<Course Assignment and Instructor Preference System (CAIPS)>
Design Document for the Business Layer Module

Authors:
<Cole Medin, Adam Comerford, Tian Liu, Zeyu Gao>

Group: 4

Document Revision History

Date	Version	Description	Author
3/27/20	1.0	Initial draft	Cole Medin
4/5/20	1.1	Updated sections 1, 2, and 3	Cole Medin, Adam Comerford, Tian Liu, Zeyu Gao
4/12/20	1.2	Updated sequence diagrams	Cole Medin, Adam Comerford, Tian Liu, Zeyu Gao
4/27/20	1.3	Adjusted and simplified sequence diagrams/use cases. Added research group attributes/methods to Faculty & Administrator classes.	Adam Comerford, Cole Medin, Tian Liu, Zeyu Gao

This page intentionally left blank.

Contents

1 Introduction

1.1 Purpose

This document outlines the object-oriented design of the business layer module (BLM) of Course Assignment and Instructor Preference System (CAIPS). The intent for this document is for system designers of CAIPS to either be able to use this document as the sole resource in developing the BLM or use this document to understand how the BLM is designed to interact with the other modules in the overall system such as the GUI.

1.2 System Overview

The *Course Assignment and Instructor Preferences System* (CAIPS - version 2.0) will be used to replace currently used university systems to manage courses, instructor assignments, and data reporting. CAIPS will be used in conjunction with other services such as data storage that will interface together through manual usage of the information that other services and CAIPS provide for each other. For example, a user who wants to translate data from CAIPS to a database can view information on CAIPS and then manually transfer it over to the database. CAIPS in conjunction with other university services together will allow the university to completely manage all past, present, and determined future information on courses and instructors.

CAIPS is designed to allow for department faculty members to be able to submit their course preferences on an online interface that can then be used by a variety of university administrators to manage and create courses and then assign professors to them based on the submitted preferences. The BLM for CAIPS will allow for the system to be able to process all of these requests made by department faculty members and administrators. It will interface with the CAIPS information database and also interface with the user interface to provide all of the functionality in the system.

1.3 Design Objectives

The CAIPS BLM shall process requests from the GUI to allow for:

- Instructors to submit course preference lists.
- Courses to be viewed, added, removed, and updated in the system.
- Courses to be assigned instructors.
- The system to display course, user, and preference list information to be transferred to data services.
- To provide interfacing for a central storage location for users, course, and preference list information entered into the system.

The CAIPS BLM system will be designed to provide the functionality and interfacing necessary for all instructors in the university to submit their course preferences, and for each department to be able to assign their department faculty to each course being taught in each semester. The goal is for this system to make all course information exist in a single interface that can be accessed by all faculty and administrators through a GUI, and have the flexibility for changing any pertinent course information. What course information is pertinent is defined in the requirements section of this document. The BLM will always interface with the data storage in CAIPS to be able to have persisting information on users, courses, and preference lists.

1.4 References

This design document is based on the CAIPS requirements specification document (version 2.0) which details the entirety of the CAIPS system requirements.

Tests from the CAIPS Test Cases document (version 1.0) are referred to by their numbering in that document.

1.5 Definitions, Acronyms, and Abbreviations

CAIPS - Course Assignment and Instructor Preference System

GUI - Graphical User Interface

BLM - Business Layer Module

OO - Object Oriented

Persist - store information in a database

Retrieve - gets information from the database

OCM - The course managers at the University

2 Design Overview

2.1 Introduction

CAIPS is designed in an object-oriented manner with a layered architecture which consists of three layers:

- GUI - Graphical User Interface, the layer controlling user interaction.
- BLM - Business Layer Module which provides the logic of the system. It takes requests from the GUI, retrieves and persists data from the Infrastructure database.
- Infrastructure - The layer controlling the system's database. This will be purposefully left abstracted as to keep the BLM independent of the specific database software.

The scope of this design will be only within BLM. The GUI and Infrastructure layers will be intentionally abstracted and any interactions between the layers will be handled through interface Facades.

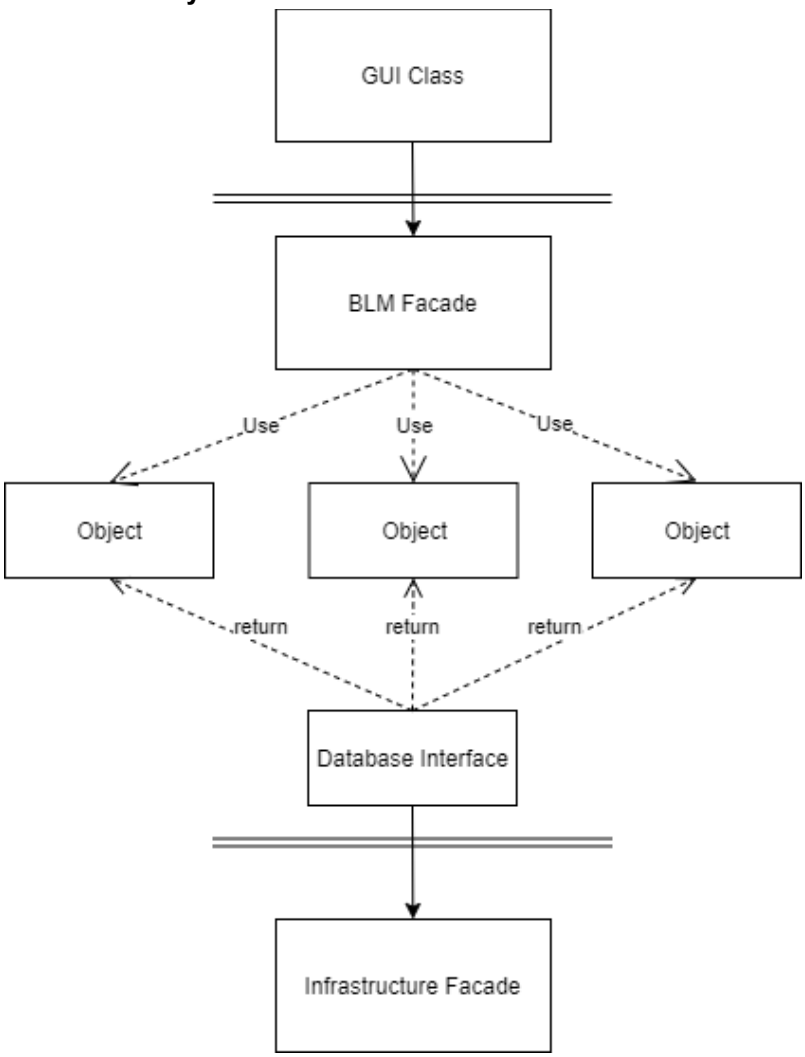
2.2 Environment Overview

This system will be operated under the environment of a higher-education institution. The primary users of this system shall be the institution's administrators and faculty. Due to varying sizes of facilities, the system should be able to handle large quantities of simultaneous user requests. The system must connect to the Institution's network and external systems such as e-mail.

The environment of the BLM, the module we will be developing, resides between the GUI and infrastructure layers. Therefore, we will be handling user requests from the GUI input and obtaining data from the infrastructure layer as needed. As such, the details of how these layers work are outside the environment of our system and will be abstracted.

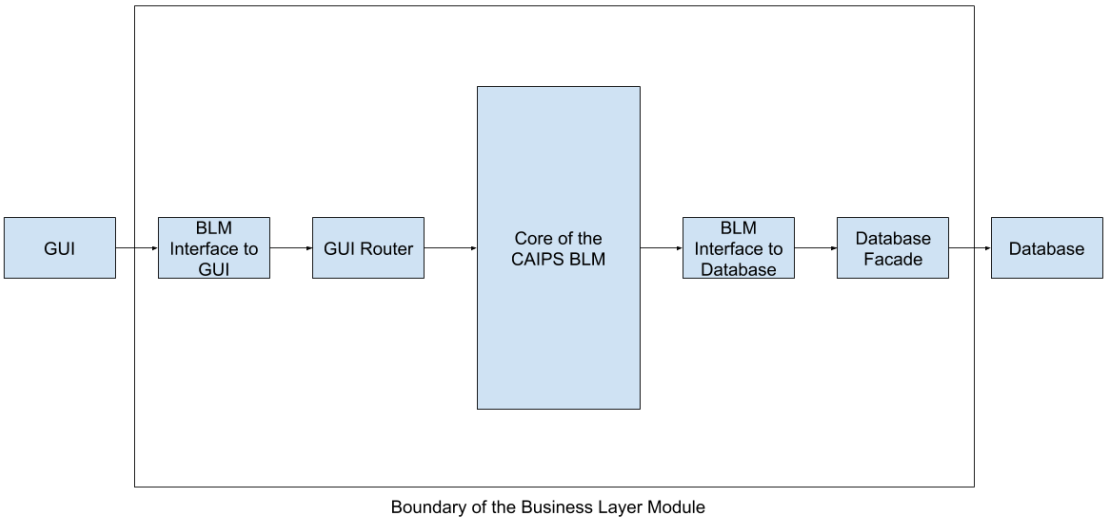
2.3 System Architecture

2.3.1 Architecture layout



The GUI sits on top of the BLM, which sits on top of the Infrastructure layer. The BLM will interface with the GUI and Infrastructure layers for vital information (user requests, database information), yet design details past the GUI and Database Facade will be abstracted. The purpose of this is to ensure that those layers may be implemented in any fashion, provided they adhere to the BLM's interface calls, and the BLM will still retain functionality.

2.3.2 Top-level system structure (more detailed than the one above): the CAIPS BLM interaction with the GUI and the database. Information flows in both directions between every major component of CAIPS.



2.4 Constraints and Assumptions

Since CAIPS is software that needs to coexist with already existing University course management software and databases, our BLM will need to be able to specifically interface with those databases and programs. We have chosen for the design of this system to mostly have University administrators manually enter information from University systems and databases into CAIPS, so we don't need to worry about how CAIPS interacts with these outside systems.

Also, we have chosen for our design to make it so that every administrator can perform most of the tasks in the system such as creating courses, updating course information, and viewing the preference lists of all of the instructors. This imposes a restriction on our design because it isn't possible for our design of CAIPS to have administrator roles imposed on the users. Instead, the users outside of the system have to enforce administrator roles within the system. We chose to do this because it allows for our system to easily handle any changes the University makes to administrator roles without having to change user permissions in the system.

3 Interfaces and Data Stores

3.1 System Interfaces

3.1.1 GUI Interface

CAIPS is separated into three systems: The GUI, the BLM, and the database. The GUI is where the users actually make the requests and have their requests validated. Once a request is generated to be processed by the system and is already validated, it is sent from the GUI to BLM interface with the GUI to process the request. Once this layer between the GUI and BLM

processes the request, it is sent one more layer deeper to the GUI request router that then decides which class of the BLM should handle the request.

3.1.2 Database Interface

All of the operations that the BLM performs either needs to retrieve information or persist information. This is done by each of the classes in the BLM that need to retrieve or persist information sending those retrieve or persist requests with the information to the database abstraction interface. This interface then communicates with the database facade class between the BLM and the database to finally communicate with the database itself. The database will then store the information given from the BLM or send information back up to the BLM through the interfacing again.

