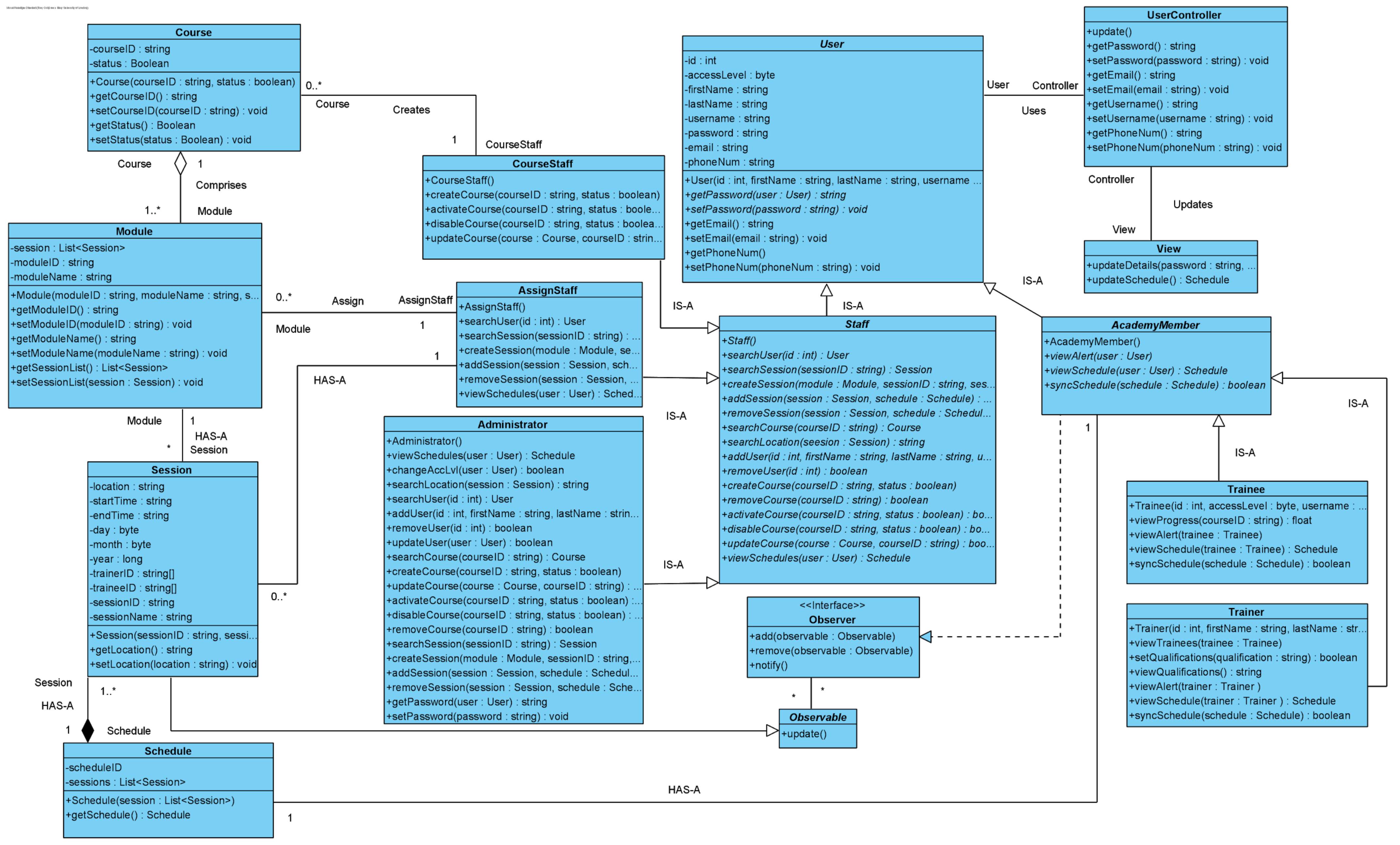
Group 30

Trainer+

ECS506U Software Engineering Group Project

Design Report


Course
-courseID : string
-status : Boolean
+Course(courseID : string, status : boolean)
+getCourseID() : string
+setCourseID(courseID : string) : void
+getStatus() : Boolean
+setStatus(status : Boolean) : void
0..*
Course
Creates
1
CourseStaff
CourseStaff
+CourseStaff()
+createCourse(courseID : string, status : boolean)
+activateCourse(courseID : string, status : boole...
+disableCourse(courseID : string, status : boolea...
+updateCourse(course : Course, courseID : strin...
Course
1
Comprises
1..*
Module
Module
-session : List<Session>
-moduleID : string
-moduleName : string
+Module(moduleID : string, moduleName : string, s...
+getModuleID() : string
+setModuleID(moduleID : string) : void
+getModuleName() : string
+setModuleName(moduleName : string) : void
+getSessionList() : List<Session>
+setSessionList(session : Session) : void
0..*
Assign
AssignStaff
Module
1
AssignStaff
+AssignStaff()
+searchUser(id : int) : User
+searchSession(sessionID : string) : ...
+createSession(module : Module, se...
+addSession(session : Session, sch...
+removeSession(session : Session, ...
+viewSchedules(user : User) : Sched...
1
HAS-A
Module
1
HAS-A
*
Session
Session
-location : string
-startTime : string
-endTime : string
-day : byte
-month : byte
-year : long
-trainerID : string[]
-traineeID : string[]
-sessionID : string
-sessionName : string
+Session(sessionID : string, sessi...
+getLocation() : string
+setLocation(location : string) : void
0..*
Session
HAS-A
1..*
1
Schedule
Schedule
-scheduleID
-sessions : List<Session>
+Schedule(session : List<Session>)
+getSchedule() : Schedule
1
HAS-A
Administrator
+Administrator()
+viewSchedules(user : User) : Schedule
+changeAccLvl(user : User) : boolean
+searchLocation(session : Session) : string
+searchUser(id : int) : User
+addUser(id : int, firstName : string, lastName : strin...
+removeUser(id : int) : boolean
+updateUser(user : User) : boolean
+searchCourse(courseID : string) : Course
+createCourse(courseID : string, status : boolean)
+updateCourse(course : Course, courseID : string) : ...
+activateCourse(courseID : string, status : boolean) :...
+disableCourse(courseID : string, status : boolean) : ...
+removeCourse(courseID : string) : boolean
+searchSession(sessionID : string) : Session
+createSession(module : Module, sessionID : string,...
+addSession(session : Session, schedule : Schedul...
