GROUP 10: MILESTONE 4

CampusCentral

Group Members:

Name: Karanvir Basson
Student ID: 1129746
Email: kbasson@uoguelph.ca

Name: Andrew Chow
Student ID: 1088114
Email: achow04@uoguelph.ca

Name: Fee Kim Ah-Poa
Student ID: 1159794
Email: fahpoa@uoguelph.ca

Name: Mazen Bahgat
Student ID: 1157821
Email: mbahgat@uoguelph.ca

Name: Muhammad Talha Sadaqat
Student ID: 1145327
Email: msadaqat@uoguelph.ca

Name: Hadi Rana
Student ID: 1107555
Email: hrana04@uoguelph.ca

Name: Joshua Jones
Student ID: 1125364
Email: jjones21@uoguelph.ca

Name: Cavaari Taylor
Student ID: 1159034
Email: cavaari@uoguelph.ca

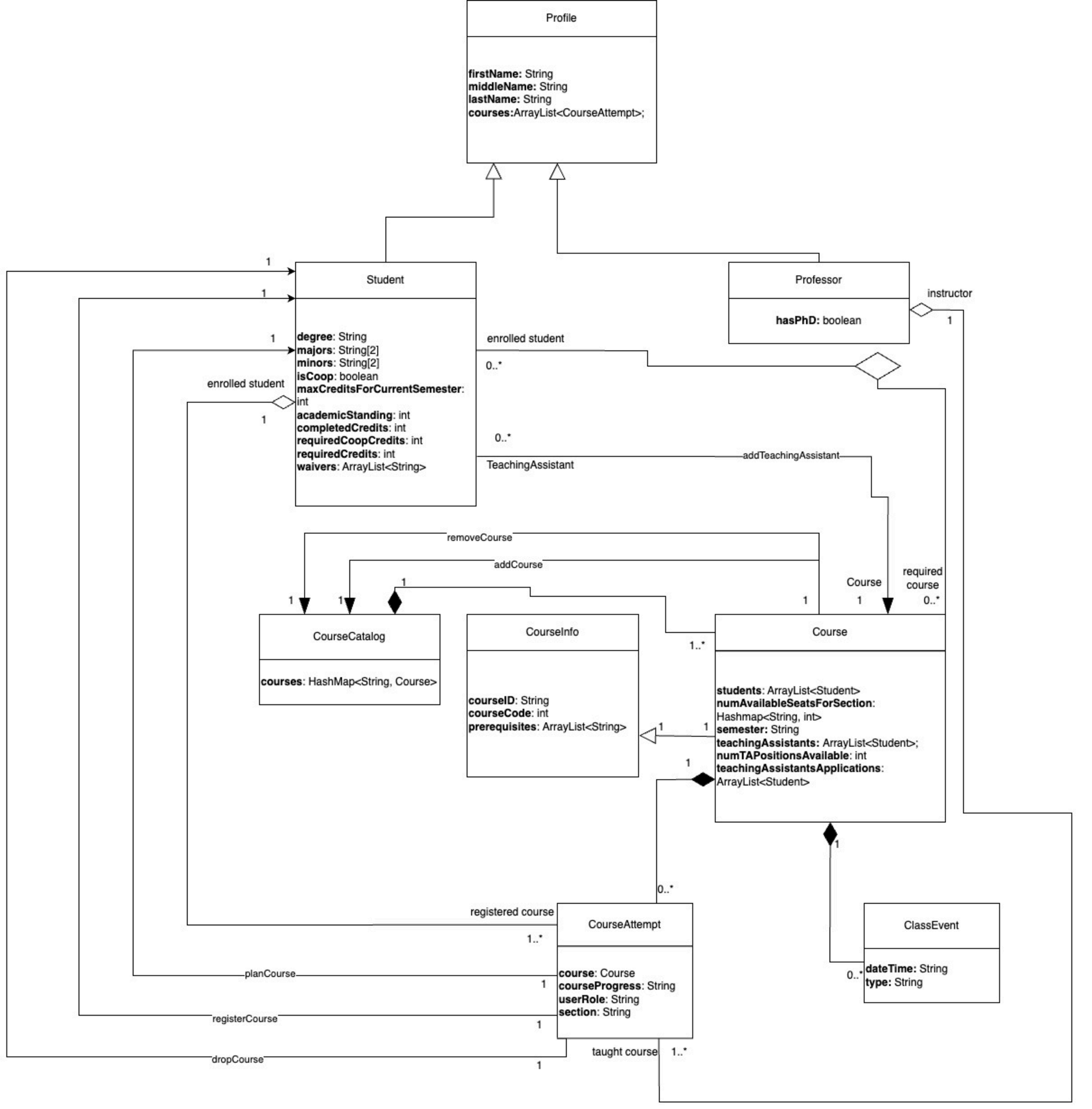
Table of Contents

3.

GROUP 10: MILESTONE 4

Class Diagram

CampusCentral


Profile
firstName: String
middleName: String
lastName: String
courses:ArrayList<CourseAttempt>;
Student
degree: String
majors: String[2]
minors: String[2]
isCoop: boolean
maxCreditsForCurrentSemester: int
academicStanding: int
completedCredits: int
requiredCoopCredits: int
requiredCredits: int
waivers: ArrayList<String>
Professor
hasPhD: boolean
instructor
1
enrolled student
0..*
enrolled student
1
1
TeachingAssistant
addTeachingAssistant
removeCourse
addCourse
required course
Course
1
0..*
CourseCatalog
courses: HashMap<String, Course>
CourseInfo
courseID: String
courseCode: int
prerequisites: ArrayList<String>
Course
students: ArrayList<Student>
numAvailableSeatsForSection: Hashmap<String, int>
semester: String
teachingAssistants: ArrayList<Student>;
numTAPositionsAvailable: int
teachingAssistantsApplications: ArrayList<Student>
registered course
CourseAttempt
course: Course
courseProgress: String
userRole: String
section: String
ClassEvent
dateTime: String
type: String
planCourse
registerCourse
dropCourse
taught course

5.

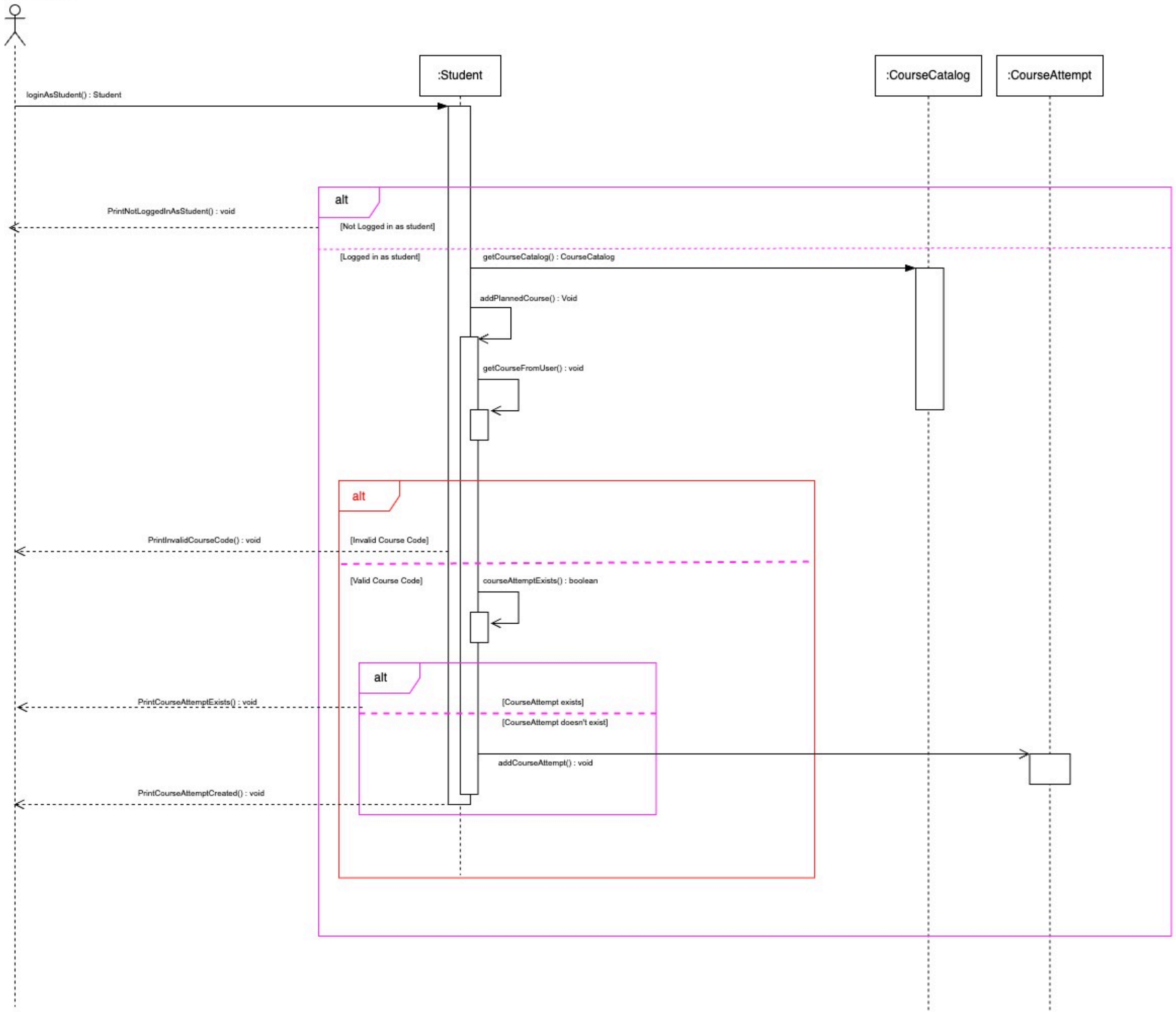
GROUP 10: MILESTONE 4

Sequence Diagrams

CampusCentral

sd: UC: Add a planned course

Main Menu



sd: UC: Remove a planned course

Main Menu

:Student

:CourseCatalog

:CourseAttempt

loginAsStudent() : Student

alt

PrintNotLoggedInAsStudent() : void

[Not Logged in as student]