3.2 Data Stores

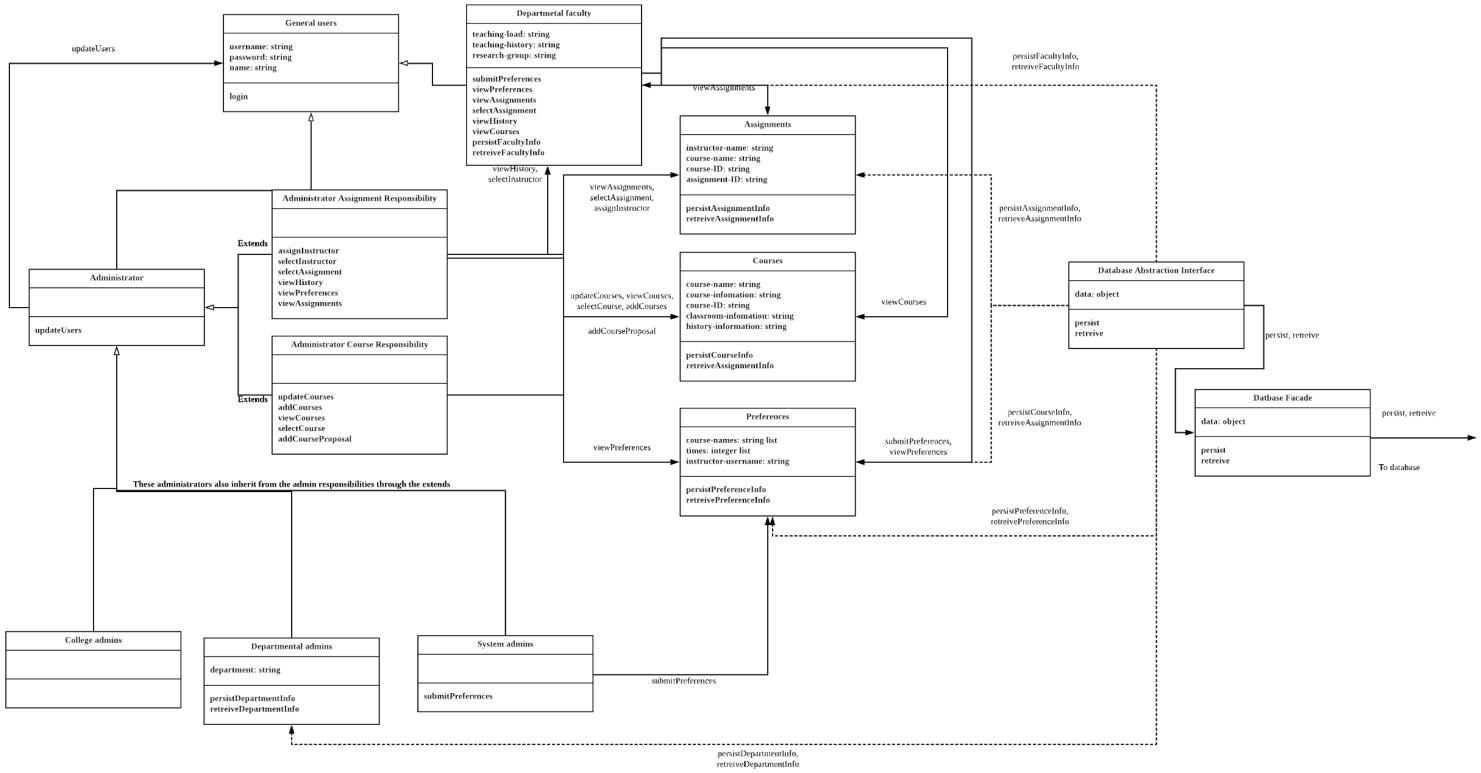
The only information that is ever kept in the BLM itself is the most recent request between the GUI and the BLM or the most recent object/object identification sent between the BLM and the database. This information is cached in the interfacing classes that connect the GUI to the BLM and the BLM to the database. The reason that these interfaces cach information is so that it doesn't need to be processed again if the system detects a failure and needs to retransmit information between interfaces of the system.

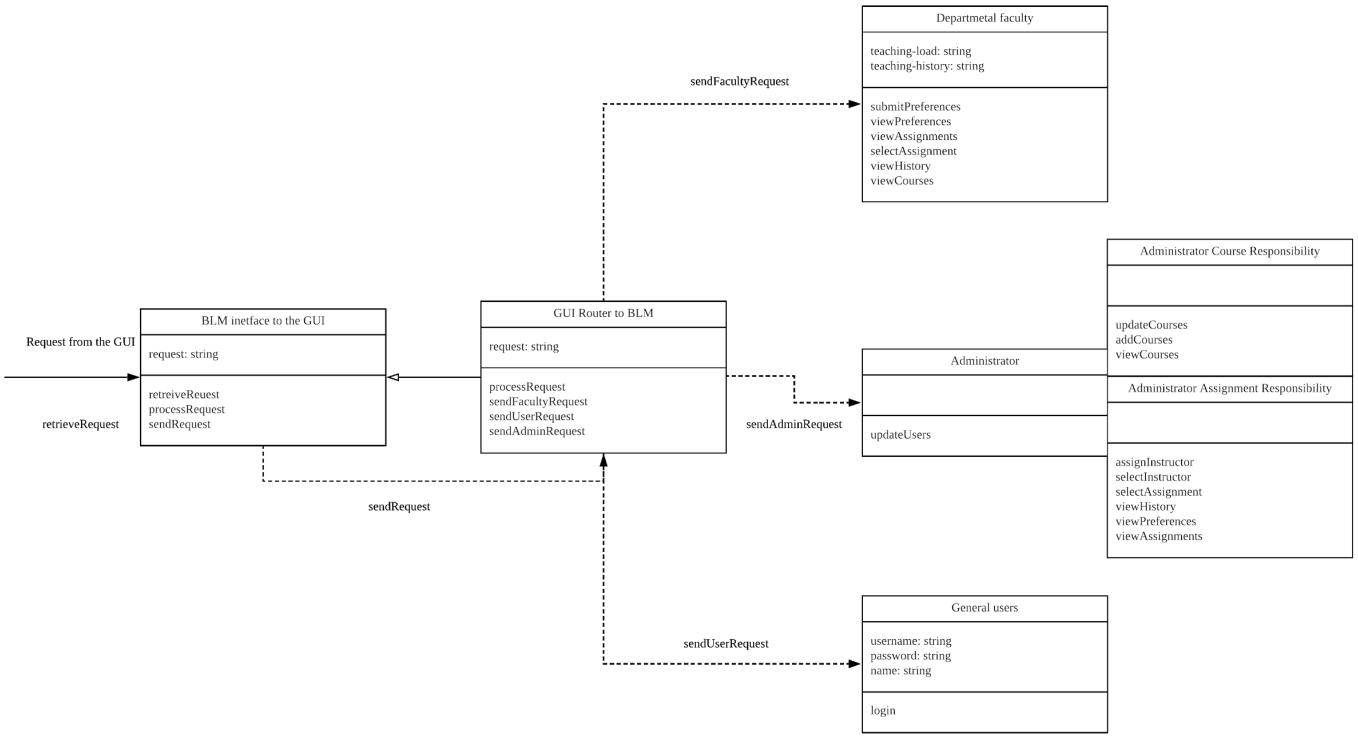
4 Structural Design

4.1 Class Diagram

The UML class diagram is split into two parts. The first part is the core of the BLM system that includes the interaction with the database through the database abstraction class on the right. The second part defines how the BLM of CAIPS interacts with the GUI to receive requests. Both the database abstraction and the GUI interfacing are intentionally made very general in order for it to work with whatever software CAIPS ends up using for its database and GUI.

Zoom in on the image while in a PDF to view the UML diagrams clearly.





The BLM is structured around the two primary users - the Administrators and Departmental Faculty. All operations go through these two user types. This makes it easily extendable because we can add any operations to the existing two user types without breaking the structure of our class interactions. By structuring it this way, the GUI router only needs to interact with these user classes rather than sending requests to each domain object.
The administrator responsibilities are split into Assignment and Course responsibilities, this is to make the administrator class modifiable as we can modify either the assignment or course responsibilities without affecting the other. This also makes it extendable as we could add another user, e.g. an OCM user, which could extend the course responsibility class if need be. It is beneficial to separate out disjoint methods of a class in this manner so that there is not one user class with a litany of methods.
We chose not to include a Research Group class itself, but rather track information from Research Groups by Faculty attributes and Administrators entering in course proposals which are sent to them from outside the system via email or other University-related external communications. If a course proposal is approved, the Administrator will manually enter in the new course via the addCourse method.

4.2 User Classes

4.2.1 **Class: General User**

- Purpose: To model the general user in the system.
- Constraints: None.
- Persistent: No (created as the user is added to the system).

4.2.1.1 Attribute Descriptions

1. Attribute: username
   Type: string
   Description: The user name of any user in the CAIPS system.
   Constraints: Shouldn't be more than 50 characters long.
2. Attribute: password
   Type: string
   Description: The password used to log into the CAIPS system for any user.
   Constraints: Shouldn't be more than 50 characters long.
3. Attribute: name
   Type: string
   Description: The full name of the user which is tied to the username.
   Constraints: Shouldn't be more than 100 characters long (not 50 because it includes the full name).

4.2.1.2 Method Descriptions

1. Method: login(string username, string password)
   The user login is a part of the GUI. The username and password are validated in the GUI and sent to the BLM so that the system can then process requests from the GUI.

4.2.2 **Class: Administrator**

- Purpose: To describe in detail the important roles and permissions of any administrator in the system.
- Constraints: None.
- Persistent: No (user is created to add it to the system).

4.2.2.1 Attribute Descriptions

1. Administrators inherit all of their attributes from the general user.

4.2.2.2 General Administrator Responsibilities Method Descriptions

1. Method: updateUsers(string username, string password, string name)
   Return Type and Value: Returns a boolean that signifies if the user update was successful.
   Parameters: The information that will be updated for the user, which could include the username, password, or the full name of the user.
   Pre-condition: The user is added to the system.
   Post-condition: The user information is updated for the specified user.
   Attributes used: user.username, user.password, user.name
   Methods called: None here, just updating the user information.
   Processing logic: Here an administrator updates user information by passing in some or all of the three parameters. The user information is then updated and the function returns true. If the user information can not be updated because the strings are too long (string lengths defined above for these attributes) then the function returns false.
   Test cases: 1.3 - Update User Information

4.2.2.3 Assignment Responsibilities Method Descriptions

1. Method: assignInstructor(string username, integer course-ID)
   Return Type and Value: Returns a boolean that signifies if the assignment was successful.
   Parameters: The username of the instructor that is being assigned and the course ID of the course that the instructor is being assigned to.
   Pre-condition: A course with the course ID being passed in exists in the system and there exists a user in the system with the username being passed in.
   Post-condition: An instructor is now assigned to a course for a semester.
   Attributes used: user.username, course.course-ID
   Methods called: user.verifyUsername(), course.verifyCourse()
   Processing logic: An administrator requests to assign an instructor to a specific course. If the username is a valid user in the system and the course ID represents a valid course in the system, then that assignment is made and the function returns true. If the username is not a valid user or the course ID is not valid, then the function returns false and the assignment is not made. Also, if the user is not an

instructor that can be assigned to a course or the instructor has already reached their teaching load, the function also returns false.
Test case: 4.1 - Assign Instructor to Course

