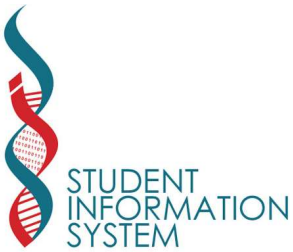
CSCI-140 Computer Science AP
CSCI-242 Computer Science Transfers
Lab 5 - SIS

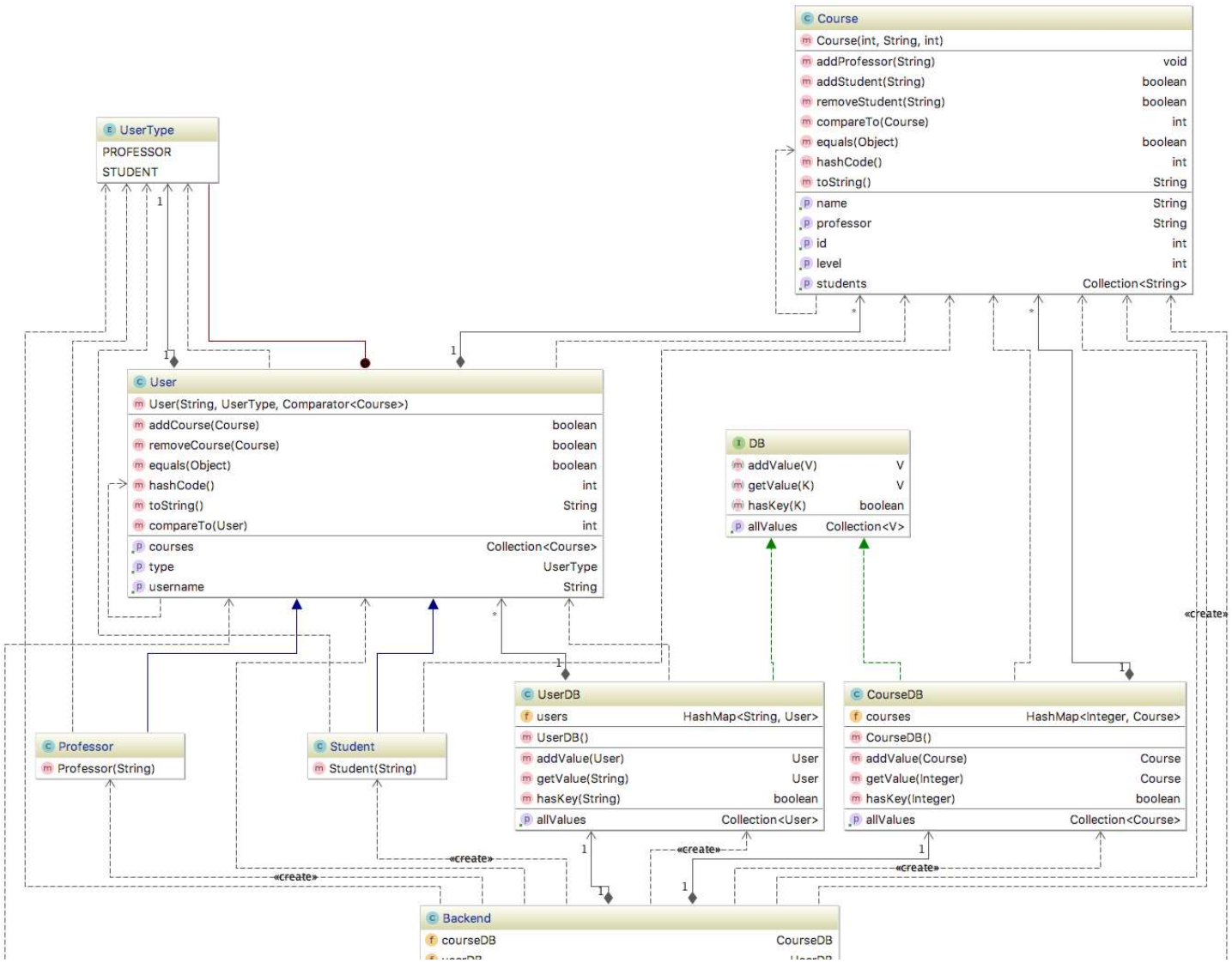


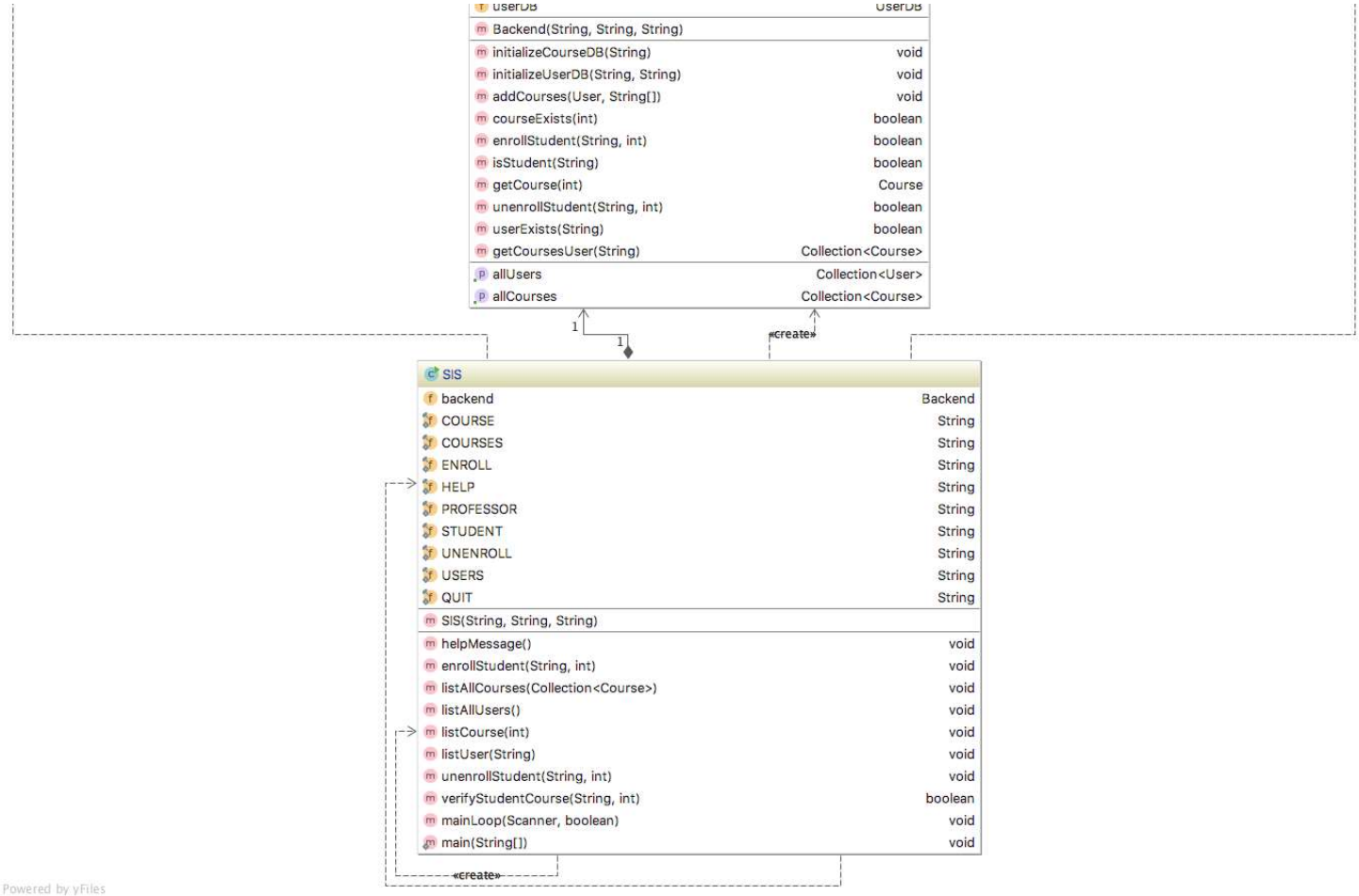
Introduction

The RIT Student Information System, SIS, is a service provided to students, faculty and staff to manage various academic activities at the university, e.g. course enrollment and schedule viewing.

Students will be designing and developing an extremely simplified version of this system that allows courses to be created, instructors to be created that teach the courses, and students that can be created who can add and remove courses from their schedule.

Design





Documentation

- SIS Documentation: The main documentation page for all classes.

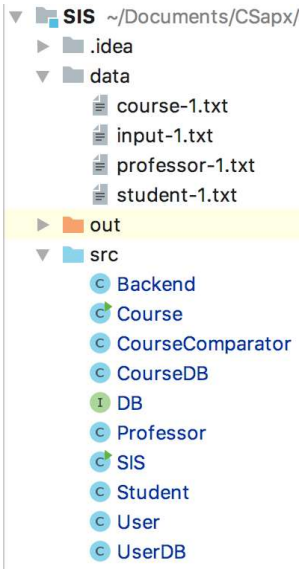
Implementation

Starter Code

Download the starter code from here. Unzip it and copy the `data` and `src` directories into your project.