[Logged in as student]

getCourseCatalog() : CourseCatalog

removePlannedCourse() : Void

getCourseFromUser() : void

alt

PrintInvalidCourseCode() : void

[Invalid Course Code]

[Valid Course Code]

courseAttemptExists() : boolean

alt

PrintCourseAttemptExists() : void

[CourseAttempt exists]

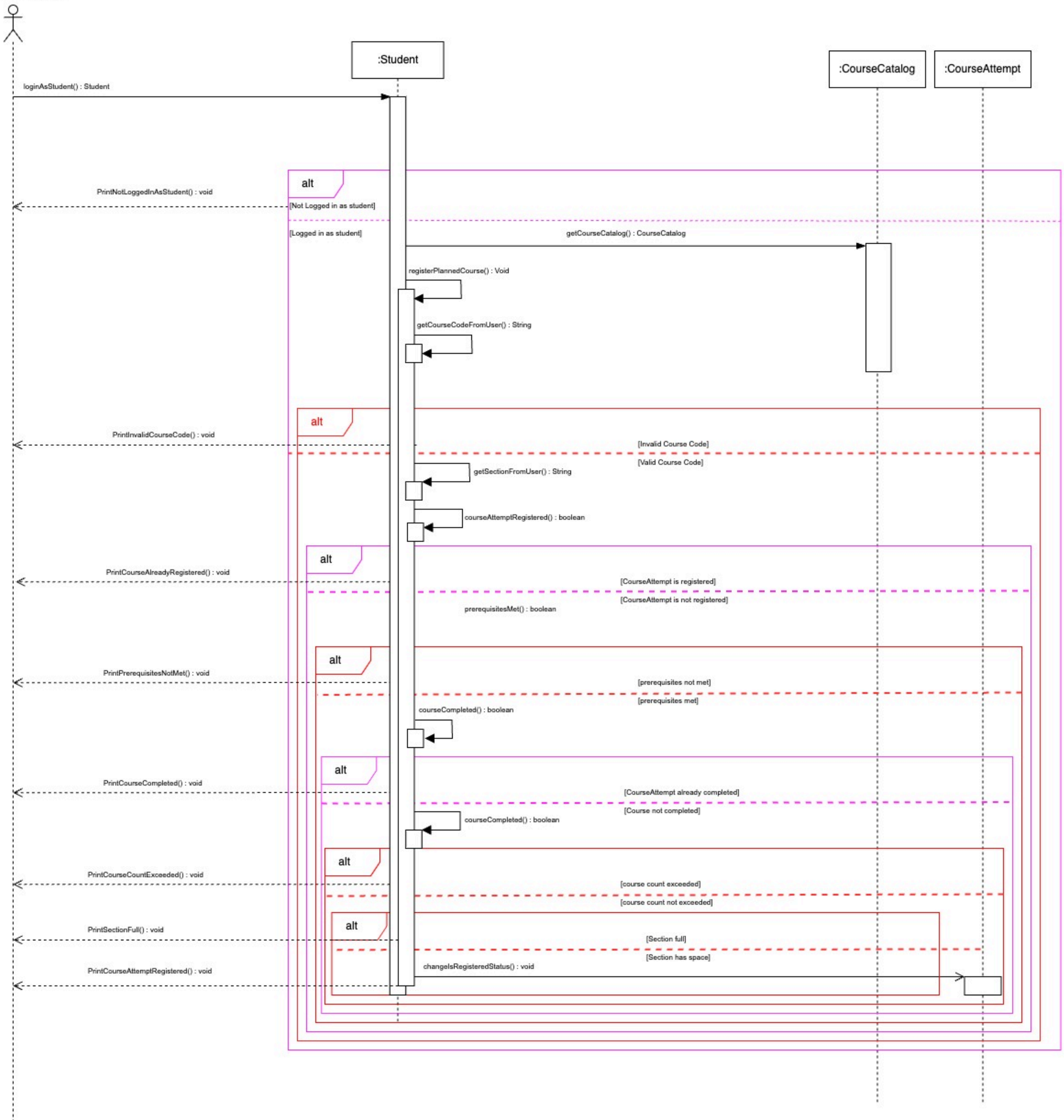
[CourseAttempt doesn't exist]

removeCourseAttempt() : void

PrintCourseAttemptRemoved () : void

sd: UC: Register for a planned course

Main Menu



sd: UC: Drop a registered course

Main Menu

:Student

:CourseCatalog

:CourseAttempt

loginAsStudent() : Student

alt

PrintNotLoggedInAsStudent() : void

[Not Logged in as student]

[Logged in as student]

getCourseCatalog() : CourseCatalog

dropPlannedCourse() : Void

getCourseCodeFromUser() : void

alt

PrintInvalidCourseCode() : void

[Invalid Course Code]

[Valid Course Code]

courseAttemptRegistered() : boolean

alt

PrintCourseAttemptNotRegistered() : void

[CourseAttempt Not Registered]

[CourseAttempt Registered]

updateCourseProgress() : void

PrintCourseAttemptDropped() : void

sd: UC: View future courses

Main Menu

:Student

loginAsStudent() : Student

alt

PrintNotLoggedInAsStudent() : void

[Not Logged in as student]

[Logged in as student]

getFutureCourseAttempts(): ArrayList<CourseAttempt>

alt

[Has no future planned courses]

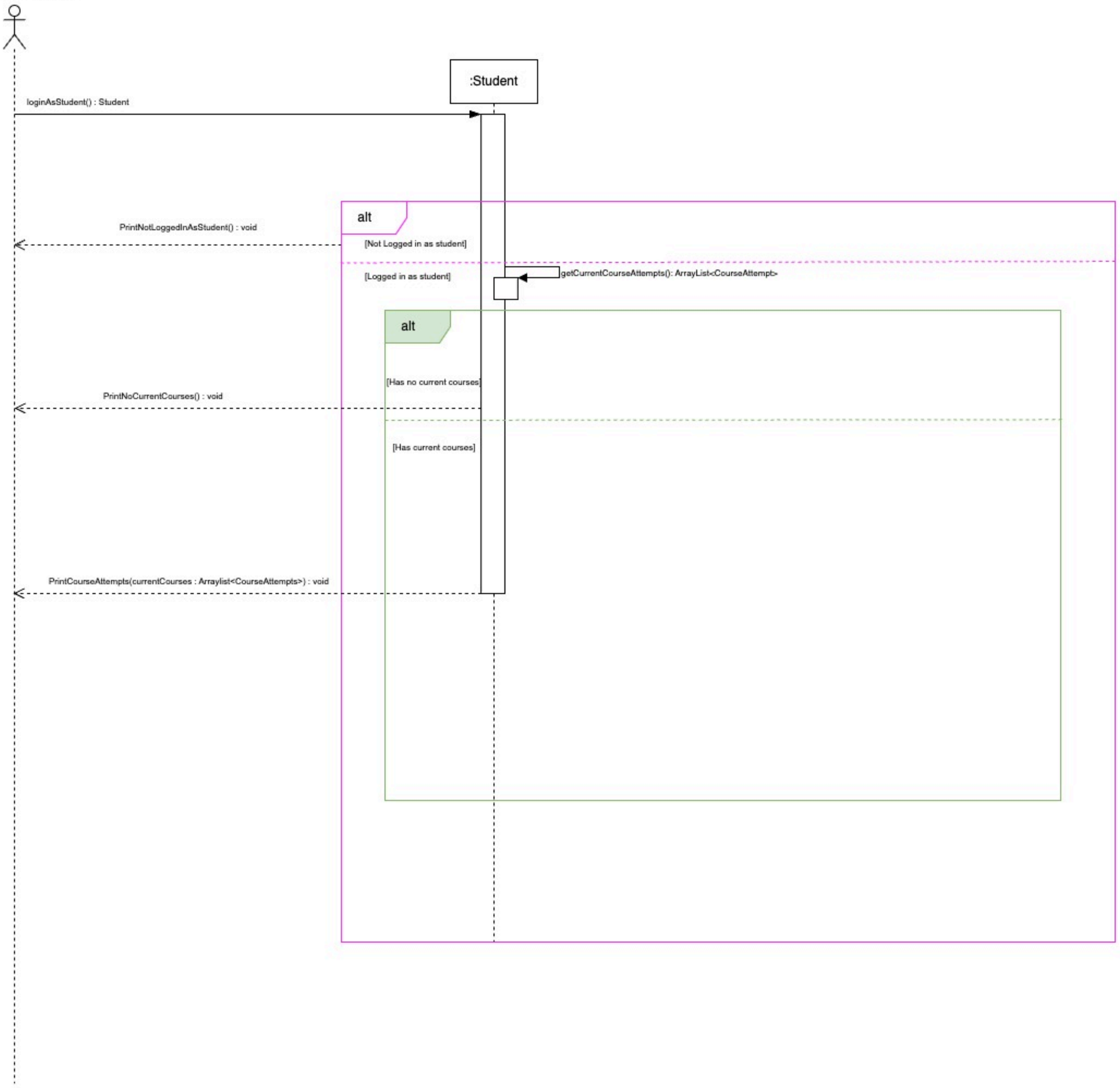
PrintNoFuturePlannedCourses() : void

[Has future planned courses]

PrintCourseAttempts(futureCourses : Arraylist<CourseAttempts>) : void

sd: UC: View current courses

Main Menu



sd: UC: View future courses

Main Menu

:Student

loginAsStudent() : Student

alt

PrintNotLoggedInAsStudent() : void

[Not Logged in as student]

[Logged in as student]

getPastCourseAttempts(): ArrayList<CourseAttempt>

alt

[Has no past courses]