2. Method: selectInstructor(string username)
   Return Type and Value: The instructor name associated with the username as a string.
   Parameters: The username of the instructor.
   Pre-condition: The instructor exists in the system and has a name associated with the username.
   Post-condition: The instructor is selected (used for menus to select the instructors).
   Attributes used: user.username
   Methods called: None.
   Processing logic: Here a user selects an instructor in an interface and the name is then available to the user who selected the instructor. If the username doesn't exist in the system then nothing is returned, otherwise the name associated with the username is returned.
   Test cases: 4.2, 4.3, 4.4, 4.6

3. Method: selectAssignment(string assignment-ID)
   Return Type and Value: The assignment information associated with the assignment.
   Parameters: The ID of the assignment being selected.
   Pre-condition: The assignment with the assignment ID exists in the system.
   Post-condition: The assignment is now selected to be used by the user.
   Attributes used: assignment.assignment-ID
   Methods called: None.
   Processing logic: Here a user gets assignment information associated with the assignment ID. If the assignment ID doesn't exist in the system then the function returns nothing. Otherwise, it returns the assignment information.
   Test cases: 4.1, 4.2, 4.3, 4.4

4. Method: viewHistory(string username)
   Return Type and Value: Returns as a string the teaching history of a department faculty.
   Parameters: The username of the department faculty member.
   Pre-condition: The department faculty member exists in the system and has teaching history.
   Post-condition: The administrator can now view the teaching history of the user.
   Attributes used: user.username, DepartmentFaculty.teaching-history
   Methods called: DepartmentFaculty.getTeachingHistory(), user.VerifyInstructor()
   Processing logic: An administrator retrieves the teaching history of a department faculty member by passing in the username of the department faculty member. If

the username doesn't exist in the system, the department faculty member doesn't have any teaching history, or the user is not a department faculty member, then the function returns nothing. Otherwise, the function returns the teaching history of the department faculty member.
Test cases: 4.6, 4.7

5. Method: viewPreferences(string instructor-username)
   Return Type and Value: Returns as a string the preferences of an instructor.
   Parameters: The username of the instructor.
   Pre-condition: The instructor exists in the system with the given username.
   Post-condition: The administrator can now view the instructor's preferences.
   Attributes used: user.username, preferences.instructor-name
   Methods called: user.VerifyInstructor(), DepartmentFaculty.findPreferences()
   Processing logic: An administrator retrieves the preferences of a department faculty member by passing in the username of the department faculty member. If the username doesn't exist in the system, the department faculty member doesn't have any preferences submitted yet, or the user is not a department faculty member, then the function returns nothing. Otherwise, the function returns the preferences of the department faculty member, starting with the most recent submission and ending with the oldest submission.
   Test cases: 2.3 - View Any Faculty Preference List

6. Method: viewAssignments(integer assignment-ID)
   Return Type and Value: Returns the assignment information associated with the assignment ID as a string.
   Parameters: The ID of the assignment.
   Pre-condition: The assignment exists in the system.
   Post-condition: The administrator can now view the assignment information.
   Attributes used: assignment.assignment-ID, assignment.course-ID, assignment.instructor-name
   Methods called: None.
   Processing logic: An administrator retrieves assignment information, which includes the course and instructor that are a part of the assignment. If the assignment ID doesn't exist in the system, then the function returns nothing. Otherwise, the function returns the assignment information.
   Test cases: 4.6 - View Instructor Course Assignments

4.2.2.4 Course Responsibility Method Descriptions

1. Method: updateCourse(integer course-ID, string course-information, string classroom-information, string history-information)
   Return Type and Value: Returns a boolean that signifies if the course update was successful.
   Parameters: The ID of the course and the course information that will be put into the course, including classroom information and course history information.

Pre-condition: The course is added into the system.
Post-condition: The course information is now updated.
Attributes used: course.course-ID, course.course-information, course.classroom-information, course.history-information.
Methods called: course.verifyID()
Processing logic: An administrator attempts to update course information by passing in some course information, which might include classroom information and history information. If the course ID is the ID of a course in the system and the information is valid then the information is updated and the function returns true. If the course ID doesn't exist in the system or the course information doesn't match any field in the course, then the function returns false and the course information isn't updated.
Test cases: 3.3, 3.4, 3.5

2. Method: selectCourse(integer course-ID)
   Return Type and Value: The course object associated with the course ID.
   Parameters: The ID of the course being selected.
   Pre-condition: The course exists in the system.
   Post-condition: The course is selected (used for menus to select the course).
   Attributes used: course.course-ID
   Methods called: None.
   Processing logic: Here a user selects a course in an interface and the course object is returned to be used for another function, such as viewing the course information. If the course ID is not tied to a course in the system then this function returns nothing, otherwise it returns the course object.
   Test cases: 3.5 - Display Course Information

3. Method: viewCourses(integer course-ID)
   Return Type and Value: Returns as a string the course information for all of the courses or just the course specified by course ID.
   Parameters: An optional course ID to get the information of a single course.
   Pre-condition: Courses exist in the system.
   Post-condition: A user can view all of the courses in the system or just the one specified.
   Attributes used: course.course-information, course.course-ID
   Methods called: None.
   Processing logic: An administrator retrieves the course information with a given course ID or all of the courses' information by not including that parameter. If the course ID parameter is used and doesn't exist in the system then the function returns nothing. It also returns nothing if no courses exist in the system. Otherwise, it returns the list of course information(s).
   Test cases: 3.4, 4.3

4. Method: addCourse(integer course-ID, string course-name, string course-information, string classroom-information)

Return Type and Value: Returns a boolean that signifies if the course addition was successful.
Parameters: The ID of the course to be added, the new course name, and the new information associated with the course including the classroom information.
Pre-condition: None.
Post-condition: A new course is now in the system.
Attributes used: course.courseID, course.course-name, course.course-information, course.classroom-information
Methods called: course.verifyID()
Processing logic: An administrator adds a course into the system by passing in the information for the new course. The course is added with all of the given information if a course isn't already in the system with the passed in course ID or course name. If the course ID or course name already exists then the function returns false, otherwise it returns true.
Test cases: 3.1 - Add Course to System

5. Method: addCourseProposal(string course-name, string course-information, string classroom-information)
   Return Type and Value: Returns a boolean that signifies if the course addition was successful.
   Parameters: The new course name, and the new information associated with the course including the classroom information.
   Pre-condition: None.
   Post-condition: A new course proposal is now in the system.
   Attributes used: course.course-name, course.course-information, course.classroom-information
   Methods called: None.
   Processing logic: An administrator adds a course proposal into the system by passing in the information for the new course proposal. The course is added with all of the given information if a course isn't already in the system with the passed in course name. If the course name already exists then the function returns false, otherwise it returns true. It is assumed that the admin acting as a course assignor has received the course proposal from a research group outside of CAIPS.
   Test cases: 3.1 - Add Course to System

4.2.2.5 Different Administrator Classes

1. Class: College Administrator
   - Purpose: To describe the general role and permissions of the college administrators.
   - Constraints: Cannot submit preferences.
   - Persistent: No (user is created to add it to the system).

4.2.2.5.1 Attribute Descriptions

1. All attributes are inherited from the administrator superclass.

4.2.2.5.2 Method Descriptions

1. All methods are inherited from the administrator superclass. College administrators have permission to perform any administrator tasks, but are generally assigned to doing statistics and some auditing jobs. They do this by viewing assignment information and course information, including teaching history and course history.

2. Class: Department Administrator
   - Purpose: To describe the general role and permissions of the department administrators.
   - Constraints: Cannot submit preferences.
   - Persistent: No (user is created to add it to the system).

4.2.2.5.1 Attribute Descriptions

1. Attribute: department
   Type: string
   Description: The department that the faculty member is a part of.
   Constraints: The string cannot be more than 50 characters long.

4.2.2.5.2 Method Descriptions

1. All methods are inherited from the administrator superclass. Department administrators have permission to perform any administrator tasks, but are generally assigned to determining course assignments in their department and managing classes in their department. Not all department administrators are assigned to assigning instructors to courses, but all have the permission to do so. The only methods that need to be defined here are the methods used to get the department attribute of the administrator from the database:

2. Method: persistDepartmentInfo(Object departmentAdminObject)
   Return Type and Value: Returns as a boolean whether or not the data persisting operation was successful.
   Parameters: The department admin object that contains the information that will be stored in the database.
   Pre-condition: The department admin object contains data about the faculty's teaching history or teaching load.
   Post-condition: The department admin's information is now stored in the database.
   Attributes used: departmentAdmin.department
   Methods called: None.
   Processing logic: An administrator performs some action that changes a department admin's information. This function is then automatically called to store the new information in the CAIPS database. If the operation is successful the function returns true and the administrator is

notified, otherwise it returns false and the administrator is warned that their information wasn't saved.
Test cases: None.

3. Method: retrieveDepartmentInfo(UUID departmentAdminIdent)
Return Type and Value: Returns the information associated with a department admin as a department admin object.
Parameters: The department admin identification that will be used to retrieve the department admin object.
Pre-condition: The department admin object contains data about the admin's department and exists in the database.
Post-condition: The department admin's information is now loaded from the database.
Attributes used: departmentAdmin.department
Methods called: None.
Processing logic: An administrator performs some action that needs a department admin's information. This function is then automatically called to retrieve the information in the CAIPS database. If the operation is successful the function returns the department admin's information, otherwise it returns nothing and the administrator is notified that there was a failure in the system and they are asked to attempt their operation again.
Test cases: None.

3. Class: System Administrator
    - Purpose: To describe the general role and permissions of the system administrators.
    - Constraints: None.
    - Persistent: No (user is created to add it to the system).

4.2.2.5.1 Attribute Descriptions

1. All attributes are inherited from the administrator superclass.

4.2.2.5.2 Method Descriptions

1. Method: submitPreferences(string list: course-name, string list: times)
This method is a department faculty member method that the system administrators can perform for testing purposes. Thus, it will be described below in the department faculty methods.

4.2.3 **Class: Department Faculty**

- Purpose: To describe the general role and permissions of the department faculty.
- Constraints: Cannot manage course, assignment, and preferences like the administrators.
- Persistent: No (user is created to add it to the system).

