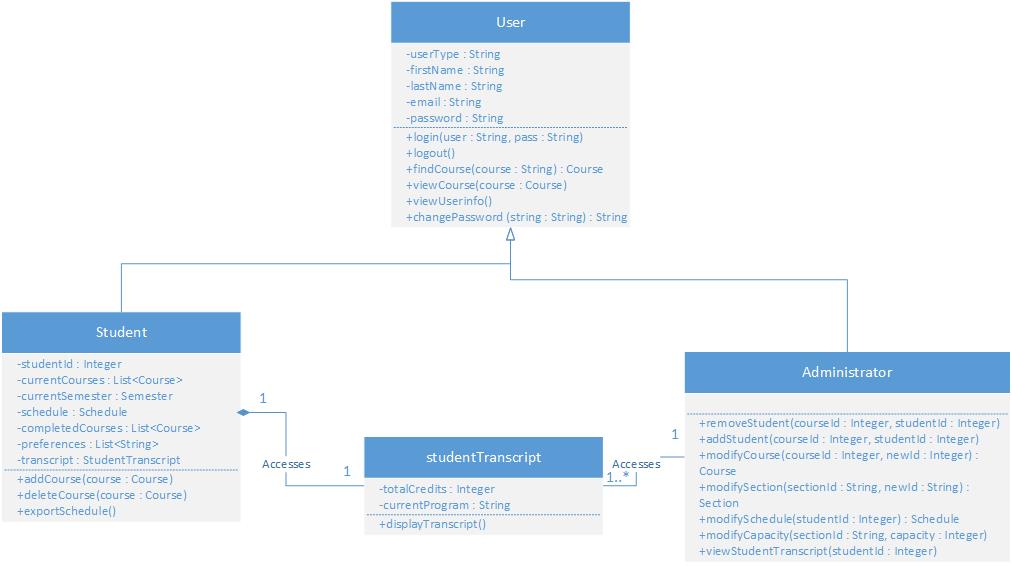
**Detailed Design**

1. User Subsystem
   1. User Subsystem Diagram

The user subsystem’s role is to store any relevant information regarding a user. While all users share certain base characteristics, such as “First Names” and “Passwords”, they will always either be an Administrator or a Student. This system allows for the creation of a Student account, after which that student may login to their profile using their credentials. Regardless of the type of account, there will always be a link to the “StudentTranscript” class. This means that the user subsystem, while secure, is not isolated from the rest of the system, and may communicate any necessary information. This transcript information is vital to maintain a proper profile, since it is the most important information regarding academic progress. The Student class contains other classes as its attributes. To see these, refer to sections 2 and 3 of the detailed design.

* 1. User subsystem unit descriptions

**Class User**

Description: Abstract class which is a template for any user in the system (Student or Administrator). A student or administrator contain the “User” attributes and methods on top of their own.

Attributes:

* userType: String
* firstName: String
* lastName: String
* email: String
* password: String

Methods:

* login(user: String, pass: String): logs the user into the system. The user parameter represents the account’s name, and the pass parameter is the password. This method does not have a return value.
* logout(): Logs the user off of the system. Does not require any parameters, and does not return any value.
* findCourse(course: String): Course: This method searches for a course in the database using the course name as the input string. The return value of this method is a Course object whose courseName attribute matches with the input value for the method.
* viewCourse(course: Course): This method takes in a course object as input value, and does not return anything. However, this method displays the course in a way that the user may understand.
* viewUserinfo(): This method does not return any value, however it displays all the information of the account using the method.
* changePassword(string: String): String: This method takes in a new password as a parameter in the form of a string, and returns it if the method is successful.

**Class Student**

Description: This class is a specialized version of User. The Student class contains the same information as the User along with its own special attributes and methods. This user is granted no special privileges.

Attributes:

* studentID: Integer
* currentCourses: List<Course>
* currentSemester: Semester
* schedule: Schedule
* completedCourses: List<Course>
* preferences: List<String>
* transcript: StudentTranscript

Methods:

* addCourse(course: Course) : This method takes in a Course object as input, and adds it to the Student’s currentCourses attribute, which is a list of Course objects. This method returns no value.
* deleteCourse(course: Course): This method takes in a Course object as input, and removes it from the Student’s currentCourses attribute, which is a list of Course objects. This method returns no value.
* exportSchedule(): This method exports the Schedule object of the account using the method so that the actual user may view it on their own as a file.

