CSCI 160
Project Specification

Student Registration System (SRS):

Requirement Specification:

We have been asked to develop an automated Student Registration System (SRS). This system will enable students to register online for courses each semester, as well as track a student's progress toward completion of his or her degree.

When a student first enrolls at the university, the student uses the SRS to set forth a plan of study as to which courses he or she plans on taking to satisfy a particular degree program, and chooses a faculty advisor. The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking. Once a plan of study has been established, then, during the registration period preceding each semester, the student is able to view the schedule of classes online, and choose whichever classes he or she wishes to attend, indicating the preferred section (day of week and time of day) if the class is offered by more than one professor. The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his or her transcript online at any time).

Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meets one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).

If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served waiting list. If a class/section that the student was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he or she will be billed for the course. Students may drop a class up to the end of the first week of the semester in which the class is being taught.

PART1: Implementing the classes and Interfaces

Implement the Student Registration System based on the class diagram given below. You should decide based on the requirement specification, the following:

1. Decide on suitable types for instance variables
2. Decide on which are concrete classes, interfaces and abstract classes.
3. Decide which methods need to be overridden in inheritance hierarchies and how
4. Implement the given classes with instance variables with their data structures, behaviors, and relationships with one another—necessary to fulfill these requirements and mission

   Your programs should be properly commented with javadoc comments and regular comments

The class definitions (Core classes):

Course: A semester-long series of lectures, assignments, exams, etc., that all relate to a particular subject area, and which are typically associated with a particular number of credit hours; a unit of study toward a degree. For example, Beginning Objects is a required course for the Master of Science degree in Information Systems Technology.

PlanOfStudy: A list of the courses that a student intends to take to fulfill the course requirements for a particular degree.

Professor: A member of the faculty who teaches sections or advises students.

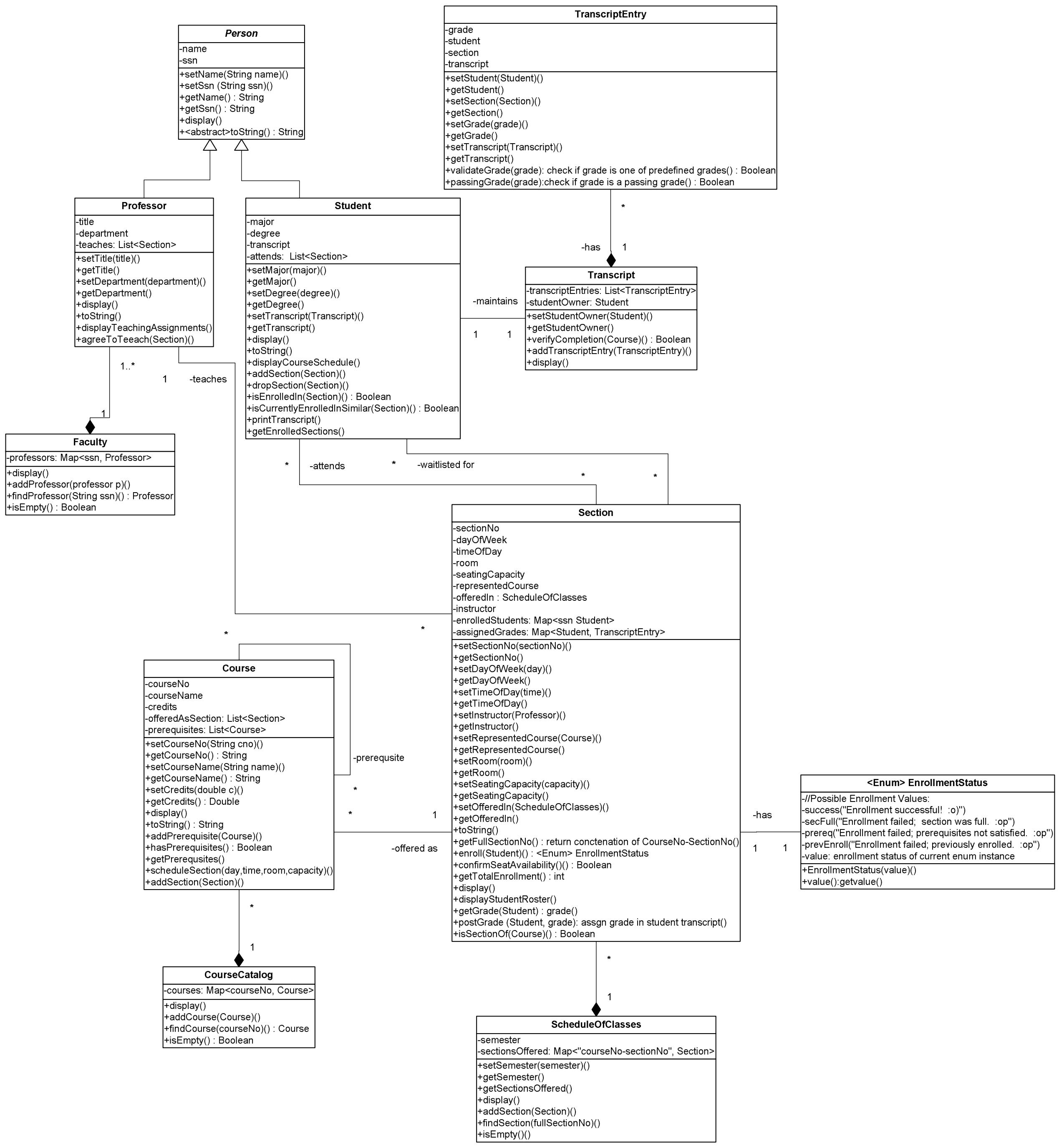
Section: The offering of a particular course during a particular semester on a particular day of the week and at a particular time of day (for example, course Beginning Objects as taught in the Spring 2005 semester on Mondays from 1:00 to 3:00 p.m.).