+removeSession(session : Session, schedule : Sche...
+getPassword(user : User) : string
+setPassword(password : string) : void
IS-A
IS-A
IS-A
User
-id : int
-accessLevel : byte
-firstName : string
-lastName : string
-username : string
-password : string
-email : string
-phoneNum : string
+User(id : int, firstName : string, lastName : string, username ...
+getPassword(user : User) : string
+setPassword(password : string) : void
+getEmail() : string
+setEmail(email : string) : void
+getPhoneNum()
+setPhoneNum(phoneNum : string) : void
User
Controller
Uses
UserController
+update()
+getPassword() : string
+setPassword(password : string) : void
+getEmail() : string
+setEmail(email : string) : void
+getUsername() : string
+setUsername(username : string) : void
+getPhoneNum() : string
+setPhoneNum(phoneNum : string) : void
Controller
Updates
View
View
+updateDetails(password : string, ...
+updateSchedule() : Schedule
IS-A
IS-A
Staff
+Staff()
+searchUser(id : int) : User
+searchSession(sessionID : string) : Session
+createSession(module : Module, sessionID : string, ses...
+addSession(session : Session, schedule : Schedule) : ...
+removeSession(session : Session, schedule : Schedul...
+searchCourse(courseID : string) : Course
+searchLocation(seesion : Session) : string
+addUser(id : int, firstName : string, lastName : string, u...
+removeUser(id : int) : boolean
+createCourse(courseID : string, status : boolean)
+removeCourse(courseID : string) : boolean
+activateCourse(courseID : string, status : boolean) : bo...
+disableCourse(courseID : string, status : boolean) : bo...
+updateCourse(course : Course, courseID : string) : boo...
+viewSchedules(user : User) : Schedule
AcademyMember
+AcademyMember()
+viewAlert(user : User)
+viewSchedule(user : User) : Schedule
+syncSchedule(schedule : Schedule) : boolean
1
IS-A
IS-A
Trainee
+Trainee(id : int, accessLevel : byte, username : ...
+viewProgress(courseID : string) : float
+viewAlert(trainee : Trainee)
+viewSchedule(trainee : Trainee) : Schedule
+syncSchedule(schedule : Schedule) : boolean
Trainer
+Trainer(id : int, firstName : string, lastName : str...
+viewTrainees(trainee : Trainee)
+setQualifications(qualification : string) : boolean
+viewQualifications() : string
+viewAlert(trainer : Trainer)
+viewSchedule(trainer : Trainer) : Schedule
+syncSchedule(schedule : Schedule) : boolean
<<Interface>>
Observer
+add(observable : Observable)
+remove(observable : Observable)
+notify()
*
*
Observable
+update()

