GRADS
Design Document

Authors:
Daniel Belling
David Cavalari
Christine Hallstrom
Guobao Sun

Group: 15

This page intentionally left blank.

Document Revision History

Date	Version	Description	Author
11/10/2014	0.0	Initial draft	Daniel Belling, David Cavalari, Christine Hallstrom, Guobao Sun

This page intentionally left blank.

Contents

1 Introduction

1.1 Purpose

This design document outlines the design for the GRADS system designed for CSCI 5801.

1.2 System Overview

The Graduation Requirement Assessment Data System will serve as an interactive database software system which grants exclusive permissions to students and program coordinators to compare and contrast student achievements (as logged on their student records) with specifically determined graduation requirements as dictated by their graduate path (Ph.D, M.S. Plan A, B or C) of choice.

- GRADS will serve as a simplified search and compile system for student records. Students and coordinators will be allowed to search the database for student records, while comparing student records with graduation requirements. Upon retrieving a students record, he or she may view a summary of their respective achievements and other characteristics including their technical GPA, identification number, committee members and other pertinent milestone dates. Students will also be able to generate 'What-If' reports to track their progress towards other graduate paths within the Computer Science department.
- Coordinators may find this system to be a convenient way to mass-produce records of all of their managed students for further processing such as, printing transcripts, amending records after important thesis or project dates, or tracking progress of all students concurrently enrolled in a graduate program.

1.3 Design Objectives

The primary objectives of this design document are defined as follows:

- To provide enough information for a novice Java programmer to implement the GRADS system.
- To provide enough detail to develop a comprehensive test suite before the code for the system has been developed.

1.4 References

[1] GRADS Requirements Document

1.5 Definitions, Acronyms, and Abbreviations

There are no new terms introduced in this document. Please refer to the GRADS Requirement Document referenced in Section 1.4 for definitions of terms.

2 Design Overview

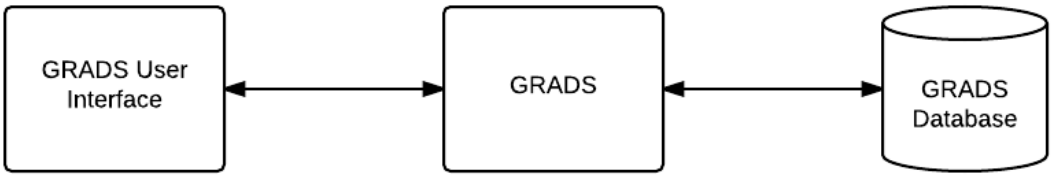
2.1 Introduction

The design of this system is a hybrid design with the main processing defined through functional decomposition and the abstract data types representing the student data and student record data designed in an object-oriented fashion. Microsoft Visio was utilized to make the diagrams included in this document.

2.2 Environment Overview

The GRADS system will interact with two other systems:

1. The GRADS User Interface, which is responsible for retrieving user requests and sending them to GRADS, and presenting the responses from GRADS to the user.
2. The GRADS Database, which GRADS will pull user and student record data from in order to respond to requests from the GRADS User Interface.



2.3 System Architecture

The software will accept user requests sent to GRADS by the GRADS User Interface, and then will extract the requested data from the GRADS Database and send the appropriate response back to the GRADS User Interface.

2.4 Constraints and Assumptions

1. This system will be written in Java.

3 Interfaces and Data Stores

3.1 System Interfaces

3.1.1 GRADS User Interface

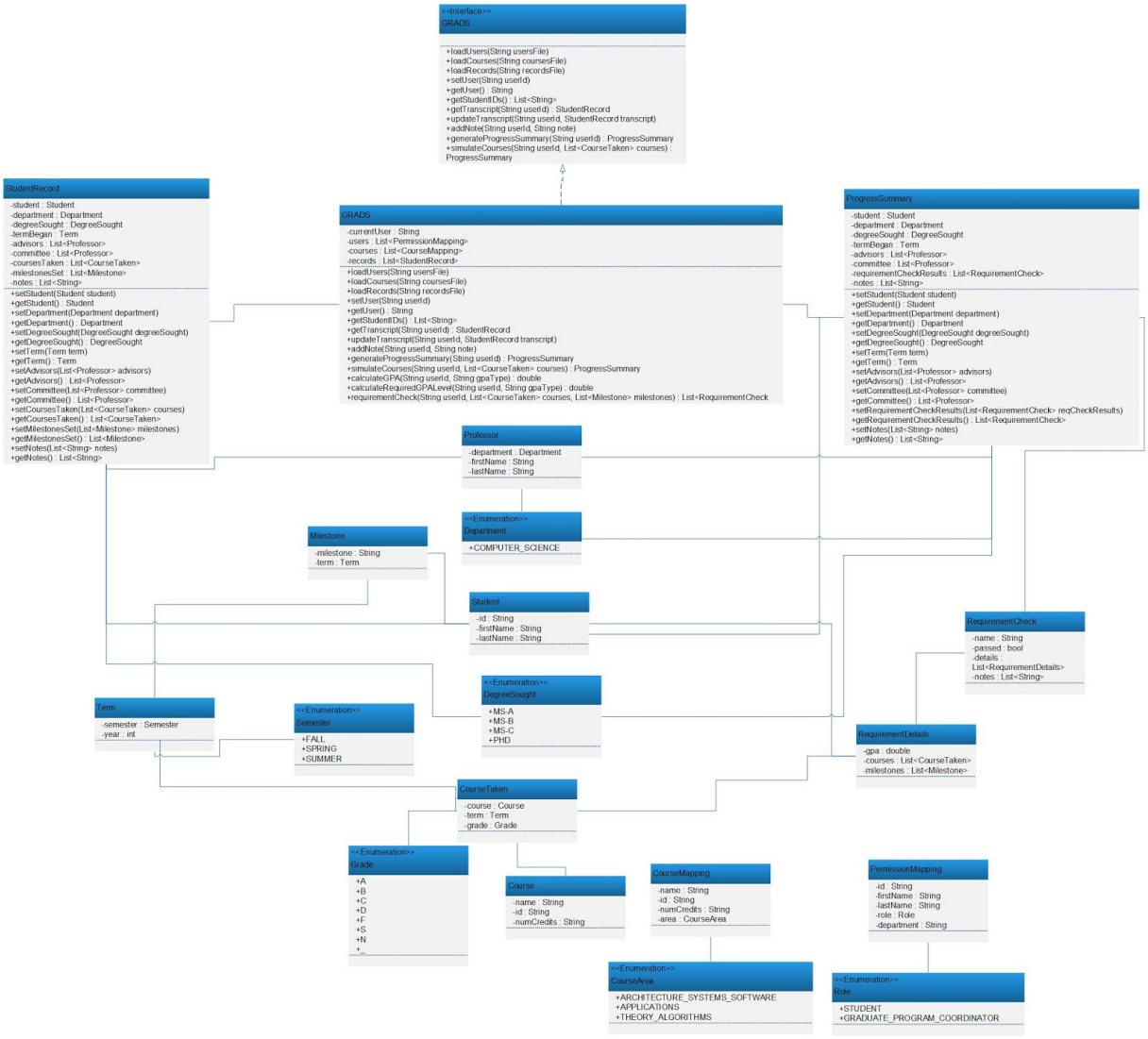
This interface is used to retrieve user requests and send them to GRADS, and interpret the responses returned by GRADS and present them to the user.