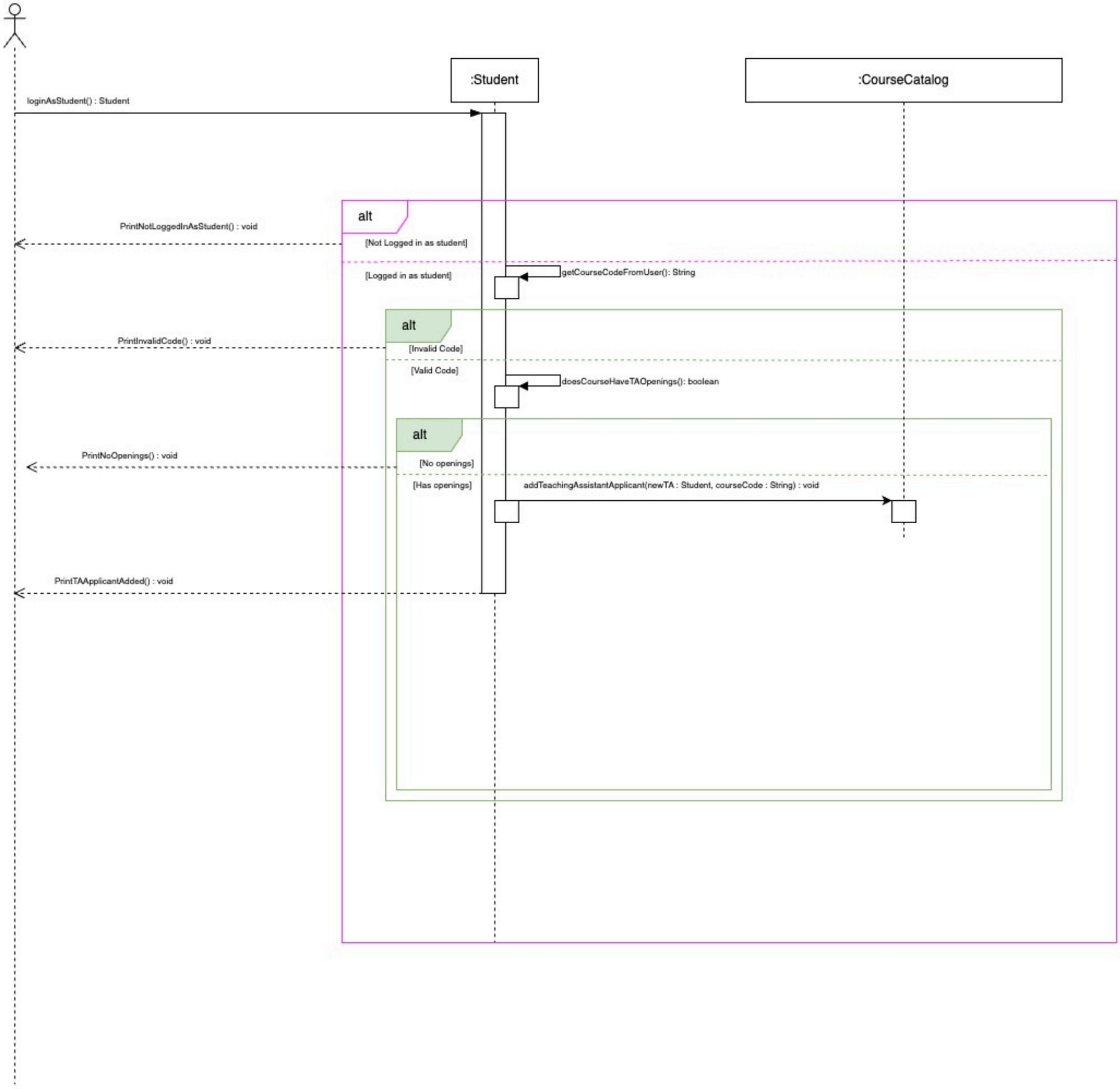
PrintNoPastCourses() : void

[Has past courses]