2. Traceability matrix

Class	Requirement	Brief Explanation
User	RQ1, RQ2, RQ5, RQ6	User stores username, password, email address and phone number as attributes
User	RQ3	User constructor allows users to create own username based on id
User	RQ4	User has a method to set password (setPassword())
CourseStaff	RQ8	CourseStaff has a method to create course (createCourse())
CourseStaff	RQ12	CourseStaff has a method to update a course's courseID an status (updateCourse())
CourseStaff	RQ15	CourseStaff has a method to search course using a courseID (searchCourse())
CourseStaff	RQ16	CourseStaff has a method to deactivate a course by setting the Boolean variable status to 'False' (disableCourse())
AssignStaff	RQ17	AssignStaff has a method to view a user's schedule (viewSchedule())

AssignStaff	RQ18, RQ19	AssignStaff has a method to search for a user which allows them to view the user details (searchUser())
AssignStaff	RQ21, RQ22	AssignStaff has a method to add sessions onto a user’s schedule (addSession())
AssignStaff	RQ23	AssignStaff has a method to create a session with a specific location (createSession())
AssignStaff	RQ24	AssignStaff has a method to remove a session (removeSession())
Trainer	RQ25	Trainer has a method to update their qualifications (setQualifications())
AcademyMember	RQ28	Both trainer and trainee have a method to view alerts (viewAlerts())
AcademyMember	RQ31, RQ32	Both trainer and trainee have a method to view their schedule (viewSchedule())
AcademyMember	RQ33	Both trainer and trainee have a method to sync their schedule to their calendar (syncSchedule())
Administrator	RQ35, RQ38, RQ39, RQ40	Admin has a method that to search a specific user in the system and view their information (searchUser())

Administrator	RQ34	Admin has a method that removes a user from the system (removeUser())
Administrator	RQ36	Admin has a method that removes an inactive course from the system (removeCourse())
Administrator	RQ37	Admin has a method that changes the access level of a user (changeAccLvl())

Requirements not satisfied (no longer needed):

RQ10, RQ11, RQ13, RQ14: Course no longer has description or content/material

3. Design Discussion

Overview and what stayed the same

Overall, the fundamental structure of our application seen in our domain model remains present in our class diagram. The idea of a central User entity specialised into the key user role entities such as Trainer, Trainee, Academy Scheduling Staff (renamed to Staff in the class diagram) and Administrator have been left untouched. As have the Calendar entity (renamed to Schedule in the class diagram) referenced across Trainer, Trainee and Academy Scheduling Staff. In addition, Course entities still consist of Modules in our class diagram.

Changes and their explanations

All key user role entities in the class diagram are no longer just specializations of the User entity alone. Two new entities, Staff and AcademyMember were added to create a layer of difference and abstraction between the two types of key user roles. Trainer and Trainee are specialisations of AcademyMember and CourseStaff, AssignStaff and Administrator are specializations of Staff. This is due to the major differences in interaction these two groups have within our system.

The Calendar entity has been changed to the Schedule entity which consists of Session entities. Calendar changed to Schedule not only because the name is more fitting but also because it might get confused with third party calendars that our users might use to sync their Schedules to. The addition of a Session entity was necessary as we needed a way of implementing Modules onto Schedules, however, a Module on a Schedule consists of more than just the Module itself but of its delivered date, time and attendants. All of which are all accounted for and accommodated by the addition of the Session entity.

Other changes include the Observer, Observable, UserController and View entities which were added to achieve our desired design patterns. These are explored more thoroughly in the next section.

Abstraction - Occurrence

In our class diagram we have implemented the “Abstraction-Occurrence” design pattern. The “Module” entity provides common information about a session such as the identification code and the name. When creating a “Module” object, it must be unique and cannot be duplicated more than once. The “Session” entity on the other hand, will have its own set of unique information every time a new instance has been created. For example, a session object can have a different start/end dates and time, different trainers & trainees. Within the Trainer+ system, the “AssignStaff” will be creating more than one “Session” object continuously. Therefore, we need an efficient way of implementing a set of objects without repeating the same common information as it would be a daunting task for the “AssignStaff” to enter and it would take up unnecessary storage space for the system.

Observer design pattern

We have implemented the observer design pattern between the “AcademyMember” and “Session” entity. Once a session is updated, the observable gets the updates from the session instance. It passes it onto the observer interface which notifies the academy member about the changes. Observer design pattern defines a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing. In our case, any changes to one session have effects on all the trainers and trainees that are participating in that session. Therefore, using the observer design pattern to notify all the trainer/trainee objects enables abstract and minimal coupling between the subject(sessions) and observer(academy members). It also eliminates the observer(academy member) independence which can cause unexpected behaviours.

MVC

The user will require a GUI in order to interact with the logic of our application i.e., the methods available to the given class. In our class diagram, we have made sure that the user can update their personal details, allow trainers & trainees to view the latest schedule and view the necessary changes taking place. When developing Trainer+, it is important to separate our work from the logic and the design of the application. To increase cohesion, logic and design are stored in different files. This is done to prevent the project becoming less manageable, harder to read and identify any mistakes during development. We can reduce coupling by providing different classes with different responsibilities therefore reducing the amount of dependency a class has to take on. During the testing phase the MVC will be vital to track for any errors within the system as we can pinpoint where the error originates from.

Justification for multiplicities

A CourseStaff can create 0 to many Courses and a Course is created by 1 CourseStaff. A CourseStaff doesn’t have to create a Course, they can be tasked with other things such as updating or disabling a Course. Thereby explaining the 0 to many relation. Similarly, an AssignStaff can assign 0 to many Modules/Sessions and a Module/Session is assigned by one AssignStaff. An AssignStaff doesn’t have to assign a Module/Session as they too can be tasked with other things, thereby supporting its 0 to many relation.

A Module has many Sessions and a Session has a Module. Therefore, multiple Sessions of the same Module can be implemented which is exactly what we needed for our Schedule. A Schedule belongs to one AcademyMember and each AcademyMember has a Schedule. This is a one-to-one relationship as both entities involved should hold reference to only one object from the other entity.

Justification for associations and aggregations (including compositions)

The aggregation relationship we have is between Course and Module. A Course will consist of Modules however if a Course object dies, the Module objects will outlive the Course's death. On the other hand, the composition relationship we have is between Schedule and Session. A Schedule is made up of Sessions and if a Schedule object dies then the Session objects die with it.

Justification for generalisations

The purpose for generalisation is to form hierarchies and encourage abstraction. Subclasses inherit attributes and methods from their superclass and are then able to expand on what they’ve inherited.

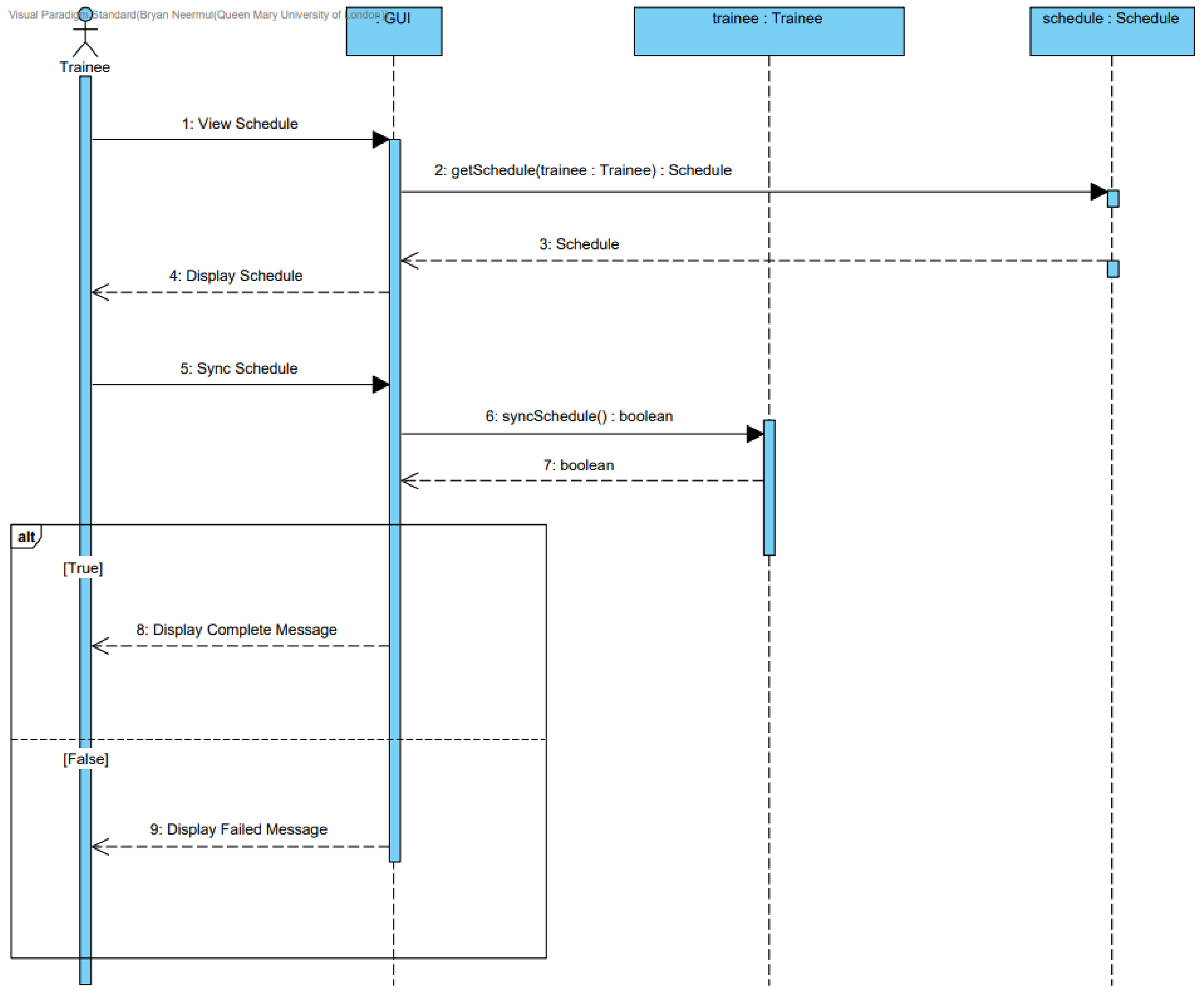
Therefore, both Staff and AcademyMember are Users as Users is their superclass and CourseStaff, AssignStaff and Administrator are Staff as Staff is their superclass.

4. Sequence Diagram 1 (15%)

Sync Schedule Use Case

This sequence diagram will represent the Sync Calendar use case for when a trainee user wants to sync their schedule to their calendar (e.g., outlook calendar).

1. The Trainee requests to **view** their schedule.
2. The GUI will get the Trainees schedule which will be returned from the Schedule object.
3. The GUI then **displays** the Schedule to the Trainee.
4. The Trainee requests to **sync** their schedule with their chosen calendar application.
5. The GUI will then sync the schedule and return a boolean from the Trainee object.
6. If the boolean returned is true the Sync was successful and a **complete message** will be **displayed**, otherwise, false will be returned and a **failed message** will be **displayed**.



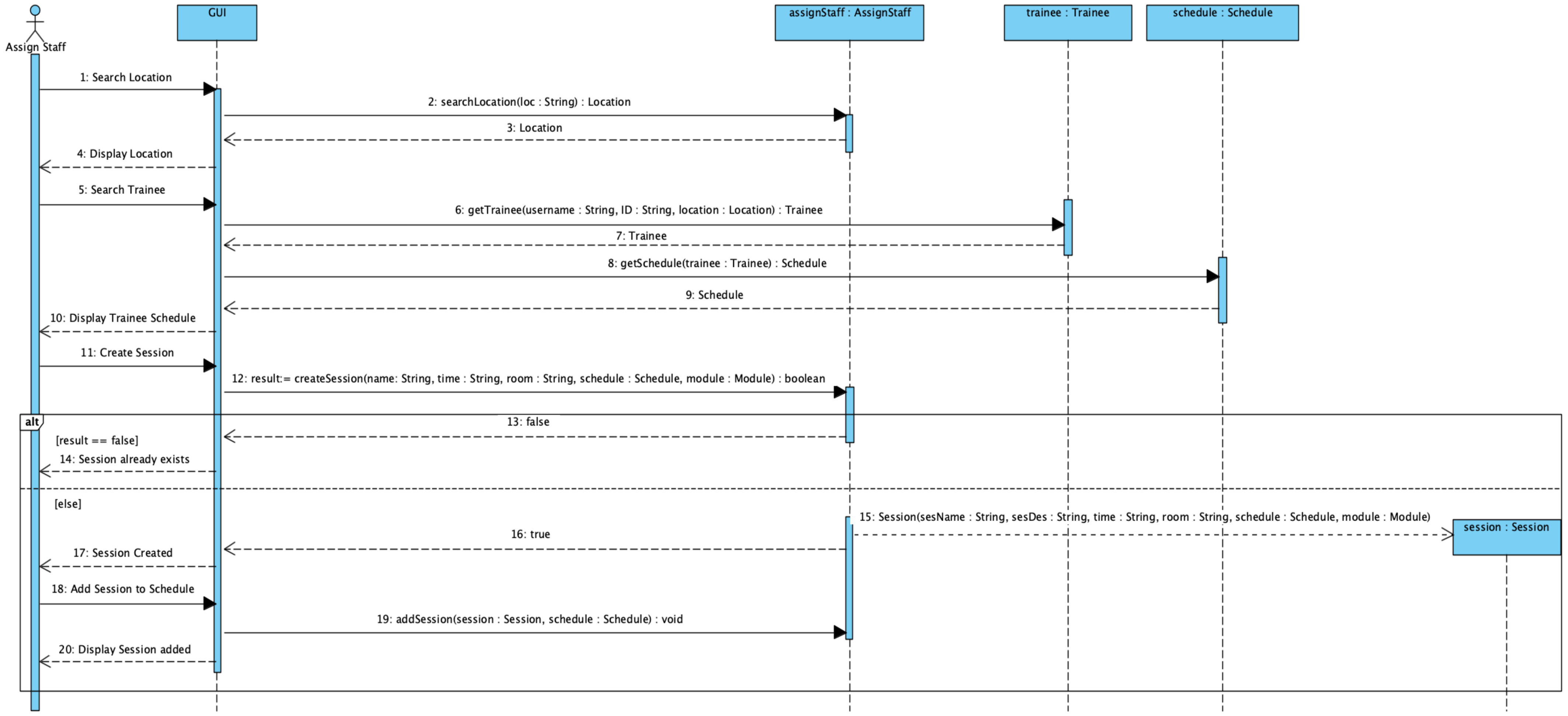
Visual Paradigm Standard(Bryan Neermul(Queen Mary University of London))
Trainee
: GUI
trainee : Trainee
schedule : Schedule
1: View Schedule
2: getSchedule(trainee : Trainee) : Schedule
3: Schedule
4: Display Schedule
5: Sync Schedule
6: syncSchedule() : boolean
7: boolean
alt
[True]
8: Display Complete Message
[False]
9: Display Failed Message

5. Sequence Diagram 2

Add Session Use Case

This sequence diagram will represent the Add Session use case for when an assign staff user is logged in to the system. It shows the tasks involved when the assign staff of the academy scheduling team will add a session to a trainee in the system.

1. An assign staff member makes a **search location** request to the GUI, with the name of the location.
2. GUI will get the correct search location object from the assign staff object.
3. The GUI then **display**s the **location** to the user.
4. The assign staff then **search**es for a specific **trainee** in that location using their username or FMD employee ID.
5. The GUI object then gets the correct trainee from the trainee object.
6. It then gets the schedule for that specific trainee from the schedule object and **display**s the **trainee schedule** to the user.
7. Assign staff user then requests GUI to **create** a **session**, with the name, room and time**.**
8. GUI calls the assign staff object where a check is performed to see if there already exists a session of the same parameters.
9. If the session already exists, then the assign staff object returns false, and GUI displays **session already exists** to the user.
10. Otherwise, a session object is created for that trainee's schedule and assign staff object returns true.
11. GUI lets the user know that a **session** has been **created**.
12. If the session is created, then the user requests to **add** the **session to** the trainee's **schedule.**
13. GUI calls assign staff object to add the session for the trainee's schedule.
14. Finally, GUI **display**s **session added** to the user.


Assign Staff
GUI
assignStaff : AssignStaff
trainee : Trainee
schedule : Schedule
1: Search Location
2: searchLocation(loc : String) : Location
3: Location
4: Display Location
5: Search Trainee
6: getTrainee(username : String, ID : String, location : Location) : Trainee
7: Trainee
8: getSchedule(trainee : Trainee) : Schedule
9: Schedule
10: Display Trainee Schedule
11: Create Session
12: result:= createSession(name: String, time : String, room : String, schedule : Schedule, module : Module) : boolean
alt
13: false
[result == false]
14: Session already exists
[else]
15: Session(sesName : String, sesDes : String, time : String, room : String, schedule : Schedule, module : Module)
session : Session
16: true
17: Session Created
18: Add Session to Schedule
19: addSession(session : Session, schedule : Schedule) : void
20: Display Session added