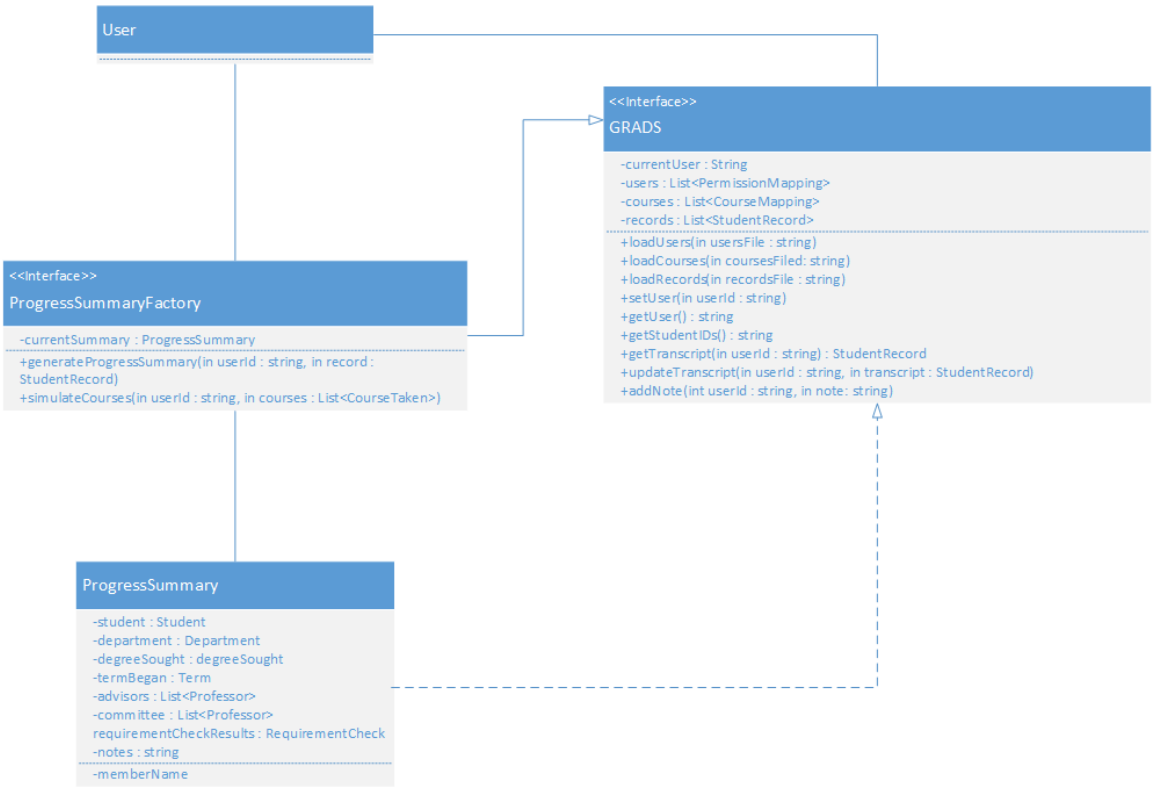
3.1.2 GRADS Database Interface

This interface is used to store user data required by GRADS.

4 Structural Design

4.1 Class Diagram





4.2 Classes in the GRADS System

4.2.1 GRADS

- Purpose: This class receives user requests from the GRADS User Interface, thus acting as a point of entry to the system, and contains the primary logic for the behavior of the system

4.2.1.1 Attribute Descriptions

1. Attribute: currentUser
   Type: String
   Description: The current user that is requesting information from GRADS.

2. Attribute: users
   Type: List<PermissionMapping>
   Description: The permission mappings that represent the users of the system. Populated by calling loadUsers(String usersFile).
   Constraints: Must not be null to call any non-setup methods in the GRADS class.

3. Attribute: courses
   Type: List<CourseMapping>
   Description: A list of all courses supported by the GRADS system. Populated by calling loadCourses(String coursesFile).
   Constraints: Must not be null to call any non-setup methods in the GRADS class.

