在完成`searchCourse()`这个方法时，写了一些辅助类和辅助方法如下：

类名/方法名	作用
`public static class CouToSec`	将Course和对应的CourseSection链接起来
`public static class searchProperty`	集合了一系列课程的性质，用于应对`searchCourse()`中的四个`boolean`参数和`COurseType`
`public CourseType getCourseType (int studentId, String courseId)`	用于辅助寻找对应course的类型

```
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);
```

`searchCourse`中有许多`@Nullable`参数，是这个方法设计的难点。由于数据库中`null`的特性，显然直接用等号来判断是十分不正确的，但是我们可以通过在sql语句中使用`where ? is null or col = ?`来较为容易地规避掉不确定是否为null的情况。部分sql语句如左图，部分对应的参数设置如右图。

```
left outer join select_course as sc on sc.section_id = sec.id" +
where (cou.id like ('%' || ? || '%') or ? is null)" +
  and (cou.name || '[' || sec.name || ']' like '%' || ? || '%' or ? is null)" +
  and (? is null or i.firstname || i.lastname like ('%' || ? || '%')" +
    or (i.firstname || ' ' || i.lastname like ('%' || ? || '%'))" +
    or i.firstname like ('%' || ? || '%') or i.lastname like ('%' || ? || '%'))"
  and (? is null or cla.dayofweek = ?)" +
  and (? is null or ? between cla.classbegin and cla.classend)" +
  and (sec.semester_id = ?)" +
  and (? is null or cla.location like ('%' || ? || '%'))" +
```

```
//@Nullable String searchCid
if (searchCid == null) {
    stmt.setNull( parameterIndex: 1, Types.VARCHAR);
    stmt.setNull( parameterIndex: 2, Types.VARCHAR);
} else {
stmt.setString( parameterIndex: 1, searchCid);
stmt.setString( parameterIndex: 2, searchCid);
}

//@Nullable String searchName
if (searchName == null) {
    stmt.setNull( parameterIndex: 3, Types.VARCHAR);
    stmt.setNull( parameterIndex: 4, Types.VARCHAR);
} else {
stmt.setString( parameterIndex: 3, searchName);
stmt.setString( parameterIndex: 4, searchName);
}
```

由于返回值是`List<CourseSearchEntry>`，在查看了注释后发现，该类有四个属性（Course，CourseSection，Set< CourseSectionClass >，List< String >）（最后的`List<String> conflictCourseNames`未在截图里面出现），前三个的关系是一个course对应一个section，一个section对应多个class（用`Set`是为了去重，防止多个相同的class在一个CourseSearchEntry中）。

```java
/**
 * The course of the searched section
 */
public Course course;

/**
 * The searched course section
 */
public CourseSection section;

/**
 * All classes of the section
 */
public Set<CourseSectionClass> sectionClasses;
```

于是，我们将处理好的Course和CourseSection对象合为一个CouToSec对象，并重写它的`equals()`和`hashCode()`方法，便于与后续class在`Map`中处理。由于可能会出现学生选课后使得CourseSection的leftCapacity不一样（但实际上仍被视为同一个CourseSection），导致`Map`中出现映射错误，于是在重写的`hasCode()`方法中只加入Course、CourseSection的id、CourseSection的name。

```java
public static class CouToSec {
    Course cou;
    CourseSection sec;

    public CouToSec(Course cou, CourseSection sec) {
        this.cou = cou;
        this.sec = sec;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || this.getClass() != o.getClass()) return false;
        CouToSec ano = (CouToSec) o;
        return Objects.equals(cou, ano.cou) && Objects.equals(sec, ano.sec);
    }

    @Override
    public int hashCode() { return Objects.hash(cou, sec.id, sec.name); }
}
```

为了应对参数中的四个`boolean`以及`CourseType`，设计了一个类来存储所有性质。并且在处理后可通过`(sp.Full && ignoreFull)||(sp.Passed && ignorePassed)||(sp.Conflict && ignoreConflict)||(sp.MissingPrerequisites && ignoreMissingPrerequisites)`来一次性解决全部`boolean`，然后用`searchCourseType == CourseType.ALL || sp.courseType == searchCourseType`来检查`CourseType`是否符合要求。

```java
public static class searchProperty {
    CourseType courseType = CourseType.ALL;
    boolean Full = false;
    boolean Conflict = false;
    boolean Passed = false;
    boolean MissingPrerequisites = false;
}
```

```java
//处理ignoreFull, ignoreConflict, ignorePassed, ignoreMissingPrerequisites
Map<CourseSearchEntry, searchProperty> propertyMap = new HashMap<>();
for (CourseSearchEntry cse : tmp1) {
    searchProperty sp = new searchProperty();
    sp.Full = isClassFull(cse.section.id);
    sp.Conflict = isCourseConflict(studentId, cse.section.id);
    sp.Passed = isCoursePassed(studentId, cse.section.id);
    sp.MissingPrerequisites = !passedPrerequisitesForCourse(studentId, cse.course.id);
    sp.courseType = getCourseType(studentId, cse.course.id);
    propertyMap.put(cse, sp);
}
List<CourseSearchEntry> tmp2 = new ArrayList<>();
for (Map.Entry<CourseSearchEntry, searchProperty> entry : propertyMap.entrySet()) {
    CourseSearchEntry cse = entry.getKey();
    searchProperty sp = entry.getValue();
    //直接跳过
    if ((sp.Full && ignoreFull) || (sp.Passed && ignorePassed)
            || (sp.Conflict && ignoreConflict) || (sp.MissingPrerequisites && ignoreMissingPrerequisites))
        continue;
    //检查CourseType
    if (searchCourseType == CourseType.ALL || sp.courseType == searchCourseType) tmp2.add(cse);
}
```