PrintCourseAttempts(pastCourses : Arraylist<CourseAttempts>) : void

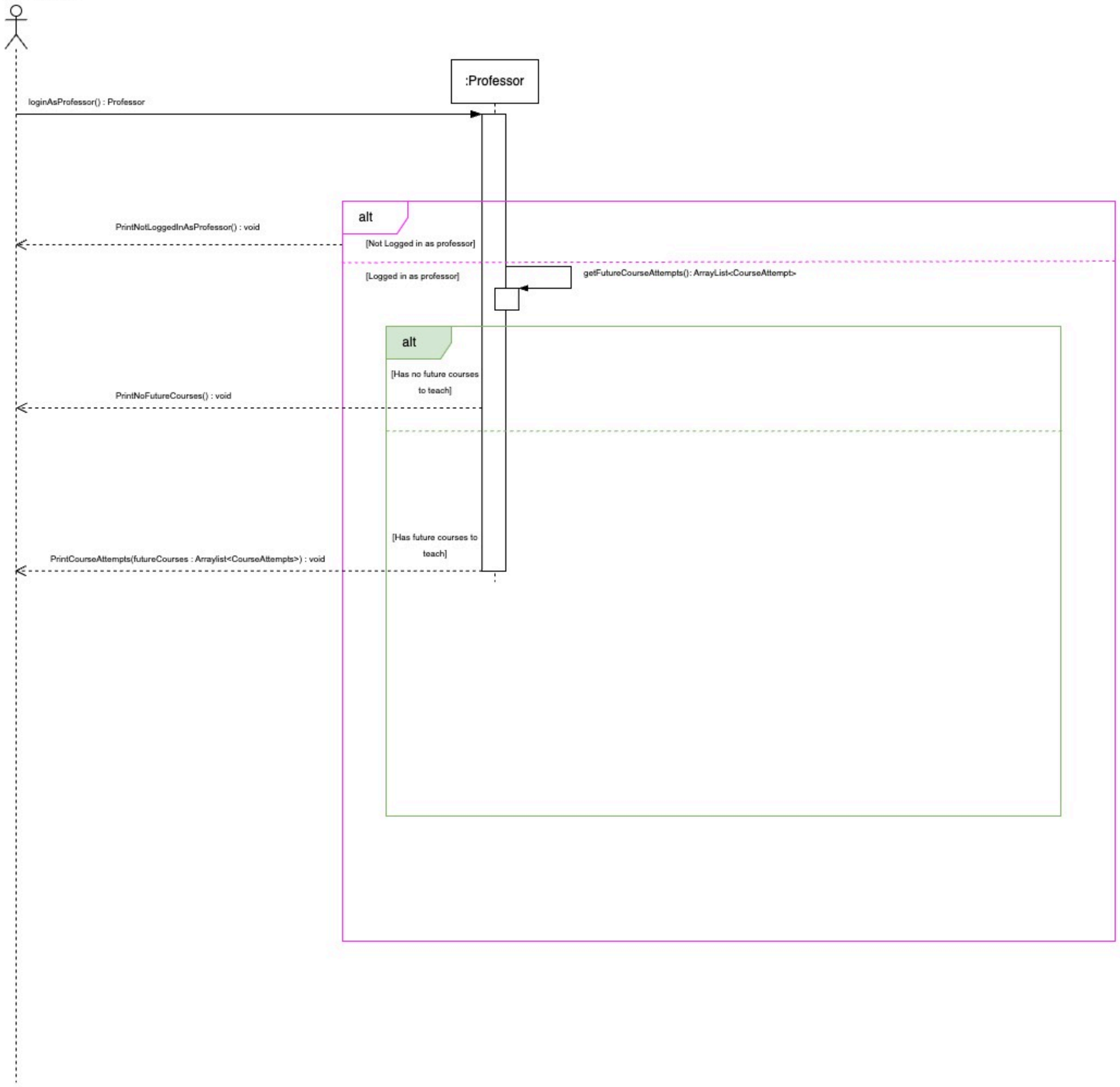
sd: UC: View TA Positions

Main Menu



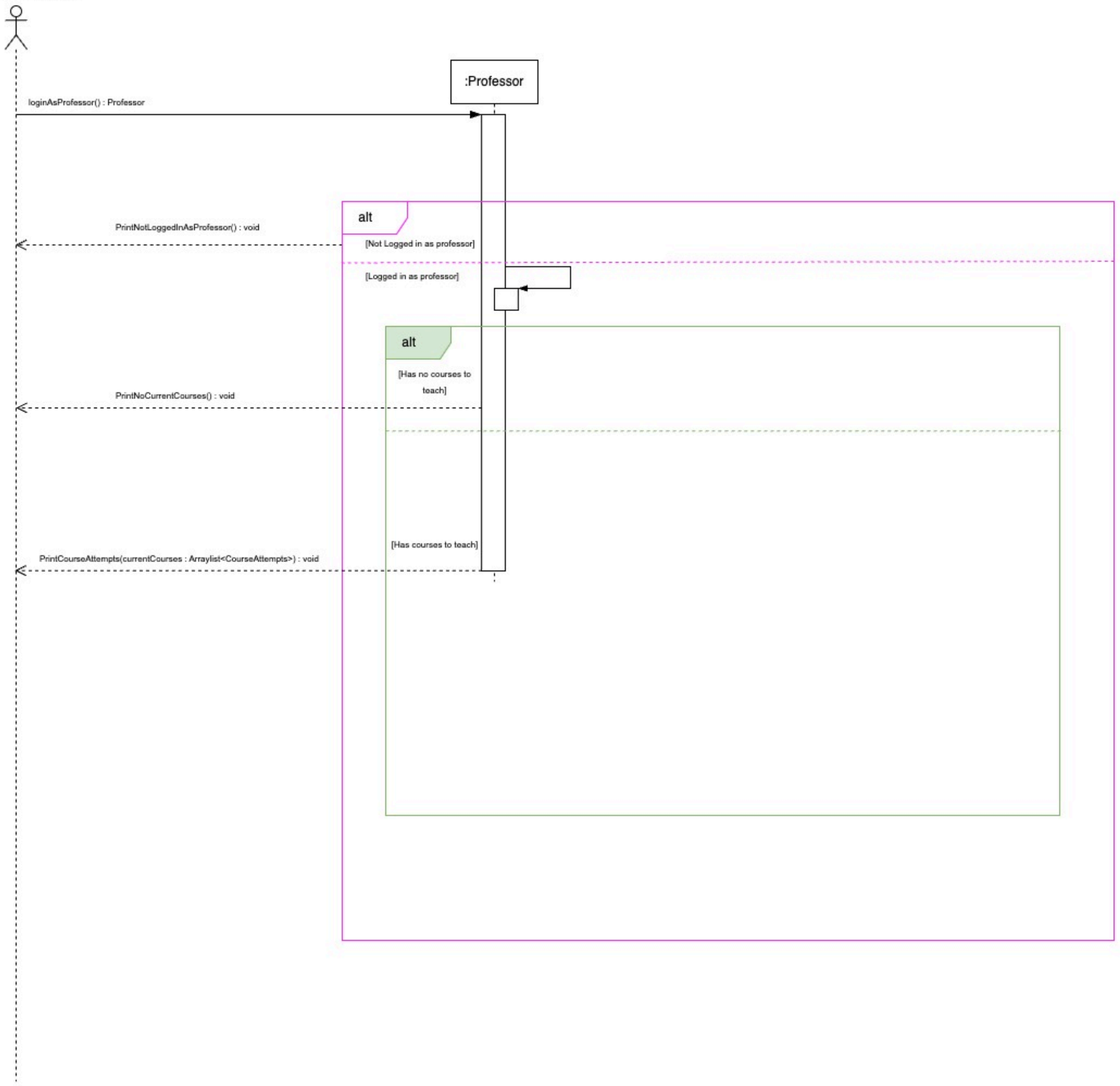
sd: UC: View future courses

Main Menu



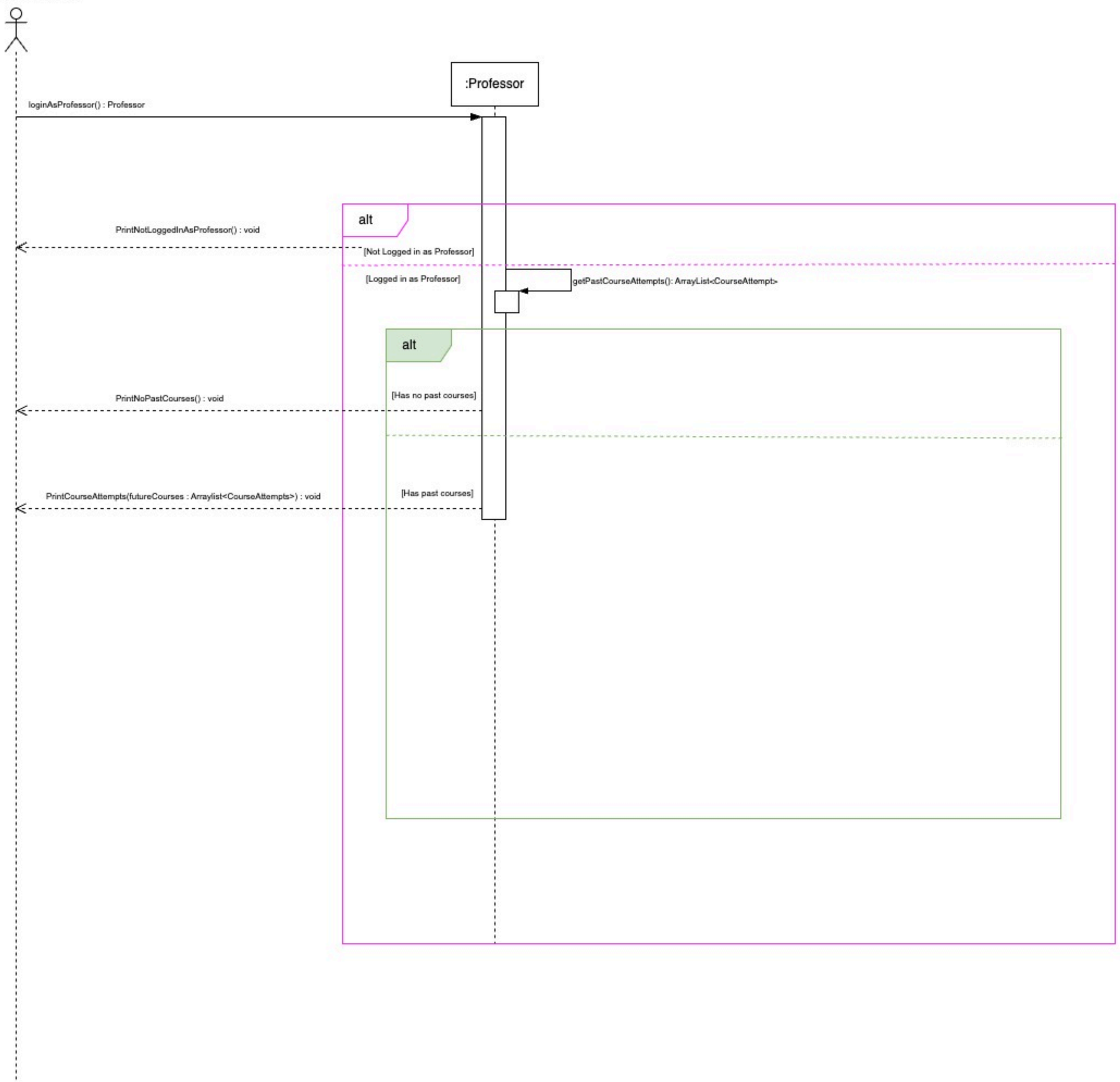
sd: UC: View current courses

Main Menu



sd: UC: View past courses

Main Menu



sd: UC: Review TA Applications

Main Menu

:Professor

:CourseCatalog

loginAsProfessor() : Professor

alt

PrintNotLoggedInAsProfessor() : void

[Not Logged in as Professor]

[Logged in as Professor]

getCourseCodeFromUser() : String

alt

PrintInvalidCourseCode() : void

Invalid Course Code

[Valid Course Code]

getListOfTeachingAssistants() : Arraylist <Student>

PrintTeachingAssistantApplicants() : void

alt

[Invalid Course Code]

[Valid Course Code]

getTeachingAssistantChoiceFromUser() : int

alt

[Invalid TA Choice]

[Valid TA Choice]

addTeachingAssistant(newTA : Student, courseCode : String) : void

PrintTeachingAssistantAdded() : void

sd: UC: View Class List

Main Menu

:Professor

loginAsProfessor() : Professor

alt

PrintNotLoggedInAsProfessor() : void

[Not Logged in as Professor]

[Logged in as Professor]

getCurrentCourses() : Arraylist <CourseAttempts>

alt

PrintNoRegisteredCourses() : void

[Has no registered courses]

[Has at least 1 registered course]

getCourseCodeFromUser() : String

alt

PrintInvalidCourseCodeError() : void

[Invalid Course Code]

[Valid Course Code]

printStudents(): void

19.

GROUP 10: MILESTONE 4

Detailed Class List

CampusCentral

Instance Variables Table

Class Names	Private Instance Variables	Descriptions
Profile	private String firstName;	Stores the First name of the Profile owner.
	private String middleName;	Stores the Middle name of the Profile owner.
	private String lastName;	Stores the Last name of the Profile owner.
	private String email;	Stores the Profile owner's university email.
	private String phoneNumber;	Stores the Profile owner's primary phone number.
	private int id;	Stores the unique university ID of the user.
	private ArrayList<CourseAttempt> courses;	Stores a list of course attempts made by the user.
Student	private String degree;	Stores the student's degree program.
	private String[2] majors;	Stores the student's program major(s). If the student is double majoring, the value at index 1 will store their second major. If not, the value at index 1 will be NULL.
	private String[2] minors;	Stores the student's minor(s). If the student doesn't have a minor, both values are NULL.
	private boolean isCoop;	Stores boolean value that represents if the student is in a Co-op program or not.
	private boolean hasCompletedDegree;	Stores boolean value that represents if the student has completed their degree or not.
	private int maxCreditsForCurrentSemester;	Stores the student's maximum allowed credits for the current semester.
	private String academicStanding;	Stores the student's current academic standing.
	private int completedCredits;	Stores the amount of credits the student has currently completed.
	private int requiredCoopCredits;	Stores the total number of Co-op credits required for graduation.

	private int requiredCredits;	Stores the total number of credits required to graduate.
	private ArrayList<String> requiredCourses;	Stores a courseCode list of the required courses for graduation.
	private ArrayList<String> waivers;	
Professor	private boolean hasPhD;	Stores a boolean value that represents if the professor has acquired their PhD.
CourseInfo (Abstract)	private String courseID;	Stores the course prefix (i.e CIS).
	private String courseName;	Stores the name of the course.
	private int courseCode;	Stores the course number for a course (i.e 2500).
	Private String department;	Stores the corresponding department of the course.
	private boolean isTransfer;	Stores a boolean value that represents if the course is a transfer course.
	private boolean isGradCourse;	Stores a boolean value that represents if the course is a graduate course.
	private ArrayList<String> prerequisites;	Stores the courses that are prerequisites for the current course.
	private String courseDescription	Stores a brief description/overview of the course.
Course (Inherits CourseInfo)	private String startDate;	Stores the start date of the course.
	private String endDate;	Stores the end date of the course.
	private ArrayList<String> sections;	Stores a list of the different sections available for the course.
	private Hashmap<String, ClassEvent[3]> sectionToLabs;	Stores a Hashmap that maps section codes to their corresponding lab ClassEvent array.
	private Hashmap<String, ClassEvent[3]> sectionToLectures;	Stores a Hashmap that maps section codes to their corresponding lecture ClassEvent array.
	private ClassEvent finalExam;	Stores the Final Exam ClassEvent.

	private Professor professor;	Stores the professor that teaches a course.
	private ArrayList<Student> students;	Stores a list of students that are registered in a course.
	private ArrayList<Student> teachingAssistants;	Stores a list of teaching assistants in a course.
	private ArrayList<Student> waitlist;	Stores the students on the course waitlist.
	private String semester;	Stores the Semester of the course offering (i.e F2023, W2023, S2023, etc.)
	private int capacity;	Stores the maximum number of students in the course.
	private int numTAPositionsAvailable;	Stores the number of available Teaching Assistant positions left for a course.
	private ArrayList<Student> teachingAssistantApplications;	Stores a list of Students who have applied for a Teaching Assistant position for the course.
	private Hashmap<String, int> numAvailableSeatsForSection;	Stores the number of available seats in a section.
CourseAttempt	private Course course;	Pointer to the Course object where the attempt was made.
	private String courseProgress;	Represents the state of the course attempt (i.e “Planned”, “Registered”, “Completed”, “Failed”, “Dropped”, etc.)
	private String userRole;	Represents the role of the course attempt owner (i.e “Student”, “Professor”, or “TA”).
	private String section;	Stores the section of the course that the user has registered for; NULL if the user is a Professor/Teaching Assistant.
ClassEvent	private String dateTime;	Represents the date and time for a meeting.
	private String type;	Stores the type of event for the class (Lecture, Lab, or Exam).
CourseCatalog	private Hashmap<String, Course> courses;	A hashmap that stores Course instances as values accessed by a CourseID String key.

