****

**TURNS**

**Developers:**

Aaron Garcia

Rigoberto Avila

Azahel Cordova Arvizu

Class: ECE 373

Date: December 4, 2017

**Problem Statement:**

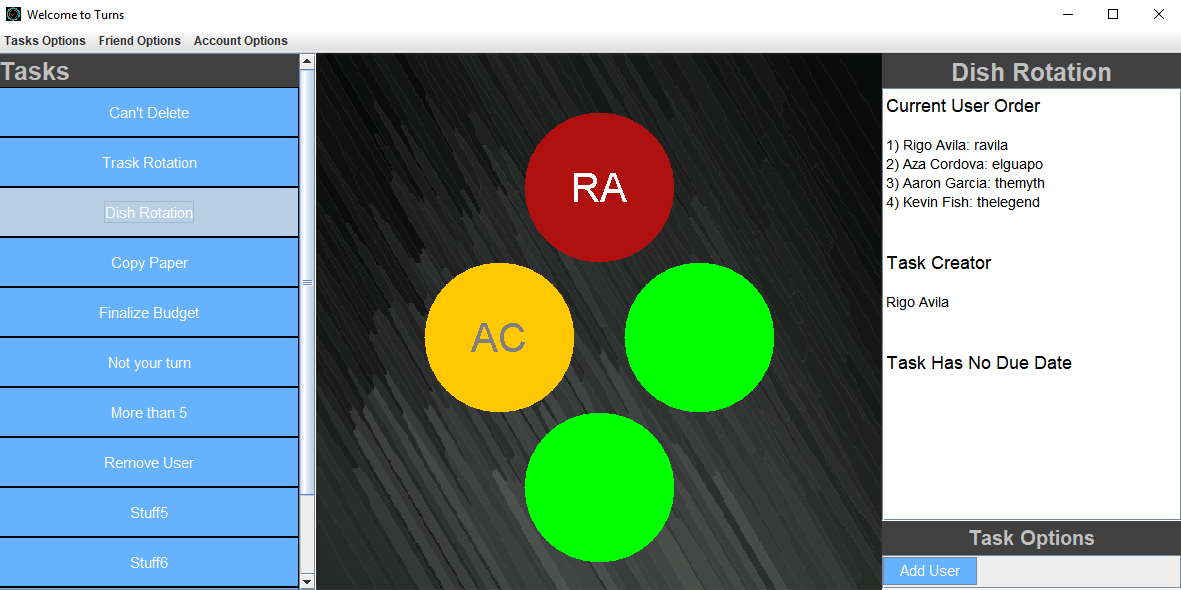
Check List are great and all but what happens if you have multiple people working on something? What if that task has to be repeated various times over a period of time? What if you forgot to open your task list on Evernote for some period of time? The solution to this problem is the application Turns. As a team we have decided to create the “Turns” application which is a simple task list meant for groups, teams, or individuals, who need tasks done in a cyclic matter.