**Class Administrator**

Description: This class is a specialized version of User. While it contains no additional attributes from the User class, it does have additional methods and elevated privileges within the system.

Attributes:

* N/A

Methods:

* removeStudent(courseId: Integer, studentId: Integer): This method takes in a course ID and a student ID as parameters, and does not return any value. The student with that given student ID is removed from the course in question.
* addStudent(courseId: Integer, studentId: Integer): This method takes in a course ID and a student ID as parameters, and does not return any value. The student with that given student ID is added to the course in question.
* modifyCourse(courseId: Integer, newId: Integer): Course: This method takes in two course IDs as parameters and returns a Course object. The course with the associated courseId has its ID changed to newId. The returned Course object has the newId as an attribute.
* modifySchedule(studentId: Integer): Schedule: This method takes a student ID as a parameter and returns a Schedule object. This method allows the Administrator to modify the schedule of the given student, and once changes are made, returns that new Schedule.
* modifyCapacity(sectionId: String, capacity: Integer): This method takes in a string parameter called sectionId and an Integer called capacity. It does not return any value. This method modifies the allowable number of students in a given section with the associated section ID to the new capacity parameter.
* viewStudentTranscript(studentId: Integer): This method takes in a student ID as a parameter and returns no value. This method displays the schedule of the student in question for the administrator to see.

**Class StudentTranscript**

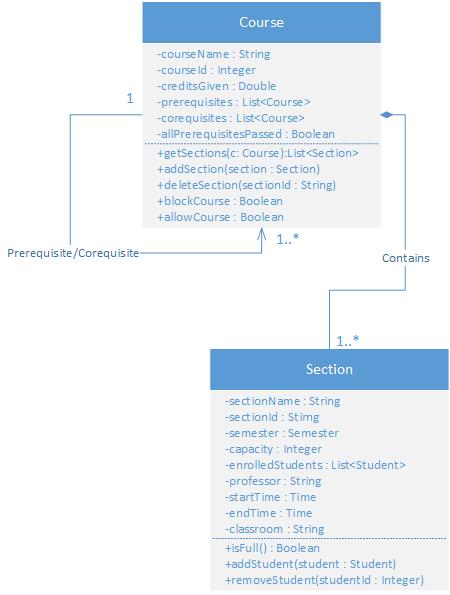
Description: This class represents the academic progress for a Student. It holds relevant information regarding program and credits.

Attributes:

* totalCredits: Integer
* currentProgram: String

Methods:

* displayTranscript(): This method takes no parameters and does not return any value. This method displays the student’s transcript to whoever is accessing it. This method is not only available to the student in question, but also to administrators.

1. Course subsystem
   1. Course subsystem diagram

The course subsystem contains all information regarding course offered in ENCS at Concordia. It is composed of all Course object instances. Every Course contains 1 to n Section objects. Furthermore, it is possible for a course to also have prerequisites or co requisites. A prerequisite is a course that a Student must complete before signing up for the course in question, and a co requisite is a course that must be taken simultaneously as the one in question.

* 1. Course Subsystem unit descriptions

**Class Course**

Description: The Course class represents a course offered at the university. Its most important attributes are the collection of Sections that compose it.

Attributes:

* courseName: String
* courseId: Integer
* creditsGiven: Double
* prerequisites: List<Course>
* corequisites: List<Course>
* allPrerequisitesPassed: Boolean

Methods:

* getSections(C: Course): List<Section>: This method takes in a Course as parameter and returns all the associated Section objects as a list. The sections returned are only the ones within the course in question.
* addSection(section: Section): This method takes in a Section object as parameter and returns nothing. This method adds a Section to the List of Sections within a course. The Course that gets the extra Section is the one using the method.
* deleteSection(sectionId: String): This method takes in a String representing a SectionId as a parameter and returns nothing. This method deletes the section with the associated section ID from the course that it is contained in.
* blockCourse(): Boolean: This method takes in no parameters and returns a true/false value. True means that the Student is blocked from taking the course, false means that the Student is allowed to do so.
* allowCourse(): Boolean: This method takes in no parameters and returns a true/false value. True means the Student is allowed to take the course in question, and false means otherwise.

**Class Section**

Description: This class represents a specific course section at Concordia. Section objects are important because a student signs up to a section instead of a course itself.