4.2.3.1 Attribute Descriptions

1. Attribute: teaching-load
   Type: string
   Description: The load that the teacher can have per year.
   Constraints: None.
2. Attribute: teaching-history
   Type: string
   Description: The history of the instructor, which is the classes that they have taught in the past.
   Constraints: None.
3. Attribute: research-group
   Type: string
   Description: Which research group is this faculty in
   Constrains: None

4.2.3.2 Method Descriptions

1. Method: submitPreferences(string list: course-name, string list: times)
   Return Type and Value: Returns a boolean that determines if the preference submission was successful or not.
   Parameters: The course names and times that the instructor prefers.
   Pre-condition: The courses exist in the system with the times that the instructor specifies.
   Post-condition: The course preference list is in the system.
   Attributes used: course.course-name, course.times
   Methods called: course.verifyTimes()
   Processing logic: An instructor submits up to 3 preferences in a list to the CAIPS system. If the instructor submits more than 3 preferences the function returns false and doesn't store the preference list in the system. If the preference list is valid, on the other hand, then the system returns true and stores the preference list to be used later by the administrators.
   Test cases: 2.1 - Submit Preference List

2. Method: viewPreferences()
   Return Type and Value: Returns the preference lists of the instructor's preferences as a tuple of string lists - the course names and the course times.
   Parameters: None.
   Pre-condition: The instructor submitted a preference list(s).
   Post-condition: The instructor can view their preference list(s).
   Attributes used: preferences.instructor-username, preferences.times, perferences.course-names
   Methods called: DepartmentFaculty.findPreferences()
   Processing logic: The instructor views their course preferences by having the system search for any preferences associated with their username. The function returns nothing if no preferences are found. Otherwise, the function returns the list of preferences to the instructor.
   Test cases: 2.3 - View any Faculty preference List

3. Method: viewAssignments()
   Return Type and Value: Returns the information associated with an assignment as a tuple of the string instructor name and integer course ID.
   Parameters: None.
   Pre-condition: The instructor has assignments in the system.
   Post-condition: The instructor can view their course assignments.
   Attributes used: assignments.instructor-name, assignments.course-ID
   Methods called: DepartmentFaculty.findAssignments()
   Processing logic: The department faculty member (the instructor) views their course assignments by having the system search for any assignments tied to their username. The function returns the information associated with the assignments if they're in the system or returns nothing if no assignments currently are in the system for this instructor.
   Test cases: 4.6 - View Instructor Course Assignments

4. Method: selectAssignment(integer assignment-ID)
   Return Type and Value: Returns the information associated with an instructor's assignment as a tuple of the course ID and the instructor name.
   Parameters: The ID of the assignment.
   Pre-condition: The assignment exists in the system.
   Post-condition: The assignment is now selected to be used by an interface.
   Attributes used: assignment.assignment-ID, assignment.instructor-name, assignment.course-ID
   Methods called: None.
   Processing logic: Here an instructor selects one of their assignments to be used in an interface. If the assignment is not found then the function returns nothing. Otherwise, it returns the information associated with the instructor's assignment (one of the possibly many assignments for the instructor).
   Test cases: 4.6 - View Instructor Course Assignments

5. Method: viewHistory()
   Return Type and Value: Returns the teaching history of the instructor as a string.
   Parameters: None.
   Pre-condition: The instructor has course history.
   Post-condition: The instructor can view their course history.
   Attributes used: DepartmentFaculty.teaching-history.
   Methods called: None.
   Processing logic: The instructor retrieves their teaching history. The function returns nothing if the instructor doesn't have any teaching history yet. Otherwise, it returns the teaching history of the instructor.
   Test cases: test 4.7 - Instructor Assignment History

6. Method: viewCourses(integer course-ID)

Return Type and Value: Returns as a string the course information for all of the courses or just the course specified by course ID.
Parameters: An optional course ID to get the information of a single course.
Pre-condition: Courses exist in the system.
Post-condition: A user can view all of the courses in the system or just the one specified.
Attributes used: course.course-information, course.course-ID
Methods called: None.
Processing logic: An administrator retrieves the course information with a given course ID or all of the courses' information by not including that parameter. If the course ID parameter is used and doesn't exist in the system then the function returns nothing. It also returns nothing if no courses exist in the system. Otherwise, it returns the list of course information(s).
Test cases: 3.2, 3.5, 3.6

7. Method: persistFacultyInfo(Object facultyObject)
   Return Type and Value: Returns as a boolean whether or not the data persisting operation was successful.
   Parameters: The faculty member object that contains the information that will be stored in the database.
   Pre-condition: The department faculty object contains data about the faculty's teaching history or teaching load.
   Post-condition: The faculty member's information is now stored in the database.
   Attributes used: departmentFaculty.teaching-load, departmentFaculty.teaching-history
   Methods called: None.
   Processing logic: An administrator performs some action that changes a department faculty member's information. This function is then automatically called to store the new information in the CAIPS database. If the operation is successful the function returns true and the administrator is notified, otherwise it returns false and the administrator is warned that their information wasn't saved.
   Test cases: None.

8. Method: retrieveFacultyInfo(UUID facultyIdent)
   Return Type and Value: Returns the information associated with a department faculty member as a department faculty member object.
   Parameters: The faculty member identification that will be used to retrieve the faculty member object.
   Pre-condition: The department faculty object contains data about the faculty's teaching history or teaching load and exists in the database.
   Post-condition: The faculty member's information is now loaded from the database.
   Attributes used: departmentFaculty.teaching-load, departmentFaculty.teaching-history
   Methods called: None.

Processing logic: An administrator performs some action that needs a department faculty member's information. This function is then automatically called to retrieve the information in the CAIPS database. If the operation is successful the function returns the department faculty member's information, otherwise it returns nothing and the administrator is notified that there was a failure in the system and they are asked to attempt their operation again.
Test cases: None.

4.3 Non-user Classes

4.3.1 Class: Assignments

- Purpose: To describe the attributes of the assignments in the CAIPS system.
- Constraints: None.
- Persistent: No (created by administrators in the system).

4.3.1.1 Attribute Descriptions

1. Attribute: instructor-name
   Type: String
   Description: The name of instructors for the assignment.
   Constraints: Shouldn't be longer than 100 characters .
2. Attribute: course-name
   Type: String
   Description: The course name of the assignment.
   Constraints: Shouldn't be longer than 100 characters.
3. Attribute: course-ID
   Type: Integer
   Description: The ID of the course.
   Constraints: Should be less than 20 digits long.
4. Attribute: assignment-ID
   Type: Integer
   Description: The ID of the assignment.
   Constraints: Should be less than 20 digits long.

4.3.1.2 Method Descriptions

1. Method: persistAssignmentInfo(Object assignmentObject)
   Return Type and Value: Returns as a boolean whether or not the data persisting operation was successful.
   Parameters: The assignment object that contains the information that will be stored in the database.
   Pre-condition: The assignment object contains data pertaining to the assignment.
   Post-condition: The assignment information is now stored in the database.
   Attributes used: assignment.instructor-name, assignment.course-name, assignment.course-ID, assignment.assignment-ID
   Methods called: None.

Processing logic: An administrator performs some action that changes an assignment's information. This function is then automatically called to store the new information in the CAIPS database. If the operation is successful the function returns true and the administrator is notified, otherwise it returns false and the administrator is warned that their information wasn't saved.
Test cases: None.

2. Method: retrieveAssignmentInfo(UUID assignmentIdent)
   Return Type and Value: Returns the information associated with an assignment as an assignment object.
   Parameters: The assignment identification that will be used to retrieve the assignment object.
   Pre-condition: The assignment object contains data and exists in the database.
   Post-condition: The assignment's information is now loaded from the database.
   Attributes used: assignment.instructor-name, assignment.course-name, assignment.course-ID, assignment.assignment-ID
   Methods called: None.
   Processing logic: An administrator performs some action that needs information about an assignment. This function is then automatically called to retrieve the information in the CAIPS database. If the operation is successful the function returns the assignment's information, otherwise it returns nothing and the administrator is notified that there was a failure in the system and they are asked to attempt their operation again.
   Test cases: None.

4.3.2 Class: Courses

- Purpose: To describe the attributes of the courses in the CAIPS system.
- Constraints: None.
- Persistent: No (created by administrators in the system).

4.3.2.1 Attribute Descriptions

1. Attribute: course-name
   Type: String
   Description: The name of the course.
   Constraints: Can't be more than 50 characters long.
2. Attribute: course-information
   Type: String
   Description: The information associated with the course, which includes the course location and time. It also includes the capacity of the class, the waitlist capacity, and the number of credits the course offers.
   Constraints: None.
3. Attribute: course-ID
   Type: Integer
   Description: The ID associated with the course, used for identification of the course in function calls.

Constraints: Should be less than 20 digits long.

4. Attribute: classroom-information
   Type: String
   Description: The information associated with the classroom, including the capacity and the location. Similar to some aspects of the course information but is used to make sure that the course is in a location that suits it.
   Constraints: None.
5. Attribute: history-information
   Type: String
   Description: The course history, which includes the past locations and times of the course and any changes in capacity or credit offerings.
   Constraints: None.

4.3.2.2 Method Descriptions

1. Method: persistCourseInfo(Object courseObject)
   Return Type and Value: Returns as a boolean whether or not the data persisting operation was successful.
   Parameters: The course object that contains the information that will be stored in the database.
   Pre-condition: The course object contains data pertaining to the course.
   Post-condition: The course information is now stored in the database.
   Attributes used: course.course-name, course.course-information, course.course-ID, course.classroom-information, course.history-information
   Methods called: None.
   Processing logic: An administrator performs some action that changes an course's information. This function is then automatically called to store the new information in the CAIPS database. If the operation is successful the function returns true and the administrator is notified, otherwise it returns false and the administrator is warned that their information wasn't saved.
   Test cases: None.

2. Method: retrieveCourseInfo(UUID courseIdent)
   Return Type and Value: Returns the information associated with a course as a course object.
   Parameters: The course identification that will be used to retrieve the course object.
   Pre-condition: The course object contains data and exists in the database.
   Post-condition: The course's information is now loaded from the database.
   Attributes used: course.course-name, course.course-information, course.course-ID, course.classroom-information, course.history-information
   Methods called: None.
   Processing logic: An administrator performs some action that needs information about a course. This function is then automatically called to retrieve the information in the CAIPS database. If the operation is successful the function returns the course's information, otherwise it returns nothing and the

administrator is notified that there was a failure in the system and they are asked to attempt their operation again.
Test cases: None.

4.3.3 Class: Preferences

- Purpose: To describe the attributes of the course preferences in the CAIPS system.
- Constraints: None.
- Persistent: No (created by administrators in the system).

4.3.3.1 Attribute Descriptions

1. Attribute: course-names
   Type: String
   Description: The name of the preference course.
   Constraints: Shouldn't be longer than 100 characters.
2. Attribute: times
   Type: String list
   Description: The times that the course is offered.
   Constraints: None.
3. Attribute: instructor-username
   Type: String
   Description: The name of the instructor that is tied to this course assignment.
   Constraints: The instructor name can not be more than 50 characters long.

4.3.3.2 Method Descriptions

1. Method: persistPreferenceInfo(Object preferenceObject)
   Return Type and Value: Returns as a boolean whether or not the data persisting operation was successful.
   Parameters: The preference object that contains the information that will be stored in the database.
   Pre-condition: The preference object contains data pertaining to the course.
   Post-condition: The preference information is now stored in the database.
   Attributes used: preferences.course-names, preferences.times, preferences.instructor-username
   Methods called: None.
   Processing logic: An administrator performs some action that changes a preference list. This function is then automatically called to store the new information in the CAIPS database. If the operation is successful the function returns true and the administrator is notified, otherwise it returns false and the administrator is warned that their information wasn't saved.
   Test cases: None.

2. Method: retrievePreferenceInfo(UUID preferenceIdent)
   Return Type and Value: Returns the information associated with a preference list as a preference object.

Parameters: The preference list identification that will be used to retrieve the preference object.
Pre-condition: The preference object contains data and exists in the database.
Post-condition: The preference list information is now loaded from the database.
Attributes used: preferences.course-names, preferences.times, preferences.instructor-username
Methods called: None.
Processing logic: An administrator performs some action that needs information about a preference list. This function is then automatically called to retrieve the information in the CAIPS database. If the operation is successful the function returns the preference list information, otherwise it returns nothing and the administrator is notified that there was a failure in the system and they are asked to attempt their operation again.
Test cases: None.

4.4 BLM Interface Classes

4.4.1 BLM Interface to GUI Class

4.4.1.1 Attribute Descriptions

1. Attribute: request
   Type: String
   Description: The request coming from the GUI that will eventually be processed by the BLM through the GUI router class.
   Constraints: None.