Student: A person who is currently enrolled at the university and who is eligible to register for one or more sections.

Transcript: A record of all of the courses taken to date by a particular student at this university, including which semester each course was taken in, the grade received, and the credits granted for the course, as well as a reflection of an overall total number of credits earned and the student's grade point average (GPA).

For complete set of classes, their attributes and methods, refer the class diagram.

Class diagram:



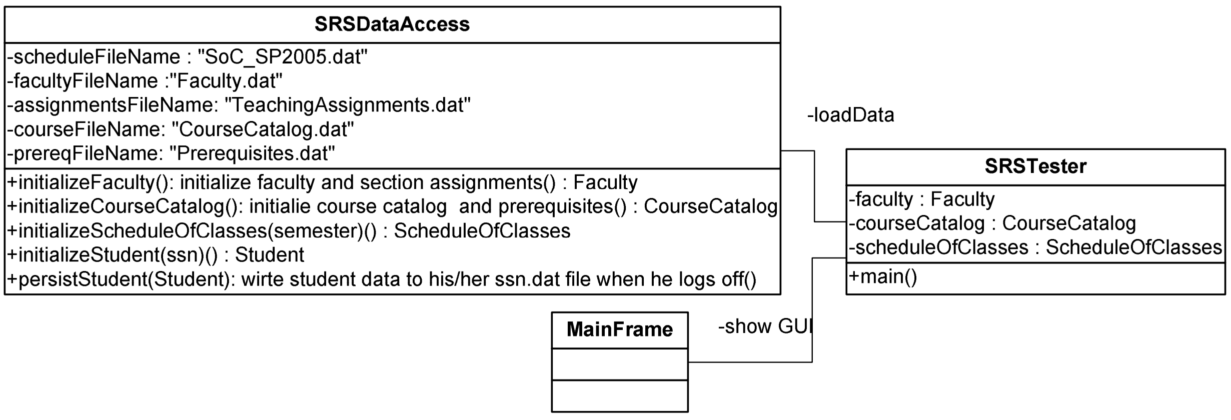
Person
-name
-ssn
+setName(String name)()
+setSsn (String ssn)()
+getName() : String
+getSsn() : String
+display()
+<abstract>toString() : String
TranscriptEntry
-grade
-student
-section
-transcript
+setStudent(Student)()
+getStudent()
+setSection(Section)()
+getSection()
+setGrade(grade)()
+getGrade()
+setTranscript(Transcript)()
+getTranscript()
+validateGrade(grade): check if grade is one of predefined grades() : Boolean
+passingGrade(grade):check if grade is a passing grade() : Boolean
Professor
-title
-department
-teaches: List<Section>
+setTitle(title)()
+getTitle()
+setDepartment(department)()
+getDepartment()
+display()
+toString()
+displayTeachingAssignments()
+agreeToTeeach(Section)()
Student
-major
-degree
-transcript
-attends: List<Section>
+setMajor(major)()
+getMajor()
+setDegree(degree)()
+getDegree()
+setTranscript(Transcript)()
+getTranscript()
+display()
+toString()
+displayCourseSchedule()
+addSection(Section)()
+dropSection(Section)()
+isEnrolledIn(Section)() : Boolean
+isCurrentlyEnrolledInSimilar(Section)() : Boolean
+printTranscript()
+getEnrolledSections()
*
-has
1
Transcript
-transcriptEntries: List<TranscriptEntry>
-studentOwner: Student
+setStudentOwner(Student)()
+getStudentOwner()
+verifyCompletion(Course)() : Boolean
+addTranscriptEntry(TranscriptEntry)()
+display()
-maintains
1
1
1..*
1
-teaches
1
Faculty
-professors: Map<ssn, Professor>
+display()
+addProfessor(professor p)()
+findProfessor(String ssn)() : Professor
+isEmpty() : Boolean
*
-attends
*
-waitlisted for
*
*
Section
-sectionNo
-dayOfWeek
-timeOfDay
-room
-seatingCapacity
-representedCourse
-offeredIn : ScheduleOfClasses
-instructor
-enrolledStudents: Map<ssn Student>
-assignedGrades: Map<Student, TranscriptEntry>
+setSectionNo(sectionNo)()
+getSectionNo()
+setDayOfWeek(day)()
+getDayOfWeek()
+setTimeOfDay(time)()
+getTimeOfDay()
+setInstructor(Professor)()
+getInstructor()
+setRepresentedCourse(Course)()
+getRepresentedCourse()
+setRoom(room)()
+getRoom()
+setSeatingCapacity(capacity)()
+getSeatingCapacity()
+setOfferedIn(ScheduleOfClasses)()
+getOfferedIn()
+toString()
+getFullSectionNo() : return concenation of CourseNo-SectionNo()
+enroll(Student)() : <Enum> EnrollmentStatus
+confirmSeatAvailability()() : Boolean
+getTotalEnrollment() : int
