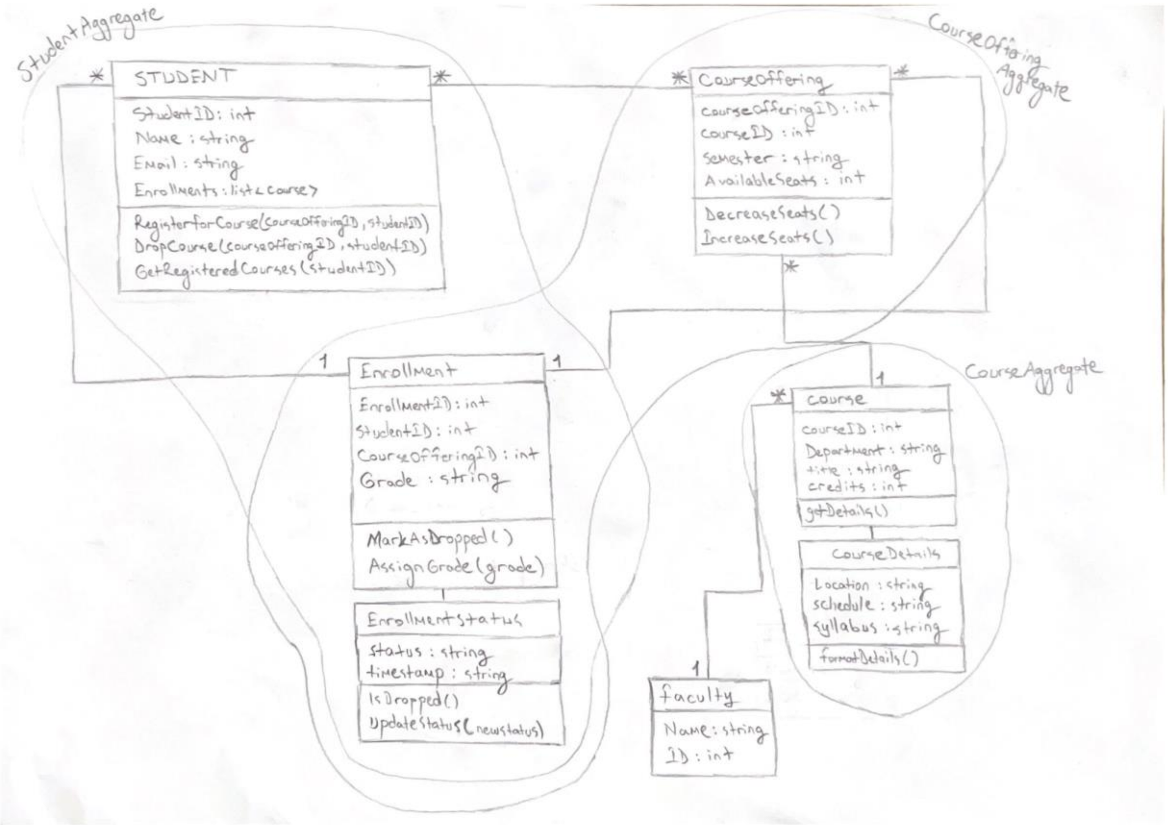
Responsibilities:

- Coordinates the logic for enrolling students in courses.
- Validates the prerequisites and ensures business rules are satisfied.

Detailed Descriptions for Each Object

- **Student**: Represents an individual student in the system, encapsulating personal information and managing relationships with course enrollments.
- **Enrollment**: Tracks the relationship between a student and a course offering, including enrollment status and grades.
- **CourseOffering**: Represents a specific instance of a course being offered in a semester, with attributes like capacity and availability.
- **Course**: Represents the core details of a course, such as its title, credits, and the department that offers it.
- **Faculty**: Represents faculty information related to the courses offered.
- **DropService**: Provides the necessary logic to handle dropping a course while maintaining consistency between the student and course offering aggregates.
- **RegistrationService**: Centralizes the logic for registering students in courses while validating business rules and maintaining consistency.

UML CLASS DIAGRAM



Part 3: Design the Use Case: Drop a Class

1. Design the Use Case by Applying Tactical Pattern Objects

Entities, Value Objects, Aggregates, and Domain Services Collaboration:

1. **Entities:**
   - **Student**:
     - GetRegisteredCourses(): Retrieves the list of courses the student is registered for.
     - DropCourse(courseOffering: CourseOffering): Drops a selected course.
   - **CourseOffering**:
     - Manages seat availability using IncreaseSeats().

  - **Enrollment**:
    - Tracks the relationship between the student and the course offering.
    - MarkAsDropped(): Marks the enrollment status as "Dropped".

2. **Value Objects:**
   - **EnrollmentStatus**:
     - Manages the enrollment status (e.g., "Active", "Dropped") and ensures consistency.