4.4.1.2 Method Descriptions

1. Method: retreiveRequest(string request)
   Return Type and Value: None.
   Parameters: The request that is coming from the GUI.
   Pre-condition: The BLM interface to the GUI is able to process the request being sent from the GUI.
   Post-condition: The request is sent one layer further into the BLM.
   Attributes used: GUIinterface.request
   Methods called: GUIinterface.processRequest().
   Processing logic: A request comes into the CAIPS BLM from the GUI and reaches this point first. Here, the BLM interface to the GUI processes the request by parsing the request string into a format that can then be passed onto the GUI router to the BLM to then send into the BLM itself.
   Test cases: None.

2. Method: processRequest(string request)
   Return Type and Value: None.
   Parameters: The request that came from the GUI.

Pre-condition: The BLM interface to the GUI is able to process the request being sent from the GUI.
Post-condition: The request is sent one layer further into the BLM.
Attributes used: GUIinterface.request
Methods called: None.
Processing logic: After this GUI interface receives the request it uses this method to parse the request string into something that can be recognized by the GUI router to then send into the BLM to perform an action requested by the user. The interface also caches the request to possibly be used later.
Test cases: None.

3. Method: sendRequest(string request)
Return Type and Value: None.
Parameters: The request that came from the GUI.
Pre-condition: The BLM interface to the GUI processed the request that came from the GUI.
Post-condition: The request is sent one layer further into the BLM.
Attributes used: GUIinterface.request
Methods called: None.
Processing logic: After this GUI interface processes the request it uses this method to send the request to the GUI router to then send into the BLM to perform an action requested by the user.
Test cases: None.

4.4.2 GUI Router to BLM Class

4.4.2.1 Attribute Descriptions

2. Attribute: request
Type: String
Description: The request coming from the GUI that will be processed and sent to a class in the BLM.
Constraints: None.

4.4.2.2 Method Descriptions

1. Method: processRequest(string request)
Return Type and Value: None.
Parameters: The request that came from the BLM interface to the GUI.
Pre-condition: The GUI router is able to process the request being sent from the GUI interface.
Post-condition: The request is sent one layer further into the BLM.
Attributes used: GUIrouter.request
Methods called: None.
Processing logic: After this GUI router receives the request it uses this method to parse the request string to then send into the BLM to perform an action requested

by the user. The parsing of the request string also determines where in the BLM to send the command for the user task to be performed. The GUI router also caches the request to possibly be used later.
Test cases: None.

2. Method: sendFacultyRequest(string request)
   Return Type and Value: None.
   Parameters: The request that came from the BLM interface to the GUI.
   Pre-condition: The GUI router already processed the request being sent from the GUI interface.
   Post-condition: The request is sent one layer further into the BLM to the department faculty member class.
   Attributes used: GUIrouter.request
   Methods called: None.
   Processing logic: After this GUI router processes the request it uses this method to send the request to the department faculty member class after it determined that the user action is a department faculty action after parsing the request.
   Test cases: None.
3. Method: sendUserRequest(string request)
   Return Type and Value: None.
   Parameters: The request that came from the BLM interface to the GUI.
   Pre-condition: The GUI router already processed the request being sent from the GUI interface.
   Post-condition: The request is sent one layer further into the BLM to the department faculty member class.
   Attributes used: GUIrouter.request
   Methods called: None.
   Processing logic: After this GUI router processes the request it uses this method to send the request to the user class after it determined that the user action is a general user action after parsing the request.
   Test cases: None.
4. Method: sendAdminRequest(string request)
   Return Type and Value: None.
   Parameters: The request that came from the BLM interface to the GUI.
   Pre-condition: The GUI router already processed the request being sent from the GUI interface.
   Post-condition: The request is sent one layer further into the BLM to the department faculty member class.
   Attributes used: GUIrouter.request
   Methods called: None.
   Processing logic: After this GUI router processes the request it uses this method to send the request to the admin class after it determined that the user action is an admin action after parsing the request.
   Test cases: None.

4.5 Database Interface Classes

4.5.1 Database Abstraction Interface Class

4.5.1.1 Attribute Descriptions

6. Attribute: Data
   Type: object
   Description: The data that is either being persisted to or retrieved from the database. Either the retrieved data is received from the database facade or the persisting data is sent to the database facade.
   Constraints: None.

4.5.1.2 Method Descriptions

1. Method: persist(Object obj)
   Return Type and Value: None.
   Parameters: The data that will be stored in the database that interfaces with the CAIPS BLM.
   Pre-condition: The BLM interfaces with a database.
   Post-condition: New information is stored in the CAIPS database.
   Attributes used: databaseInterface.data
   Methods called: None.
   Processing logic: Information created in the CAIPS BLM is automatically stored in the CAIPS database through this interface and the database facade as well. The classes that generate information in the BLM send information here to be sent to the database facade that then persists the data in the database.
   Test cases: None.

2. Method: retrieve(UUID ident)
   Return Type and Value: None.
   Parameters: The identification of the data that is being requested from the database.
   Pre-condition: The BLM interfaces with a database.
   Post-condition: New information is one step further to being retrieved from the database.
   Attributes used: databaseInterface.data
   Methods called: None.
   Processing logic: The BLM requests information from the database using the identification of the data object. The request is sent to the database abstraction interface that then forwards the request to the database facade.
   Test cases: None.

4.5.2 Database Facade Class

4.5.2.1 Attribute Descriptions

7. Attribute: Data
   Type: object
   Description: The data that is either being persisted to or retrieved from the database. Either the retrieved data is being received from the database and sent to the database abstraction interface or the persisting data is being received from the database abstraction interface and is being sent to the database.
   Constraints: None.

4.5.2.2 Method Descriptions

1. Method: persist(Object obj)
   Return Type and Value: None.
   Parameters: The data that will be stored in the database that interfaces with the CAIPS BLM.
   Pre-condition: The BLM interfaces with a database.
   Post-condition: New information is stored in the CAIPS database.
   Attributes used: databaseFacade.data
   Methods called: None.
   Processing logic: Information created in the CAIPS BLM is automatically stored in the CAIPS database through this facade and the database interface as well. The classes that generate information in the BLM send information to the interface that then sends it to here to be persisted in the CAIPS database.
   Test cases: None.

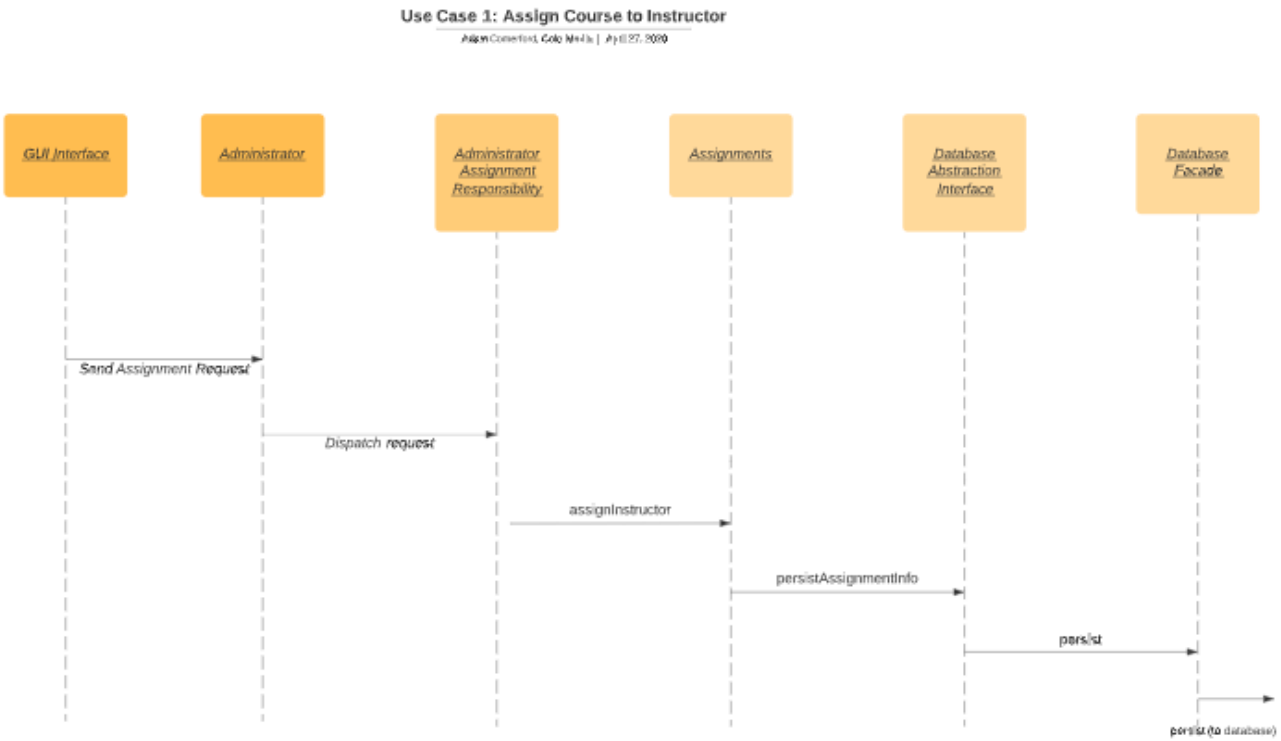
2. Method: retrieve(UUID ident)
   Return Type and Value: None.
   Parameters: The identification of the data that is being requested from the database.
   Pre-condition: The BLM interfaces with a database.
   Post-condition: New information is one step further to being retrieved from the database.
   Attributes used: databaseFacade.data
   Methods called: None.
   Processing logic: The BLM requests information from the database using the identification of the data object. The request is sent to the database interface interface that then forwards the object to the BLM.
   Test cases: None.

5 Dynamic Model

For all of our sequence diagrams we are assuming that the database returns information to the GUI and all methods return rightfully to their caller so we don't include return statements from the database or other objects.

