`Hausaufgabe 2 - Deadline (7te Woche)`

Ziel: Implementieren Sie die Anwendungslogik, die in Hausaufgabe 1 beschrieben wurde.

Aufgaben und Anforderungen:

1. Erstellen einer Konsolenanwendung

Entwickeln Sie eine Konsolenanwendung, die als Benutzeroberfläche für die grundlegende Funktionalität der Anwendung dient. Diese Konsole wird die Interaktion zwischen dem Benutzer und der Anwendung ermöglichen und Funktionen für das Hinzufügen, Ändern, Löschen und Anzeigen von verwalteten Entitäten bieten (z. B. Personen, Objekte, Aktivitäten).

2. Implementierung einer Schichtenarchitektur basierend auf MVC

Die Anwendung muss in **einer vierstufigen Schichtenarchitektur** strukturiert sein: **`Presentation Layer`**, **`Controller Layer`**, **`Service Layer`** und **`Repository Layer`**.

Die Eigenschaften der einzelnen Schichten sind:

- **`Presentation Layer (Konsole)`**: Stellt die Benutzerschnittstelle dar, die Benutzereingaben empfängt und an den Controller weitergibt.
- **`Controller Layer`**: Vermittelt Anfragen aus dem Presentation Layer, führt Basisvalidierungen durch und leitet Anfragen an das Service Layer weiter.
- **`Service Layer`**: Enthält die Anwendungslogik und verwaltet die Geschäftsoperationen. Alle erweiterten Validierungsprüfungen (z. B. Zugangsgrenzen, Berechtigungsregeln) müssen hier implementiert werden. Das Service Layer darf nur mit dem Repository Layer zur Datenverwaltung interagieren.
- **Repository Layer:** Verwaltet den Datenzugriff und implementiert ein **`Repository-Muster`**, das mehrere Speicherarten unterstützt:
  - **`InMemoryRepository`**: Speichert Daten in der Arbeitsspeicher mithilfe einer Datenstruktur (z. B. Map) für die Verwaltung der Daten.
  - **`FileRepository`**: Speichert Daten in Dateien und ermöglicht so eine Datenpersistenz zwischen den Sitzungen.

- **`DBRepository`**: Speichert Daten in einer relationalen Datenbank für eine dauerhafte und sichere Datenspeicherung.

`Hinweis:` In dieser Projektiteration ist **`NUR`** die Implementierung des **`InMemoryRepository`** erforderlich.

3. Erstellen eines Repository Interfaces

Implementieren Sie ein generisches **`IRepository<T>`** Interface, das die grundlegenden CRUD-Methoden (`create, read, update, delete`) für die Verwaltung der Daten definiert. Dieses Interface muss von allen Repository-Typen (`InMemory`, `File` und `DB`) implementiert werden, um einen Wechsel zwischen verschiedenen Speicherarten ohne Änderungen an der Anwendungslogik zu ermöglichen.

4. Verwendung des Collections Framework

Verwenden Sie das **`Collections Framework`** (z. B. Listen, Sets, Maps), um Daten in der Anwendung zu strukturieren und zu verwalten.

5. Generieren der Javadocs

Dokumentieren Sie den gesamten Quellcode mit **`Javadocs`**, um die Funktion und das Verhalten jeder Klasse, Methode und jedes Interfaces zu erklären. Jede Klasse und Methode muss klar dokumentiert sein, um das Verständnis der implementierten Logik und Funktionen zu gewährleisten.

6. Einhalten der Java-Benennungskonventionen

Wenden Sie die Java-Benennungskonventionen auf alle Klassen, Methoden und Variablen im Projekt an.

Weitere Informationen:

- https://dzone.com/articles/an-introduction-to-the-java-collections-framework
- https://reflectoring.io/howto-format-code-snippets-in-javadoc/
- https://www.baeldung.com/solid-principles
- https://okso.app/showcase/solid

- https://refactoring.guru/design-patterns
- https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Beispiel zur Veranschaulichung der Architektur mit einer Universitätsanwendung

- **Presentation Layer (ConsoleApp):**
  - Dient als Einstiegspunkt für Benutzerinteraktionen wie die Anmeldung zu Kursen, das Anzeigen von Kursen und das Auflisten von Studierenden. Die Benutzereingaben werden an den **UniversityController** weitergeleitet.

- **Controller Layer (UniversityController):**
  - **UniversityController** empfängt Anfragen von der **ConsoleApp**, verarbeitet sie und ruft die entsprechenden Methoden im **UniversityService** auf.
  - Wichtige Methoden:
    - `enrollStudent()`: Ruft **UniversityService** auf, um einen Studierenden für einen Kurs anzumelden.
    - `viewCourses()`: Ruft **UniversityService** auf, um alle verfügbaren Kurse anzuzeigen.
    - `viewEnrolled()`: Ruft **UniversityService** auf, um die Studierenden eines bestimmten Kurses anzuzeigen.
    - `deleteCourse()`: Ruft **UniversityService** auf, um einen Kurs zu entfernen und gleichzeitig die Studierenden von diesem Kurs abzumelden.