**System Boundary:**

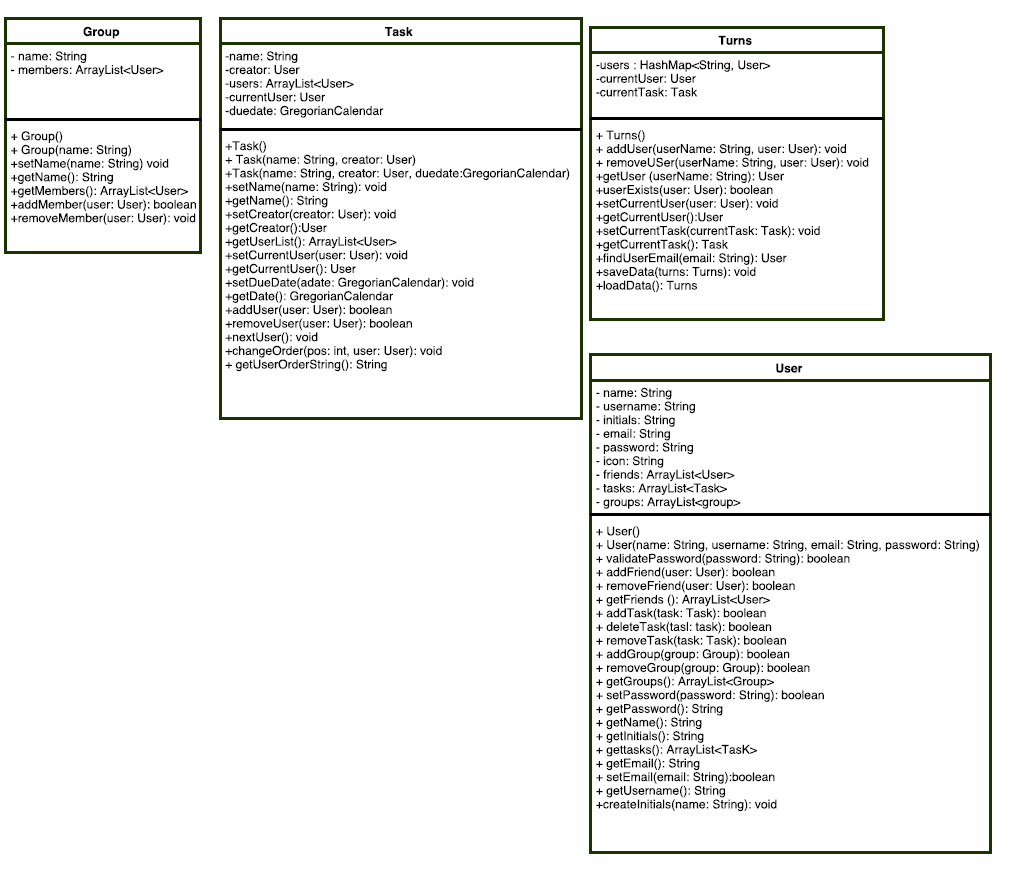
The initial idea was to create a mobile app that would allow teams to better organize themselves as well as helping them to assign responsibilities and keep track of their completion. However, after going through multiple changes, the result will not work (as of right now) on mobile devices. Also, initially we wanted the tasks to me modifiable at any time. For example the user who created the task, would be able to add more people to it or modify its description even if the task was already happening. Users won’t be able to do that anymore. Now, when a task is created, it remains that way until the task is finalized at due date or date of completion. Another boundary is that our app completely avoids any type of social interaction among users. This means that users cannot message one another or communicate in any way through the application. At the beginning we had an idea of having a button so that users could remind the user in turn to complete the assigned task. However, since this could be considered social interaction in some way, we decided to take it off. After all, the purpose of TURNS is to increase productivity, not to create distractions for the users. One last thing that limits the functionality of our application is the limit of users per task. Initially we had not thought about this, however in the final implementation, 10 users per task is the limit.