Public Getter Methods Table

Class Names	Public Getter Methods	Descriptions
Profile	public String getFirstName();	Returns the **firstName** String attribute from a Profile instance.
	public String getMiddleName();	Returns the **middleName** String attribute from a Profile instance.
	public String getLastName();	Returns the **lastName** String attribute from a Profile instance.
	public String getEmail();	Returns the **email** String attribute from a Profile instance.
	public String getPhoneNumber();	Returns the **phoneNumber** String attribute from a Profile instance.
	public int getID();	Returns the **ID** int attribute from a Profile instance.
	private ArrayList<CourseAttempt> getCourses();	Returns the **courses** Arraylist attribute from a Profile instance.
Student	public String getDegree();	Returns the **degree** String attribute from a Student instance.
	public String[2] getMajors();	Returns the **majors** String array attribute from a Student instance.
	public String[2] getMinors();	Returns the **minors** String array attribute from a Student instance.
	public boolean getIsCoop();	Returns the **isCoop** boolean attribute from a Student instance.
	public boolean getHasCompletedDegree();	Returns the **hasCompletedDegree** boolean attribute from a Student instance.
	public int getMaxCreditsForCurrentSemester()	Returns the **maxCreditsForCurrentSemester** int attribute from a Student instance.
	public String getAcademicStanding();	Returns the **academicStanding** String attribute from a Student instance.
	public int getCompletedCredits();	Returns the **completedCredits** int attribute from a Student instance.
	public int getRequiredCoopCredits();	Returns the **requiredCoopCredits** int attribute from a Student instance.
	public int getRequiredCredits();	Returns the **requiredCredits** int attribute from a Student instance.

	public ArrayList<String> getRequiredCourses();	Returns the **requiredCourses** Arraylist attribute from a Student instance.
Professor	public ArrayList<CourseAttempt> getHasPhD();	Returns the **hasPhD** boolean attribute from a Professor instance.
CourseInfo	public String getCourseID();	Returns the **courseID** String attribute from a CourseInfo instance.
	public String getCourseName();	Returns the **courseName** String attribute from a CourseInfo instance.
	public int getCourseCode();	Returns the **courseCode** int attribute from a CourseInfo instance.
	Private String getDepartment();	Returns the **department** String attribute from a CourseInfo instance.
	public boolean getIsTransfer();	Returns the **isTransfer** boolean attribute from a CourseInfo instance.
	public boolean getIsGradCourse();	Returns the **isGradCourse** boolean attribute from a CourseInfo instance.
	public ArrayList<String> getPrerequisites();	Returns the **prerequisites** ArrayList attribute from a CourseInfo instance.
	public String getCourseDescription();	Returns the **courseDescription** String attribute from a CourseInfo instance.
Course	public String getStartDate();	Returns the **startDate** String attribute from a Course instance.
	public String getEndDate();	Returns the **endDate** String attribute from a Course instance.
	public ArrayList<String> getSections();	Returns the **sections** Arraylist attribute from a Course instance.
	public Hashmap<String, ClassEvent[3]> getSectionToLabs();	Returns the **sectionToLab** Hashmap attribute from a Course instance.
	public Hashmap<String, ClassEvent[3]> getSectionToLectures();	Returns the **sectionToLecture**; Hashmap attribute from a Course instance.

	public ClassEvent getFinalExam();	Returns the **finalExam** ClassEvent attribute from a Course instance.
	public Professor getProfessor();	Returns the **professor** Professor attribute from a Course instance.
	public ArrayList<Student> getStudents();	Returns the **Students** ArrayList attribute from a Course instance.
	public ArrayList<Student> getTeachingAssistants();	Returns the **TeachingAssistants** ArrayList attribute from a Course instance.
	public ArrayList<Student> getWaitlist();	Returns the **Waitlist** ArrayList attribute from a Course instance.
	public String getSemester();	Returns the **semester** String attribute from a Course instance.
	public int getCapacity();	Returns the **capacity** int attribute from a Course instance.
	public int getNumTAPositionsAvailable();	Returns the **numTAPositionsAvailable** int attribute from a Course instance.
	public ArrayList<Student> getTeachingAssistantApplications();	Returns the **teachingAssistantApplications** Arraylist attribute from a Course instance.
Course Attempt	public Course getCourse();	Returns the **course** Course attribute from a Course Attempt instance.
	public Integer getGrade();	Returns the **grade** Integer attribute from a Course Attempt instance.
	public String getCourseProgress();	Returns the **courseProgress** String attribute from a Course Attempt instance.
	public String getUserRole();	Returns the **userRole** String attribute from a Course Attempt instance.
	public boolean getHasWaiver();4	Returns the **hasWaiver** boolean attribute from a Course Attempt instance.

	public String getSection();	Returns the **section** String attribute from a Course Attempt instance.
Class Event	public String getDateTime();	Returns the **dateTime** String attribute from a Class Event instance.
	public String getType();	Returns the **type** String attribute from a Class Event instance.
Course Catalog	public HashMap<String, Course> getCourses()	Returns the **Courses** HashMap attribute from a Course Catalog instance.

Public Setter Methods Table

Class Names	Public Setter Methods	Descriptions
Profile	public void setFirstName(String firstName);	Sets the **firstName** String attribute on a Profile instance.
	public void setMiddleName(String middleName);	Sets the **middleName** String attribute on a Profile instance.
	public void setLastName(String lastName);	Sets the **lastName** String attribute on a Profile instance.
	public void setEmail(String email);	Sets the **email** String attribute on a Profile instance.
	public void setPhoneNumber(String phoneNumber);	Sets the **phoneNumber** String attribute on a Profile instance.
	public void setID(int id);	Sets the **id** int attribute on a Profile instance.
	public void setCourses(ArrayList<CourseAttempt> courses);	Sets the **courses** ArrayList attribute on a Profile instance.
Student	public void setDegree(String degree);	Sets the **degree** String attribute on a Student instance.
	public void setMajors(String[2] majors);	Sets the **majors** String array attribute on a Student instance.
	public void setMinors(String[2] minors);	Sets the **minors** String array attribute on a Student instance.
	public void setIsCoop(boolean isCoop);	Sets the **isCoop** boolean attribute on a Student instance.
	public void setHasCompletedDegree(boolean hasCompletedDegree);	Sets the **hasCompletedDegree** boolean attribute on a Student instance.
	public void setMaxCreditsForCurrentSemester(int maxCreditsForCurrentSemester);	Sets the **maxCreditsForCurrentSemester** int attribute on a Student instance.
	public void setAcademicStanding(String academicStanding);	Sets the **academicStanding** String attribute on a Student instance.
	public void setCompletedCredits(int completedCredits);	Sets the **completedCredits** int attribute on a Student instance.
	public void setRequiredCoopCredits(int requiredCoopCredits);	Sets the **requiredCoopCredits** int attribute on a Student instance.
	public void setRequiredCredits(int requiredCredits);	Sets the **requiredCredits** String int on a Student instance.

	public void setRequiredCourses(ArrayList<String> requiredCourses);	Sets the **requiredCourses** ArrayList attribute on a Student instance.
Professor	Public void setHasPhD(boolean hasPhD);	Sets the **hasPhD** boolean attribute on a Professor instance.
CourseInfo	public void setCourseID(String courseID);	Sets the **courseID** String attribute on a CourseInfo instance.
	public void setCourseName(String courseName);	Sets the **courseName** String attribute on a CourseInfo instance.
	public void setCourseCode(int courseCode);	Sets the **courseCode** int attribute on a CourseInfo instance.
	public void setDepartment(String department);	Sets the **department** String attribute on a CourseInfo instance.
	public void setIsTransfer(boolean isTransfer);	Sets the **isTransfer** boolean attribute on a CourseInfo instance.
	public void setIsGradCourse(boolean isGradCourse);	Sets the **isGradCourse** boolean attribute on a CourseInfo instance.
	public void setPrerequisites(ArrayList<String> Prerequisites);	Sets the **prerequisites** ArrayList attribute on a CourseInfo instance.
	public void setCourseDescription(String courseDescription);	Sets the **courseDescription** String attribute on a CourseInfo instance.
Course	public void setStartDate(String startDate);	Sets the **startDate** String attribute on a Course instance.
	public void setEndDate(String endDate);	Sets the **endDate** String attribute on a Course instance.
	public void setSections(ArrayList<String> sections);	Sets the **sections** Arraylist attribute on a Course instance.
	public void setSectionToLabs(Hashmap<String, ClassEvent[3]> sectionToLabs);	Sets the **sectionToLabs** ArrayList attribute on a Course instance.
	public void setSectionToLectures (Hashmap<String, ClassEvent[3]> sectionToLectures);	Sets the **sectionToLectures** ArrayList attribute on a Course instance.
	public void setFinalExam(ClassEvent finalExam);	Sets the **FinalExam** ClassEvent attribute on a Course instance.
	public void setProfessor(Professor professor);	Sets the **professor** Professor attribute on a Course instance.

	public void setStudents(ArrayList<Student> students);	Sets the **students** ArrayList attribute on a Course instance.
	public void setTeachingAssistants (ArrayList<Student> teachingAssistants);	Sets the **teachingAssistants** ArrayList attribute on a Course instance.
	public void setWaitlist(ArrayList<Student> waitlist);	Sets the **waitlist** ArrayList attribute on a Course instance.
	public void setSemester(String semester);	Sets the **semester** String attribute on a Course instance.
	public void setCapacity(int capacity);	Sets the **capacity** int attribute on a Course instance.
	public void setNumTAPositionsAvailable(int capacity);	Sets the **numTAPositionsAvailable** int attribute on a Course instance.
	Public void setTeachingAssistantApplications (ArrayList<Student> teachingAssistantApplications);	Sets the **teachingAssistantApplications** ArrayList attribute on a Course instance.
CourseAttempt	public void setCourse(Course course);	Sets the **course** Course attribute on a CourseAttempt instance.
	public void setGrade(Integer grade);	Sets the **grade** Integer attribute on a CourseAttempt instance.
	public void setCourseProgress(String courseProgress);	Sets the **courseProgress** String attribute on a CourseAttempt instance.
	public void setUserRole(String userRole);	Sets the **userRole** String attribute on a CourseAttempt instance.
	public void setHasWaiver(boolean hasWaiver);	Sets the **hasWaiver** boolean attribute on a CourseAttempt instance.
	Public void setSection(String section);	Sets the **section** String attribute on a CourseAttempt instance.
ClassEvent	public void setDateTime(String dateTime);	Sets the **dateTime** String attribute on a ClassEvent instance.
	public void setType(String type);	Sets the **type** String attribute on a ClassEvent instance.
CourseCatalog	public void setCourses(Hashmap<String, Course> courses);	Sets the **courses** Hashmap attribute on a CourseCatalog instance.