在`enrollCourse()`中，同样设计了一些辅助方法。

类名/方法名	作用
`public EnrollResult firstTwoResult(int studentId, int sectionId)`	用于判断`COURSE_NOT_FOUND`和`ALREADY_ENROLLED`
`public boolean isCoursePassed(int studentId, int sectionId)`	用于判断`ALREADY_PASSED`
`public boolean isCourseConflict(int studentId, int sectionId)`	用于判断`COURSE_CONFLICT_FOUND`
`public boolean isClassFull(int sectionId)`	用于判断`COURSE_IS_FULL`

另外，对`PREREQUISITE_NOT_FULFILLED`则调用了本类中的另一个方法`passedPrerequisitesForCourse()`。而`SUCCESS`和`UNKNOWN_ERROR`则在前面所有判断都不符合时，通过一个简单的sql语句和`executeUpdate()`来判断。

```java
//SUCCESS UNKNOWN_ERROR
stmt = con.prepareStatement( sql: "insert into select_course(student_id,section_id) values (?,?)");
stmt.setInt( parameterIndex: 1, studentId);
stmt.setInt( parameterIndex: 2, sectionId);
int cnt = stmt.executeUpdate();
if (cnt == 1) return EnrollResult.SUCCESS;
else return EnrollResult.UNKNOWN_ERROR;
```

在`getCourseTable()`中，比较难的是怎么能获得date的所在周。通过查看semester.json文件可以发现，每个学期的begin_date刚好是该学期第一周的周一，于是我们可以通过`(? - sm.begin_date) / 7 + 1`来获得date的所在周（? 填入date）。

源码如下。

```
@Override
    public CourseTable getCourseTable(int studentId, Date date) {
        try (Connection con = SQLDataSource.getInstance().getSQLConnection()) {
            String sql = "select sm.id                                as smid,"
+
                    "       (? - sm.begin_date) / 7 + 1          as smweek," +
                    "       cou.name                             as coursename,"
+
                    "       i.id                                 as iid," +
                    "       getfullname(i.firstname, i.lastname) as iname," +
                    "       cla.classbegin," +
                    "       cla.classend," +
                    "       cla.location," +
                    "       cla.dayofweek," +
                    "       sec.name                             as sectionname"
+
                    "from semesters as sm" +
                    "         inner join sections sec on sec.semester_id =
sm.id" +
                    "         inner join classes cla on sec.id = cla.section_id
and cla.dayofweek = (? - sm.begin_date) / 7 + 1" +
                    "         inner join courses cou on cou.id = sec.course_id"
+
                    "         inner join class_instructor ci on ci.class_id =
cla.id" +
                    "         inner join instructors i on i.id =
ci.instructor_id" +
                    "         inner join" +
                    "     ((select sc.section_id from select_course sc where
sc.student_id = ?)" +
                    "      union all" +
                    "      (select sgp.section_id from student_grades_pf sgp
where sgp.student_id = ?)" +
                    "      union all" +
                    "      (select sgh.section_id from student_grades_hundred
sgh where sgh.student_id = ?)) scs" +
                    "     on scs.section_id = sec.id" +
                    "where ? between sm.begin_date and sm.end_date";

            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setDate(1, date);
            stmt.setDate(2, date);
            stmt.setInt(3, studentId);
            stmt.setInt(4, studentId);
            stmt.setInt(5, studentId);
            stmt.setDate(6, date);

            ResultSet rs = stmt.executeQuery();
```

```
            CourseTable courseTable = new CourseTable();
            courseTable.table = new HashMap<>();
            for (DayOfWeek d : DayOfWeek.values()) {
                courseTable.table.put(d, new HashSet<>());
            }

            while (rs.next()) {
                CourseTable.CourseTableEntry entry = new
CourseTable.CourseTableEntry();
                entry.courseFullName = String.format("%s[%s]", rs.getString(3),
rs.getString(10));

                Instructor instructor = new Instructor();
                instructor.id = rs.getInt(4);
                instructor.fullName = rs.getString(5);
                entry.instructor = instructor;

                entry.classBegin = rs.getShort(6);
                entry.classEnd = rs.getShort(7);
                entry.location = rs.getString(8);

                DayOfWeek day = DayOfWeek.of(rs.getInt(9));
                courseTable.table.get(day).add(entry);
            }
            return courseTable;
        } catch (Exception e) {
            e.printStackTrace();
            throw new IntegrityViolationException();
        }
    }
```