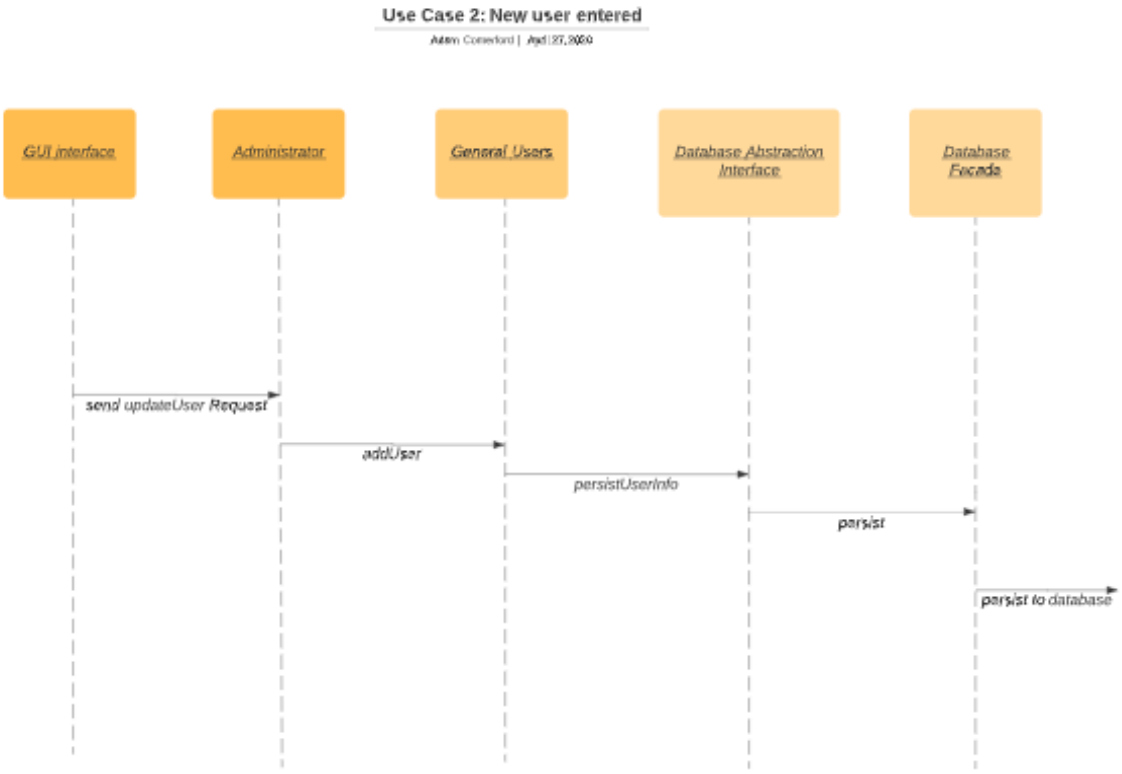
5.1 Scenario 1

- *Scenario Name:* Assign Course to Instructor
- *Scenario Description:* Once an administrator logins in through the GUI, they can send a request from the GUI that reaches the administrator class. In this case, the request from the GUI is to assign a course to an instructor, so the request eventually leads to that operation taking place. After this, the new assignment information is automatically persisted into the database.
- *Sequence Diagram:*



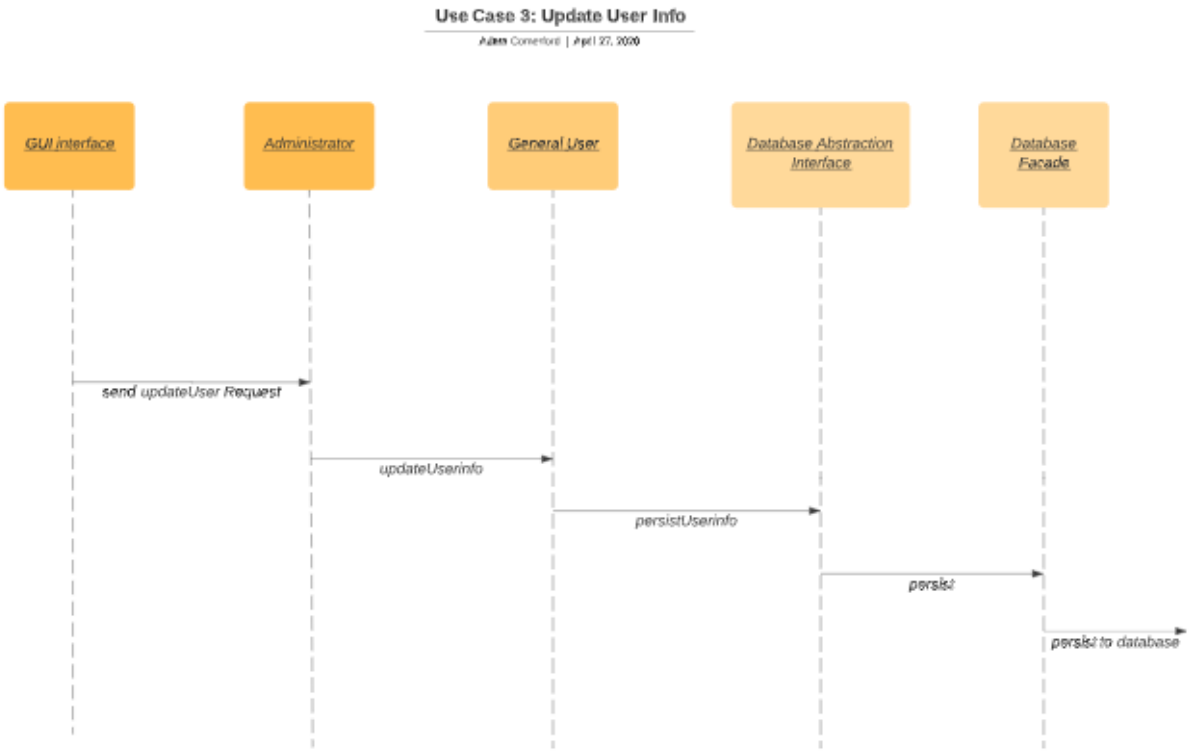
5.2 Scenario 2

- *Scenario Name:* New User Entered in System
- *Scenario Description:* An administrator logins into CAIPS through the GUI and then sends a request to the BLM to enter a user into the system. This request also contains the user information to create the new user. The request is sent into the BLM where the administrator class takes care of putting the user in the system. Once the operation is complete, the new user is automatically persisted into the CAIPS database through its interface with the BLM.
- *Sequence Diagram:*



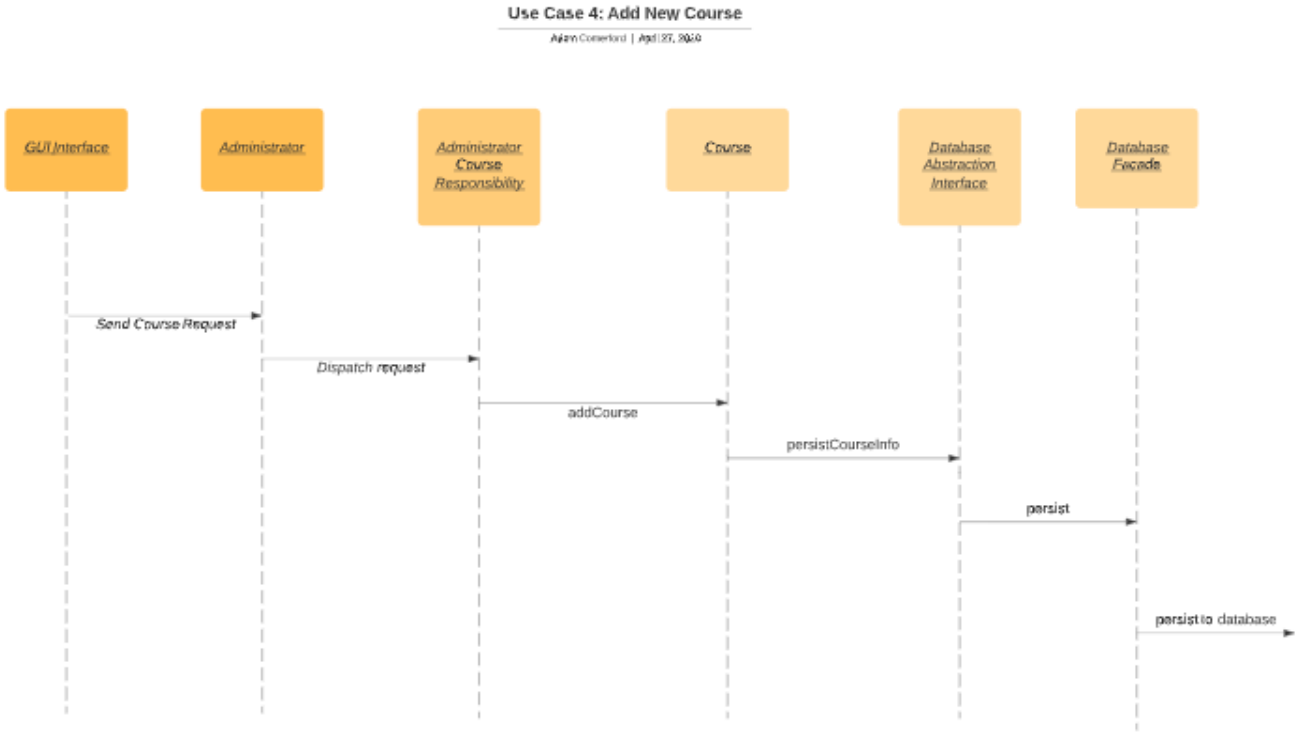
5.3 Scenario 3

- *Scenario Name:* Update User Info
- *Scenario Description:* An administrator logins into CAIPS through the GUI and then sends a request to the BLM to enter new user information into the system. This request also contains the user information to update a user. The request is sent into the BLM where the administrator class takes care of putting the user information in the system. Once the operation is complete, the updated user is automatically persisted into the CAIPS database through its interface with the BLM to update the database's information on the newly updated user object.
- *Sequence Diagram:*



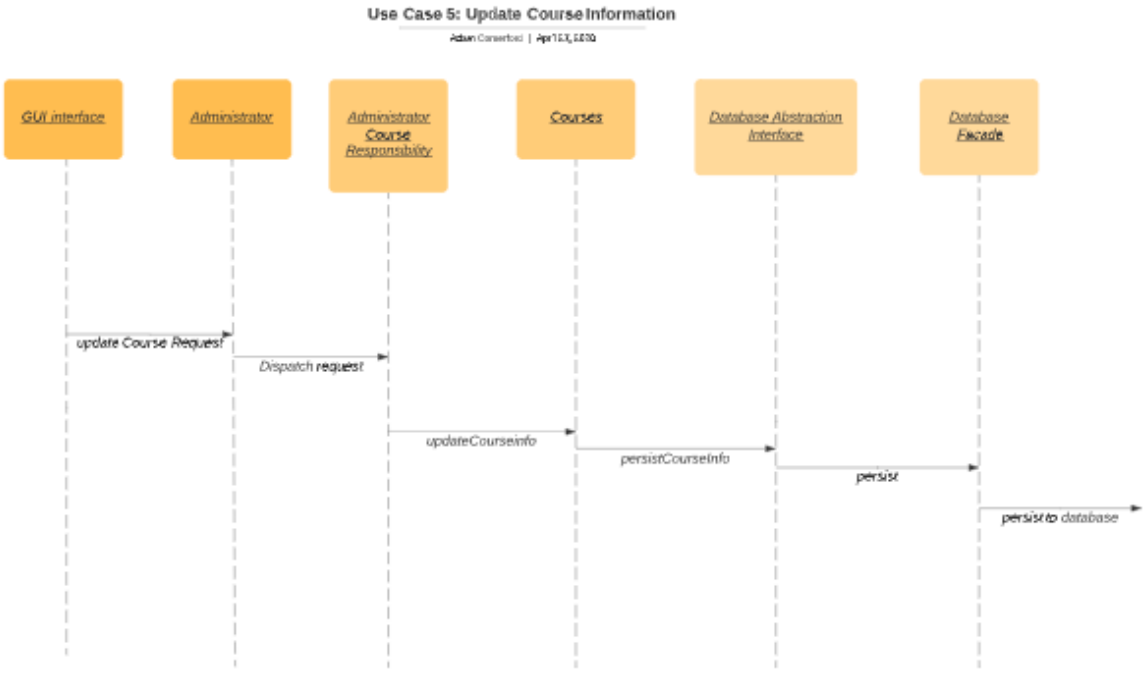
5.4 Scenario 4

- *Scenario Name:* Add New Course
- *Scenario Description:* An administrator logins into CAIPS through the GUI and then sends a request to the BLM to enter a new course into the system. This request also contains the course information to create the new course. The request is sent into the BLM where the administrator class takes care of putting the course information in the system. Once the operation is complete, the new course is automatically persisted into the CAIPS database through its interface with the BLM.
- *Sequence Diagram:*