4. Attribute: records
   Type: List<StudentRecord>
   Description: A list of student records in the GRADS system. Populated by calling loadRecords(String recordsFile).

Constraints: Must not be null to call any non-setup methods in the GRADS class.

4.2.1.2 Method Descriptions

1. Method: loadUsers
Return Type: Void
Parameters: *String usersFile*
Return value: *None*
Post Condition: The *users* field *contains a PermissionMapping for each record read from the file.*
Attributes read/used: *users*
Methods called: *None*

Processing logic:
GRADS attempts to open the users file. If file is not found, GRADS returns an error. Otherwise, GRADS opens the file and seeks records containing id, firstName, lastName, role, and department. If GRADS encounters a record in which it cannot locate all five data items, it throws an exception. Otherwise, it loads each set of five data items into a new instance of PermissionMapping and adds it to the list *users*.

Test case 1: Try loading a user file. Expected result: user data will be loaded into *users* list.

Test case 2: Try loading an improperly-formatted user file. GRADS should throw an exception.

Test case 3: Try loading a nonexistent file. GRADS should return an error.

2. Method: loadCourses
Return Type: Void
Parameters: String coursesFile
Return value: *None*
Post-condition: The *courses* field contains an instance of CourseMapping for each course in the file.
Attributes read/used: *courses*
Methods called: *None*

Processing logic:
GRADS attempts to open the courses file. If file is not found, GRADS returns an error. Otherwise, GRADS opens the file and seeks records containing name, id, numCredits, and courseArea. If GRADS encounters a record in which it cannot locate all four data items, it throws an exception. Otherwise, it loads each set of four data items into a new instance of CourseMapping and adds it to the list *courses*.

Test case 1: Try loading a course file. Expected result: course data will be loaded into *courses* list.

Test case 2: Try loading an improperly-formatted courses file. GRADS should throw an exception.

Test case 3: Try loading a nonexistent file. GRADS should return an error.

3. Method: loadRecords
Return Type: *Void*
Parameters: String *recordsFile*
Return value: *None*

Post-condition: The *records* field contains an instance of RecordMapping for each record in the file.
Attributes read/used: *records*
Methods called: *None*

Processing logic:
GRADS attempts to open the records file. If file is not found, GRADS returns an error. Otherwise, GRADS opens the file and seeks records for student, department, degreeSought, termBegan, advisors, committee, coursesTaken, milestonesSet and notes.. If GRADS encounters a record in which it cannot locate any of the student, department, degreeSought or termBegan fields, it throws an exception. Otherwise it loads the student records into a new instance of RecordMapping and adds it to the list *records.*

Test case 1: Try loading a record file. Expected result: record data will be loaded into *records* list.

Test case 2: Try loading an improperly-formatted records file. GRADS should throw an exception.

Test case 3: Try loading a nonexistent file. GRADS should return an error.

4. Method: setUser
Return Type: *Void*
Parameters: String *userId*
Return value: *None*
Post-condition: The *currentUser* field is set to the *userId* input.
Attributes read/used: *currentUser*
Methods called: *None*

Processing logic:
GRADS attempts to set the current user string to the x500 of the user. If no string, or an unsuitable data type (other than a string) is provided, GRADS returns an error. Otherwise GRADS sets the currentUser field.

Test case 1: Try reading the userId string. Expected result: currentUser field will be set to userId.

Test case 2: Try reading no input. GRADS should return an error.

Test case 3: Try reading a userId which is not a string. GRADS should return an error.

5. Method: getUser
Return Type: String
Parameters: None
Return value: *currentUser*
Post-condition: The currentUser string is returned to the caller.
Attributes read/used: *currentUser*

Processing logic:
GRADS attempts to retrieve the current user string. If currentUser string is unset, GRADS returns the empty string “”.

Test case 1: Try retrieving the currentUser string. Expected result: currentUser string is returned to the caller.

Test case 2: Try retrieving the currentUser string before it has been set with setUser. GRADS should return the empty string.

6. Method: getStudentIDs
Return Type: List<String>
Parameters: None
Return value: *List<String> studentIDs*
Pre-condition: The *users* field must be populated with *loadUsers.*
Post-condition:
Attributes read/used: *users*
Methods called:

Processing logic:
GRADS attempts to generate a list of ID's of students currently in the *users* field. If the *users* field is currently unset or cannot be retrieved, GRADS throws an exception. GRADS scans the *users* list for users with an ID field which is currently set (non-null) and populates the return list with these ID values. If GRADS encounters a student in the *users* list who currently has an unset ID, GRADS throws an exception.

Test case 1: Try retrieving the IDs of students currently in the GRADS system. Expected result: GRADS returns a list of IDs to the caller.

Test case 2: Try retrieving the IDs from a users list which is currently unset. GRADS throws an exception.

Test case 3: Try retrieving the IDs from a users list which is currently empty. GRADS throws an exception.

Test case 4: Try retrieving the IDs from a users list which contains a student with an unset ID. GRADS throws an exception.

Test case 5: Try retrieving the IDs from a users list which is populated with students, but none of them have set ID fields. GRADS throws an exception.

7. Method: getTranscript
Return Type: StudentRecord
Parameters: *String userId*
Return value: *StudentRecord transcript*
Pre-condition: *records* field has been set with *loadRecords* and *users* has been populated with *loadUsers*
Post-condition: *transcript* is returned to the caller.
Attributes read/used: *records, users*
Methods called:

Processing logic:
GRADS attempts to retrieve a StudentRecord belonging to userId. If the *records* field is unset or there is no such StudentRecord in *records* which belongs to *userId*, GRADS throws an exception. Otherwise, *transcript* is returned.