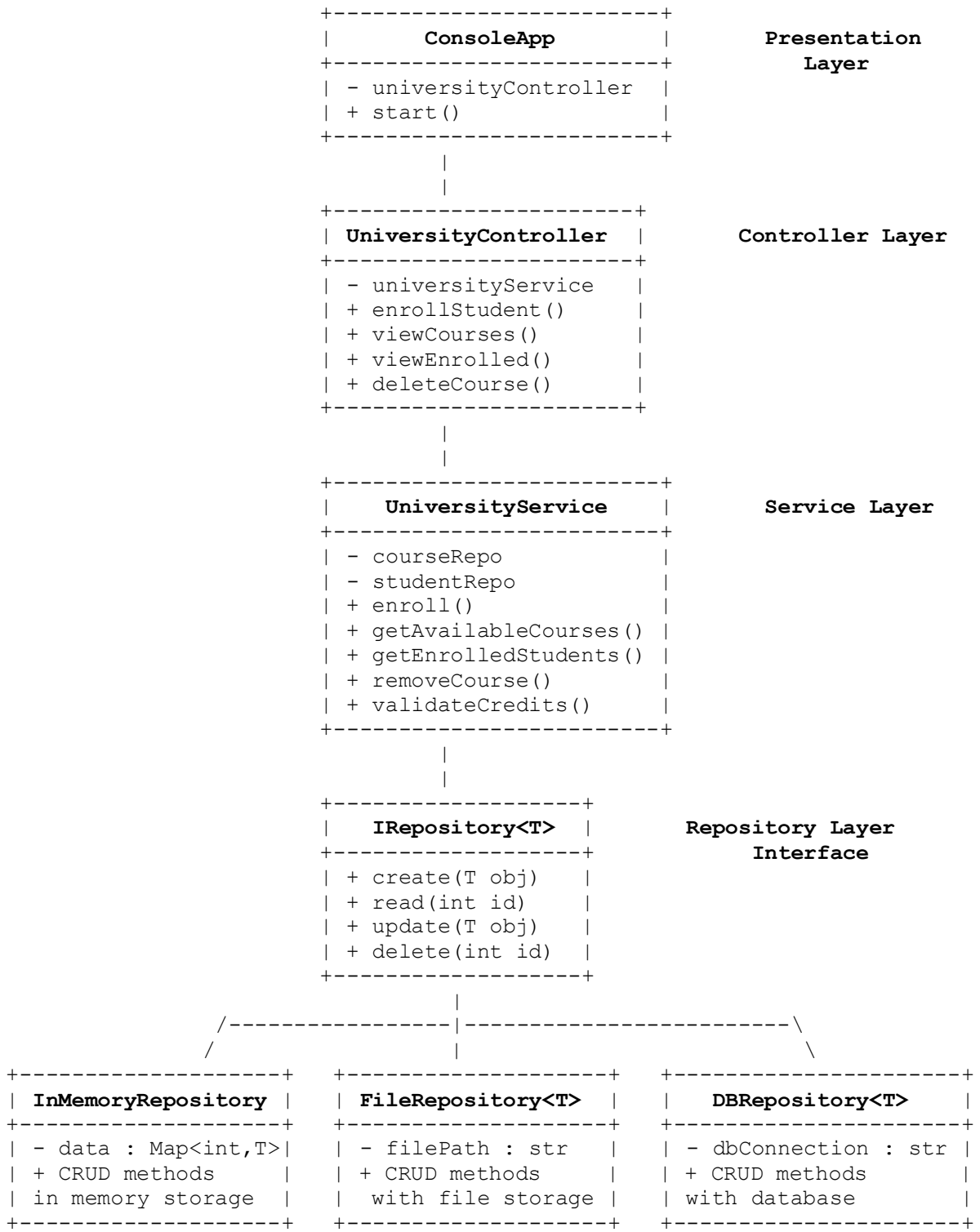
- **Service Layer (UniversityService):**
  - Enthält die Hauptlogik, wie die Verwaltung der Einschreibungen, Verfügbarkeit von Kursen und Verwaltung der Credits.
  - Wichtige Methoden:
    - `enroll()`: Fügt einen Studierenden zu einem Kurs hinzu, sofern Plätze verfügbar und das Creditlimit nicht überschritten ist.
    - `getAvailableCourses()`: Ruft eine Liste der Kurse mit offenen Plätzen ab.
    - `getEnrolledStudents()`: Ruft die Liste der in einem bestimmten Kurs eingeschriebenen Studierenden ab.

