**3.1 Subsystems**

In the system, there are many different classes that serve different purposes. However, the different classes are not independent of one another. While they serve different purposes, they work together in groups to achieve bigger actions than basic methods. Of course, the program does not only have one feature, so we can separate the system into groups that serve the same function. There are five groups, that can be called subsystems, which focus on a certain aspect;

* IUserInteraction: This subsystem is responsible to every interaction made between the User and the Program, such as logging in, browsing the course list, setting preferences, adding courses, logging out, etc. The classes in this subsystem are the Student, Administrater and User classes.
* IScheduleManagement: The Schedule Management subsystem, consisting of the Schedule class, is responsible for all schedule operations. In other words, whenever another one of the subsystems wants to modify or interact with a schedule or the list of schedules, they interact with this subsystem which in turn does the operations.
* ICourseManagement: The Course, CoursesPassed, Section and Classroom classes make up this subsystem. Because the Courses are much more complex than a name (they contain sections, times, prerequisites, etc.), we cannot have a single class take care of everything. Instead, we have the Course Management subsystem which uses the above classes to modify and interact with the Courses.
* IPreferenceManagement: Though this subsystem only contains one class: the Preference class. It is responsible for taking the students preferences and giving it to another subsystem to help create a schedule according to the student’s needs.
* ISequenceManagement: This subsystem also contains one class, the Sequence class. It is responsible for returning the Sequence of a student to a user who wishes to view it, or to update the sequence by removing courses (when a student completes a course).

**3.1.1 IUserInteraction**

**Student Class**

|  |
| --- |
| **Description** |
| Students are users that interact with the program directly. They log in, make changes in the schedule, then log out. |
| **Attributes** |
| studentID: long  the studentID is a unique identifier for every student  cgpa: double  the cgpa is the GPA of each student |
| **Methods** |
| * getID() studentID * returns the student’s ID * getGPA() cgpa * returns the student’s GPA * getAllSavedSchedules(studentID:long) list of Schedule objects * returns a list of all the previous and upcoming schedules of a student * getSavedSchedule(scheduleID:long) Schedule * returns the saved schedule of a student * addCourse(courseID:long, sectionID:long) Schedule * adds a course with section sectionID to the student’s sequence * removeCourse(sectionID:long, studentID:long) Schedule * removes a course from the student’s sequence * deleteSchedule(scheduleID: long) Schedule * deletes the entire schedule * browseCourseList() list of Course objects * returns a list of all the available courses * getPreferenceList() list of Preference objects * returns the student’s preferences * getSection(sectionID:long) Section * returns the section * login(studentID:long, password:String) LogonPage * gives the student access to the system * logout(studentID:long) LogoutPage * disconnects student from the system * saveSchedule(scheduleID:long) * saves changes to the schedule * searchCourse(courseID:long) list of Section objects * takes a course ID as input and returns the matched results * selectWeek(day:int, month:int) Schedule * returns a schedule of the selecteddate * viewAcademicRecord(studentID:long) AcademicRecord * returns the student’s academic record * viewCourseSequence(studentID:long) Sequence * returns the student’s sequence |

**Administrator Class**

|  |
| --- |
| **Description** |
| Administrators are users that interact with the program directly. They log in, make changes in the schedule, then log out. |
| **Attributes** |
| adminID: long  the adminID is a unique identifier for each administrator |
| **Methods** |
| * login(adminID:long, password:String) LogonPage * logout(adminID:long) LogoutPage * addUser(username:String, firstname:String, lastname:String, netname: String, password:String, priviledge:int) User * creates and adds user to userlist * deleteUser(userID:long) list<User> * removes user with ID userID and returns the list of users * browseCourses() list<Course> * returns list of courses * createCourse(courseCode:int, description:String, credits:Double, type:String) Course * creates a course * deleteCourse(courseID:long) list<Course> * deletes course with ID courseID * updateCourse(courseCode:int, description:String, credits:Double, type:String) Course * changes a course information * updateUser(username:String, firstname:String, lastname:String, netname: String, password:String, priviledge:int) User * changes a user’s information * userSearch(userID:String, username:String, firstname:String, lastname:String, netname: String) list<User> * searches for users with according information and returns a list * viewCourse(courseID:long) Course * returns course with ID courseID * viewUser(userID:long) User * returns user with ID userID |

**User Class**

|  |
| --- |
| **Description** |
| User is the class that handles basic user info and verifies if a user is legitimate or not, and allows it to logout. |
| **Attributes** |
| verified: Boolean  this attribute is the state of the user which determines whether his login is successful or not |
| **Methods** |
| * login(studentID:long, password:String) LogonPage * gives the student access to the system * logout() LogoutPage * disconnects student from the system * changeUsername(username:String) * changes username * changePassword(password:String) * changes password * disconnect() void * disconnects the student from the system * requestLogout() void * makes a request to disconnect * verify(adminID: long, password: String) verified * verifies if the login info is correct |