Public Constructor Methods Table

Class Name	Constructor Method	Description
Profile (Abstract)	public Profile(String firstName, String middleName, String lastName, String email, int phoneNumber, int id, ArrayList<CourseAttempt> courses);	Creates a new **Profile** object with the given parameters.
Student (Inherits Profile)	public Student(String degree, String[2] majors, String[2] minors, boolean isCoop, boolean hasCompletedDegree, int maxCreditsForCurrentSemester, String academicStanding, int completedCredits, int requiredCoopCredits, int requiredCredits, ArrayList<String> requiredCourses);	Creates a new **Student** object with the given parameters.
Professor (Inherits Profile)	public Professor(boolean hasPhD);	Creates a new **Professor** object with given parameters.
Course Info (Abstract)	public CourseInfo(String courseID, String courseName, int courseCode, String department, boolean isTransfer, boolean isGradCourse, ArrayList<String> prerequisites, String courseDescription);	Creates a new **CourseInfo** object with the given parameters.
Course (Inherits CourseInfo)	public Course(CourseInfo courseInfo, String startDate, String endDate, ArrayList<String> sections, Hashmap<String, ClassEvent[3]> sectionToLabs, Hashmap<String, ClassEvent[3]> sectionToLectures, ClassEvent finalExam, Professor professor, ArrayList<Student> students, ArrayList<Student> teachingAssistants, ArrayList<Student> waitlist, String semester, int capacity, int numTAPositionsAvailable, ArrayList<Student> teachingAssistantApplications);	Creates a new **Course** object with the given parameters.
CourseAttempt	public CourseAttempt(Course course, Integer grade, String courseProgress, String userRole, boolean hasWaiver; String section);	Creates a new **CourseAttempt** object with the given parameters.
ClassEvent	public ClassEvent(String dateTime, String type);	Creates a new **ClassEvent** object with given parameters.
CourseCatalog	public CourseCatalog();	Creates a new **CourseCatalog** Object. **courses** is set to a Hashmap with (key, value) pairs from courseID to Course object.

Public Methods Table

Class Name	Public Methods (No Constructors/Getters/Setters)	Descriptions
Main	public static void main();	The entry point of the application. This sets up the Main object and shows the GUI display.
	Public void addError(String error)	Accesses the Main instance and appends the error message to the back of the errorLog ArrayList.
	public void update(Object obj);	Updates the GUI to reflect changes in the provided object (i.e. Course, Profile, etc.).
Student (Inherits Profile)	public void planCourse(CourseAttempt newCourse);	Accesses the Student instance and appends the Course to the back of the courses ArrayList.
	public void registerCourse(String programID, int courseCode);	Accesses the Student instance and sets the isRegistered attribute of the corresponding ClassAttempt to true.
	public void dropCourse(String programID, int courseCode);	Drops a course from the student's registered courses using the provided program ID and course code.
Course (Inherits Course Info)	public addTeachingAssistant(Student newTA);	Adds a new teaching assistant to the course.
Course Catalog	Public void addCourse(Course course);	Adds a new (key, value) pair to the course Hashmap in the CourseCatalog.
	Public void removeCourse(String programID, int courseCode);	Removes a course from the course catalog using the provided program ID and course code.
	Public void returnFilteredCourse(String program, int courseLevel);	Returns an array of filtered courses based on the provided filtering parameters.

Main Class Public Methods Algorithm Table

Public Methods	Algorithm Details

public static void main();	public static void main() {
	  courseCatalog ← new CourseCatalog();
	  currentUser ← NULL;
	
	  while (currentUser is NULL)
	  printLoginPrompt();
	  userLogin ← read stdin;
	  currentUser ← getCurrentUser(userLogin);
	
	  isStudent ← (currentUser instanceOf Student);
	  printWelcomeMessage(currentUser);
	
	  do {
	  printOptions(isStudent);
	  userInput ← read stdin;
	
	  if (isQuit(userInput)) break;
	  handleAction(isStudent, userInput);
	  } while(true);
	
	  printExitMessage();
	}

Main Class Private Methods Algorithm Table

Private Methods	Algorithm Details

private void printLoginPrompt();	private void printLoginPrompt() {
	  Print “Enter ‘s’ if you are a student, or ‘p’ if you are a professor: ”;
	}

private Profile getCurrentUser(String userLogin);	private Profile getCurrentUser(String userLogin) {
	  if (userLogin == “s) return getStudentUser(); // LOW LEVEL FUNCTIONALITY
	  else return getProfessorUser(); // LOW LEVEL FUNCTIONALITY
	}

private void printWelcomeMessage()	private void printWelcomeMessage(Profile profile) {
	  Print “Welcome to Campus Central Course Scheduler”;
	  Print “Currently signed in as”;
	  if (profile is instance of Professor AND profile.getHasPhD()) Print “ Dr.”;
	  Print Profile.getName();
	  Print “Enter ‘q’ or ‘Quit’ to exit Course Scheduler”;
	}

private void printOptions(boolean isStudent);	private void printOptions(boolean isStudent)
	  Print “What would you like to do today?”;
	  Print “Please enter a number to select an option:”;
	  if (isStudent) printStudentOptions();
	  else printProfessorOptions();

private void printStudentOptions();	private void printStudentOptions()
	  Print “1. Add a planned course to the course schedule”;
	  Print “2. Remove a planned course from the course schedule”;
	  Print “3. Register for a planned course to course schedule”;
	  Print “4. Drop a registered course from the course schedule”;
	  Print “5. View Future Courses”;
	  Print “6. View Current Courses”;
	  Print “7. View Past Courses”;
	  Print “8. View TA positions available”;
	  Print “9. Apply for TA position”;

private void printProfessorOptions();	private void printProfessorOptions()
	  Print “1. View the courses I am teaching next semester”;
	  Print “2. View my past courses taught”;
	  Print “3. Review TA Applications”;
	  Print “4. View class list for course”;

private void handleAction(boolean isStudent, String userInput, Student student)	private void handleAction(CourseCatalog courseCatalog, boolean isStudent, String userInput, Profile user) {
	  if (isStudent) handleStudentAction(userInput, user, courseCatalog);
	  else handleProfessorAction(userInput, user, courseCatalog);
	}

private void handleStudentAction(String userInput, Student student)	private void handleStudentAction(String userInput, Student student, CourseCatalog courseCatalog)
	  if (userInput == “1”) handleAddPlannedCourse(student);
	  if (userInput == “2”) removePlannedCourse(student);
	  if (userInput == “3”) registerPlannedCourse(student);
	  if (userInput == “4”) dropPlannedCourse(student);
	  if (userInput == “5”) printFutureCourses(student);
	  if (userInput == “6”) printCurrentCourses(student);
	  if (userInput == “7”) printPastCourses(student);
	  if (userInput == “8”) printTAPositions(courseCatalog);
	  if (userInput == “9”) applyForTAPosition(student);

private void handleProfessorAction(userInput, user, courseCatalog);	private void handleProfessorAction(String userInput, Professor professor, courseCatalog)
	  if (userInput == “1”) printCurrentCourses(professor);
	  if (userInput == “2”) printPastCourses(professor);
	  if (userInput == “3”) reviewTAApplications(professor);
	  if (userInput == “4”) printClasslist(professor);

private void handleAddPlannedCourse(Student student);	private void handleAddPlannedCourse(Student student)
	  userCourseCode ← NULL;
	
	  do {
	  printAddPlannedCoursePrompt(student);
	
	  userCourseCode ← read stdin;
	  if (isQuit(userCourseCode)) return;
	
	  newCourse ← courseCatalog.get(userCourseCode);
	
	  if (newCourse is NULL)
	  Print “Invalid Course Code. Please try again!”
	  continue;
	
	  if (alreadyInCourses(student, userCourseCode))
	  Print “This course is already in your schedule. Please try again!”

	continue;
	
	Print “Enter desired section: ”;
	desiredSection ← getUserInput();
	
	if (newCourse.getSectionCapacities().get(desiredSection) <= 0)
	Print “No more seats left in section. Please try another section!”
	continue;
	newCourseAttempt = new
	CourseAttempt(newCourse, “Planned”, “Student”, desiredSection);
	
	student.addCourse(newCourseAttempt);
	break;
	} while (true)

private void printAddPlannedCoursePrompt(Student student)	private void printAddPlannedCoursePrompt(Student student, CourseCatalog courseCatalog) Print “Which course would you like to plan? Enter ‘q’ or ‘quit’ to exit to main menu”; Print “Please enter the course code and ID (i.e. CIS*1300):”; Print “Courses you currently have planned:”; For each courseAttempt in student.getCourses() course ← courseAttempt.getCourse(); Print course.getFullCourseCode();
private void isQuit(String userInput)	private void isQuit(String userInput) Return (userInput.toLower() == ‘q’ OR userInput.toLower() == ‘quit’)
private boolean alreadyInCourses(Arraylist<CourseAttempts> courses)	private boolean alreadyInCourses(Arraylist<CourseAttempts> courses, String courseCode) For each courseAttempt in student.getCourses() currentCourse ← courseAttempt.course; currentCourseCode ← currentCourse.getFullCourseCode(); if (currentCourseCode == courseCode) return true; return false;

private boolean studentHasWaiverForCourse(Student student, String userCourseCode);	private boolean studentHasWaiverForCourse(Student student, String userCourseCode) {
	For each courseCode in student.getWaivers()
	if (courseCode === userCourseCode) return true;
	
	return false;
	}

private void removePlannedCourse(Student student);	private void removePlannedCourse(Student student)
	do {
	printRemovePlannedCoursePrompt();
	userCourseCode ← read stdin;
	if (isQuit(userCourseCode)) return;
	