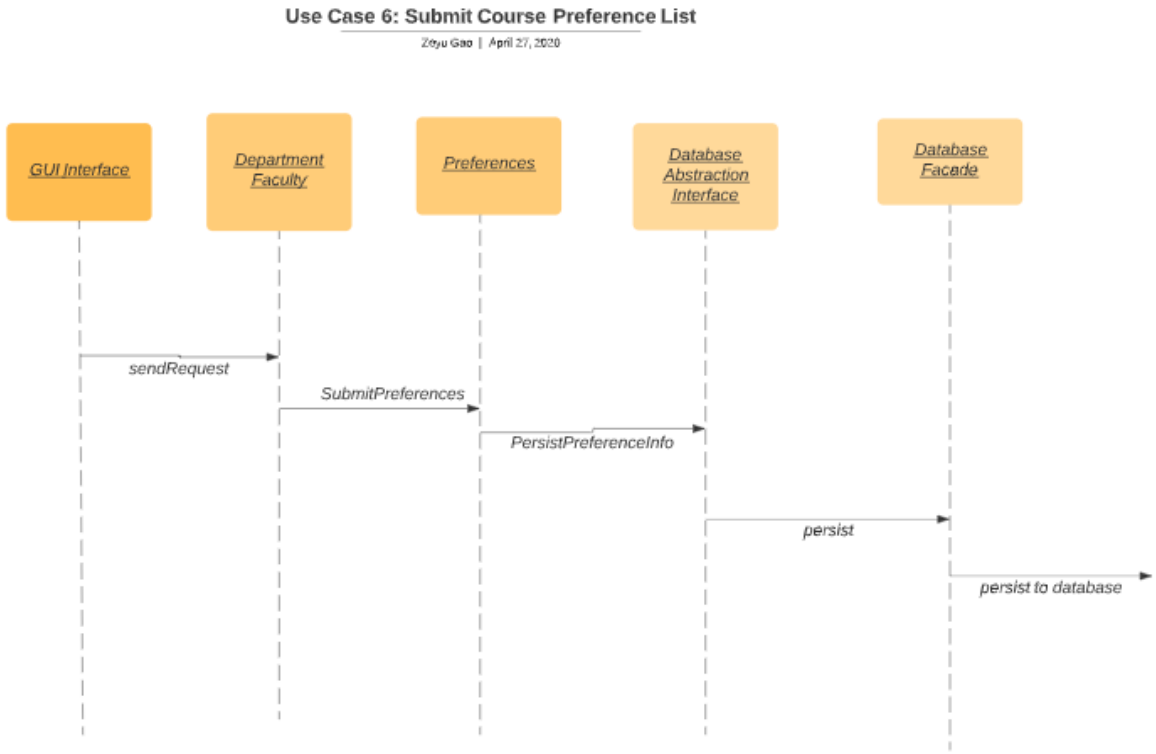
5.5 Scenario 5

- *Scenario Name:* Update Course Information
- *Scenario Description:* An administrator logins into CAIPS through the GUI and then sends a request to the BLM to enter new course information into the system. This request also contains the course information to update a course. The request is sent into the BLM where the administrator class takes care of putting the course information in the system. Once the operation is complete, the updated course is automatically persisted into the CAIPS database through its interface with the BLM to update the database's information on the newly updated course object.
- *Sequence Diagram:*



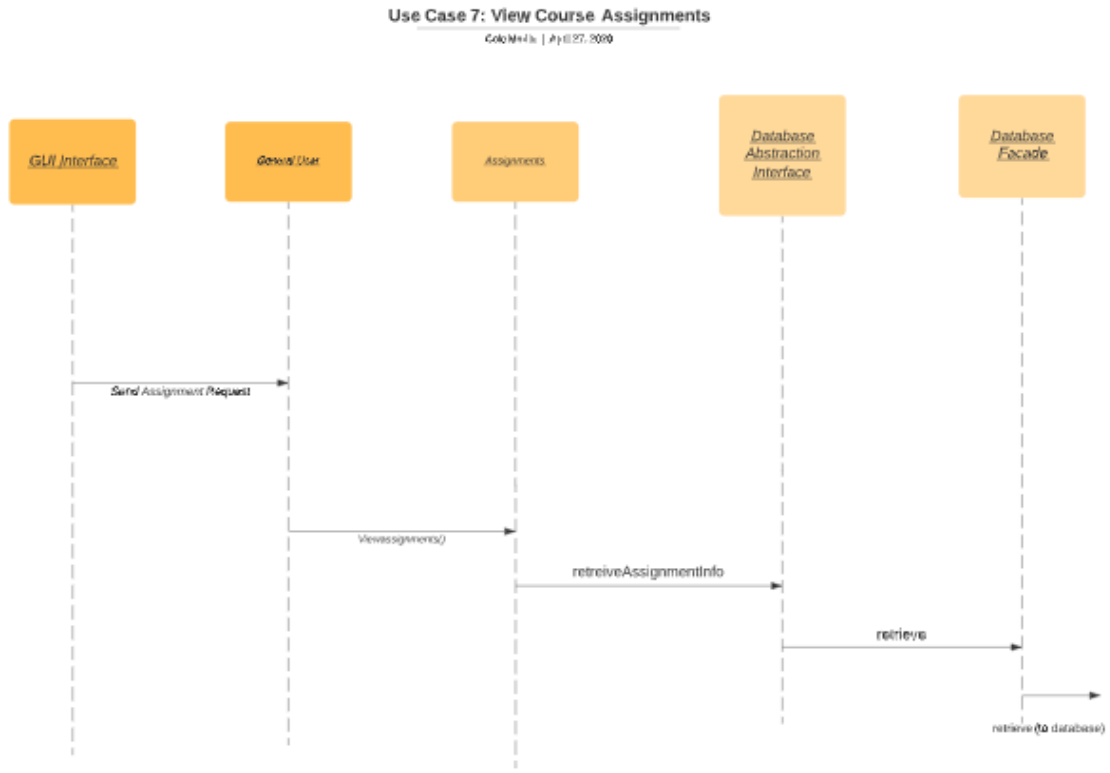
5.6 Scenario 6

- *Scenario Name:* Submit Course Preference List
- *Scenario Description:* A system administrator or department faculty member logins into CAIPS through the GUI and then sends a request to the BLM to enter a new preference list into the system. This request also contains the preference list information. The request is sent into the BLM where the department faculty member class or the system administrator class takes care of putting the preference list in the system. Once the operation is complete, the new preference list is automatically persisted into the CAIPS database through its interface with the BLM.
- *Sequence Diagram:*



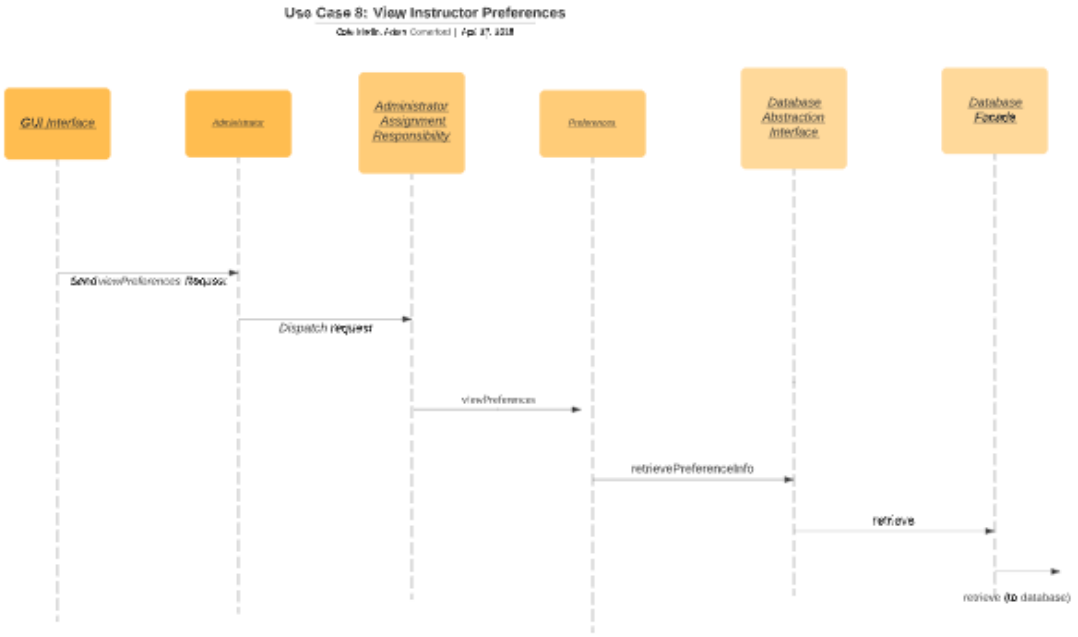
5.7 Scenario 7

- *Scenario Name:* View Course Assignments
- *Scenario Description:* A user logins into CAIPS through the GUI and requests to view course assignments. If the user is a department faculty member, then the GUI router will forward the request to the department faculty member to view that faculty's course assignments. If the user is an administrator, then the GUI router will forward the request to the administrator class to view all faculty members' course assignments. The course assignments are retrieved from the database through the BLM's interface with the database.
- *Sequence Diagram:*



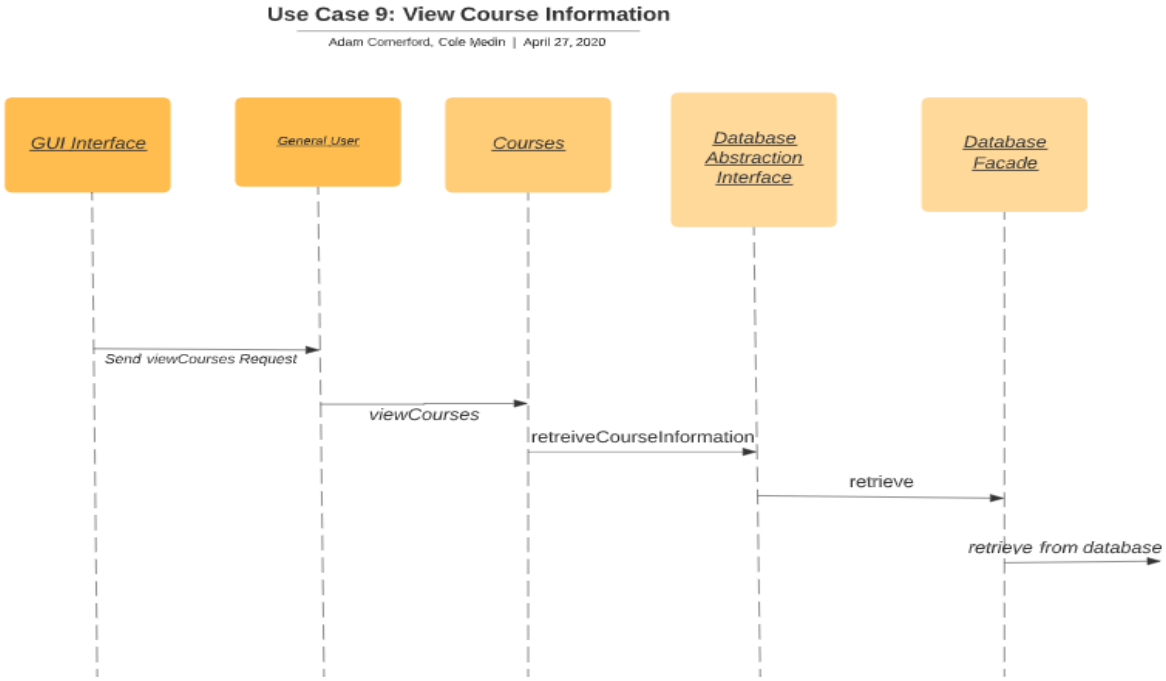
5.8 Scenario 8

- *Scenario Name:* View Instructor Preferences
- *Scenario Description:* A user logins into CAIPS through the GUI and requests to view preference lists. If the user is a department faculty member, then the GUI router will forward the request to the department faculty member to view that faculty's preference lists. If the user is an administrator, then the GUI router will forward the request to the administrator class to view all faculty members' preference lists. The preference lists are retrieved from the database through the BLM's interface with the database.
- *Sequence Diagram:*