**3.1.2 IScheduleManagement**

**Schedule Class**

|  |
| --- |
| **Description** |
| This class manages courses in a schedule, and ensures that courses don’t overlap |
| **Attributes** |
| scheduleID: long  every schedule has a unique long identifier  freeCreds: int  freeCreds is the amount of remaining free credits that the student can take in a semester  freeBlock: boolean  freeBlock is a Boolean which determines whether the time for a class is free or not |
| **Methods** |
| * removeStudent(studentID) * removes the student * updateSequence(courseID) * updates the sequence after changes are made * addCourse(courseID:long, sectionID:long) Schedule * adds a course to the schedule * bookTimeSlot(startTime:int, endTime:int) * books a timeslot for a course to prevent overlap * freeTimeBlock(startTime:int, endTime:int) * frees a timeslot that was booked * deleteSchedule(scheduleID:long) * deletes a schedule * getFreeCredits(studentID:long) int * returns the amount of credits still available * getWeeklyView(startDay:int, startMonth:int) Schedule * returns a weekly view of the schedule * isBlockFree(startTime:int, endTime:int) Boolean * computes if the timeslot is available and returns true or false * removeCourse(sectionID:long, studentID:long) Schedule * removes a course from the schedule * getAllSavedSchedules(studentID:long) list of Schedule objects * returns all schedules available in the system * getSchedule(scheduleID:long) Schedule * returns a single schedule that has the id scheduleID * saveSchedule(scheduleID:long) list of Schedule objects * saves changes made to schedules |

**3.1.3 ICourseManagement**

**Course Class**

|  |
| --- |
| **Description** |
| The Course class contains all the information on the classes available. |
| **Attributes** |
| courseID: String  every course has a unique String identifier  courseName: String  every course has a name  courseDescription: String  courses have a String description  credits: int  this attribute is an integer value of the credits the course is worth |
| **Methods** |
| * getSection() section * returns a section of a class * getCourseList() list of Course objects * returns a list of available courses * getAllSections(courseID:long) list of Section objects * returns a list of all available sections * createSequence() Sequence * initializes a sequence * generateSchedule() Schedule * generates a schedule based on preferences * getCourseList() list<Course> * gets a list of courses in a schedule * getCredits(courseID:long) int * returns the credits of a course * selectCourse(courseID:long) list of Section objects and Prerequisite objects * selects a course that has the ID courseID |

**CoursesPassed Class**

|  |
| --- |
| **Description** |
| This class is responsible for confirming and returning the passed courses. |
| **Attributes** |
| isPassed: Boolean  this attribute is a Boolean value that determines if a course is already passed or not |
| **Methods** |
| * checkPassed(prereqs:Course, studentID:long) boolean * checks if a course is passed by student with ID studentID * gatherCoursesPassed(studentID:long) list of Course objects * returns a list of passed courses for student with ID studentID |

**Section Class**

|  |
| --- |
| **Description** |
| A course can have multiple sections; this class contains the different sections and is responsible for adding and removing students from the sections. |
| **Attributes** |
| semester: String  the semester String is a value which determines which semester the course is in |
| **Methods** |
| * addToList(studentID:long) * adds a student to a section * getSection(sectionID:long) Section * returns a section with the ID sectionID * getSections() list of Section objects * returns a list of available sections * getTimes(sectionID:long) list of int * returns the time of section with ID sectionID * removeStudent(studentID:long) * removes student with ID studentID from a section * getSemester(sectionID:long) semester * returns the semester of a section |

**Classroom Class**

|  |
| --- |
| **Description** |
| Each section has a room number and availability |
| **Attributes** |
| * roomNumber: int * availability: boolean |
| **Methods** |
| * setRoomNumber(roomNumber:int) * sets a room number to a classroom * getRoomNumber() roomNumber * returns room number of a classroom * getAvailability(time:double, day:int, month:int) availability * returns the time/date availability of a course |

**3.1.4 IPreferenceManagement**

**Preference Class**

|  |
| --- |
| **Description** |
| This class manages the user’s preference and generates a sequence, and is capable of returning all the preferences in the system. |
| **Attributes** |
| numOfCourses: int  numOfCourses is an integer attribute that determines how many courses a student wishes to take |
| **Methods** |
| * choosePreference(pref:Preference) Sequence * applies the preferences set by the user and returns a sequence * getPreferenceList() list of Preference objects * gets the preferences set by the user |

**3.1.5 ISequenceManagement**

**Sequence Class**

|  |
| --- |
| **Description** |
| This class allows other classes to get the Sequence of a student, and the class also is capable of updating the sequence whenever a course is completed. |
| **Attributes** |
| None |
| **Methods** |
| * getSequence() Sequence * returns a sequence of a student * updateSequence(courseID:long) * whenever a course is passed, this method is used to remove the course from the sequence |