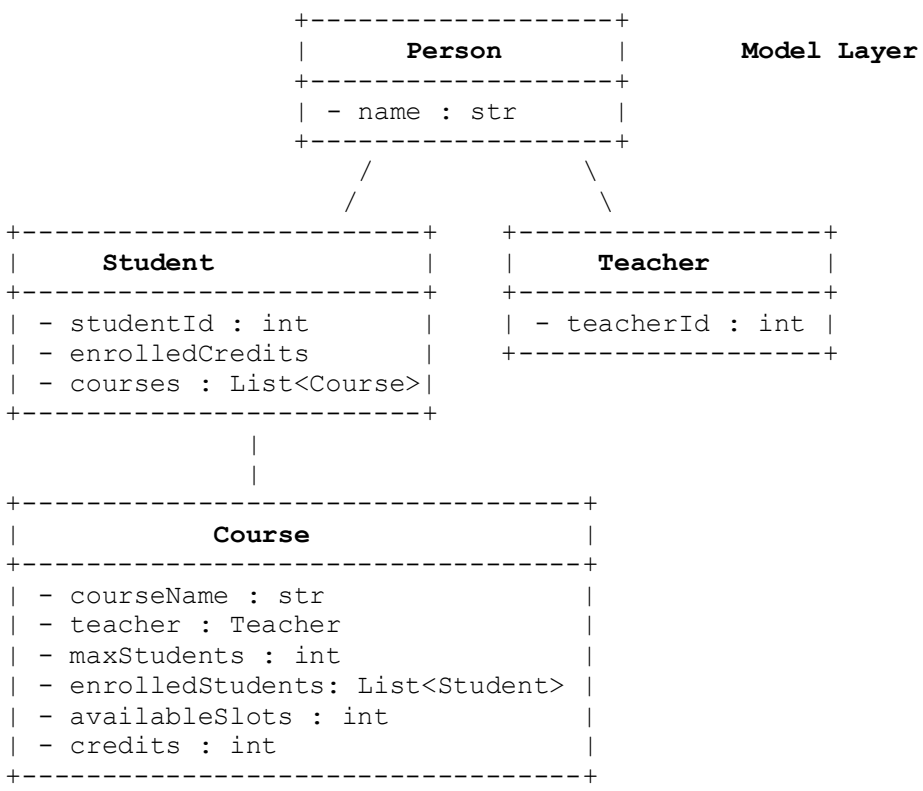
- `removeCourse()`: Löscht einen Kurs und entfernt ihn aus der Liste der eingeschriebenen Kurse der Studierenden.
- `validateCredits()`: Stellt sicher, dass Studierende das Maximum von 30 Credits nicht überschreiten.

- **Repository Interface Layer (IRepository<T>):**
  - Definiert die Standard `CRUD`-Operationen, die von allen Repository-Typen implementiert werden müssen:
    - `create(T obj)`: Fügt eine neue Entität hinzu.
    - `read(int id)`: Ruft eine Entität anhand ihrer `ID` ab.
    - `update(T obj)`: Aktualisiert eine bestehende Entität.
    - `delete(int id)`: Entfernt eine Entität anhand ihrer `ID`.

  - **Repository Implementierungen:**
    - Jedes Repository implementiert **`IRepository<T>`** für den Datenzugriff und `CRUD`-Operationen.
      - **`InMemoryRepository<T>`**: Speichert Daten in einer `Map<int, T>` für schnellen Zugriff.
      - **`FileRepository<T>`**: Speichert Daten in einer Datei (z. B. `JSON, CSV`) und führt dateibasierte `CRUD`-Operationen durch.
      - **`DBRepository<T>`**: Verwendet eine relationale Datenbank zur sicheren Speicherung und Verwaltung der Daten.

  - **Model Layer:**
    - **`Person`**: Eine Basisklasse mit gemeinsamen Feldern, die von **`Student`** und **`Teacher`** erweitert wird.
    - **`Student`**: Repräsentiert einen Studierenden, mit Attributen wie `studentId`, eingeschriebene Credits und eingeschriebene Kurse.
    - `Teacher`: Repräsentiert einen Lehrer, mit Attributen wie `teacherId` und den Kursen, die er verwaltet.
    - **`Course`**: Repräsentiert einen Kurs mit Feldern wie Kursname, Lehrer, maximale Teilnehmerzahl, eingeschriebene Studierende, Credits und verfügbaren Plätzen.


ConsoleApp
- universityController
+ start()
Presentation Layer
UniversityController
- universityService
+ enrollStudent()
+ viewCourses()
+ viewEnrolled()
+ deleteCourse()
Controller Layer
UniversityService
- courseRepo
- studentRepo
+ enroll()
+ getAvailableCourses()
+ getEnrolledStudents()
+ removeCourse()
+ validateCredits()
Service Layer
IRepository<T>
+ create(T obj)
+ read(int id)
+ update(T obj)
+ delete(int id)
Repository Layer Interface
InMemoryRepository
- data : Map<int,T>
+ CRUD methods
in memory storage
FileRepository<T>
- filePath : str
+ CRUD methods
with file storage
DBRepository<T>
- dbConnection : str
+ CRUD methods
with database


Person
Model Layer
- name : str
Student
- studentId : int
- enrolledCredits
- courses : List<Course>
Teacher
- teacherId : int
Course
- courseName : str
- teacher : Teacher
- maxStudents : int
- enrolledStudents: List<Student>
- availableSlots : int
- credits : int