Test case 1: Try retrieving a StudentRecord which is a member of *records* and whose owner is a member of *users.* Expected result: GRADS returns *transcript.*

Test case 2: Try retrieving a StudentRecord belonging to userId which is a member of *records*, but whose owner is not a member of *users*. GRADS throws an exception.

Test case 3: Try retrieving a StudentRecord belonging to userId which is not found, but whose owner is a member of *users*. GRADS throws an exception.

8. Method: updateTranscript
Return Type: Void
Parameters: *String userId, StudentRecord transcript*
Return value: *None*
Pre-condition: *userId* must be a member of *users.*
Post-condition:
Attributes read/used: *users, records*
Methods called: *getTranscript*

Processing logic:
GRADS attempts to update the StudentRecord on file belonging to *userId*. GRADS fetches the StudentRecord currently stored for userId and compares it to *transcript*. If any discrepancies are found, fields in StudentRecord are overwritten by those found in *transcript*. If there is insufficient space in the GRADS database for *transcript, transcript* doest not belong to *userId*, or if the *currentUser* is a non-GPC GRADS throws an exception. Otherwise, StudentRecord is updated.

Test case 1: Try updating the StudentRecord belonging to *userId* who is a member of *users,* and who has a StudentRecord which is a member of *records.* Expected result: *userId's* StudentRecord is replaced by transcript.

Test case 2: Try updating the StudentRecord belonging to *userId* who is a member of *users*, but there is no StudentRecord in *records* belonging to *userId*, GRADS throws an exception.

Test case 3: Try updating the StudentRecord belonging to *userId* who is not a member of *users*, but there exists a StudentRecord in *records* whose owner is *userId*, GRADS throws an exception.

Test case 4: Try updating a StudentRecord belonging with a *transcript* which passes a validity check, by a *currentUser* who is a non-GPC. GRADS throws an exception.

Test case 5: Try updating a StudentRecord with a full *records* field. GRADS throws an exception.

9. Method: addNote
Return Type: Void
Parameters: *String userId, String note*
Return value: *None*
Pre-condition: *records* list must be populated with *loadRecords*
Post-condition: StudentRecord belonging to userId has note appended to notes field.
Attributes read/used: *records*
Methods called: *getTranscript, updateTranscript*

Processing logic:
GRADS attempts to append *note* to the *notes* list in the current StudentRecord in *records* belonging to userId. If the *currentUser* is a non-GPC, or if there is insufficient space to write the note to StudentRecord, GRADS throws an exception. Otherwise, *note* is added to *notes*.

Test case 1: Try appending *note* to the *notes* field of StudentRecord with owner *userId.* Expected result: *transcript* in *records* is updated with *note* appended to *notes* list.

Test case 2: Try appending *note* to the *notes* field of StudentRecord with owner *userId* by a *currentUser* who is a non-GPC. GRADS throws an exception.

Test case 3: Try appending *note* to the *notes* field of StudentRecord with owner *userId* with a *notes* field which is currently full. GRADS throws an exception.

10. Method: generateProgressSummary
Return Type: ProgressSummary
Parameters: String userId
Return value: *ProgressSummary progressSummary*
Pre-condition: *records* field has been set with *loadRecords*, *users* has been set with *loadUsers*
Post-condition:
Attributes read/used: *records*
Methods called: *getTranscript*

Processing logic:
GRADS attempts to generate a ProgressSummary of userId's currently stored StudentRecord in records. If the ProgressSummary could not be generated by a valid StudentRecord in *records*, GRADS throws an exception. Otherwise, GRADS retrieves the transcript for *userId* with the *getTranscript* method, and uses this data to populate a unique ProgressSummary for *userId.*

Test Case 1: Call the method with a valid student userId. Expected result: A progress summary is returned.

Test Case 2: Call the method with a non-existent userId. Expecded result: GRADS throws an exception.

Test Case 3: Call the method with a userId which does not correspond to an existing StudentRecord. Expected result: GRADS throws an exception.

11. Method: simulateCourses
Return Type: ProgressSummary
Parameters: String userId, List<CourseTaken> courses
Return value: *ProgressSummary whatIfSummary*
Pre-condition: *records* field has been set with *loadRecords, users* field has been set with *loadUsers*
Post-condition:
Attributes read/used: *records*
Methods called: *generateProgressSummary, getTranscript, updateTranscript*

Processing logic:
GRADS attempts to generate a ProgressSummary which simulates the impact of courses on userId's StudentRecord. If the ProgressSummary could not be generated, or if the CourseTaken list *courses* is unset, or contains courses which are currently invalid or not members of the *allowedCourses* list reocgnized by GRADS, GRADS throws an exception.

Test case 1: Call simulateCourses method with a valid student userId corresponding to a valid student record. Expected result: GRADS returns the simulated progress report.

Test case 2: Call method with an invalid student userId (so that the ProgressSummary will fail). Expected result: GRADS throws an exception.

Test case 3: Call the method with a valid student userId, but with the CourseTaken list *courses* unset. Expected result: GRADS throws an exception.

Test case 4: Call the method with a valid student userId, but with a course number that is in the *allowedCourses* list and is no longer valid. Expected result: GRADS throws an exception.

Test case 5: Call the method with a valid student userId, but with a course number that is not on the *allowedCourses* list. Expected result: GRADS throws an exception.

12. Method: calculateGPA
Return Type: double
Parameters: String userId, String gpaType
Return value: *double gpa*
Attributes read/used: *records*

Processing logic: GRADS attempts to retrieve course info appropriate for the type of GPA being calculated, and then calculates the student's GPA for that type as follows:

- OVERALL - calculate GPA for all courses taken by the input student
- IN_PROGRAM - calculate GPA for all CSCI courses taken by the input student
- BREADTH_REQUIREMENT - calculate GPA for all courses that contribute to the input students breadth requirement

Test case 1: Call method using valid student ID for OVERALL GPA type. Verify that every course the student has taken appears in the list of courses across which the grades are averaged.