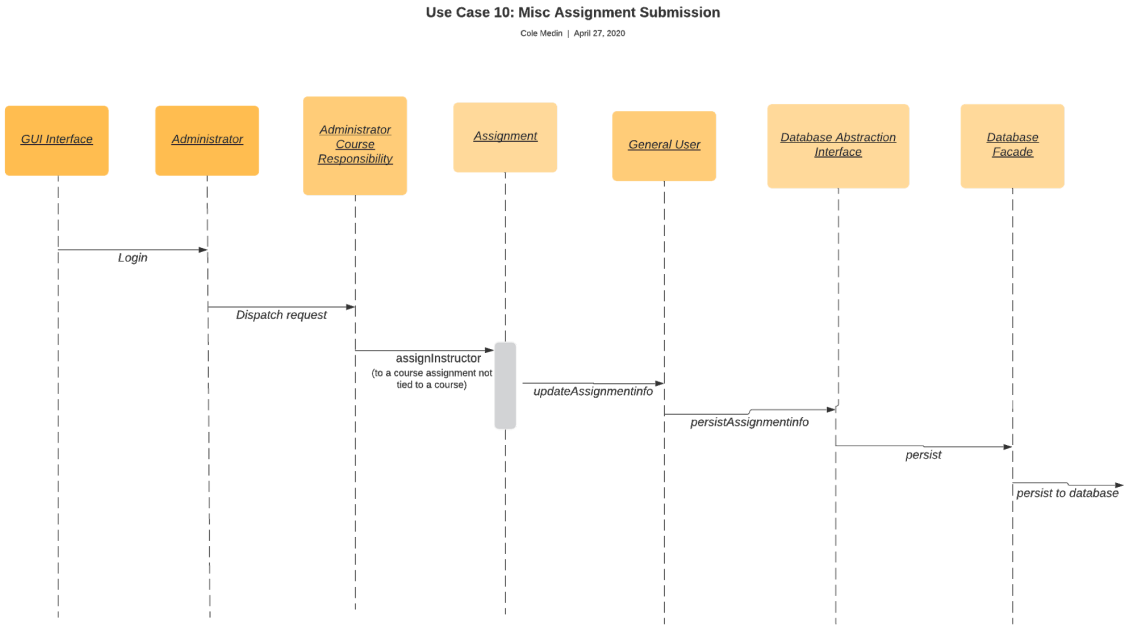
5.9 Scenario 9

- *Scenario Name:* View Course Information
- *Scenario Description:* A user logins into CAIPS through the GUI and requests to view course information. The user can request to either view the course information of select courses or all courses. Either way, the information of the course(s) is retrieved from the database through the BLM's interface with the database.
- *Sequence Diagram:*



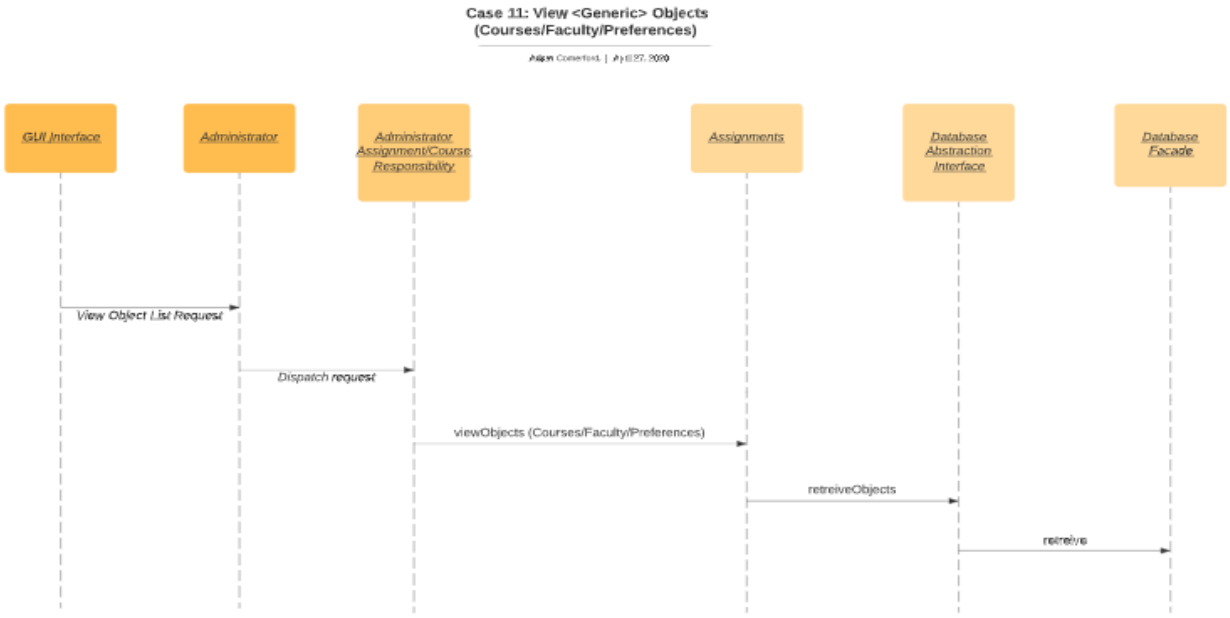
5.10 Scenario 10

- *Scenario Name:* Misc Assignment Submission
- *Scenario Description:* This scenario is for administrators who login into the CAIPS system through the GUI and request to submit assignments that are not tied to courses. After the GUI routes this request to the administrator class and the assignment submission is made, it is persisted in the CAIPS database through its interfacing with the BLM.
- *Sequence Diagram:*



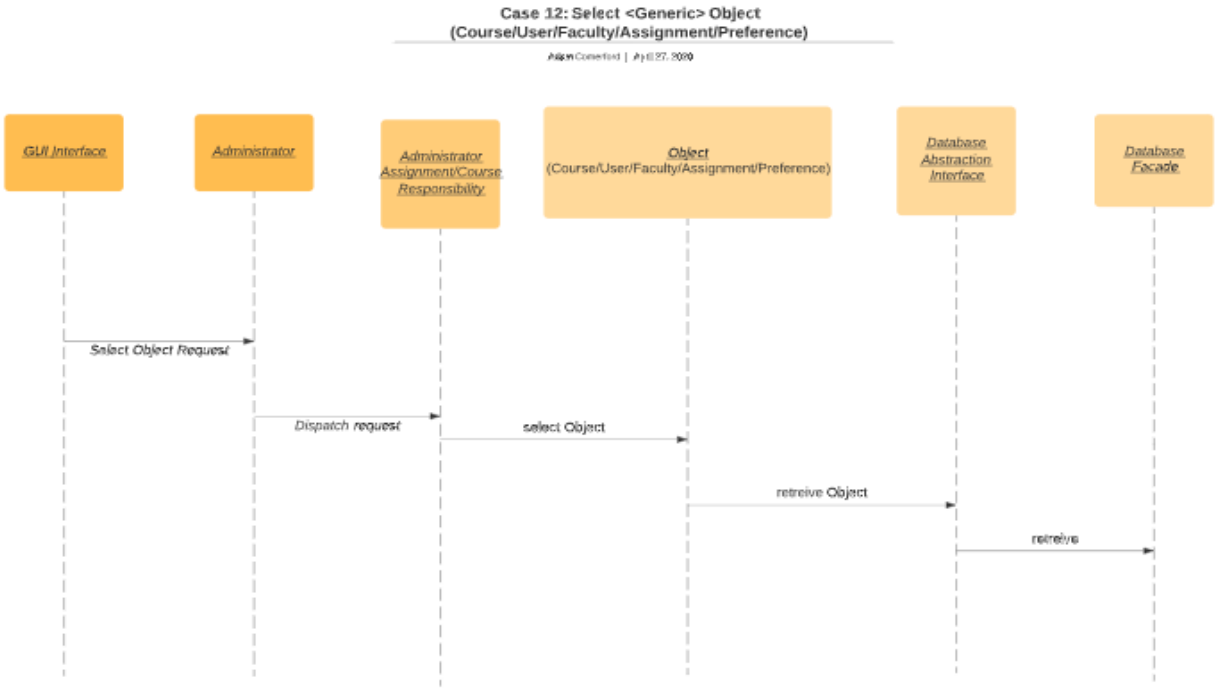
5.11 Scenario 11

- *Scenario Name:* View Object List
- *Scenario Description:* This scenario is for Administrators who need to view a list of Courses/Faculty/Preferences before selecting one to update. When a View Object List request has been made, the BLM retrieves a list of all of those Objects within the system in which they can then select one (Scenario 12).
- *Sequence Diagram:*



5.12 Scenario 12

- *Scenario Name:* Select Object
- *Scenario Description:* This scenario is for Administrators who need to select an object from a list after a view<Object> request has been made. From a list of objects, the Administrator can select any specific object which can then be followed up by any method specific to that object (e.g. select a course, then view Course Information).
- *Sequence Diagram:*



6 Non-functional requirements

6.1: The system will always detect a failure in sending information to the database through the database interface in the BLM.

- No matter the program, retrieving information from a database should always either be a success or a known failure so that the program can attempt again to retrieve the information. The users in the CAIPS system should never be given null or false information from the database, so it must always detect failures.

6.2: The courses created and assigned to instructors in the system are cleared by OCM ahead of time. This is done outside the BLM. Courses are cleared with OCM through external communications such as email.

- This assumption is used for CAIPS because that way the system doesn't need to include an OCM class and have to manage all of the OCM input into the system. In reality it is simple enough for all of the OCM clearing of classes to take place outside of the system. OCM and other parties involved needing to go through the system would only bog down the process of clearing classes to be put on the schedule.

6.3: The BLM does not take any security measures. This is all done before the GUI sends requests to BLM. Therefore, only valid requests will be processed.

- This will reduce the workload of BLM, it will be more easy for administrators of the CAIPS to maintain the system running well.

6.4: The BLM will retrieve and persist data from/to the infrastructure layer within a reasonable amount of waiting time from the perspective of the user - no longer than 10 seconds.

- No request for information should take longer than 10 seconds. Any user who experiences that much delay in a simple request would definitely be disgruntled.

6.5: If a request from the BLM fails or succeeds to process, the GUI will give a visible notification to the user.

- Any user in any system should always be notified of the success of their requests. If a user doesn't know if their request was successful or not, they'll be anxious and probably attempt their request again.

6.6: The number of concurrent users the system can support at its peak is 3000.

- Since students aren't a part of this system, it won't need to support a super large number of users accessing the system during course scheduling season. This number will be enough for all of the administrators accessing the system.