Project Structure

Your project structure should look like this. Do not use packages for this assignment.



SIS and Backend

SIS is the main program. Read the documentation to see how the program is run and how the commands work. It requires three data files that are processed by Backend. Read the documentation to see the format of these files.

In the starter code you were given a small sample of each data file to be stored in the `data` directory at the root of your project, `course-1.txt`, `professor-1.txt` and `student-1.txt`. In addition, a sample input file, `input-1.txt` is provided. You can run the main program with all the files as:

```
$ java SIS data/course-1.txt data/professor-1.txt data/student-1.txt data/input-1.txt
```

The main program is set up to read the command line arguments in and then pass control to the main loop. The structure of reading the input in the main loop is given to you, as well as the handling of the `help` and `quit` commands. Your job is to implement the remaining commands. Look carefully at the private helper methods as they explain how to deal with error conditions, as well as how they communicate with the backend.

Course, CourseDB and DB

Your first task should be implementing the constructor for the backend and reading in the course data file. Each line of the course file corresponds to a Course object. You should read each of the courses and store them inside CourseDB.

Since we haven't covered File I/O in Java yet, here is an example of how you can open a file and attach it to a Scanner. Assume that `courseFile` is a string that points to the course file from your project root, e.g. `"data/course-1.txt"`:

```
try (Scanner in = new Scanner(new File(courseFile))) {
    while (in.hasNext()) {
        String[] fields = in.nextLine().split(",");
        // fields[0] is the course id, as a String
        // fields[1] is the course name, as a String
        // fields[2] is the course level, as a String
        ...
    }
}
```

Because an exception can be thrown if the file does not exist, you must declare the Backend constructor, and the private helper methods it uses, to throw `FileNotFoundException`, e.g.:

```
public Backend(String courseFile, String professorFile, String studentFile) throws FileNotFoundException {
    ...
}
```

You can use the String split method to split a string with a comma delimeter by doing:

```
// assume s is the String, "1,Computer_Science_1,100"
String[] fields = s.split(",");
// fields[0] = "1"
// fields[1] = "Computer_Science_1"
// fields[2] = "100"
```

The course database implements the generic DB interface. It represents a simplified interface to an associative array, e.g. Map. The course database uses course id's (integers) as the key, and the course object as a value. Because the lookup time for courses should be in constant time, it must be implemented as a HashMap.

You should complete the implementation of the Course class that you started in the in-lab activity. Then you can focus on implementing your first commands, `course` for listing a single course, and `courses` for listing all the courses. Look over the documentation in SIS and Backend to understand how to do this.

User, Student, Professor and UserDB

After you finish working on courses, you can now focus on the second type of object in our system, User. There are two types of users, Professor and Student. Read over the documentation to understand how they interpret the courses they either teach or are enrolled in.

Just like with courses there is a separate database for users, UserDB. This collection associates usernames (strings) with user objects.

To start with you should implement the user, student and professor classes. You will either need to make two more comparator classes for the different ordering of courses that students and professors require, or you are allowed to use the new Java 8 lambda (anonymous function) syntax to create the comparators inline without needing new classes.

Note that the javadocs for User.UserType makes it seem like you need to do a lot more work when defining this than you actually need to do. You simply define it in your User class as:

```
/** The type of user */
public enum UserType {
    PROFESSOR,
    STUDENT
}
```

Next you can read the data files in with the backend. And finally you can implement the remaining commands in the system, `enroll`, `professor`, `student` and `unenroll`.

Sample Run

Here is the expected output when running the sample data and input files from the supplied code. You are encouraged to make more samples to test your code functions properly.

Submission

Submit to try your source code and Git log:

```
$ try csapx-grd lab5-1 *.java log.txt
```

You can verify your submission with the command:

```
$ try -q csapx-grd lab5-1
```

Grading

The grade breakdown for this lab is as follows:

- Problem Solving: 15%
- In-Lab Activity: 10%
- Functionality: 65%
  - `course` command: 5%
  - `courses` command: 10%
  - `enroll` command: 10%
  - `professor` command: 10%
  - `student` command: 10%
  - `unenroll` command: 10%
  - `users` command: 10%
- Code Style & Version Control: 10%