Test case 2: Call method as above for IN_PROGRAM GPA type. Verify that the list of courses being averaged contains in-program courses only.

Test case 3: Call method as above for BREADTH_REQUIREMENT GPA type. Verify that the list of courses being averaged contains only breadth requirement courses.

13. Method: calculateRequiredGPALevel
Return Type: double
Parameters: String userId, String gpaType
Return value: *double gpa*
Attributes read/used: *records*

Processing logic: GRADS retrieves the DegreeSought for the input students, and then returns the required GPA level for that type of degree, based on the GPA type as follows:

- PHD
    - BREADTH_REQUIREMENT: return 3.45
    - IN_PROGRAM: return 3.45
    - OVERALL: return 3.45
- MS_A, MS_B, MS_C
    - BREADTH_REQUIREMENT: return 3.25
    - IN_PROGRAM: return 3.25
    - OVERALL: return 3.25

Test case 1: Call the method with each type of degree. Expected result: the correct GPA types are returned.

14. Method: requirementCheck
Return Type: List<RequirementCheck>
Parameters: String userId, List<CourseTaken> courses, List<Milestone> milestones
Return value: *List<RequirementCheck> requirementCheckResults*
Attributes read/used: *records*

Processing logic: GRADS uses lists of taken courses and completed milestones for the current student userId to retrieve a list of RequirementChecks specifying which requirements the student has passed/not passed. It does this by checking the associated required courses, required GPA (where applicable), and required milestones for the student's chosen degree program, then checking the corresponding student record to see if it contains those values. If so, it generates a RequirementCheck with *passed* set to true. If a degree requirement exists for which not all of the required courses and/or milestones can be found, or for which the GPA is less than the minimum required GPA, it generates a RequirementCheck with *passed* set to false. If GRADS cannot establish any requirements for some reason (for example, if the student has not chosen a degree program), then this method will simply return an empty list.

Test case 1: Call the method with a valid student userId. Expected result: A list of RequirementChecks is returned.

Test case 2: Call the method using a student record where the student has not selected a degree program. Expected result: GRADS returns empty list.

Test case 3: Call the method using student record with no courses or milestones, but with a degree program chosen. Expected result: GRADS returns list of RequirementChecks with *passed* values set to false.

Test cas 4: Call the method using student record with a chosen degree program and all courses and milestones completed. Expected result: GRADS returns list of RequirementChecks with *passed* values set to true.

4.2.2 Class: StudentRecord

- Purpose: This class stores information regarding a student record for a particular student.

4.2.2.1 Attribute Descriptions

1. Attribute: student
   Type: Student
   Description: An instance of the Student class containing relevant information about a student.

2. Attribute: department
   Type: Department
   Description: The department which the student currently belongs to.

3. Attribute: degreeSought
   Type: DegreeSought
   Description: The degree path currently being pursued by the student.

4. Attribute: termBegan
   Type: Term
   Description: The term during which a student began taking graduate courses at the university.

5. Attribute: advisors
   Type: List<Professor>
   Description: A list of Professors currently acting as advisors for the student.

6. Attribute: committee
   Type: List<Professor>
   Description: A list of Professors currently on the student's committee.

7. Attribute: coursesTaken
   Type: List<CourseTaken>
   Description: A list of the courses taken by the student.

8. Attribute: milestonesSet
   Type: List<Milestone>
   Description: A list of milestones achieved by the student.

9. Attribute: notes
   Type: List<String>
   Description: A list of notes appended to the student's record.

4.2.2.2 Method Descriptions

1. Method: setStudent
Return Type: void
Parameters: Student student

Processing logic:
Set the instance of Student contained in the Student class to the input value.

2. Method: getStudent
Return Type: Student
Parameters:

Processing logic:
Retrieve the instance of Student contained in the Student class.

3. Method: setDepartment
Return Type: void
Parameters: Department department

Processing logic:
Set the instance of Department contained in the Student class to the input value.

4. Method: getDepartment
Return Type: Department
Parameters:

Processing logic:
Retrieve the instance of Department contained in the Student class.

5. Method: setDegreeSought
Return Type: void
Parameters: DegreeSought degreeSought

Processing logic:
Set the instance of DegreeSought contained in the Student class to the input value.

6. Method: getDegreeSought
Return Type: DegreeSought
Parameters:

Processing logic:
Retrieve the instance of DegreeSought contained in the Student class.

7. Method: setTerm
Return Type: void
Parameters: Term term

Processing logic:
Set the instance of Term *contained in the Student class to the input value.*

8. Method: getTerm
Return Type: Term
Parameters:

Processing logic:
Retrieve the instance of Term *contained in the Student class.*

9. Method: setAdvisors
Return Type: void
Parameters: List<Professor> advisors

Processing logic:
Set the list of advisors contained in the Student class to the input value.

10. Method: getAdvisors
Return Type: List<Professor>
Parameters:

Processing logic:
Retrieve the list of advisors contained in the Student class.

11. Method: setCommittee
Return Type: void
Parameters: List<Professor> committee

Processing logic:
Set the list of committee members contained in the Student class to the input value.

12. Method: getCommittee
Return Type: List<Professor>
Parameters:

Processing logic:
Retrieve the list of committee members contained in the Student class.

13. Method: setCoursesTaken
Return Type: void

Parameters: List<CourseTaken> courses

Processing logic:
Set the list of courses taken contained in the Student class to the input value.