	For each courseAttempt in student.getCourses()
	if (courseAttempt.getCourse().getFullCourseCode() == userCourseCode)
	Remove courseAttempt from courses;
	Print “Course Removed. Back to the main menu!”;
	return;
	
	print “Invalid Course Code. Please try again!”
	} while (true)

private void printRemovePlannedCoursePrompt();	private void printRemovePlannedCoursePrompt()
	  Print “Which planned course would you remove from your schedule?”
	  Print “Enter ‘q’ or ‘quit’ to exit to main menu”
	  Print “Please enter the course code and ID (i.e CIS*1300): ”

private void registerPlannedCourse(Student student);	private void registerPlannedCourse(Student student) {
	  do {
	  printRegisterPlannedCoursePrompt();
	  userCourseCode ← read stdin;
	  if (isQuit(userCourseCode)) return;
	
	  For each courseAttempt in student.getCourses()
	  if (courseAttempt.getCourse().getFullCourseCode() == userCourseCode)
	  currentCourse.setCourseProgress(“Registered”);
	  Print “Course Registered. Back to the main menu!”
	  return;
	
	  print “Invalid Course Code. Please try again!”
	  } while (true)
	}

private void printRegisterPlannedCoursePrompt();	private void printRegisterPlannedCoursePrompt() {
	  Print “Which planned course would you register for?”
	  Print “Enter ‘q’ or ‘quit’ to exit to main menu”
	  Print “Please enter the course code and ID (i.e CIS*1300): ”
	}

private void dropPlannedCourse (Student student)	private void dropPlannedCourse(Student student) {
	  do {
	  printDropPlannedCoursePrompt();
	  userCourseCode ← read stdin;
	  if (isQuit(userCourseCode)) return;
	
	  For each courseAttempt in student.getCourses()
	  if (courseAttempt.getCourse().getCourseCode() == userCourseCode)
	  Remove courseAttempt from courses;
	  Print “Course Dropped. Back to the main menu!”
	  return;
	
	  print “Invalid Course Code. Please try again!”
	  } while (true)
	}

private void printCurrentCourses (Profile profile)	private void printCurrentCourses(Profile profile) {
	  currentSemester = getCurrentSemester() // LOW LEVEL FUNCTIONALITY
	
	  For each courseAttempt in student.getCourses()
	  if (courseAttempt.getCourse().getSemester() == currentSemester)
	  Print courseAttempt.toString()
	}

private void printFutureCourses (Profile profile)	private void printFutureCourses(Profile profile)
	  nextSemester = getNextSemester() // LOW LEVEL FUNCTIONALITY
	
	  For each courseAttempt in student.getCourses()
	  if (courseAttempt.getCourse().getSemester() == nextSemester)
	  Print courseAttempt.toString()

private void printPastCourses (Profile profile)	private void printPastCourses(Profile profile) {
	  prevSemester = getPreviousSemester() // LOW LEVEL FUNCTIONALITY
	
	  For each courseAttempt in student.getCourses()
	  if (courseAttempt.getCourse().getSemester() == prevSemester)
	  Print courseAttempt.toString(); // LOW LEVEL FUNCTIONALITY
	}

private void printTAPositions(Course Catalog courseCatalog)	private void printTAPositions(CourseCatalog courseCatalog) {
	  numCoursesPrinted ← 0;
	
	  For each course in courseCatalog.getCourses()
	  if (courseCatalog.getNumTAPositionsAvailable() > 0)
	  Print course.toString();
	  numCoursesPrinted++;
	
	  if (numCoursesPrinted <= 0)
	  Print “No TA Positions available”;
	}

private void applyForTAPosition (Student student, CourseCatalog courseCatalog)	private void applyForTAPosition(Student student, CourseCatalog courseCatalog) {
	  do {
	  printApplyAsTAPrompt();
	  userCourseCode ← read stdin;
	  if (isQuit(userCourseCode)) return;
	
	  userCourse ← courseCatalog.getCourses().get(courseCatalog);
	
	  if (userCourse == NULL)
	  Print “Invalid Course! Please enter again!”;
	  continue;
	
	  alreadyApplicant ←
	  userCourse.getTeachingAssistantApplications().contains(student);
	
	  if (alreadyApplicant)
	  Print “You are already an applicant! Returning to the main menu!”;
	  else
	  userCourse.addTAApplicant(student)
	
	  return;
	  } while (true)
	}

private void printApplyAsTAPrompt()	private void printApplyAsTAPrompt() {
	  Print “Which course would you like to apply as a TA for?”;
	  Print “Enter ‘q’ or ‘quit’ to exit to main menu”;
	  Print “Please enter the course code and ID (i.e CIS*1300): ”;
	}

private void reviewTAApplications(Pr ofessor professor)	private void reviewTAApplications(Professor professor) {
	  do {
	  printReviewTAApplicationsPrompt();
	
	  userCourseCode ← read stdin;
	  if (isQuit(userCourseCode)) return;
	  userCourse ← NULL;
	
	  For each courseAttempt in professor.getCourses()

	if (courseAttempt.getCourse().getFullCourseCode() == userCourseCode)
	  userCourse ← courseAttempt.getCourse();
	
	if (userCourse is NULL)
	  Print “Invalid code! Try again!”;
	  continue;
	
	For each applicant in userCourse.getTeachingAssistantApplications()
	  Print applicant.toString();
	
	do {
	  printChooseTAApplicationPrompt();
	  userTAChoice ← read stdin;
	  if (isQuit(userCourseCode)) break;
	
	  TA ← userCourse.getTeachingAssistantApplications().get(userTAChoice);
	  if (TA is NULL)
	  Print “Invalid choice! Try again!”;
	  Continue;
	
	  userCourse.getTeachingAssistants().append(TA);
	  userCourse.getTeachingAssistantApplications().remove(userTAChoice);
	  return;
	 } while (true)
	
	 return;
	} while (true)
	}

private void printReviewTAApplicationsPrompt()	private void printReviewTAApplicationsPrompt() {
	  Print “Which course would you like to view TA applications for?”;
	  Print “Enter ‘q’ or ‘quit’ to exit to main menu”;
	  Print “Please enter the course code and ID (i.e CIS*1300): ”;
	}

private void printChooseTAApplicationPrompt()	private void printChooseTAApplicationPrompt() {
	  Print “Which student would you like to accept as TA?”;
	  Print “Enter ‘q’ or ‘quit’ to exit to main menu”;
	  Print “Please enter the list number of the candidate: ”;
	}

private void printExitMessage()	private void printExitMessage() Print “Thank you for using CampusCentral. See you later!”;

Course Class Private Methods Algorithm Table

private String getFullCourseCode (Course course)	private String getFullCourseCode() {
	  return this.getCourseID() + ‘*’ + this.getCourseCode();
	}

private void addTAApplicant (Student student)	private void addTAApplicant(Student student) {
	  applicants ← this.getTeachingAssistantApplications();
	  applicants.append(student);
	}

GROUP 10: MILESTONE 4

Use Cases

CampusCentral

UCCS1: Registering for a planned course

Purpose: To allow students to register for courses they have previously added as planned courses in the CampusCentral course management system.
Primary Actor: Student
Stakeholders List:

- Program Counselors
    - Wants to be able to provide guidance and support to students when course registration is available.
- System Administrator:
    - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student is an authenticated user of CampusCentral.
- The student has added the desired course as a planned course on their schedule

Postconditions:

- The course has been added to the student's schedule for the corresponding term.
- The student's course schedule is updated with the correct course details (section, ID, and time)

Basic Flow (Main Success Scenario):

1. The student logs into CampusCentral.
2. The user navigates the system to the Course Scheduler.
3. User enters the option to register for a planned course.
4. The student enters the desired course to register.
5. The system displays the available sections for the selected course.
6. The student enters the section that fits their schedule.
7. The student verifies and registers for the course.
8. The system updates the student's schedules to reflect their enrollment in the course.

Alternate Flow (Exceptions):

*a. At any time, System fails:
To support recovery, the state of the student's schedule from any step of the scenario

1. The user re-logs onto Campus Central, the schedule remains in the same state as before

4a. The selected course is completely full:

1. The system displays an error message that indicates the course they have selected is full.
2. Control is returned to user.

4b. The selected courses' prerequisites have not been met:

1. The system displays an error message that states that the prerequisites for this course have not been met.
2. The student must complete the course's prerequisites before enrolling in the desired course.
3. Control is returned to user.

6a. The selected course section is full:

1. The system displays an error message that states the course section they've selected is full.
2. The student must select a different section or course.
3. Control is returned to user.

6b. The course exceeds the maximum number of courses they can take

1. The system displays an error message that states that the course cannot be registered as the student has too many courses registered already
2. Control is returned to user

UCCS2: Dropping a registered course

Purpose: To allow students to drop courses they have previously registered in through the CampusCentral course management system.

Primary Actor: Student

Stakeholders List:

- Program Counselors
    - Wants to be able to provide guidance and support to students when a student is debating on dropping a course.
- System Administrator:
    - Wants to be able to provide technical support for students if any problems occur during course selection.
- University Registrar:
    - Wants to be able to update the student's academic record and course availability once a course has been dropped.

Preconditions:

- The student is currently registered for a course.
- The student has identified what course they would like to drop for the semester

Postconditions:

- The student's academic record is updated to reflect the dropped course.
- The course capacity is updated in the university course management system.

Basic Flow (Main Success Scenario):

1. The student logs into CampusCentral.
2. The user navigates the system to the Course Scheduler.
3. The system displays the student's registered courses.
4. The student enters the course they want to drop.
5. The system asks the student to confirm whether they want to drop the course.
6. The system removes the course from the student's registered courses.
7. The system updates the student's record to reflect the dropped course.
8. The system updates the courses available for the student.
9. The system displays a confirmation message regarding the dropped course.