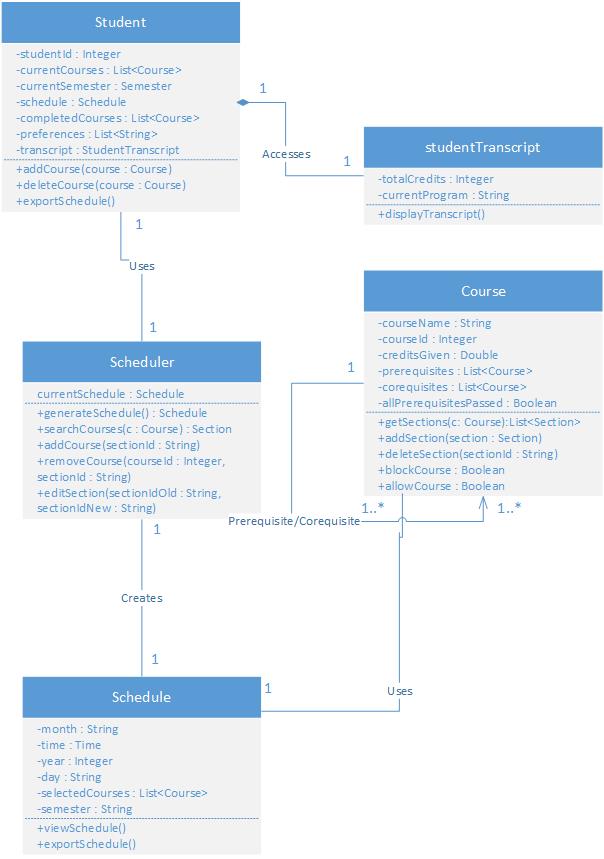
Attributes:

* sectionName: String
* sectionId: String
* semester: Semester
* capacity: Integer
* enrolledStudents: List<Student>
* professor: String
* startTime: Time
* endTime: Time
* classroom: String

Methods:

* isFull(): Boolean: This method takes in no parameters and returns a true/false value. It checks to see if the length of the List of Students enrolled in a section (enrolledStudents) is equal to the capacity of the section. A return value of true means that there is no more room in a section, and false means that there is still room.
* addStudent(student: Student): This method take sin a student as a parameter and returns no value. The Student object inputted into the method is added to the section using the method if and only if the isFull() method returns false.
* removeStudent(studentId: Integer): This method takes in an integer as a parameter representing a student ID and returns nothing. It checks the enrolledStudents list and removes the student with the associated student ID.  
    
  3.Scheduler Subsystem

1. Scheduler Subsystem diagram



The Scheduler subsystem encompasses aspects of the other subsystems and packages them in order to add real functionality. In essence, its goal is to create a Schedule, which makes it the most important subsystem of all, because the end goal of “The Force” is to provide a scheduling service to all of its users. The Scheduler class provides the service to the Student, whereas the Schedule class itself accesses Course information.

1. Scheduler Subsystem unit descriptions

**Class Student**

See section 1-b of detailed design

**Class studentTranscript**

See section 1-b of detailed design

**Class Course**

See section 2-b of detailed design

**Class Scheduler**

Description: The Scheduler class effectively serves as a kind of interface for a student to interact with which provides the student with their desired schedule. In order to achieve this, the scheduler has access to both the Student class and the Schedule class.

Attributes:

* currentSchedule: Schedule

Methods:

* generateSchedule(): Schedule: This method takes in no parameters and returns a Schedule. This is the method that provides the Student with the end product.
* searchCourses(c: Course): Section: This method takes in a Course as a parameter and returns a Section. This method searches the database for a Course (input parameter) and returns the best fit of a section to the Student’s preferences and schedule.
* addCourse(sectionId:String): This method takes a section ID String as a parameter and returns nothing. It adds a section to the student’s schedule.
* removeCourse(courseId: Integer, sectionId: String): This method takes a courseId Integer and a sectionId String as parameters. It removes the section with the associated ID from the Course with the associated course ID from the Student’s schedule.
* editSection(sectionIdOld: String, sectionIdNew: String): This method takes in two section ID strings as parameters and returns nothing. This method swaps sections for a student in their schedule. The section with the associated sectionIdOld is replaced with the section with the associated sectionIdNew.

**Class Schedule**

Description: This class is the end product provided to the user. It is effectively the goal of the entire system.

Attributes:

* month: String
* time: Time
* year: Integer
* day: String
* selectedCourses: List<Course>
* semester: String

Methods:

* viewSchedule(): This method has no parameters and no return values. It displays the schedule using the method to the user.
* exportSchedule(): This method has no parameters and no return values. It exports the Schedule as a file that the user can then use in whatever way they see fit.