+display()
+displayStudentRoster()
+getGrade(Student) : grade()
+postGrade (Student, grade): assgn grade in student transcript()
+isSectionOf(Course)() : Boolean
*
*
Course
-courseNo
-courseName
-credits
-offeredAsSection: List<Section>
-prerequisites: List<Course>
+setCourseNo(String cno)()
+getCourseNo() : String
+setCourseName(String name)()
+getCourseName() : String
+setCredits(double c)()
+getCredits() : Double
+display()
+toString() : String
+addPrerequisite(Course)()
+hasPrerequisites() : Boolean
+getPrerequsites()
+scheduleSection(day,time,room,capacity)()
+addSection(Section)()
-prerequsite
*
*
1
-offered as
-has
1
1
<Enum> EnrollmentStatus
-//Possible Enrollment Values:
-success("Enrollment successful! :o)")
-secFull("Enrollment failed; section was full. :op")
-prereq("Enrollment failed; prerequisites not satisfied. :op")
-prevEnroll("Enrollment failed; previously enrolled. :op")
-value: enrollment status of current enum instance
+EnrollmentStatus(value)()
+value():getvalue()
*
1
CourseCatalog
-courses: Map<courseNo, Course>
+display()
+addCourse(Course)()
+findCourse(courseNo)() : Course
+isEmpty() : Boolean
*
1
ScheduleOfClasses
-semester
-sectionsOffered: Map<"courseNo-sectionNo", Section>
+setSemester(semester)()
+getSemester()
+getSectionsOffered()
+display()
+addSection(Section)()
+findSection(fullSectionNo)()
+isEmpty()()

PART2: Data Access

Write a class SRSDataAccess.java that takes care of loading some initial data from files into the application. This class methods should be called by main initialize course catalog, faculty list and schedule of classes.

Following files are provided in class website along with this project specification document:
download at: http://www.users.csbsju.edu/~rdissanayaka/courses/f14/csci160/SRSDatFiles.zip

1. **SoC_SP2005.dat**
   Schedule of classes data file for Spring 2005
   Single course description per line
   Format of one line: <courseNo> TAB <sectionNo> TAB <DayOfWeek> TAB <timeOfDay> TAB <room> TAB <roomCaacity>
2. **Faculty.dat**
   List of faculty
   Single professor per line
   Format of line: <name> TAB <ssn> TAB <title> TAB <department>
3. **TeachingAssignments.dat**
   Teaching assignments of professors
   Single section assignment per line
   Format of line: <professor ssn> TAB <fullSectionNo>
4. **CourseCatalog.dat**
   Course lists
   Single course per line
   Format of line: <courseNo> TAB <courseName> TAB <credits>
5. **Prerequisites.dat**
   Prerequisites for courses
   Single prerequisite per course per line
   Format of line: <prerequisite> TAB <course>
6. set of dat files each with the ssn of student ad file name
   111-11-1111.dat
   222-22-2222.dat
   333-33-3333.dat
   the initializeStudent method of SRSDataAccess should use correct dat file based on input parameter ssn to the method to create a student object.