14. Method: getCoursesTaken
Return Type: List<CourseTaken>
Parameters:

Processing logic:
Retrieve the list of courses taken contained in the Student class.

15. Method: setMilestonesSet
Return Type: void
Parameters: List<Milestone> milestones

Processing logic:
Set the list of milestones contained in the Student class to the input value.

16. Method: getMilestonesSet
Return Type: List<Milestone>
Parameters:

Processing logic:
Retrieve the list of milestones contained in the Student class.

17. Method: setNotes
Return Type: void
Parameters: List<String> notes

Processing logic:
Set the list of notes contained in the Student class to the input value.

18. Method: getNotes
Return Type: List<String>
Parameters:

Processing logic:
Retrieve the list of notes contained in the Student class.

4.2.4 Class: ProgressSummary

- Purpose: This class stores information relating to a particular student's progress summary.

4.2.3.1 Attribute Descriptions

1. Attribute: student
   Type: Student
   Description: An instance of the Student class containing relevant information about a student.

2. Attribute: department
   Type: Department
   Description: The department which the student currently belongs to.

3. Attribute: degreeSought

Type: DegreeSought
Description: The degree path currently being pursued by the student.

4. Attribute: termBegan
   Type: Term
   Description: The term during which a student began taking graduate courses at the university.

5. Attribute: advisors
   Type: List<Professor>
   Description: A list of Professors currently acting as advisors for the student.

6. Attribute: committee
   Type: List<Professor>
   Description: A list of Professors currently on the student's committee.

7. Attribute: requirementCheckResults
   Type: List<RequirementCheck>
   Description: A list of RequirementCheck instances, containing information related to the student's requirements for graduation.

8. Attribute: notes
   Type: List<String>
   Description: A list of notes appended to the student's record.

4.2.3.2 Method Descriptions

19. Method: setStudent
Return Type: void
Parameters: Student student

Processing logic:
Set the instance of Student contained in the ProgessSummary class to the input value.

20. Method: getStudent
Return Type: Student
Parameters:

Processing logic:
Retrieve the instance of Student contained in the ProgessSummary class.

21. Method: setDepartment
Return Type: void
Parameters: Department department

Processing logic:
Set the instance of Department contained in the ProgessSummary class to the input value.

22. Method: getDepartment
Return Type: Department
Parameters:

Processing logic:
Retrieve the instance of Department contained in the ProgessSummary class.

23. Method: setDegreeSought
Return Type: void
Parameters: DegreeSought degreeSought

Processing logic:
Set the instance of DegreeSought contained in the ProgessSummary class to the input value.

24. Method: getDegreeSought
Return Type: DegreeSought
Parameters:

Processing logic:
Retrieve the instance of DegreeSought contained in the ProgessSummary class.

25. Method: setTerm
Return Type: void
Parameters: Term term

Processing logic:
Set the instance of Term *contained in the ProgessSummary class to the input value.*

26. Method: getTerm
Return Type: Term
Parameters:

Processing logic:
Retrieve the instance of Term *contained in the ProgessSummary class.*

27. Method: setAdvisors
Return Type: void
Parameters: List<Professor> advisors

Processing logic:
Set the list of advisors contained in the ProgessSummary class to the input value.

28. Method: getAdvisors
Return Type: List<Professor>
Parameters:

Processing logic:
Retrieve the list of advisors contained in the ProgessSummary class.

29. Method: setCommittee
Return Type: void
Parameters: List<Professor> committee

Processing logic:
Set the list of committee members contained in the ProgessSummary class to the input value.

30. Method: getCommittee
Return Type: List<Professor>
Parameters:

Processing logic:
Retrieve the list of committee members contained in the ProgessSummary class.

31. Method: setRequirementCheckResults
Return Type: void
Parameters: List<RequirementCheck> reqCheckResults

Processing logic:
Set the list of requirement check results contained in the ProgessSummary class to the input value.

32. Method: getRequirementCheckResults
Return Type: List<RequirementCheck>
Parameters:

Processing logic:
Retrieve the list of requirement check results contained in the ProgessSummary class.

33. Method: setNotes
Return Type: void
Parameters: List<String> notes

Processing logic:
Set the list of notes contained in the ProgessSummary class to the input value.

34. Method: getNotes
Return Type: List<String>
Parameters:

Processing logic:
Retrieve the list of notes contained in the ProgessSummary class.

4.2.4 Class: Professor

- Purpose: A simple data structure to define a professor

4.2.4.1 Attribute Descriptions

1. Attribute: department
   Type: Department

2. Attribute: firstName
   Type: String

3. Attribute: lastName
   Type: String

4.2.4.2 Method Descriptions

NONE

4.2.5 Class: Milestone

- Purpose: Simple data type to record a completed milestone

4.2.5.1 Attribute Descriptions

1. Attribute: milestone
   Type: String
   Description: Describes the milestone completed (e.g. “THESIS_DEFENSE_PASSED”).

2. Attribute: term
   Type: Term
   Description: Reflects the term in which the milestone was completed

4.2.6 Class: Student

- Purpose: A simple data structure to record an individual student

4.2.6.1 Attribute Descriptions

1. Attribute: id
   Type: String

2. Attribute: firstname
   Type: String

3. Attribute: lastname
   Type: String

4.2.7 Class: CourseTaken

- Purpose: Represents a course that has been completed by a specific student.