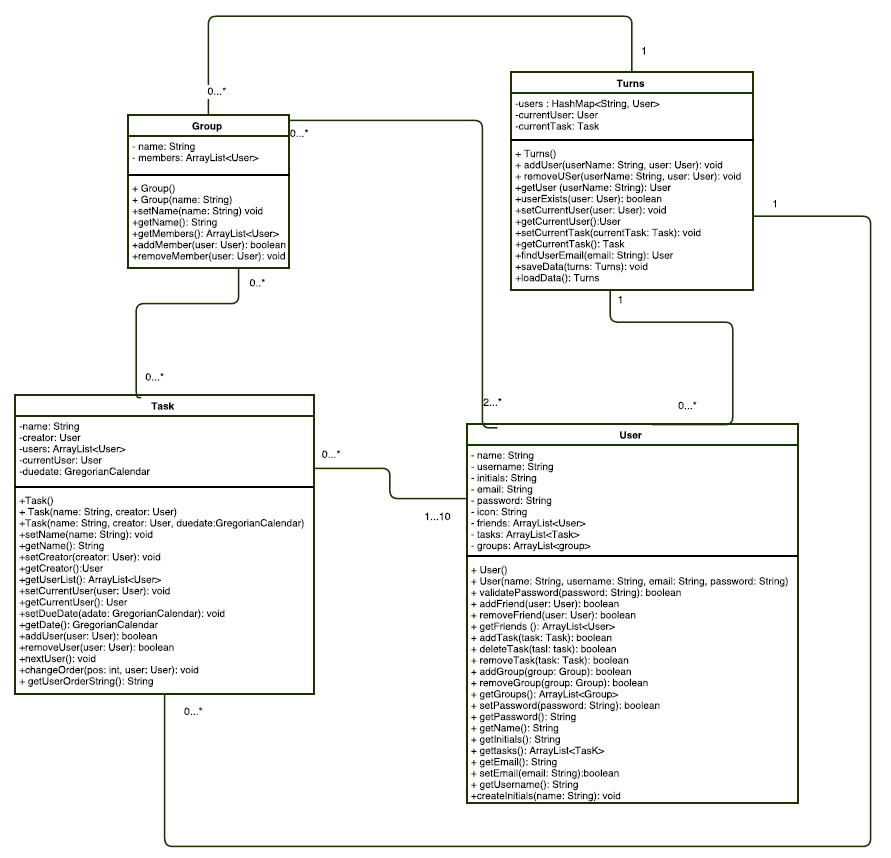
**Domain Analysis:**

Turns is a Java application that will rely on tasks, users and circles to facilitate and increase effectivity in completions of task. With the TURNS app, similar to a conventional to-do list, users will be able to create tasks along with their due date and a brief description of what has to be done. However, users will see a major benefit in using Turns as opposed to using any other to-do list or reminder app when they need to split the work between multiple people or collaborate on the same task at a time. This “Turns” functionality will work the following way. Users will have the option of creating a task and share with multiple people so that everyone is involved. For example, when living with roommates, responsibilities such as sweeping or taking out the trash are often rotated among the inhabitants. But how can you keep track if the person in turn already did what he/she had to? Or how do you keep track of who is next? This is where the rotation functionality from “Turns” comes in place. With turns, users will be able to easily assign a rotation pattern within the members of a task. Also, by using color coding it will be easy for users to see who is next, and who is in turn is at the moment.

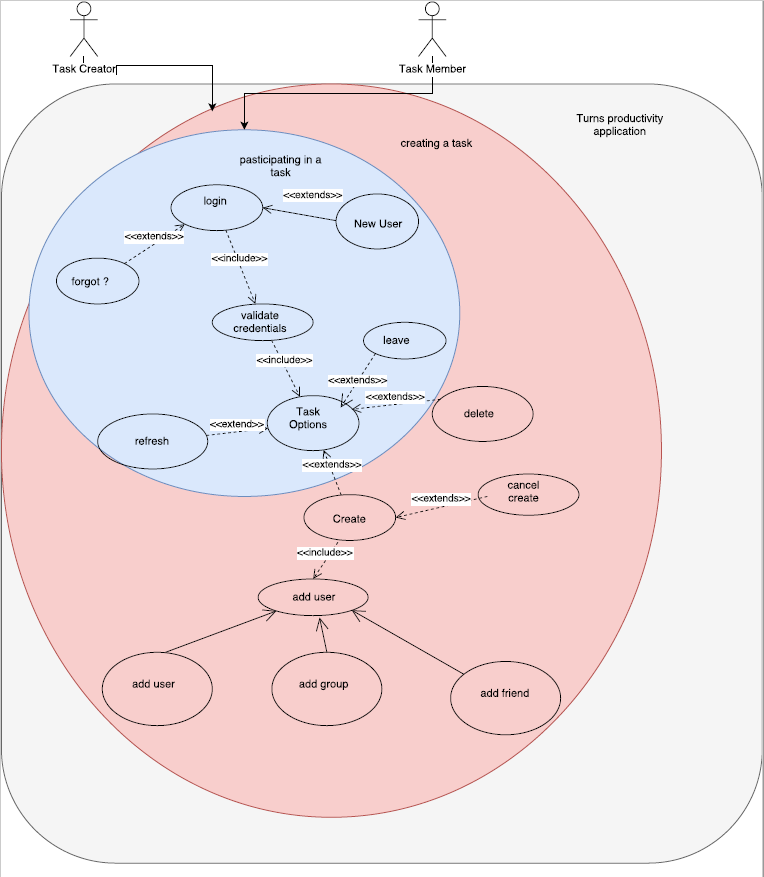


In the diagram above, the user in red is the person in turn, the user in yellow is the next user in turn and the rest of the users would correspond to green, meaning they are free of responsibility at the moment. Turns allows users to create groups and add other users to them. This facilitates creating the tasks that will implement the rotation functionality. The user who creates the task and/or group is able to add and eliminate members from it. This application is not only limited to household chores. It can be used in the office, at restaurants, and much more. Users can also create tasks rotations that only involve themselves.

**Detailed UMLs and Class Model:**



**Use Case Diagram:**



**Use Cases:**

***Use Case Name****:*Create a Task

***Actors*:** Task Creator

***Preconditions*:** Turns is waiting for a user to login and have credentials validated. Friends and groups must have already been added and created

***Description*:** Turns starts in the waiting state where it waits for a user to login and validate its credentials. Once validated, the creator can click on "Task Options" in the menu bar where a menu pops down. From there the user can click on "Create Task" to initialize a new task. At this point the user can now specify the name of the task. The user now also has the option to add other users by username or click on two other links that would take the user to a screen where friends of the user or pre-created groups can be added. Once filled out and users are added, the user can hit okay to create the new task. Once the task is created, the user can view the new task in the "Task Lists" panel by clicking "Refresh".

***Exceptions****:*

*Canceled:* At any time, the user can hit a cancel button to terminate the task creation process.

*Limit 10 Usernames:* Only 10 usernames can be added at one time. However, once the task is created more creators can add more tasks

*Non-existing User:* Only users that already have a Turns account can be added to a task item. Otherwise an error will occur.

*Group/Friend Additions:* While in the process of creating a new task, no new groups or friends can be added. The created would need to cancel and add this using other menu bar options

***Post conditions****:*Turns is waiting for the user to close the window or logout. Alternatively, a new action can be started from any of the other menus/ GUI Items

**Use Case Name:** Leaving a Task

**Actors:** Task Member

**Preconditions:** A user must already be enrolled in a task with other members to be able to leave.

**Description:** Turns starts in the waiting state where it waits for a user to login and validate its credentials. Once validated, the user can select a task that the user is enrolled in. From there, the user can select the "Leave Task" option to be removed from the task. A confirm dialog will pop out to make sure that the user actually wants to leave the group.

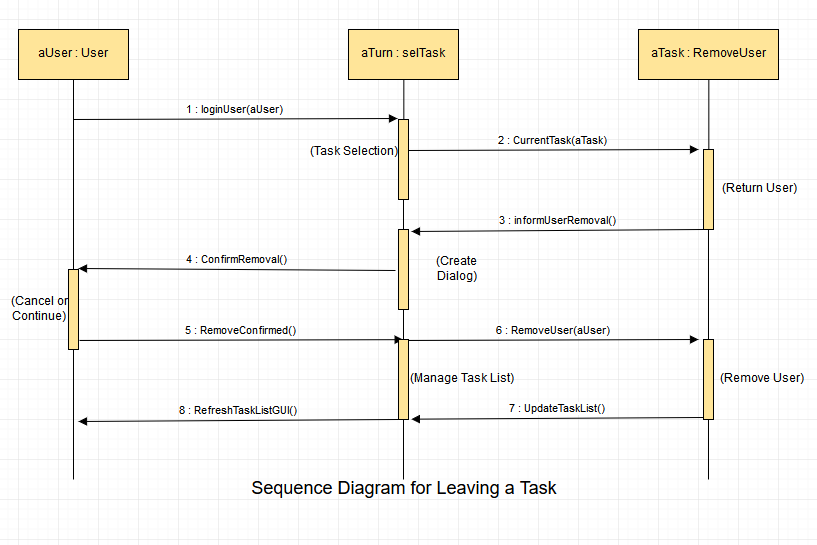
**Exceptions:**

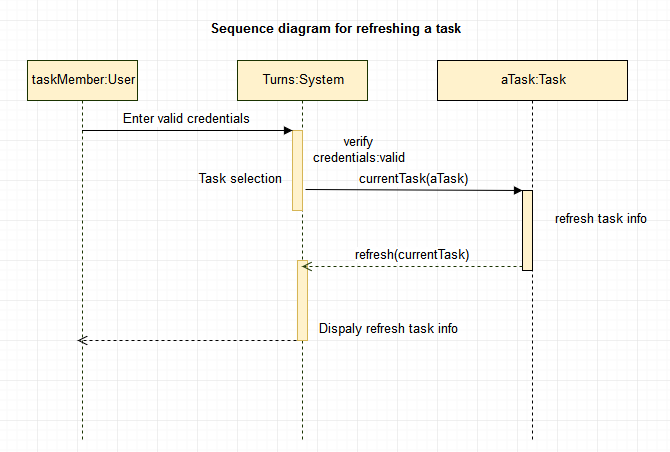
*Canceled:* At any time, the user can hit a cancel button to terminate the leaving task process

