DATABASE DIAGRAM

SOFTWARE DIAGRAM SPRINT 2

STATE DIAGRAM

SPRINT 2

UNIVERSAL

applies to ALL users...

ADMIN

INSTRUCTOR

TA

BEGIN USER

User.login(id, password)

Preconditions: given an ID that exists in the system and the associated password for that ID
Postconditions: opens application for the role associated with the given ID. Error message displayed if invalid username or password. Return boolean: true if matching password/username in database and false otherwise.
Side-Effects: ---
ID: input – an integer to uniquely identify each user Password: input – a string to unlock the application for each unique ID.

User.getUsername()

Preconditions: name fields for a user have been correctly validated in the constructor (they are not blank)
Postconditions: No change to host object, an email address (string) which is used as the Username
Side-Effects: ---

User.getPassword()

Preconditions: password fields for a user have been correctly validated in the constructor (the password is more than 4 characters long (minimum requirement))
Postconditions: No change to host object, returns a string representing the password for the User's login.
Side-Effects: ---

User.getName()

Preconditions: name fields for a user have been correctly validated in the constructor (they are not blank)
Postconditions: No change to host object, returns a dictionary with first name and last name.
Side-Effects: ---

User.getRole()

Preconditions: must exist TA & Instructor objects.
Postconditions: No change to host object, return a String representing what the user represents (TA, or instructor, admin)
Side-Effects: ---

User.editMyContactInfo(contactDict)

Preconditions: Any assignments to the phone number key inside the dictionary must be 10 numbers, they user does not necessarily have to provide “all” fields.
Postconditions: If valid changes, the user’s contact information is set to the input fields, all assignments to this user will be updated, the GUI will display all the changes. If invalid inputs, it will display an error message.
Side-Effects: ---
contactDict: input – dictionary contains keys named after all of the user’s fields who’s values are used to update the user’s fields.

BEGIN ADMIN

Admin.createCourse(courseInfo)

Preconditions: The course to be created cannot contain an ID of an already existing course/section. All fields required for a course must be provided in the argument.
Postconditions: Course with empty assignments and no sections is created. It is displayed in the GUI and users can now be assigned to it. If provided duplicate ID or missing inputs, displays error message.
Side-Effects: ---
courseInfo: input – dictionary: keys are the course fields, values are used to enter the course’s specific information for each field.

Admin.createUser(userInfo)

Preconditions: The user to be created cannot contain an ID of an already existing User. All fields required for an user must be provided in the argument.
Postconditions: User with empty assignments is created, the user can now be assigned to courses. If provided duplicate ID, bad phone number, missing inputs, displays error message.
Side-Effects: ---
userInfo: input – dictionary: keys are the user’ fields, values are used to enter the user’s specific information for each field.

Admin.createSection(sectionInfo)

Preconditions: The section to be created cannot contain an ID of an already existing course/section, the provided host course must also exist. All fields required for a section must be provided in the argument.
Postconditions: Section with empty assignments is created. It is displayed in the GUI and users can now be assigned to it. If provided duplicate section ID, missing inputs, or invalid host course, it displays error message.
Side-Effects: ---
sectionInfo: input – dictionary contains keys named after all of the sections’ fields who’s values are used to enter the section’s fields.

Admin.removeCourse(activeCourse)

Preconditions: The course trying to remove must exist.
Postconditions: the course will be removed from the schedule of classes GUI, and no user can be assigned to it. If provided non-existent course, displays error message.
Side-Effects: The respective sections and user assignments to this course will also be remove
activeCourse: in out - This course will be used to in a search to traverse the courses table to remove. The argument's state will be mutated: all user assignments/sections to this course are removed.

Admin.removeUser(activeUser)

Preconditions: The account trying to remove must exist.
Postconditions: the account will be removed from the schedule of classes GUI, and no course/section can link to this user. If provided non-existent account, displays error message.
Side-Effects: The respective course/section assignments to this user will also be removed.
activeUser: in-out - This user will be used to in a search to traverse the user table to remove. The parameter's state will be mutated: all course/section assignments are removed.

Admin.removeSection(activeSection)

Preconditions: The section trying to remove must exist.
Postconditions: The section will be removed from the schedule of classes GUI, and no user can be assigned to it. If provided non-existent section, displays error message.
Side-Effects: The respective course and user assignments to this course will also be removed.
activeSection: in-out -This section will be used to in a search to traverse the section table to remove. The argument's state will be mutated: all user assignments/course to this section are removed.

