```java
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.*;
import cn.edu.sustech.cs307.dto.prerequisite.Prerequisite;

import javax.annotation.Nullable;
import javax.annotation.ParametersAreNonnullByDefault;
import java.time.DayOfWeek;
import java.util.List;

@ParametersAreNonnullByDefault
public interface CourseService {
    /**
     * Add one course according to following parameters.
     * If some of parameters are invalid, throw {@link
cn.edu.sustech.cs307.exception.IntegrityViolationException}
     *
     * @param courseId represents the id of course. For example, CS307,
CS309
     * @param courseName the name of course
     * @param credit the credit of course
     * @param classHour The total teaching hour that the course spends.
     * @param grading the grading type of course
     * @param coursePrerequisite The root node of prerequisite.{@link
cn.edu.sustech.cs307.dto.prerequisite.Prerequisite}
     */
    void addCourse(String courseId, String courseName, int credit, int
classHour,
                   Course.CourseGrading grading, @Nullable Prerequisite
coursePrerequisite);

    /**
     * Add one course section according to following parameters:
     * If some of parameters are invalid, throw {@link
cn.edu.sustech.cs307.exception.IntegrityViolationException}
     *
     * @param courseId represents the id of course. For example, CS307,
CS309
     * @param semesterId the id of semester
     * @param sectionName the name of section {@link
cn.edu.sustech.cs307.dto.CourseSection}
     * @param totalCapacity the total capacity of section
```

```
     * @return the CourseSection id of new inserted line, if adding
process is successful.
     */
    int addCourseSection(String courseId, int semesterId, String
sectionName, int totalCapacity);




    /**
     * Add one course section class according to following parameters:
     * If some of parameters are invalid, throw {@link
cn.edu.sustech.cs307.exception.IntegrityViolationException}
     *
     * @param sectionId
     * @param instructorId
     * @param dayOfWeek
     * @param weekList
     * @param classStart
     * @param classEnd
     * @param location
     * @return the CourseSectionClass id of new inserted line.
     */
    int addCourseSectionClass(int sectionId, int instructorId, DayOfWeek
dayOfWeek,List<Short> weekList,
                              short classStart, short classEnd, String
location);


    void removeCourse(String courseId);

    void removeCourseSection(int sectionId);

    void removeCourseSectionClass(int classId);

    List<CourseSection> getCourseSectionsInSemester(String courseId, int
semesterId);

    Course getCourseBySection(int sectionId);

    /**
     *
     * @param sectionId the id of {@code CourseSection}
```

```
     * @return
     */
    List<CourseSectionClass> getCourseSectionClasses(int sectionId);

    CourseSection getCourseSectionByClass(int classId);

    List<Student> getEnrolledStudentsInSemester(String courseId, int
semesterId);
}
```

```
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.Department;

import javax.annotation.ParametersAreNonnullByDefault;
import java.util.List;

@ParametersAreNonnullByDefault
public interface DepartmentService {
    int addDepartment(String name);

    void removeDepartment(int departmentId);

    List<Department> getAllDepartments();

    Department getDepartment(int departmentId);
}
```

```
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.CourseSection;

import javax.annotation.ParametersAreNonnullByDefault;
import java.util.List;
```

```java
@ParametersAreNonnullByDefault
public interface InstructorService {
    void addInstructor(int userId, String firstName, String lastName);

    /**
     *
     * @param instructorId
     * @param semesterId
     * @return
     */
    List<CourseSection> getInstructedCourseSections(int instructorId, int
semesterId);
}
```

```java
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.Major;

import javax.annotation.ParametersAreNonnullByDefault;
import java.util.List;

@ParametersAreNonnullByDefault
public interface MajorService {
    int addMajor(String name, int departmentId);

    void removeMajor(int majorId);

    List<Major> getAllMajors();

    Major getMajor(int majorId);

    /**
     * Binding a course id {@code courseId} to major id {@code majorId},
and the selection is compulsory.