Alternate Flow (Exceptions):

*a. At any time, System fails:
To support recovery, the state of the student’s schedule from any step of the scenario

1. The user re-logs onto Campus Central, the schedule remains in the same state as before

3a. The student is not currently registered in any courses:

1. The system will not have the option to drop courses without a registered course.
2. Control is returned to user.

6a. The system encounters an error while dropping the course:

1. The system displays an error message indicating that the course could not be dropped due to an error.
2. The system prompts the student to try again later or contact technical support for assistance.
3. Control is returned to user.

UCCS3: Adding a planned course to the student's schedule

Purpose: To assist students to plan their courses to meet their academic requirements by distributing courses across the remaining semesters in their program.

Primary Actor: Student

Stakeholders List:

- Academic Advisors:
    - Providing assistance and guidance to help the student's schedule planning with choosing appropriate courses to meet program requirements
- System Administrator:
    - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student is an authenticated user of CampusCentral

Postconditions:

- The course has been added to the student's planned courses for the selected semester.
- The student's course schedule is updated with the new planned course.

Basic Flow (Main Success Scenario):

1. The user logs into CampusCentral as an authenticated student.
2. The user navigates the system to the Course Scheduler.
3. The user searches for their desired course in the course catalog.
4. The user adds their desired course to their plan entering the course
5. The course is added to the student's plan for that semester.

Alternate Flow (Exceptions):

*a. At any time, System fails:

1. To support recovery, the state of the student's schedule from any step of the scenario.
2. The user logs onto Campus Central, the schedule remains in the same state as before.
3. Control is returned to user.

3a. User enters incorrect course

1. The system indicates that the user entered an incorrect course
2. System prompts user for another course
3. Control is returned to user

UCCS4: Removing a planned course from the student's schedule

Purpose: To enable students to fix mistakes made on their academic plan or reschedule their previous plans for any reason.

Primary Actor: Student

Stakeholders List:

- Program Counselor:
    - To provide guidance and assistance for students that have conflicts within their academic schedule
- System Administrator:
    - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student is an authenticated user of CampusCentral

Postconditions:

- The course has been removed from the student’s planned courses for the selected semester.
- The student’s course schedule is updated to reflect the removal of the course.

Basic Flow (Main Success Scenario):

1. The user logs into CampusCentral as an authenticated student.
2. The user navigates the system to the Course Scheduler.
3. The user enters the course they want to remove from planned courses
4. The course is removed from the student’s plan.

Alternate Flow (Exceptions):

*a. At any time, System fails:
1. To support recovery, the state of the student’s schedule from any step of the scenario
2. The user logs onto Campus Central, the schedule remains in the same state as before

4a. The selected course is a prerequisite for another course on the student’s plan.
1. The user is notified with an error message that states the course is a prerequisite for other courses on their plan.
2. The process will be canceled.
3. Control is returned to user.

UCCS5: Viewing Future Courses

Primary Actor: Student

Purpose: To allow students to preview future courses which they are required to take and be aware of any necessary prerequisites that they might need to take prior to taking a said course to allow them to plan semesters accordingly.

Stakeholders List:

- **Program Counselor**
  - To be able to advise students in their specific program about the courses which are mandatory to complete the degree.
- **Academic Advisor**
  - To be able to advise students on the best possible courses to take in specific semesters and preview which need to be taken for specific programs.
- **System Administrator:**
  - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student is enrolled in a program and is a student at the university.
- The student is an authenticated user of Campus Central.

Postconditions:

- Students are able to view courses they are taking in the future, for the upcoming semester

Basic Flow (Main Success Scenario):

1. The user logs into CampusCentral as an authenticated student.
2. The user navigates the system to the Course Scheduler.
3. The user enters the option to view future future courses
4. The system shows the user the courses they are registered in for the next semester

Alternate Flow (Exceptions):

*a. At any time, System fails:

1. To support recovery, the state of the student's schedule from any step of the scenario.
2. The user logs onto Campus Central, the schedule remains in the same state as before

4a. The student has no future courses

1. System messages indicating the user has no future courses
2. Control is returned to user.

UCCS6: Viewing Past Courses

Purpose: To allow students to be able to see all courses they have completed in the past

Primary Actor: Student

Stakeholders List:

- **Program Counselor**
  - To be able to guide students in their specific program about any mandatory course which may have been missed to complete the degree.
- **System Administrator:**
  - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student has fully completed at least 1 course whilst being enrolled.
- The student is an authenticated user of Campus Central.

Postconditions:

- Students are able to view their progress thus far with all completed courses and their grades received for said courses.

Basic Flow (Main Success Scenario):

1. The user logs into CampusCentral as an authenticated student.
2. The user navigates the system to the Course Scheduler.
3. The user enters the option to view past courses
4. The system shows courses completed, marks and when they were completed.

Alternate Flow (Exceptions):

*a. At any time, System fails:
1. To support recovery, the state of the student's schedule from any step of the scenario.
2. The user logs onto Campus Central, the schedule remains in the same state as before.

4a. The user has not completed any courses whilst being fully registered as a student
1. A brief message appears stating that no courses have been completed.
2. The student is allowed to see in order every course which is required to be taken and relevant information pertaining to each course to finish the specified program.
3. Control is returned to user

UCCS7: Viewing Current Courses

Purpose: To allow students to be able to see all courses they are currently taking

Primary Actor: Student

Stakeholders List:

- **Program Counselor**
  - To be able to guide students in their specific program about any mandatory course which may have been missed to complete the degree.
- **System Administrator:**
  - Wants to be able to provide technical support for students if any problems arise during course selection.

Preconditions:

- The student has fully completed at least 1 course whilst being enrolled.
- The student is an authenticated user of Campus Central.

Postconditions:

- Students are able to view their progress thus far with all completed courses and their grades received for said courses.

Basic Flow (Main Success Scenario):

1. The user logs into CampusCentral as an authenticated student.
2. The user navigates the system to the Course Scheduler.
3. The user enters the option to view current courses
4. The system shows courses the student is currently taking during the semester

Alternate Flow (Exceptions):

*a. At any time, System fails:

1. To support recovery, the state of the student's schedule from any step of the scenario.
2. The user logs onto Campus Central, the schedule remains in the same state as before.

4a. The user has not completed any courses whilst being fully registered as a student

1. System indicates that the user is not taking any courses for the semester
2. Control is returned to user

UCCS8: View Class Lists

Purpose: To allow Professors to view a class list for a particular course.

Primary Actor: Professor

Stakeholders List:

- System Administrator:
    - Wants to be able to provide technical support for students if any problems arise during course selection.
- Department Chair / Program Counselor **:**
    - Wants to ensure that the class list for a course includes all necessary information for each student.
- Students**:**
    - Wants to ensure that their personal information is secure and protected.

Preconditions:

The Professor is an authenticated user of the CampusCentral system.
The Professor is assigned to teach the course for which they want to download the class list.

Postconditions:

The Professor has successfully downloaded the class list for the corresponding course.
The downloaded class list includes all the necessary information for each student.

Basic Flow (Main Success Scenario):

1. The Professor logs into CampusCentral.
2. The Professor navigates the system to the Course Scheduler.
3. The Professor enters the option to view class list.
4. The Professor enters the course they want to view the class list for.
5. The system displays a list of courses the Professor is assigned to teach.
6. The Professor enters the desired course.
7. The system outputs the students registered in a course, including student's name and student id

Alternate Flow (Exceptions):

3a. The Professor is not assigned to teach the selected course:
1. The system displays an error message indicating that the Professor is not assigned to teach the selected course.
2. The Professor selects a different course or contacts the department chair for assistance.
3. Control is returned to user.

4b. There are no students registered in the course:
1. The system displays an error message indicating there are no students in the class list.
2. Control is returned to user.

6a. The professor enters an incorrect course ID/code
1. The system prompts an error message and asks the user to re-enter the course
2. Control is returned to user.

UCCS9: View courses they are teaching

Primary Actor: Professor

Purpose: To allow professors to be able to view courses they are teaching in the past, and future, and to provide accessibility to summaries and class lists for courses at any given time.

Stakeholders List:

- System Administrator
    - Wants to provide technical support to professors if the courses displayed don't match their scheduled plan
- Program Counselor
    - Wants to know the courses that a professor is currently teaching, taught in the past, and going to teach in the upcoming semesters

Preconditions:

- The professor is an authenticated user of CampusCentral.

Postconditions:

- The course has been added to the professor's schedule for the corresponding term.
- The professor's course schedule is updated.

Basic Flow (Main Success Scenario)

1. The professor logs into CampusCentral.
2. The Professor navigates the system to the Course Scheduler.
3. The professor selects if they want to view future (next semester courses) or past courses that they will be teaching or have taught.

Alternate Flow (Exceptions)

*a. At any time, System fails:
1. To support recovery, the state of the professor's schedule from any step of the scenario is saved
2. The user logs onto Campus Central, the schedule remains in the same state as before

3a. If the professor does not have any scheduled courses:
1. The system will display a message that no course has been scheduled for the semester.
2. Control is returned to user.

UCCS10: Review TA Applications

Primary Actor: Professor

Purpose: To allow professors to be able to view TA Applications for the courses they are teaching next semester

Stakeholders List:

- System Administrator
    - Wants to provide technical support to professors if the courses displayed don't match their scheduled plan
- Program Counselor
    - Wants to know the who the teaching assistants are for the courses the professor is teaching in the upcoming semester

Preconditions:

- The professor is an authenticated user of CampusCentral.
- The professor's schedule has been updated for the upcoming courses for the courses they will be teaching

Postconditions:

- The Teaching assistants for the courses have been updated

Basic Flow (Main Success Scenario)

1. The Professor logs into Campus Central
2. The Professor navigates the system to the Course Scheduler.
3. The Professor enters the option to view TA applications
4. The Professor enters a course to review TA applications for
5. The Professor selects which students they would like to approve for a teaching assistant position
6. Control is returned back to the user

Alternate Flow (Exceptions)

*a. At any time, System fails:

1. To support recovery, the state of the professor's schedule from any step of the scenario is saved
2. The user logs onto Campus Central, the schedule remains in the same state as before

4a. There are no Teaching Assistant Applications

1. The system prints out a message indicating that there are no applications for the selected course
2. Control is returned to the user

4b. The professor enters an incorrect course id/code

1. The system prints out a message indicating that the course entered is invalid
2. System prompts to ask the professor to re-enter the course
3. Control is returned to the user