Admin.editCourse(activeCrse, newInfo)

Preconditions: the course to edit must exist in the system and the course's input meets formatting requirements for contact information (number of labs must be int, no missing fields)
Postconditions: valid changes to course's state are saved, all GUI elements with this course are updated, user/sections also reflect any changes. If provided non-existent course, or wrong input, displays error message.
Side-Effects: ---
activeCrse: output – This course's information, state, will be modified according to the user's input. newInfo: input – contains the information to apply to the course

Admin.editSection(activeSection)

Preconditions: the section to edit must exist in the system and the section's input meets formatting requirements for contact information (no missing fields)

Postconditions: valid changes to section's state are saved, all GUI elements with this section's are updated, user/course's also reflect any changes. If provided non-existent section, displays error message.
Side-Effects: ---
activeSection: output – This section's information, state, will be modified according to the user's input.

Admin.editUser (activeUser)

Preconditions: the account to edit must exist in the system and the user's input meets formatting requirements for contact information
Postconditions: valid changes to account's state are saved, all GUI elements with this account are updated, course's/sections also reflect any changes. If provided non-existent user, displays error message.
Side-Effects: ---
activeUser: output – This account's information, state, will be modified according to the user's input.

Admin.courseUserAsgmt(activeUser, activeCourse)

Preconditions: The user must exist in the system, the user has not exceeded their maximum course assignments, must be an user.
Postconditions: The user will receive the course assignment and the course will receive the user assignment. If max capacity instructor, nonexistent instructor/course provided, user provided not a user, then raise error message.
Side-Effects: ---
activeUser: in – out: the user is used as an input for the course's user field and the user itself will be assigned to a course. activeCourse: in – out: used for assigning user to

Admin.secUserAsgmt(activeUser, activeSec)

Preconditions: The User must exist in the system, user can't already be assigned to section, the user (if TA) has appropriate grader status, and the section doesn't already have a user.
Postconditions: The user will receive the section assignment and the section will receive the user assignment. If nonexistent user/section, full section, wrong grader status then error message raised.
Side-Effects: ---
activeUser: in–out: the user is used as an input for the section's user field and the user itself will be assigned to the section. activeSec: in–out: the section is used as an input for the user field and the user itself will be assigned to the section.

BEGIN TA

TA.hasMaxAsgmts ()

Preconditions: Maximum course assignment field has been instantiated in constructor (it is non-negative real number) and any assignments to this TA must be reflected accurately in the TA's state.
Postconditions: returns boolean whether the user has reached it's maximum course assignments.
Side-Effects: ---

TA.getTACrseAsgmts()

Preconditions: The TA must contain at least one assignment to a course and the TA's assignments to courses reflect any assignments from courses to this TA.
Postconditions: Returns a list of the Courses assigned to this TA. Displays if no current assignments
Side-Effects: ---

TA.getTASecAsgmts()

Preconditions: The TA must contain at least one assignment to a section and the TA's assignments to section reflect any assignments from section to this TA.
Postconditions: Returns a list of the section assigned to this TA. Displays if no current assignments.
Side-Effects: ---

TA.getGraderStatus()

Preconditions: Unique TA has been instantiated.
Postconditions: Returns whether or not the TA has been assigned a grader status.
Side-Effects: ---

TA.setSkills (skills)

Preconditions: Skills must not be empty
Postconditions: set the TAs skills the arguments
Side-Effects: ---
Skills: in – used to set the TAs skill field to this string.

BEGIN INSTRUCTOR

Instructor.hasMaxAsgmts ()

Preconditions: Maximum course assignment field has been instantiated in constructor (it is non-negative real number) and any assignments to this Instructor must be reflected accurately in the Instructor state
Postconditions: returns whether the instructor has reached it's maximum course assignments.
Side-Effects: ---

Instructor.getInstrCrseAsgmts()

Preconditions: Any assignments, from a course to this instructor, must be reflected by the instructor's assignments to courses.
Postconditions: no change to the host object or the respective course, returns a list of the courses the instructor is assigned to.
Side-Effects: ---

BEGIN COURSE

Course.addInstructor(activeInstructor)

Preconditions: Course hasn't exceeded max instructor assignments, the instructor hasn't exceeded their max course assignments, the argument must be an existing instructor.
Postconditions: course is assigned to instructor and instructor assigned to course. Error displayed for maximum capacity reached for instructor or course or nonexistent instructor.
Side-Effects: ---
activeInstructor: in out - This instructor will used to modify the course's instructor list and the instructor will have to be assigned to this course, it's state will be modified in the process.

Course.addTA(activeTA)

Preconditions: Course hasn't exceeded max TA assignments, the TA hasn't exceeded their max course assignments, the argument must be an existing TA.
Postconditions: course is assigned to TA and TA assigned to course. Error displayed for maximum capacity reached for TA or course or nonexistent TA.
Side-Effects: ---
activeTa: in out - This TA will used to modify the course's TA list and the TA will have to be assigned to this course, it's state will be modified in the process.

Course.removeAssignment(activeUser)

Preconditions: The user we're trying to remove exists, and the user is assigned to this course

Postconditions: the user's assignments to this course will be removed and the courses assignments to the user will be removed. Error message displayed if non existent user or user that isn't assigned to the course.
Side-Effects: ---
activeUser: in out – This user will be used to search for a matching user in the course so it can be removed, then all the user's assignments to/from this course will be removed.

Course.removeCourse()

Preconditions: course must be active in the schedule of classes system.
Postconditions: this course will be removed from the schedule of classes GUI and no user can be assigned to it.
Side-Effects: The respective sections and assignments to this course will also be removed.

Course.editCourseInfo (Dict)

Preconditions: Course's input meets formatting requirements for contact information (number of labs must be int, no missing fields)
Postconditions: valid changes to course's state are saved, all GUI elements with this course are updated, user/sections also reflect any changes. If provided wrong input, displays error message.
Side-Effects: ---
Dict: input – This course's information, state, will be modified according to the user's input.

Course.getAsgmtsForCrse()

Preconditions: All of the course's user assignments must have a matching assignment to this course. Must be at least one user assignment to this course.
Postconditions: no change to the host object or the respective users, displays all of the course's users who are assigned to this specific course.
Side-Effects: ---

Course.getSectionsForCrse()

Preconditions: Sections must be correctly instantiated to belong to the course and the course must respectively contain any section assigned to this course.
Postconditions: no change to the host object or the respective sections, displays all of the course's sections who are a part of this specific course.
Side-Effects: ---

Course. getCrseInfo()

Preconditions: All course information must be correctly instantiated in the constructor.
Postconditions: no change to the host object or the respective sections, displays all of the course information (name, ID, course description, semester,modality, sections,assignments)
Side-Effects: …

BEGIN SECTION

Section.getID()

Preconditions: Section has been assigned a unique ID at instantiation.
Postconditions: returns an integer representing the sections unique ID
Side-Effects: …

Section.getParentCourse()

Preconditions: Section must always be assigned a parent course, as per composition, and the section should not exist if the course is removed.
Postconditions: returns a course object representing the course that the section belongs to.
Side-Effects: …

BEGIN LAB

Lab.getLabTAAsgmt()

Preconditions: the TA field within the lab section has been correctly set to a TA that is assigned to this lab. Must be a TA assigned to return.
Postconditions: no changes to the lab's state, just return the TA that is currently assigned. If no TA assigned then error message displayed.
Side-Effects: ---

Lab.addTA(activeTA)

Preconditions: the lab cannot already have a TA assigned, the TA cannot be assigned to grader status.
Postconditions: the TA will be assigned to this section, and the section will be assigned to the TA. Will display error message if assigning a TA with grader status, or if the Lab is already full.
Side-Effects: ---
activeTA: in out: the specific TA we want to assign to the lab and have the lab assigned to the TA

Lab.removeTA()

Preconditions: there must be a TA assigned to the lab section
Postconditions: the TA is removed from the lab section, the lab and TA itself still exist. Error displayed if no TA present.
Side-Effects: ---

BEGIN LECTURE

Lecture.getLecInstrAsgmt()

Preconditions: the instructor field within the lecture section has been correctly instantiated to be a instructor.
Postconditions: no changes to the lecture's state, just return the instructor that is currently assigned to the lecture. Error displayed if there isn't an instructor assigned.
Side-Effects: ---
…

Lecture.addUser(activeUser)

Preconditions: have to send in user object.
Postconditions: the User will be assigned to this section, and the section will be assigned to the TA/instructor
Side-Effects: ---
activeUser: in out: the specific User we want to assign to the lecture and have the lecture assigned to the User.

Lecture.removeUser()

Preconditions: there must be a user assigned to the lecture section
Postconditions: the user is removed from the lecture section, the lecture and user itself still exist; they're just not connected any more (as per aggregation).
Side-Effects: ---