3. **Aggregates**

StudentAggregate:

- **Root Entity**: Student.
- **Description**:
  - Manages the student and their associated Enrollments.
  - Responsible for operations such as retrieving the list of registered courses (GetRegisteredCourses) and initiating or canceling enrollments (RegisterForCourse, DropCourse).

CourseOfferingAggregate:

- **Root Entity**: CourseOffering.
- **Description**:
  - Manages the details of a specific course offering (e.g., semester, available seats).
  - Handles operations related to enrollments, such as increasing or decreasing seat availability and associating enrollments with course offerings.

CourseAggregate:

- **Root Entity**: Course.
- **Description**:
  - Represents the general details of a course, such as its title, credits, and department.

  - Handles operations and information not specific to a single offering, such as retrieving prerequisites, managing the syllabus, or linking to the faculty offering the course.
  - Provides an abstraction for shared data used across multiple course offerings.
- **Responsibilities**:
  - Serves as the centralized entity for course-specific information.
  - Links with the Faculty entity to define which department or faculty owns the course.
  - Supports queries for general course details (GetDetails) and manages prerequisites or other requirements.

4. **Domain Services:**
   - **DropService**:
     - Orchestrates the process of dropping a course.
     - Workflow:
       1. Retrieves the Enrollment associated with the student and course offering.
       2. Updates the Enrollment status to "Dropped" using MarkAsDropped().
       3. Updates the course offering's seat availability using IncreaseSeats().

Flow of the Use Case:

1. **View Registered Courses**:
   - The student uses GetRegisteredCourses() to retrieve their current enrollments.
2. **Select a Course to Drop**:
   - The student chooses a course from the list of registered courses.
3. **Confirm the Drop**:
   - DropClassService is invoked to handle the drop operation:
     - Calls Student.DropCourse() to update the enrollment status.
     - Calls CourseOffering.IncreaseSeats() to update seat availability.

2. GRASP Responsibility Assignment

Basic GRASP Patterns

1. Creator

- **Responsibility**:
    - The object responsible for creating an instance of another object should:
        - Contain or aggregate the created object.
        - Use the created object.
- **Application in Drop a Class Use Case**:
    - **DropService**:
        - Responsible for creating or managing Enrollment objects when needed (e.g., retrieving and marking an Enrollment as "Dropped").
    - **Course**:
        - Responsible for creating or managing CourseDetails objects when course-specific details are needed.

---

2. Information Expert

- **Responsibility**:
    - Assign responsibility to the object that has the necessary information to fulfill the task.
- **Application in Drop a Class Use Case**:
    - **Student**:
        - Knows the list of registered courses (Enrollments).
        - Responsible for providing the list of registered courses through GetRegisteredCourses().
    - **Enrollment**:
        - Knows its status and manages changes (e.g., MarkAsDropped()).
    - **CourseOffering**:

        - Knows the available seats and is responsible for updating them (IncreaseSeats()).
    - **Course**:
        - Knows its title, credits, and other general information. Responsible for retrieving course-specific details.
    - **Faculty**:
        - Knows the list of courses offered under its department and provides that information when needed.

---

3. Low Coupling

- **Responsibility**:
    - Minimize dependencies between objects to reduce the impact of changes in one object on others.
- **Application in Drop a Class Use Case**:
    - **DropService**:
        - Acts as an intermediary between Student, Enrollment, and CourseOffering, ensuring that these entities interact minimally with each other.
        - Direct repository calls (StudentRepository, CourseOfferingRepository) reduce dependencies within entities.
    - **Faculty**:
        - Keeps its relationship with Course modular, providing information about its courses without tightly coupling with other aggregates.

---

4. Controller

- **Responsibility**:
    - Handle system events and coordinate tasks by delegating work to other objects.
- **Application in Drop a Class Use Case**:
    - **DropController**:
        - Handles the user's request to drop a class.

        - Delegates the task to the DropClassService to perform the required operations.

---

5. High Cohesion

- **Responsibility**:
    - Keep related responsibilities together within a single object, promoting a focused purpose and reducing complexity.
- **Application in Drop a Class Use Case**:
    - **Student**:
        - Focuses only on student-related data and operations (e.g., managing enrollments, retrieving registered courses).
    - **Enrollment**:
        - Manages enrollment-specific responsibilities such as status and grades.
    - **CourseOffering**:
        - Maintains responsibilities related to the specific course offering, such as seat availability and semester information.
    - **Course**:
        - Manages general course-related responsibilities, such as retrieving details or prerequisites.
    - **Faculty**:
        - Manages faculty-specific responsibilities, such as maintaining the list of courses offered by its department.

3. Architectural Design with the Port and Adapter Pattern

Port and Adapter Usage

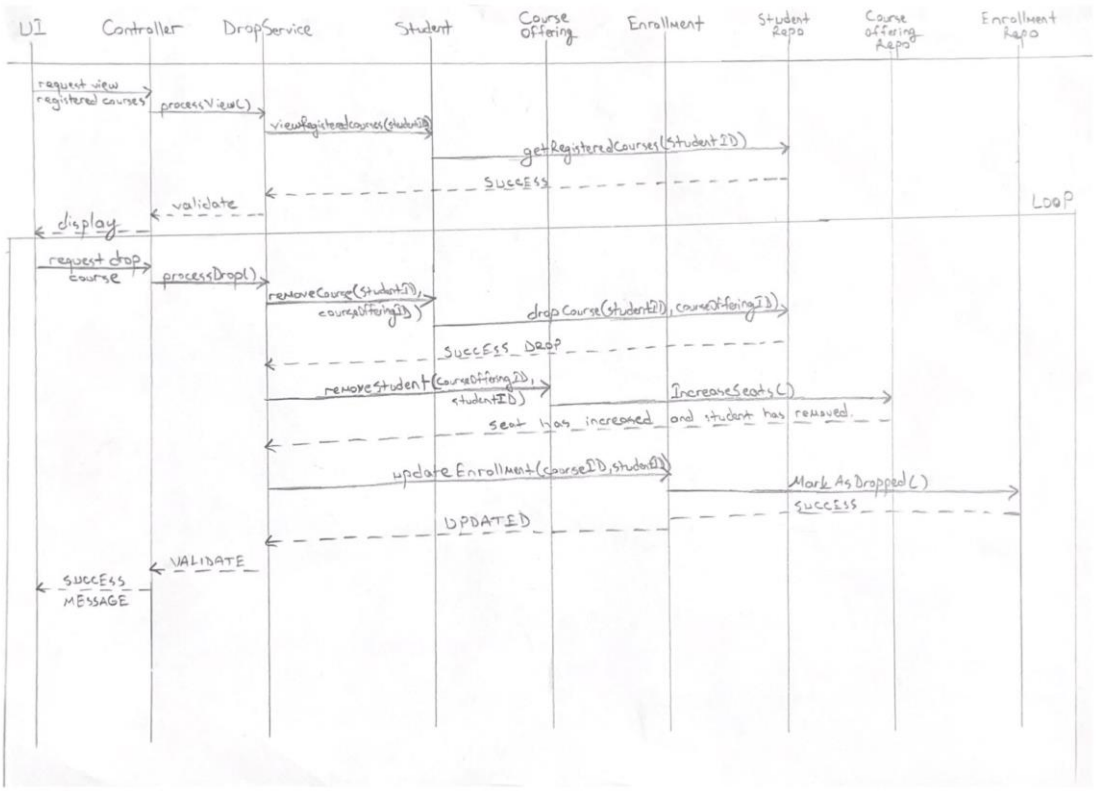
1. **Port**:
    - Repository interfaces provide an abstraction between the domain model and the infrastructure:
        - StudentRepository: Provides access to student data.
        - CourseOfferingRepository: Provides access to course offering data.

        - EnrollmentRepository: Provides access to enrollment data.
    - The application layer uses these ports to manage domain objects without directly interacting with the infrastructure.
2. **Adapter**:
    - Infrastructure-specific implementations of the repository interfaces:
        - SQLStudentRepository: Handles student data in the database.
        - SQLCourseOfferingRepository: Handles course offering data in the database.
        - SQLEnrollmentRepository: Handles enrollment data in the database.

Advantages of the Port and Adapter Pattern

1. **Independence**:
    - The domain model is not affected by changes in the infrastructure (e.g., database, user interface, or external systems).
2. **Flexibility and Maintainability**:
    - Changes made in the infrastructure layer (e.g., switching to a different database or persistence technology) have minimal impact on the domain model.
3. **Testability**:
    - The domain model can be tested independently of the infrastructure, thanks to the abstraction provided by ports and adapters.

UML SEQUENCE DIAGRAM


UI
Controller
DropService
Student
Course Offering
Enrollment
Student Repo
Course offering Repo
Enrollment Repo
request view registered courses
processView()
viewRegisteredcourses(studentID)
getRegisteredCourses(studentID)
SUCCESS
validate
display
LOOP
request drop course
processDrop()
removeCourse(studentID, courseOfferingID)
dropCourse(studentID, courseOfferingID)
SUCCESS_DROP
removeStudent(courseOfferingID, studentID)
IncreaseSeats()
seat has increased and student has removed.
updateEnrollment(courseID, studentID)
MarkAsDropped()
SUCCESS
UPDATED
VALIDATE
SUCCESS MESSAGE