     * @param majorId the id of major
     * @param courseId the course id
     */
```

```
    void addMajorCompulsoryCourse(int majorId, String courseId);

    /**
     * Binding a course id{@code courseId} to major id {@code majorId},
and the selection is elective.
     * @param majorId the id of major
     * @param courseId the course id
     */
    void addMajorElectiveCourse(int majorId, String courseId);
}
```

```
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.Semester;

import javax.annotation.ParametersAreNonnullByDefault;
import java.sql.Date;
import java.util.List;

@ParametersAreNonnullByDefault
public interface SemesterService {
    /**
     * Add one semester according to following parameters:
     * If some of parameters are invalid, throw {@link
cn.edu.sustech.cs307.exception.IntegrityViolationException}
     * @param name
     * @param begin
     * @param end
     * @return the Semester id of new inserted line, if adding process is
successful.
     */
    int addSemester(String name, Date begin, Date end);

    void removeSemester(int semesterId);

    List<Semester> getAllSemesters();

    Semester getSemester(int semesterId);
```

```
}
```

```
package cn.edu.sustech.cs307.service;

import cn.edu.sustech.cs307.dto.*;
import cn.edu.sustech.cs307.dto.grade.Grade;

import javax.annotation.Nullable;
import javax.annotation.ParametersAreNonnullByDefault;
import java.sql.Date;
import java.time.DayOfWeek;
import java.util.List;
import java.util.Map;

/**
 *
 */
@ParametersAreNonnullByDefault
public interface StudentService {
    enum EnrollResult {
        /**
         * Enrolled successfully
         */
        SUCCESS,
        /**
         * Cannot found the course section
         */
        COURSE_NOT_FOUND,
        /**
         * The course section is full
         */
        COURSE_IS_FULL,
        /**
         * The course section is already enrolled by the student
         */
        ALREADY_ENROLLED,
        /**
         * The course (of the section) is already passed by the student
```

```
         */
        ALREADY_PASSED,
        /**
         * The student misses prerequisites for the course
         */
        PREREQUISITES_NOT_FULFILLED,
        /**
         * The student's enrolled courses has time conflicts with the
section,
         * or has course conflicts (same course) with the section.
         */
        COURSE_CONFLICT_FOUND,
        /**
         * Other (unknown) errors
         */
        UNKNOWN_ERROR
    }

    enum CourseType {
        /**
         * All courses
         */
        ALL,
        /**
         * Courses in compulsory courses of the student's major
         */
        MAJOR_COMPULSORY,
        /**
         * Courses in elective courses of the student's major
         */
        MAJOR_ELECTIVE,
        /**
         * Courses only in other majors than the student's major
         */
        CROSS_MAJOR,
        /**
         * Courses not belong to any major's requirements
         */
        PUBLIC
    }

    /**
     * Add one student according to following parameters.
```

```
     * If some of parameters are invalid, throw {@link
cn.edu.sustech.cs307.exception.IntegrityViolationException}
     *
     * @param userId
     * @param majorId
     * @param firstName
     * @param lastName
     * @param enrolledDate
     */
    void addStudent(int userId, int majorId, String firstName, String
lastName, Date enrolledDate);

    /**
     * Search available courses (' sections) for the specified student in
the semester with extra conditions.
     *
     * @param studentId
     * @param semesterId
     * @param searchCid                  search course id. Rule: searchCid
in course.id
     * @param searchName                 search course name. Rule:
searchName in "course.name[section.name]"
     * @param searchInstructor           search instructor name.
     *                                   Rule: firstName + lastName begins
with searchInstructor
     *                                   or firstName + ' ' + lastName
begins with searchInstructor
     *                                   or firstName begins with
searchInstructor
     *                                   or lastName begins with
searchInstructor.
     * @param searchDayOfWeek            search day of week. Matches *any*
class in the section in the search day of week.
     * @param searchClassTime            search class time. Matches *any*
class in the section contains the search class time.
     * @param searchClassLocations       search class locations. Matches
*any* class in the section contains *any* location from the search class
locations.
     * @param searchCourseType           search course type. See {@link
cn.edu.sustech.cs307.service.StudentService.CourseType}
     * @param ignoreFull                 whether or not to ignore full
course sections.