*Current Turn:* If it is currently the user's turn to complete the task, the user cannot leave until the task is completed.

**Post conditions:** Turns is waiting for the user to close the window or logout. Alternatively, a new action can be started from any of the other menus/ GUI Items

**Sequence Diagrams:**

****

****

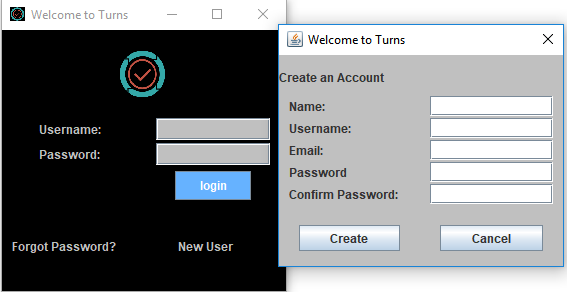
**Implementation:**

During the implementation we ran into some issues that made us change some of the functionality. Fortunately, these were minor tweaks and changes that we had to make and nothing serious enough that would compromise the overall project. For example, the idea was to have all user names displaying in their corresponding turns circles as well as having a photo for every user. However, in the finalized project we ended up with only the usernames for the users in the red and yellow circles, and none of user has photos. One data structure we used in our project is a hash-map. We use the hash-map in order to save the users and their username guaranteeing that a username won’t be repeated.

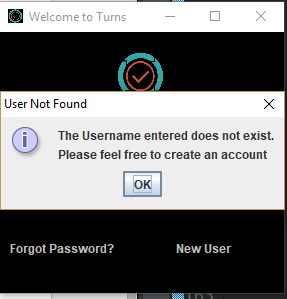
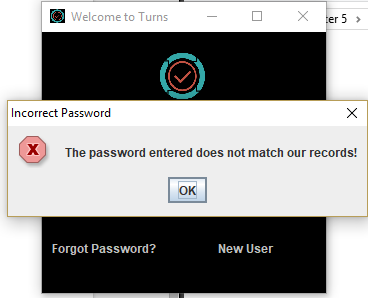
**Testing and Results:**

For unit testing, we created individual drivers for the classes. With this, we tested all the methods of each class were working as expected. To summarize it, we just filled all the fields for each class with values, then we printed them as expected values. Then we printed the values that actually, were stored in those fields in order to verify that they were the same. If they indeed ended up being the same, then the methods worked as expected. In appendix\_User and Appendix\_Task you will find example of the code used in the drivers followed by their respective output.

For integration testing, we tested all the different GUIs in order to make sure that classes were working in conjunction as they were supposed to. There is a lot of information that is shared and passed around among classes, therefore we thought that doing the testing that way would be the most convenient. In the next image you will see the login window and new user window side by side. We tested them by creating a new account and then trying to log in with the account we had just created. If we were allowed to the main screen, it meant that the account was successfully created and the test worked.



For the systems testing aspect of the Turns software, the Domain Analysis was used as a requirements section to see if the systems implemented satisfy these requirements. The 2 main systems were to login and main screen systems. The login system was tested by making sure that only the right username and password combination allowed access to the Turns application and the respected user account. Error messages popped up to handle an incorrect password or a non-existing user.

Turns wanted to ensure a secure system. To ensure this, we checked that the password was more than 6 characters long. If it was shorter than this the user account was not created. To further improve security, but also give a means to a user to recover a password, a system was created to show the user their password temporarily. To access the password a user has to input an existing username with a corresponding email address. To verify that this works we attempted multiple different combinations of both existing and non-existing usernames with various emails. It was confirmed that only the right combination would give access to the password. The show password screen timed out after 7 seconds to further fulfill the security requirement.