SRSDataAccess
-scheduleFileName : "SoC_SP2005.dat"
-facultyFileName :"Faculty.dat"
-assignmentsFileName: "TeachingAssignments.dat"
-courseFileName: "CourseCatalog.dat"
-prereqFileName: "Prerequisites.dat"
+initializeFaculty(): initialize faculty and section assignments() : Faculty
+initializeCourseCatalog(): initialie course catalog and prerequisites() : CourseCatalog
+initializeScheduleOfClasses(semester)() : ScheduleOfClasses
+initializeStudent(ssn)() : Student
+persistStudent(Student): wirte student data to his/her ssn.dat file when he logs off()
-loadData
SRSTester
-faculty : Faculty
-courseCatalog : CourseCatalog
-scheduleOfClasses : ScheduleOfClasses
+main()
MainFrame
-show GUI

PART3: Event Driven Object Collaboration/ Graphical User Interface (GUI)

your class objects should collaborate at runtime in an appropriate manner. Instantiating these classes to create the appropriate types and number of object instances Setting these objects in motion through external triggering events. To develop SRS GUI use javax.swing package.

SRS application's GUI may involve the collaboration of many different objects. Following are few different use cases that must be handled by the GUI you develop:

- log in
- log off: write updated/new data back to appropriate files.
- Allow student register for a Course
- Allow Student drop a Course
- Determine a Student's Course Load
- Choose a Faculty Advisor for a student
- Establish a Plan of Study for a student
- View the Schedule of Classes
- Allow Professor request a Student Roster for a Given Course
- Allow Professor request a Transcript for a Given Student
- Maintain Course Information (e.g., Allow change the course description, reflect a different instructor for the course, and so on)
- Determine a Student's Eligibility for Graduation
- Post Final Semester Grades for a Given Course

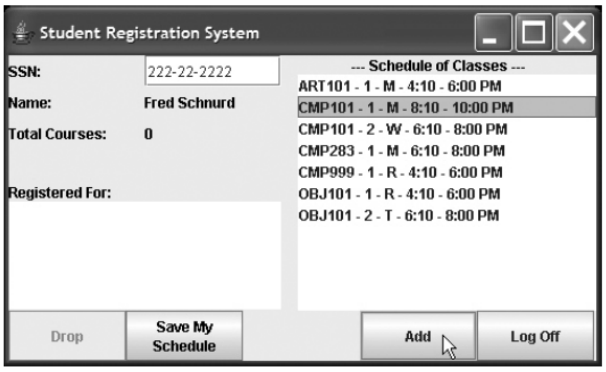
We may decompose any one of the use cases into steps, with each step representing a more detailed use case. For example, "Register for a Course" may be decomposed into these steps:
1. Verify that a student has met the prerequisites.
2. Check student's plan of study to ensure that this course is required.
3. Check for availability of a seat in the course.
4. (Optionally) Place student on a wait list.
and so forth.

Each use case may have several scenarios involved. All these must be correctly handled by GIU and backend application.

You can develop a GUI menu which shows all above as functions as a drop down list. Clicking on menu item "register for course" for example will display a JFrame GUI for handling this task inside menu Frame. This GUI should handle all possible scenarios when registering a student. For main menu GUI, write a class MainFrame (which is also a JFrame) .
Write another class SRSTester containing just the main method. The main method should load data using SRSDataAccess and then initialize MainFrame class to display GUI.

Scenario #1 for the "Register for a Course"Use Case
In this first scenario, a student by the name of Fred successfully registers for a course. The specific
sequence of events is as follows:
1. Fred, a student, logs on to the SRS.
2. He views the schedule of classes for the current semester to determine which section(s) he wishes to register for.
3. Fred requests a seat in a particular section of a course entitled "Beginning Computer Concepts," course number CMP101, section 1.
4. Fred's plan of study is checked to ensure that the requested course is appropriate for his overall degree goals. (We assume that students are not permitted to take courses outside of their plans of study.)
5. His transcript is checked to ensure that he has satisfied all of the prerequisites for the requested course, if there are any.
6. Seating availability in the section is confirmed.
7. The section is added to Fred's current course load.

From Fred's vantage point (sitting in front of a computer screen!), here's what he perceives to be occurring: after logging on to the SRS, he indicates that he wishes to register for CMP101, section 1 by choosing it from the available course list, and then clicks the Add button.


Student Registration System

SSN: 222-22-2222

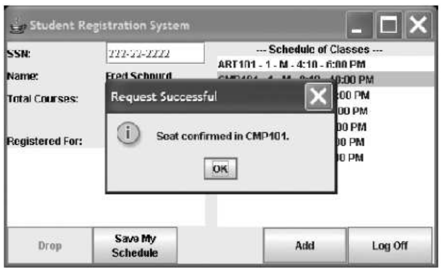
Name: Fred Schnurd

Total Courses: 0

Registered For:

--- Schedule of Classes ---

ART101 - 1 - M - 4:10 - 6:00 PM
CMP101 - 1 - M - 8:10 - 10:00 PM
CMP101 - 2 - W - 6:10 - 8:00 PM
CMP283 - 1 - M - 6:10 - 8:00 PM
CMP999 - 1 - R - 4:10 - 6:00 PM
OBJ101 - 1 - R - 4:10 - 6:00 PM
OBJ101 - 2 - T - 6:10 - 8:00 PM


Request Successful

Seat confirmed in CMP101.

Scenario #2 for the "Register for a Course"Use Case
In this scenario, Fred once again attempts to register for a course. While he meets all of the requirements, the requested section is unfortunately full. The SRS offers Fred the option of putting his name on a wait list. The specific sequence of events is as follows:
1. Fred, a student, logs on to the SRS.
2. Fred views the schedule of classes for the current semester to determine which section(s) he wishes to register for.
3. Fred requests a seat in a particular section of a course entitled "Beginning Computer Concepts," course number CMP101, section 1.
4. Fred's plan of study is checked to ensure that the requested course is appropriate for his overall degree goals.
5. His transcript is checked to ensure that he has satisfied all of the prerequisites for the requested course, if any.
6. ***Seating availability in the section is checked, but the section is found to be full***.
7. ***Fred is asked if he wishes to be put on a first come, first served wait list***.
8. ***Fred elects to be placed on the wait list***.

With a little imagination, you can undoubtedly think of numerous other scenarios for this use case, involving such circumstances as Fred having requested a course that isn't called for by his plan of study, or a course for which he hasn't met the prerequisites

Some good references/tutorials for swing are as follows:
Oracle Swing Tutorial: http://docs.oracle.com/javase/tutorial/uiswing/
Beginner swing tutorial: http://www.javabeginner.com/java-swing
JFrame Example: http://www.javabeginner.com/java-swing/java-jframe-class-example
JLabel Example: http://www.javabeginner.com/java-swing/java-jlabel-class-example
JTextField Example: http://www.javabeginner.com/java-swing/java-jtextfield-class-example
JButton Example: http://www.javabeginner.com/java-swing/java-jbutton-class-example
JList Example: http://www.javabeginner.com/java-swing/java-jlist-class-example
JInternalFrame Example: http://www.javabeginner.com/java-swing/java-jinternalframe-class-example
JMenu Example: http://www.javabeginner.com/java-swing/java-jmenu-class-example
AddressBook GUI: http://www.javabeginner.com/java-swing/java-swing-address-book

PART4: Exception Handling

Provide exception handling wherever appropriate.

E.g.

- reading/writing files
- getting user input

For **extra credit**, implement appropriate custom exceptions and throw them wherever suitable.

What to submit

Submit a folder <yourfirstNameLastName>Project containing following:

- subfolder *javadoc* with javadoc files for all classes
- subfolder *src* with source code java files
- a executable jar file *srs.jar* which contain all class files and dat files.

  To learn how to create an executable jar, refer following:

  http://docs.oracle.com/javase/tutorial/deployment/jar/appman.html

  I should be able to run your application by typing following command:

  ```
  java -jar srs.jar
  ```

  You will also need to demonstrate to the class your running application (~15 minutes). The presentation date will be announced in the class.

When to submit

listed in class website.

How to submit

Copy your folder to /usr/people/handins/CS160/Project.