```

```
     * @param ignoreConflict             whether or not to ignore time-
conflicting course sections.
     * @param ignorePassed               whether or not to ignore the
student's passed courses.
     * @param ignoreMissingPrerequisites whether or not to ignore courses
with missing prerequisites.
     * @param pageSize                   the page size, effectively `limit
pageSize`.
     * @param pageIndex                  the page index, effectively
`offset pageIndex * pageSize`.
     * @return a list of search entries. See {@link
cn.edu.sustech.cs307.dto.CourseSearchEntry}
     */
    List<CourseSearchEntry> searchCourse(int studentId, int semesterId,
@Nullable String searchCid,
                                         @Nullable String searchName,
@Nullable String searchInstructor,
                                         @Nullable DayOfWeek
searchDayOfWeek, @Nullable Short searchClassTime,
                                         @Nullable List<String>
searchClassLocations,
                                         CourseType searchCourseType,
                                         boolean ignoreFull, boolean
ignoreConflict,
                                         boolean ignorePassed, boolean
ignoreMissingPrerequisites,
                                         int pageSize, int pageIndex);

    /**
     * It is the course selection function according to the studentId and
courseId.
     * The test case can be invalid data or conflict info, so that it can
return 8 different
     * types of enroll results.
     *
     * @param studentId
     * @param sectionId the id of CourseSection
     * @return See {@link
cn.edu.sustech.cs307.service.StudentService.EnrollResult}
     */
    EnrollResult enrollCourse(int studentId, int sectionId);

    /**
```

```
     * Drop a course section for a student
     *
     * @param studentId
     * @param sectionId
     * @throws IllegalStateException if the student already has a grade
for the course section.
     */
    void dropCourse(int studentId, int sectionId) throws
IllegalStateException;

    /**
     * It is used for importing existing data from other sources.
     * <p>
     * With this interface, staff for teaching affairs can bypass the
     * prerequisite fulfillment check to directly enroll a student in a
course
     * and assign him/her a grade.
     *
     * @param studentId
     * @param sectionId
     * @param grade     Can be null
     */
    void addEnrolledCourseWithGrade(int studentId, int sectionId,
@Nullable Grade grade);

    /**
     * For teachers who can give student a grade
     *
     * @param studentId student id is in database
     * @param sectionId section id in test cases that have selected by the
student
     * @param grade     a new grade
     */
    void setEnrolledCourseGrade(int studentId, int sectionId, Grade
grade);

    /**
     * Queries grades of all enrolled courses in the given semester for
the given student
     *
     * @param studentId
     * @param semesterId the semester id, null means return all semesters'
result.
```

```
     * @return A map from enrolled courses to corresponding grades.
     * If the grade is a hundred-mark score, the value should be wrapped
by a
     * {@code HundredMarkGrade} object.
     * If the grade is pass or fail, the value should be {@code
PassOrFailGrade.PASS}
     * or {@code PassOrFailGrade.FAIL} respectively.
     * If the grade is not set yet, the value should be null.
     */
    Map<Course, Grade> getEnrolledCoursesAndGrades(int studentId,
@Nullable Integer semesterId);

    /**
     * Return a course table in current week according to the date.
     *
     * @param studentId
     * @param date
     * @return the student's course table for the entire week of the date.
     * Regardless which day of week the date is, return Monday-to-Sunday
course table for that week.
     */
    CourseTable getCourseTable(int studentId, Date date);

    /**
     * check whether a student satisfy a certain course's prerequisites.
     *
     * @param studentId
     * @param courseId
     * @return
     */
    boolean passedPrerequisitesForCourse(int studentId, String courseId);

    Major getStudentMajor(int studentId);
}
```

```
package cn.edu.sustech.cs307.service;
```

```java
import cn.edu.sustech.cs307.dto.User;

import javax.annotation.ParametersAreNonnullByDefault;
import java.util.List;

@ParametersAreNonnullByDefault
public interface UserService {
    void removeUser(int userId);

    List<User> getAllUsers();

    User getUser(int userId);
}
```