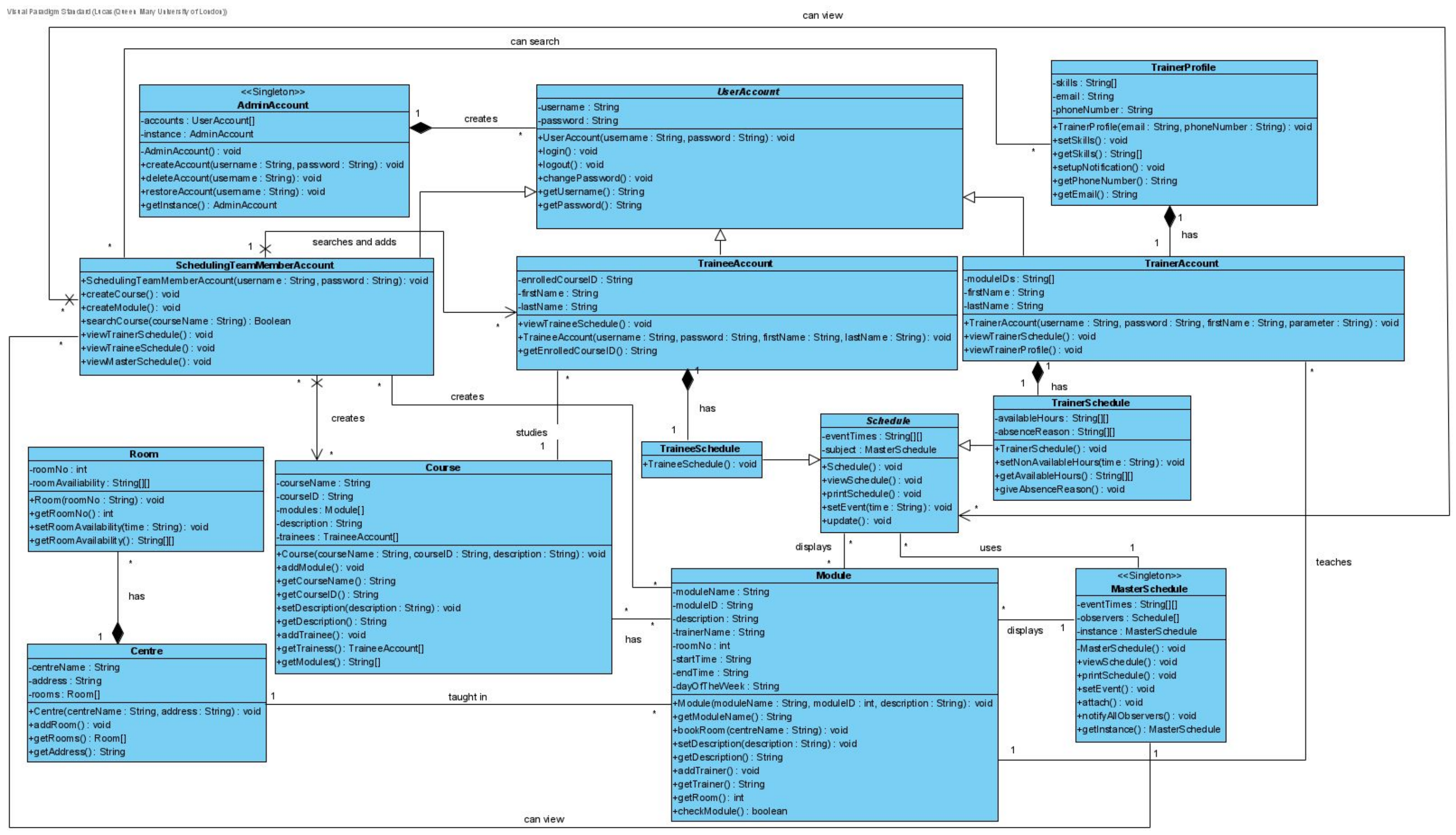
Group 1

FDM Trainer Skills and Availability

ECS506U Software Engineering Group Project

Design Report

1. Class Diagram (40%)


Visual Paradigm Standard (Lucas (Queen Mary University of London))
can view
can search
<<Singleton>>
AdminAccount
-accounts : UserAccount[]
-instance : AdminAccount
-AdminAccount() : void
+createAccount(username : String, password : String) : void
+deleteAccount(username : String) : void
+restoreAccount(username : String) : void
+getInstance() : AdminAccount
creates
UserAccount
-username : String
-password : String
+UserAccount(username : String, password : String) : void
+login() : void
+logout() : void
+changePassword() : void
+getUsername() : String
+getPassword() : String
TrainerProfile
-skills : String[]
-email : String
-phoneNumber : String
+TrainerProfile(email : String, phoneNumber : String) : void
+setSkills() : void
+getSkills() : String[]
+setupNotification() : void
+getPhoneNumber() : String
+getEmail() : String
has
searches and adds
SchedulingTeamMemberAccount
+SchedulingTeamMemberAccount(username : String, password : String) : void
+createCourse() : void
+createModule() : void
+searchCourse(courseName : String) : Boolean
+viewTrainerSchedule() : void
+viewTraineeSchedule() : void
+viewMasterSchedule() : void
TraineeAccount
-enrolledCourseID : String
-firstName : String
-lastName : String
+viewTraineeSchedule() : void
+TraineeAccount(username : String, password : String, firstName : String, lastName : String) : void
+getEnrolledCourseID() : String
TrainerAccount
-moduleIDs : String[]
-firstName : String
-lastName : String
+TrainerAccount(username : String, password : String, firstName : String, parameter : String) : void
+viewTrainerSchedule() : void
+viewTrainerProfile() : void
creates
creates
studies
has
has
TraineeSchedule
+TraineeSchedule() : void
Schedule
-eventTimes : String[][]
-subject : MasterSchedule
+Schedule() : void
+viewSchedule() : void
+printSchedule() : void
+setEvent(time : String) : void
+update() : void
TrainerSchedule
-availableHours : String[][]
-absenceReason : String[][]
+TrainerSchedule() : void
+setNonAvailableHours(time : String) : void
+getAvailableHours() : String[][]
+giveAbsenceReason() : void
Room
-roomNo : int
-roomAvailiability : String[][]
+Room(roomNo : String) : void
+getRoomNo() : int
+setRoomAvailability(time : String) : void
+getRoomAvailability() : String[][]
Course
-courseName : String
-courseID : String
-modules : Module[]
-description : String
-trainees : TraineeAccount[]
+Course(courseName : String, courseID : String, description : String) : void
+addModule() : void
+getCourseName() : String
+getCourseID() : String
+setDescription(description : String) : void
+getDescription() : String
+addTrainee() : void
+getTrainess() : TraineeAccount[]
+getModules() : String[]
displays
uses
teaches
has
has
Module
-moduleName : String
-moduleID : String
-description : String
-trainerName : String
-roomNo : int
-startTime : String
-endTime : String
-dayOfTheWeek : String
+Module(moduleName : String, moduleID : int, description : String) : void
+getModuleName() : String
+bookRoom(centreName : String) : void
+setDescription(description : String) : void
+getDescription() : String
+addTrainer() : void
+getTrainer() : String
+getRoom() : int
+checkModule() : boolean
displays
<<Singleton>>
MasterSchedule
-eventTimes : String[][]
-observers : Schedule[]
-instance : MasterSchedule
-MasterSchedule() : void
+viewSchedule() : void
+printSchedule() : void
+setEvent() : void
+attach() : void
+notifyAllObservers() : void
+getInstance() : MasterSchedule
Centre
-centreName : String
-address : String
-rooms : Room[]
+Centre(centreName : String, address : String) : void
+addRoom() : void
+getRooms() : Room[]
+getAddress() : String
taught in
can view

2. Traceability matrix (10%)

Class	Requirement	Brief Explanation
UserAccount	RQ20	This is an abstract class that all other accounts inherit from, it has username as an attribute and is set in the constructor.
UserAccount	RQ21	This is an abstract class that all other accounts inherit from, it has password as an attribute and is set in the constructor when the account is created, user password input is matched to validate logon.
Module	RQ27	The Module class contains a roomNo attribute that will correspond to the room number and will be stored in the module class.
Course	RQ28	The Course class has a CourseID attribute which will uniquely identify each course on creation, and has a description attribute to store the courses description.
Room	RQ29	The Room class is responsible for storing information on all unique rooms, the roomNo attribute identifies each room.
Module	RQ33	The Module class has a moduleID and description attribute to store the unique module id and the description of the module.
TraineeSchedule	RQ41	TraineeSchedule class provides the data for the trainees schedule
TrainerSchedule	RQ45	This class has an availableHours attribute responsible for storing the hours that the trainer is free to be booked for.

TrainerProfile	RQ46	The trainerProfile class has a skills attribute that the trainer will be able to edit to store an array of the skills they are proficient in.
TrainerProfile	RQ51	The TrainerProfile class is responsible for storing information about the trainer and has email and phoneNumber attributes for storing the trainers phone number and email.
Module	RQ57	The module class will have the room number stored where the module will be taking place, trainers will be able to view the module and find out what room it is taking place in.
TrainerSchedule	RQ58	The TrainerSchedule class has an absenceReason attribute that stores the reason the trainer is unavailable at a certain time, the absenceReason is mapped with the availableHours to show why and when the trainer is unavailable.
TraineeAccount	RQ63	Both of these classes have firstName, lastName attributes that must be declared on creation, will store the first names and last names of all trainers and trainees.

3. Design Discussion (20%)

Explanation of design decisions such as why you did include the attributes or ignored others in classes that represent key entities

Each course and module have a distinct courseID and moduleID attribute respectively. This ensures that each course and module created can be easily distinguished and identified. These classes also contain attributes: courseName and moduleName. This means that two of the same courses and or modules can co-exist while being distinguishable through their ID's. The UserAccount class contains both a username and password attribute, allowing each user to have distinct login details regardless of their job title. The unique usernames allow the system to identify which subclass of UserAccount is accessing the application. The system then allows the given user to access certain features depending on the functionalities they require.

The design includes numerous array attributes to group up moduleID's and courseID's with common associations and or features. For example, the TraineeAccount class possesses the moduleIDs attribute which is an array of strings that the given instance of TraineeAccount is assigned to teach. AdminAccount includes an array of UserAccount objects in order to allow the admin to access any current users' accounts. Each course object will hold references to the modules it consists of and the trainees that are enrolled. This is accomplished through an array of module objects and an array of TraineeAccount objects which are both stored in Course. Another crucial attribute of the Schedule class is eventTimes, which is a 2-dimensional array of strings. The first dimension will contain the time of day and the second dimension will hold the moduleID for which the event is reserved.

Briefly explain associations (including aggregations) in your diagram and justify their correctness (especially using analysis from your earlier reports):

UserAccount is the generalization of TraineeAccount, TrainerAccount, and SchedulingTeamMemberAccount meaning they inherit all attributes and methods from UserAccount. TrainerSchedule and TraineeSchedule are subclasses of Schedule and thus also inherit its methods and attributes. There is a composition between TraineeSchedule and TraineeAccount, as well as between TrainerAccount and TrainerSchedule. This means that without Account objects, there can be no schedule objects. This is due to the fact that each individual schedule corresponds to a specific Account object. This relationship also exists between Room and Center as well as AdminAccount and UserAccount. There is a many to many association between SchedulingTeamMemberAccount and Course called creates, except SchedulingTeamMemberAccount is not navigable. This means that Course cannot access SchedulingTeamMemberAccount but the opposite is true. There also exists an association named can view between SchedulingTeamMemberAccount and Schedule but once again SchedulingTeamMemberAccount is non-navigable meaning it cannot be accessed by Schedule.

Discussion on the differences between the entities in domain model and class diagram (if any differences):

As the team spent more time developing the software, we noticed a number of adjustments and additions that needed to be made between our domain model and class diagram. Following our first client meeting with an FDM Group employee, we were informed of the importance of an entity and primary user that we initially chose not to include in the domain model: the trainee/consultant. This refers to the students of the modules that will be taught by trainers. The primary functionality they will be using is viewing their own individual schedules which contain all modules from their enrolled courses.

Our initial domain model included a general schedule entity and a specialization of that class in the form of "TrainerSchedule". We edited the schedule class so that there are 3 specializations: "TrainerSchedule", "TraineeSchedule" and "MasterSchedule", all possessing different operations and or attributes. TrainerSchedule will be capable of viewing their own individual schedules, setting the days and hours in which they are unavailable based on other job responsibilities, and submitting absence excuses. TraineeSchedule will only display their own schedules. MasterSchedule however, displays all modules from all courses at a given academic centre.

An aspect of this trainer skills and availability application we did not take into account during the domain analysis was where the modules were going to take place. Therefore, in the construction of the class diagram we chose to include the "Centre" class and "Room" class. These academic centres consist of rooms that must be reserved during the add module use case. Otherwise, modules can be hosted online on platforms such as Zoom or Microsoft Teams.

Discuss any design patterns you applied. Why did you apply that certain pattern and explain why you think that your design is a correct application of this pattern?

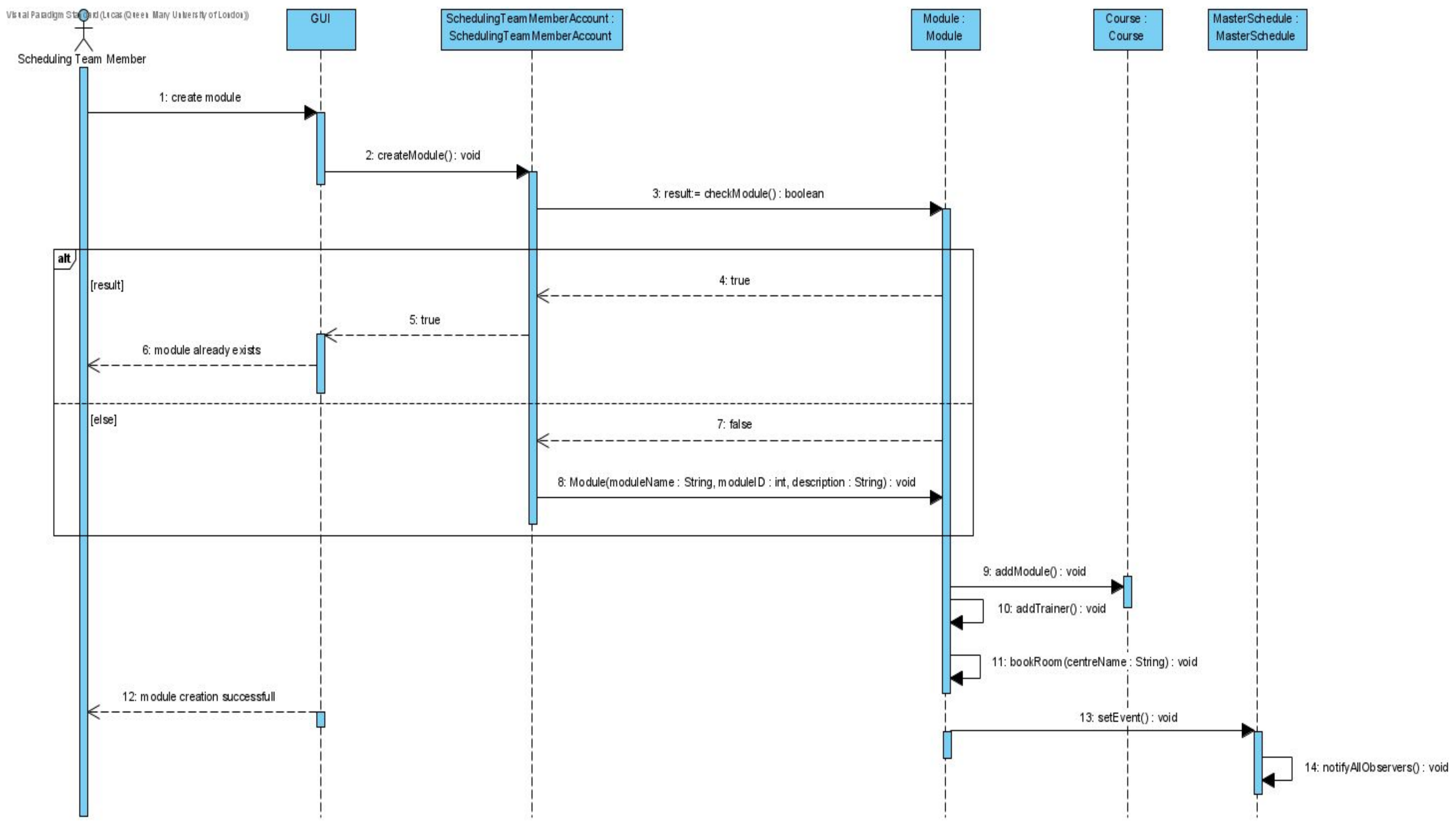
We implemented two design patterns in our construction of the class diagram: the Singleton pattern and the Observer pattern. The singleton pattern was especially crucial in its application in the adminAccount and MasterSchedule classes as only one instance of those objects can exist. With that said, we implemented the pattern by creating private static instances of AdminAccount and MasterSchedule. We included private constructors to ensure instances could not be created from outside of the class. Finally in order to provide access to the instances we created public methods to get instances of AdminAccount and MasterSchedule.

The second design pattern we used to enhance the structure of our system was the observer pattern. When a module is created, designated certain time slots and added to the MasterSchedule, the appropriate individual Trainee and Trainer schedules must be notified and updated accordingly. In the basic observer pattern template, the MasterSchedule class would be the subject while the observer would be the Schedule class and hence its subclasses. We ensured these roles by creating the private attribute: subject of type MasterSchedule in the Schedule class and the private attribute: observer of type array of Schedule objects. Whenever a TraineeAccount or TrainerAccount instance is created, it is added to the observers attribute in MasterSchedule which is an array of Schedule objects. Therefore when a change is made in MasterSchedule, the public notifyAllObservers method in MasterSchedule is called which calls the public update method in Schedule. This method updates the appropriate TraineeSchedule and TrainerSchedule instances with the changes in their modules.

4. Sequence Diagram 1 (15%)

This sequence diagram represents the steps in the process of adding a module to an already existing course.

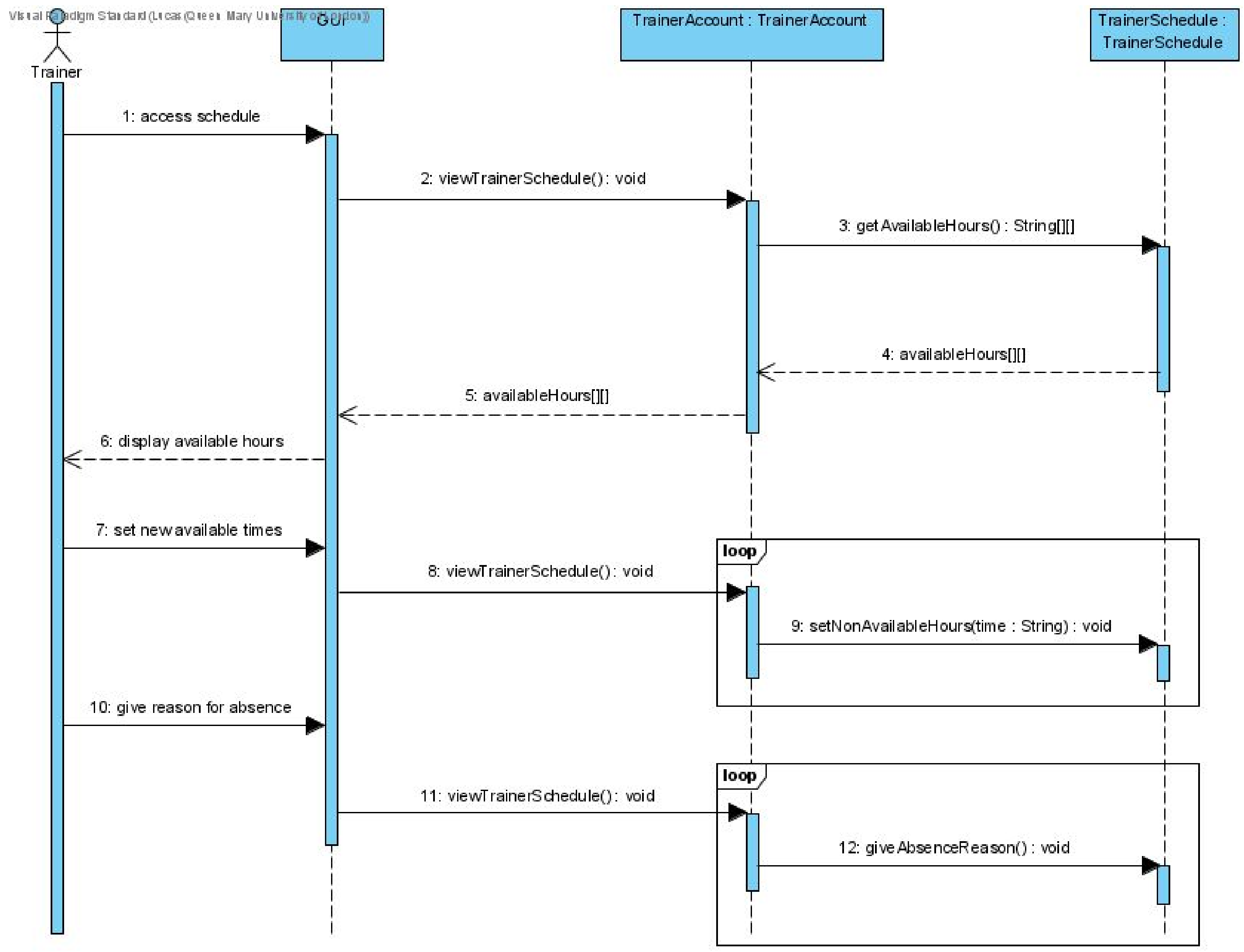
The prerequisites are that the scheduling team member already has access to a Scheduling Team Member Account and that the related course already exists


Visual Paradigm Standard (Lucas (Queen Mary University of London))
Scheduling Team Member
GUI
SchedulingTeamMemberAccount : SchedulingTeamMemberAccount
Module : Module
Course : Course
MasterSchedule : MasterSchedule
1: create module
2: createModule() : void
3: result:= checkModule() : boolean
alt
[result]
4: true
5: true
6: module already exists
[else]
7: false
8: Module(moduleName : String, moduleID : int, description : String) : void
9: addModule() : void
10: addTrainer() : void
11: bookRoom(centreName : String) : void
12: module creation successfull
13: setEvent() : void
14: notifyAllObservers() : void

5. Sequence Diagram 2 (15%)

This sequence diagram represents the steps in order for a trainer to declare a time and reason for non availability in a time slot of their timetable.

The only prerequisite is that the trainer already has access to a Trainer Account.


Trainer
GUI
TrainerAccount : TrainerAccount
TrainerSchedule : TrainerSchedule
1: access schedule
2: viewTrainerSchedule() : void
3: getAvailableHours() : String[][]
4: availableHours[][]
5: availableHours[][]
6: display available hours
7: set new available times
loop
8: viewTrainerSchedule() : void
9: setNonAvailableHours(time : String) : void
10: give reason for absence
loop
11: viewTrainerSchedule() : void
12: giveAbsenceReason() : void