4.2.7.1 Attribute Descriptions

1. Attribute: course
   Type: CourseMapping

2. Attribute: term
   Type: Term

3. Attribute: grade
   Type: Grade

4.2.8 Class: CourseMapping

- Purpose: This class is used to store information about a course offered in the catalog

4.2.8.1 Attribute Descriptions

1. Attribute: name
   Type: String

2. Attribute: id
   Type: String
   Description: The course id. Four lowercase letters followed by four numbers.

3. Attribute: numCredits
   Type: String
   Description: Can be either a single number or a range (e.g. “1-3”)

4. Attribute: area

Type: CourseArea
Description: CourseArea is an enum which can be ARCHITECTURE_SYSTEMS_SOFTWARE, APPLICATIONS, or THEORY_ALGORITHMS

4.2.9 Class: PermissionMapping

- Purpose: Describes permissions for a given user

4.2.9.1 Attribute Descriptions

1. Attribute: id
   Type: String
   Description: The userID of a particular user. A string of letters and numbers.

2. Attribute: firstName
   Type: String

3. Attribute: lastName
   Type: String

4. Attribute: role
   Type: Role
   Description: Role is an enum which can be either GRADUATE_PROGRAM_COORDINATOR or STUDENT

4.2.10 Class: RequirementCheck

- Purpose: Specifies a specific graduation requirement as well as whether it is completed

4.2.9.1 Attribute Descriptions

1. Attribute: name
   Type: String
   Description: Must be all uppercase with no spaces. Underscore characters are allowed (e.g. “BREADTH_REQUIREMENT”).

2. Attribute: passed
   Type: Boolean
   Description: Specifies whether requirement has been passed

3. Attribute: details
   Type: List<RequirementDetails>
   Description: Details on the gpa, courses, and milestones the student achieved to complete the requirement

4. Attribute: notes
   Type: List<string>
   Description: Miscellaneous notes associated with the requirement

4.2.11 Class: RequirementDetails

- Purpose: Describes specific gpa, course, and milestone requirements

4.2.11.1 Attribute Descriptions

1. Attribute: gpa
   Type: double
   Description: A numeric value denoting a student’s GPA for a specific course.

Constraints: Must be between 0.0 and 4.0

2. Attribute: courses
   Type: List<CoursesTaken>
   Description: The courses that are being applied to the current milestone requirement

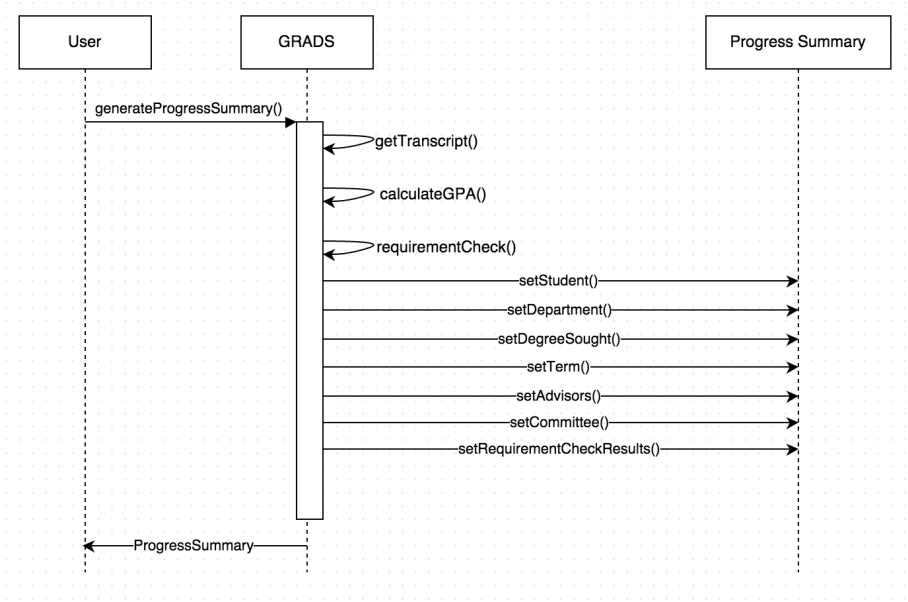
3. Attribute: milestones
   Type: List<Milestone>
   Description: List of completed milestones being applied to the current requirement

5 Dynamic Model

5.1 Scenarios

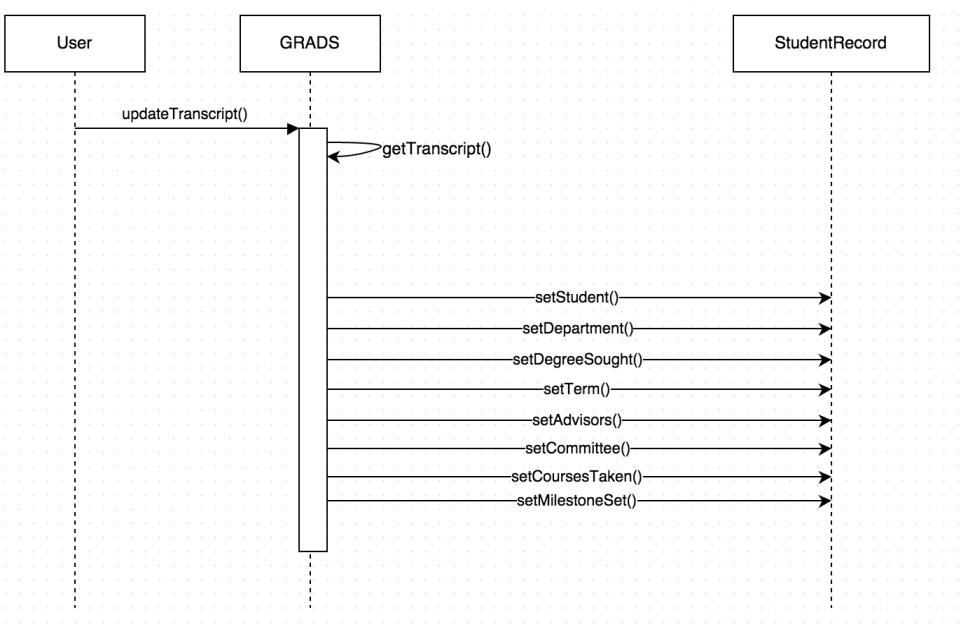
5.1.1

- Scenario Name: Show Degree Progress
- Scenario Description: This scenario shows the interaction process when a user requests the GRADS system to show the degree progress. When user requests generateProgressSummary() function, it will first pull the user's transcript. Then it will calculate the corresponding GPA based on the transcript and check milestones for the degree requirements. At last, the system will initialize a Progress Summary object and set the corresponding attributes to their values, based on the information it fetched and its own calculation based on the transcript. The user will be returned an object of class Progress Summary.
- Sequence Diagram:



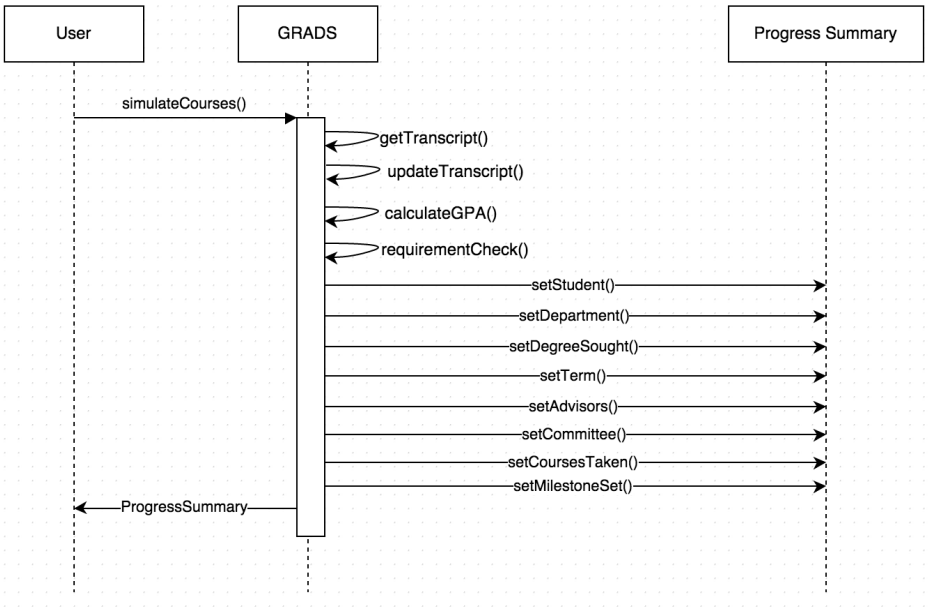
5.1.2

- Scenario Name: Amend Record
- Scenario Description: This scenario shows the interaction process when a user requests the GRADS system to modify records for a given student. When user requests updateTranscript() function, it will first pull the transcript of the given ID. Then it will take user's input to update the object of StudentRecord class. At last, the system will set those modified attributes to new values.
- Sequence Diagram:



5.1.3

- Scenario Name: Check a course's impact
- Scenario Description: This scenario shows the interaction process when a user requests the GRADS system to show the impact after assuming one certain course is added into his/her transcript. When user requests simulateCourses() function, it will first pull the transcript of the given ID. Then it will add the given course. At last, the system will show the new degree progress (with the given course included).
- Sequence Diagram:

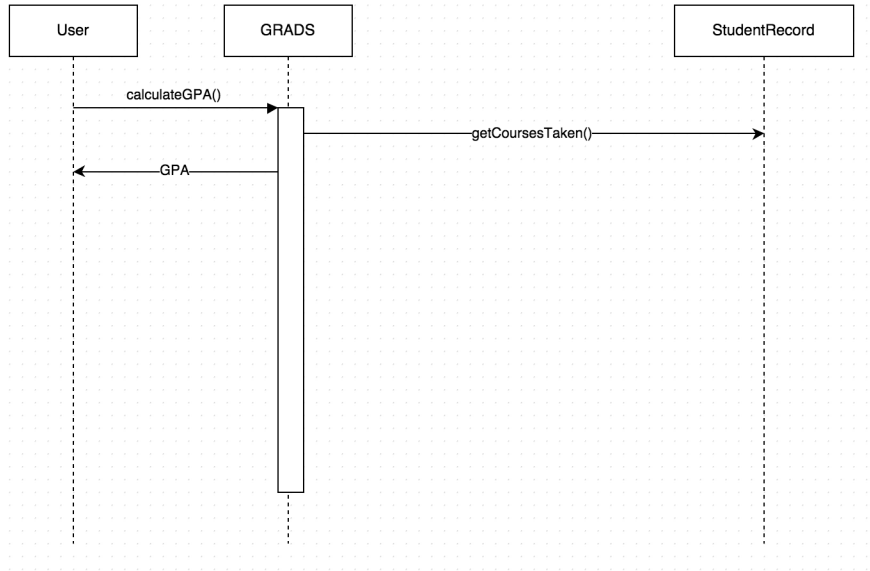


5.1.4

- Scenario Name: Show GPA
- Scenario Description: This scenario shows the interaction process when a user requests the GRADS system to show a student's GPA. When user requests calculateGPA() function, it

will first pull the transcript of the given ID. Then it will calculate the GPA and return to user.

- Sequence Diagram:



5.1.4

- Scenario Name: Show Transcript
- Scenario Description: This scenario shows the interaction process when a user requests the GRADS system to show a student's transcript. When user requests getTranscript() function, it will pull the transcript of the given ID. Then it return it to user.
- Sequence Diagram: