Assumptions made about the system:

I have assumed that teaching staffs (professors and lecturers) are not required to make any actions through the system except that they can check into the system using their University id and check the appointments that were booked by researchers and visitors. Researchers use the system to display the appointments booked by visitors and to search for teaching staffs to book appointments. Students use the system to search for classes offered by teaching staffs, and to sing up in the classes. They can also see the classes that they have signed up for after entering the id in the system. Students are not required to make any appointments, and they have no related actions with researchers and visitors. There is no related action between researchers and visitors; and between professors and lecturers. Classes have maximum capacity of ten students. Teaching staffs have fixed appointment hours Monday through Friday during the term of 3 months. But each teaching staff has different appointment hours. Teaching staffs are busy professionals, so they have appointment hours fixed throughout the term. When a visitor is booking the appointment, full name and email must be filled out so that teaching staffs and researchers can communicate with them to cancel the appointment. The system doesn't have functionalities for canceling a registered class by students and canceling the appointment by teaching staffs, researchers and visitors. I have assumed that two visitors cannot make appointment at the same time with the same teaching staff or the same researcher. Same rule holds for the researchers for making appointments.

Only last name is required for 'SEARCH BY NAME' (Step3). Alphabetic inputs (ID, Expertise Search, Name search, class code) must be in lowercase letter. Appointment string must exactly match the one listed to the user. User can just copy and paste the appointment date and time to the input text field to check out. For booking appointment, 'SEARCH BY EXPERTISE' only returns the teaching staff or researcher. To check the available appointments, user must enter name and click 'SEARCH BY NAME' again. For Student, 'SEARCH BY EXPERTISE' will display a list of teaching staffs as well as available classes offered by teaching staffs in two different text panes.

The overall structure and design of your program:

Use Cases

Student:

- Chooses 'Student' type from menu.
- Enters University ID
- Searches teaching staffs by Expertise and sees a list of classes offered by teaching staffs under an expertise
- Searches By name and sees a list of classes offered by a teaching staff
- Enters class code and sign up
- Exits University UI

Researcher:

- Chooses 'Researcher' type from menu.
- Enters University ID
- Searches teaching staffs by Expertise and sees a list of teaching staffs under an expertise
- Searches By name and sees a list of available appointments of teaching staff.
- Enters a string of date and time to book an appointment
- Exits University UI

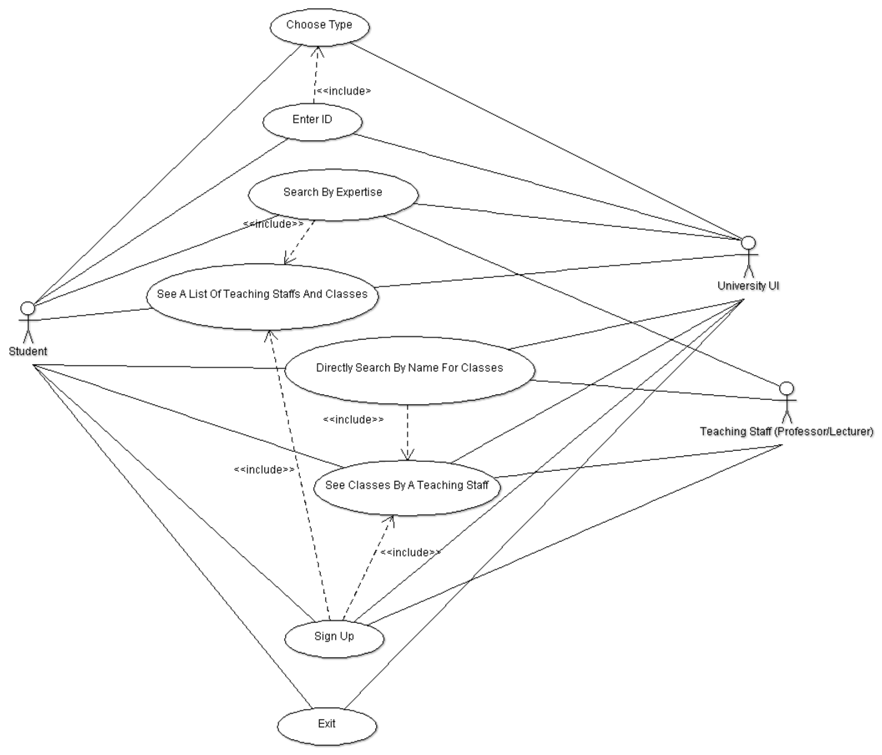

Choose Type
<<include>
Enter ID
Search By Expertise
<<include>>
See A List Of Teaching Staffs And Classes
Student
University UI
Directly Search By Name For Classes
Teaching Staff (Professor/Lecturer)
<<include>>
<<include>>
See Classes By A Teaching Staff
<<include>>
Sign Up
Exit

Figure 1: UML use-case diagram for Student

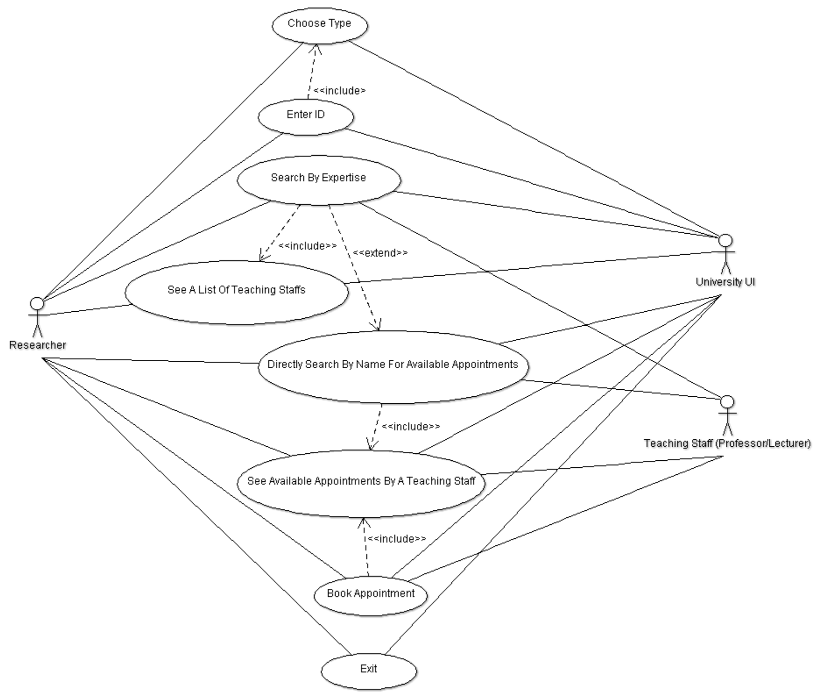

Choose Type
<<include>
Enter ID
Search By Expertise
<<include>>
<<extend>>
See A List Of Teaching Staffs
Researcher
University UI
Directly Search By Name For Available Appointments
Teaching Staff (Professor/Lecturer)
<<include>>
See Available Appointments By A Teaching Staff
<<include>>
Book Appointment
Exit

Figure 2: UML use-case diagram for Researcher

Visitor:

- Chooses ‘Visitor’ type from menu.
- Enters name and email
- Searches teaching staffs by Expertise and sees a list of teaching staffs under an expertise
- Searches By name and sees a list of available appointments of teaching staff.
- Enters a string of date and time to book an appointment
- Exits University UI

Teaching Staff (Professor and Lecturer)

- Chooses ‘Professor/Lecturer’ type from menu.
- Enters University ID
- Checks for booked appointments
- Exits University UI

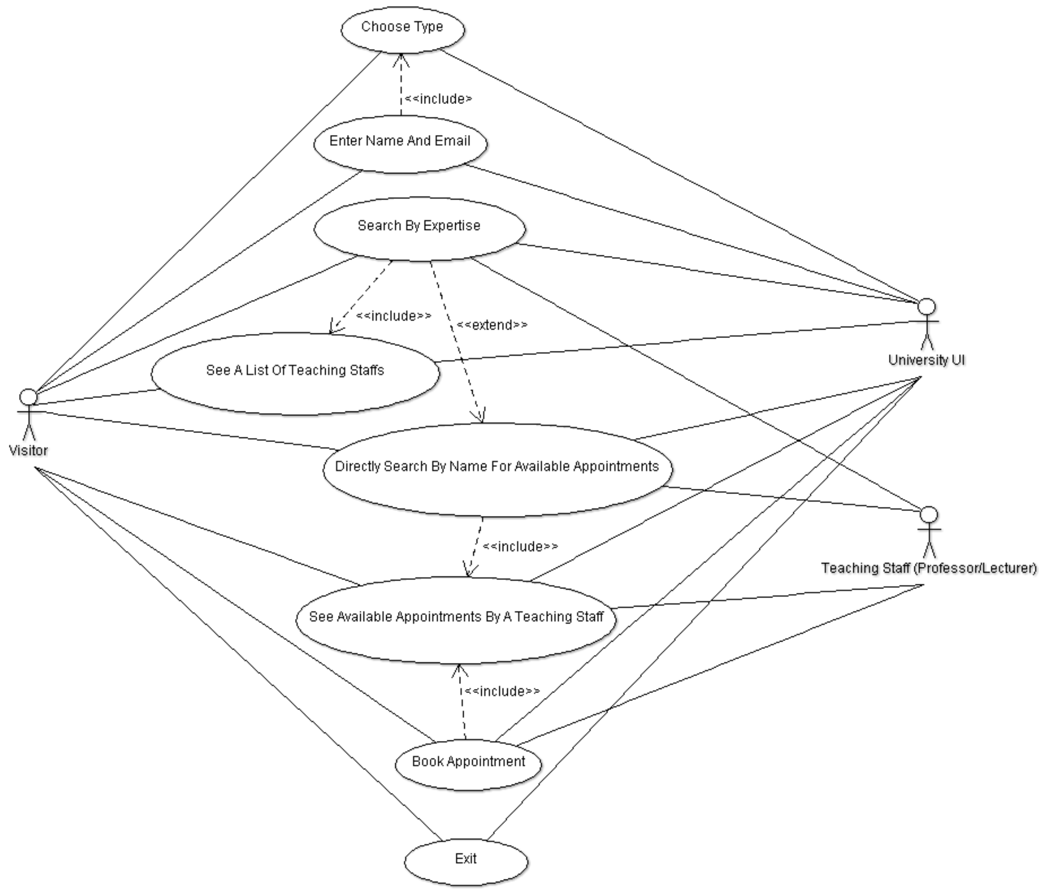

Choose Type
<<include>
Enter Name And Email
Search By Expertise
<<include>>
<<extend>>
See A List Of Teaching Staffs
University UI
Visitor
Directly Search By Name For Available Appointments
<<include>>
Teaching Staff (Professor/Lecturer)
See Available Appointments By A Teaching Staff
<<include>>
Book Appointment
Exit

Figure 3: UML use-case diagram for Visitor

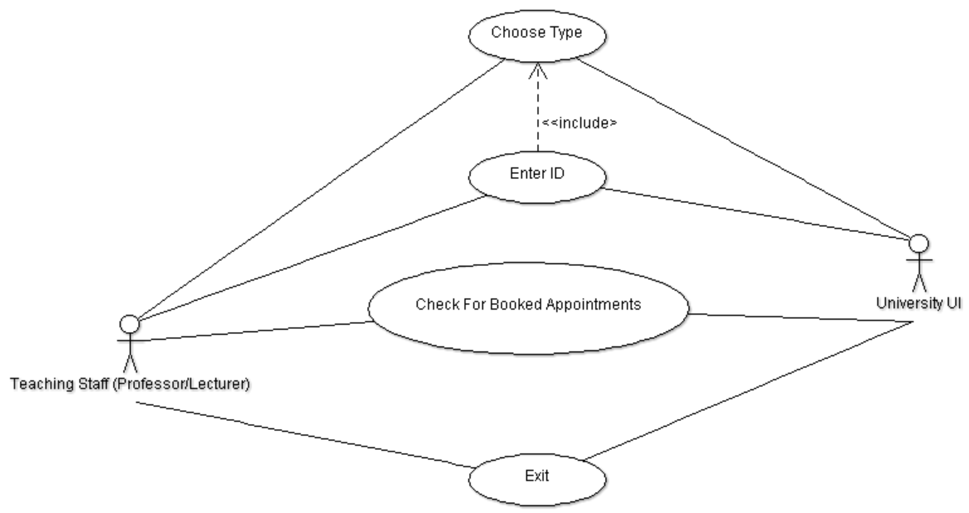

Choose Type
<<include>
Enter ID
Check For Booked Appointments
University UI
Teaching Staff (Professor/Lecturer)
Exit

Figure 4: UML use-case diagram for Teaching Staff

University member was created as an Abstract Super class which is inherited by concrete classes of Professor, Lecturer, Researcher and Student because they have ‘is a’ relationship with UniversityMember with common attributes (unique id, full name, address, email).

StaffProperty class is defined with attributes (office room, expertiseList, availableAppointments) which belong to Professor, Lecturer and Researcher. It is ‘has-a relationship’ that they have with StaffProperty and compositon principle is used. They are staff members who implement StaffMember interface. MyClasses is defined with attributes class name, room location, class schedule (day and time) and room capacity.

I used Thread to run two tasks at run; taskA to set up the university members in the system and store these objects in UniversityMemberDatabase and taskB to run the front-end GUI part.

Class Diagram:

Class diagram must be zoomed to in read the text clearly.

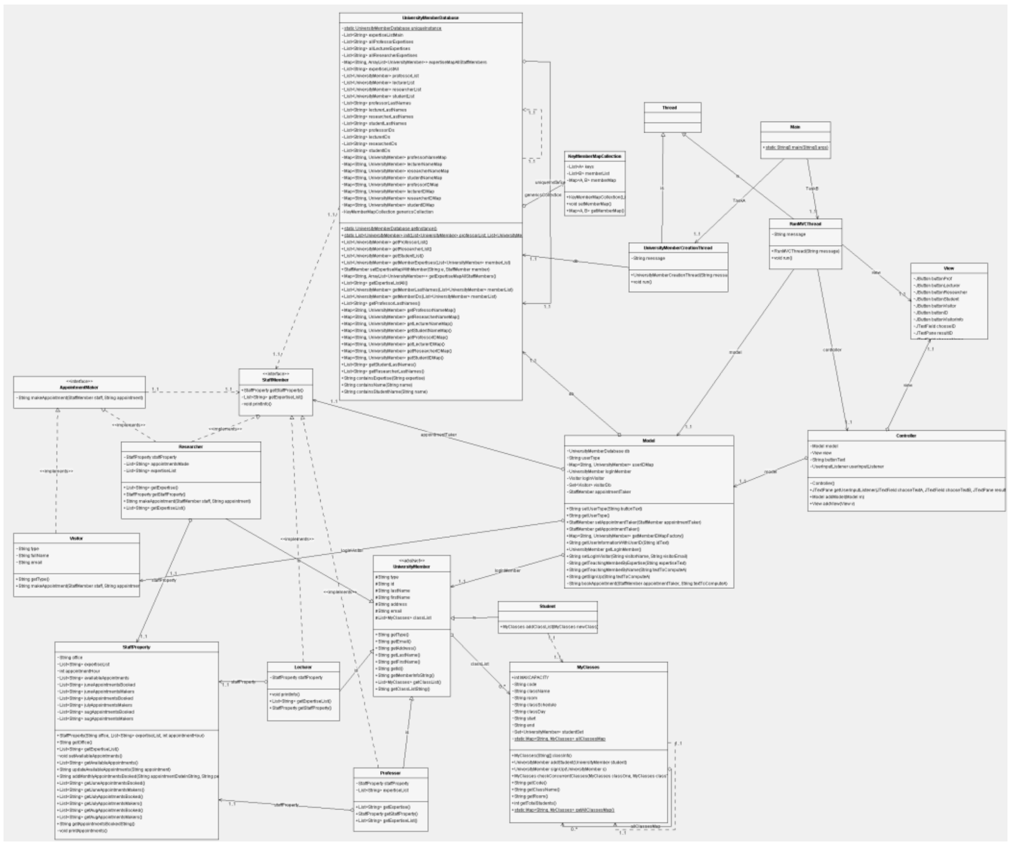


Figure 5: Class Diagram

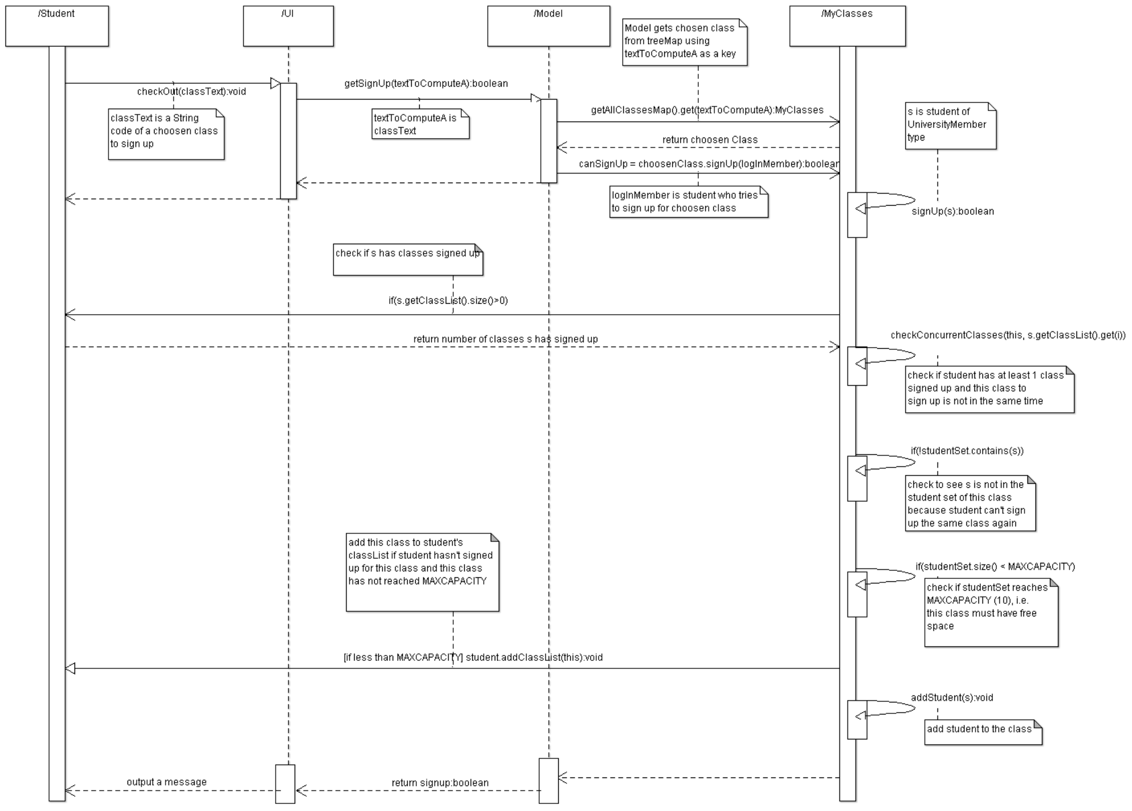

/Student
/UI
/Model
/MyClasses
Model gets chosen class from treeMap using textToComputeA as a key
checkOut(classText):void
getSignUp(textToComputeA):boolean
getAllClassesMap().get(textToComputeA):MyClasses
classText is a String code of a choosen class to sign up
textToComputeA is classText
s is student of UniversityMember type
return choosen Class
canSignUp = choosenClass.signUp(logInMember):boolean
logInMember is student who tries to sign up for choosen class
signUp(s):boolean
check if s has classes signed up
if(s.getClassList().size()>0)
return number of classes s has signed up
checkConcurrentClasses(this, s.getClassList().get(i))
check if student has at least 1 class signed up and this class to sign up is not in the same time
if(!studentSet.contains(s))
check to see s is not in the student set of this class because student can't sign up the same class again
add this class to student's classList if student hasn't signed up for this class and this class has not reached MAXCAPACITY
if(studentSet.size() < MAXCAPACITY)
check if studentSet reaches MAXCAPACITY (10), i.e. this class must have free space
[if less than MAXCAPACITY] student.addClassList(this):void
addStudent(s):void
add student to the class
output a message
return signup:boolean

Figure 6: Sequence Diagram

When the system displays the classes including number of free spaces to sign up to a Student. Student can sign up the class by entering a class code to the UI.

Design patterns/design principles used

Singleton pattern is used for creating an instance of UniversityMemberDatabase because this instance is created when setting up university members in the UniversityMemberCreation thread and the same instance is invoked when a person tries to use the system. Using Singleton pattern prevents creating another instance of UniversityMemberDatabase.

Factory Method is used in getMemberIDMapFactory method in Model to assign desired instance of UniversityMember to logInMember. Factory method is also used in getTeachingMemberByName.

Model-View-Controller pattern is used to develop the system. Observer pattern is used between Model and View where Model is Observable and View is Observer. For event handling in GUI, Listener pattern with ActionListener interface is used. Controller has an inner class, UserInputListener which implements ActionListener to handle the user input events in the system.

SwingWorker is used to perform the computation when user enters an input in GUI. Controller has an inter class of ComputeWorker which extends SwingWorker.

Strategy Pattern is used among Professor, Lecturer and Researcher who have instance of StaffProperty and attribute of expertiseList. These concrete classes implement StaffMember Interface which has two abstract methods; getStaffProperty and getExpertiseList. Strategy Pattern is also used for Researcher and Visitor who make appointments with Teaching Staff; so, concrete classes, Researcher and Visitor, implements AppointmentMaker Interface which has makeAppointment method to be overridden.

Refactoring used during the development of the system

Creating university members such as ten students required refactoring. A method, createStudent(), was implemented to avoid duplicate codes (DRY). Refactoring was also performed during the implementation to get last names of the members from four different collections, and getMemberLastNames() was implemented to get lastNames of these members instead of looping over four different member lists.

Generics is used in KeyAndMemberMapCollection to remove duplicate codes while creating a map of key and member pairs. Generics collection was refactored, so it can accept keys of names or ids and values of members pair and return the Map.

There were many arguments passed for creating university members and when there was a change on these arguments, changes had to be made at many locations. For this, String arguments are passed in an array. Previously, there were multiple listener classes such as GetUserTypeListener, GetUserIDListener, GetVisitorInfoListener, etc. But I have refactored the Controller so that there is only one Listener class, UserInputListener which implements ActionListener.

The overall process you followed in developing the software

I have followed through a series of the phases, gradually adding more features to the program. At first, I started the back-end for creating university members, then I added complex features afterwards. I have followed Test-Driven Development in developing the software. I added the front-end GUI part as the last step.

UnitTest

UniversityMemberDatabaseTest

testGetMemberExpertises(): compares an expertise with the return value of getMemberExpertises() which is called with professorList, lecturerList or researcherList. It also tests the size of the collections.

testSetExpertiseMapWithMember(): compares instance of professor, lecturer or researcher with the return value of getExpertiseMapAllStaffMembers().

testGetExpertiseListAll(): a given expertise, containsExpertise() should return true. If an expertise is not in the list, the method returns false.

testGetMemberLastNames(): a given last name, containsName() should return true. If last name is not in the list, the method returns false.

testGetProfessorNameMap(): compares an instance of Professor with the return value of getProfessorNameMap() method. It also tests the size of the map.

testGetLectureNameMap(): compares an instance of Lecturer with the return value of getProfessorNameMap() method. It also tests the size of the map.

testGetResearcherNameMap(): compares an instance of Researcher with the return value of getProfessorNameMap() method. It also tests the size of the map.

testGetStudentNameMap(): compares an instance of Student with the return value of getProfessorNameMap() method. It also tests the size of the map.

testGetProfessorIDMap(), testGetLecturerIDMap(), testGetResearcherIDMap(), testGetStudentIDMap(): these tests perform the same manner as previous 4 tests but using the IDs.

StaffPropertyTest

testGetOffice(): tests if staffProperty.getExpertiseList() returns the correct office.

testGetAvailableAppointments(): compares a string of Appointment date and time with the return value of staffProperty.getAvailableAppointments().get(i). It also tests the size of the collection where available appointments are stored.

testSetAvailableAppointments(): performs the same way as previous test.

testAddMonthlyAppointments(): sets up an appointment by calling staffProperty.addMonthlyAppointmentsBooked("2018 Jun 15 09:00", "Steve Pellerin") and it compares size of the collection by calling staffProperty.getJuneAppointmentsBooked().size(). It also test the appointment maker with the return value of staffProperty.getJuneAppointmentsMakers().get(i).

testUpdateAvailableAppointments(): calls the method as staffProperty.updateAvailableAppointments("2018 Jun 01 09:00"); which remove the appointment from the collection. Then, it tests the size of the collection to check if the size has reduced. It also checks if the appointment following the removed appointment has moved up by an index in the collection.

KeyMemberMapCollectionTest

testSetMemberMap(): sets up the instance of UniversityMemberDatabase and creates an instance of KeyMemberMapCollection and checks the size of the map, if the instance of university member, which was added in the database and member map, contains in the map.

MyClassesTest()

testSignUp(): tests if a given student can sign up in class, tests if a student can sign up a class which student already signed up. It returns true if it is the first time that student is signing up for the class; otherwise, it returns false. It also checks to make sure that a student can't sign up for a class that overlaps with a class that he has signed up.

testAddStudent(): checks if a student was added in the student list of the class that student was able to sign up.

testCheckConcurrentClasses(): returns true if two given classes overlap each other. It returns false if two classes occur the different times.

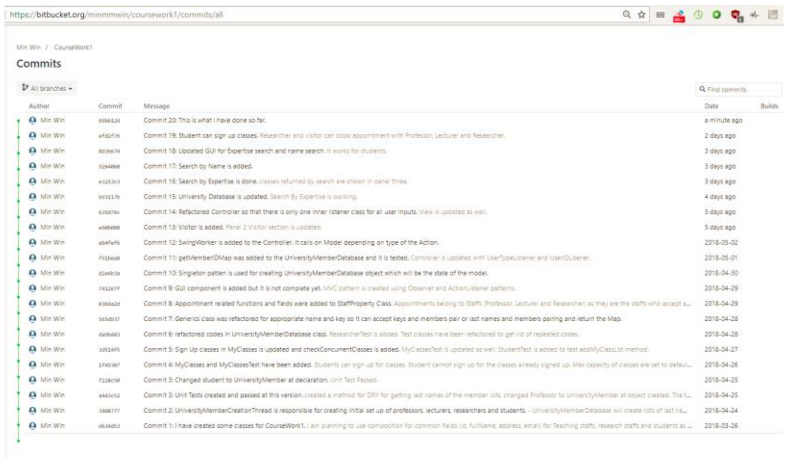
testGetClassName(), testGetClassRoom(), testGetClassSchedule(): tests string of name, room, schedule of a given class with the return value of getters.

testGetAllClassesMap(): Classes are stored in a map using class code as the key and class itself as the value. It checks the size of the map where classes are stored, checks if a given class is inside the map by calling with a key.

StudentTest

testAddClassList(): tests if a student was able to add a class, and checks the size of the class list and name of the class that student added.

Commit Log



https://bitbucket.org/minmmwin/coursework1/commits/all

Min Win / CourseWork1

Commits

All branches

Author	Commit	Message	Date	Builds
Min Win	69b6124	Commit 20: This is what I have done so far.	a minute ago	
Min Win	efd2f76	Commit 19: Student can sign up classes. Researcher and visitor can book appointment with Professor, Lecturer and Researcher.	2 days ago	
Min Win	8336674	Commit 18: Updated GUI for Expertise search and name search. It works for students.	3 days ago	
Min Win	9264868	Commit 17: Search by Name is added.	3 days ago	
Min Win	e3253c3	Commit 16: Search by Expertise is done. classes returned by search are shown in panel three.	3 days ago	
Min Win	b93117b	Commit 15: University Database is updated. Search By Expertise is working.	4 days ago	
Min Win	635d76c	Commit 14: Refactored Controller so that there is only one inner listener class for all user inputs. View is updated as well.	5 days ago	
Min Win	e6b6488	Commit 13: Visitor is added. Panel 2 Visitor section is updated.	5 days ago	
Min Win	eb4fef6	Commit 12: SwingWorker is added to the Controller. It calls on Model depending on type of the Action.	2018-05-02	
Min Win	f51be68	Commit 11: getMemberIDMap was added to the UniversityMemberDatabase and it is tested. Controller is updated with UserTypeListener and UserIDListener.	2018-05-01	
Min Win	d2e4b3a	Commit 10: Singleton patten is used for creating UniversityMemberDatabase object which will be the state of the model.	2018-04-30	
Min Win	7412677	Commit 9: GUI component is added but it is not complete yet. MVC pattern is created using Observer and ActionListener patterns.	2018-04-29	
Min Win	03b4a2d	Commit 8: Appointment related functions and fields were added to StaffProperty Class. Appointments belong to Staffs (Professor, Lecturer and Researcher) as they are the staffs who accept a...	2018-04-29	
Min Win	565d037	Commit 7: Generics class was refactored for appropriate name and key so it can accept keys and members pair or last names and members pairing and return the Map.	2018-04-28	
Min Win	da9b483	Commit 6: refactored codes in UniversityMemberDatabase class. ResearcherTest is added. Test classes have been refactored to get rid of repeated codes.	2018-04-28	
Min Win	3d534f5	Commit 5: Sign Up classes in MyClasses is updated and checkConcurrentClasses is added. MyClassesTest is updated as well. StudentTest is added to test addMyClassList method.	2018-04-27	
Min Win	1f49387	Commit 4: MyClasses and MyClassesTest have been added. Students can sign up for classes. Student cannot sign up for the classes already signed up. Max capacity of classes are set to defaul...	2018-04-26	
Min Win	f228c58	Commit 3: Changed student to UniversityMember at declaration. Unit Test Passed.	2018-04-25	
Min Win	a443c52	Commit 3: Unit Tests created and passed at this version. created a method for DRY for getting last names of the member lists. changed Professor to UniversityMember at object created. The t...	2018-04-25	
Min Win	3488777	Commit 2: UniversityMemberCreationThread is responsible for creating initial set up of professors, lecturers, researchers and students. - UniversityMemberDatabase will create lists of last na...	2018-04-24	
Min Win	d634d53	Commit 1: I have created some classes for CourseWork1. I am planning to use composition for common fields (id, fullName, address, email) for Teaching staffs, research staffs and students as ...	2018-03-26	