Another requirement was to make turns as user friendly as possible. To accomplish this, we made sure that there was large fonts and easy to read lists. The main contribution to this, however, was the big Turns wheel in the center. To test simplicity, we created various different tasks and added different amounts of users. This was done to see if Turns kept its simplistic user interface while still organizing large amounts of information.

Finally, we wanted to make the Turns task information easily accessible to all those involved. To validate this, we created multiple user accounts and added them to different tasks. We then exhaustively checked them all to make sure that the right information was still displayed even though we had logged in and logged out multiple different times.

**State Model:**

Main Screen

Do/update turns diagram and task information

Login Window

New User

Forgot Password

Main Screen

Forgot password Button

Login button/invalid credentials

New user button

Ok button

Login button/valid credentials

Task name button

Task name button

Main Screen

Do/update turns diagram

Task name button

Turn Completed button

**Extensibility:**

There’s three main things that could be added to our implementation. First, it would be very useful to be able to create new groups as we create a task. Also, searching for friends even if the user doesn’t know what the actual username is. Being able to add friends while creating a task would also make things faster and easier for the user. Next, as mentioned in the video, adding a due date to tasks as part of the info required to create the task, would make sense. A significant addition that could be made to the project, is to add methods that allow the users to modify their settings and information. For example, many times a user might want to change the password initially chosen during the account creation process. One last change that impact the app is arranging the tasks by priority as well as color coding the, depending on priority just as we did we the turn circles in the main screen.

**Appendix Task:**

**Code:**

package testDrivers;

import framework.Task;

import framework.User;

public class DriverTask

{

public static void main(String args[])

{

User user1 = new User("Jim", "thelegend27", "youknow@gmail.com", "1234");

User user2 = new User("Aza", "elguapo", "aza@gmail.com", "2468");

User user3 = new User("Aaron", "themyth", "aaron@aol.com", "1357");

Task task1 = new Task();

task1.setName("TestTask1");

task1.setCreator(user1);

//User should also appear in user list

//Adding other users to user list

task1.addUser(user2);

task1.addUser(user3);

//Sets the user that needs to complete the task

task1.setCurrentUser(user1);

//print method to check what is being output

//prints out all users

System.out.println("Users in this task are: ");

for(int i=0; i < task1.getUserList().size(); i++) {

System.out.println(task1.getUserList().get(i).getName());

}

//Prints out creator Jim

System.out.println("\nThe creater for " + task1.getName() + " is: "

+ task1.getCreator().getName());

System.out.println("Correct: Jim");

//Prints the current users and changes. should be jim

System.out.println("\nThe current user for the task is: " + task1.getCurrentUser().getName());

System.out.println("Correct: Jim");

//updates current user. should be aza

task1.nextUser();

System.out.println("\nThe current user for the task is: " + task1.getCurrentUser().getName());

System.out.println("Correct: Aza");

//updates current user. should be aaron

task1.nextUser();

System.out.println("\nThe current user for the task is: " + task1.getCurrentUser().getName());

System.out.println("Correct: Aaron");

//removes user aaron from task

task1.removeUser(user3);

//prints out all users

System.out.println("\nThe remaining users in this task are: ");

for(int i=0; i < task1.getUserList().size(); i++) {

System.out.println(task1.getUserList().get(i).getName());

}

System.out.println("Correct: \nJim\nAza");

//Testing to change the order

task1.changeOrder(0, user2);

//Aza should now be first

//prints out all users

System.out.println("\nNew position for users in this task are: ");

for(int i=0; i < task1.getUserList().size(); i++) {

System.out.println(task1.getUserList().get(i).getName());

}

System.out.println("Correct: \nAza\nJim");

//Removing missing user should return false since user is not there

System.out.println("\nRemoving Aaron who was already removed");

System.out.println("Output: " + task1.removeUser(user3) + "\nCorrect: false")

}

}

**Output:**

Users in this task are:

Jim

Aza

Aaron

The creator for TestTask1 is: Jim

Correct: Jim

The current user for the task is: Jim

Correct: Jim

The current user for the task is: Aza

Correct: Aza

The current user for the task is: Aaron

Correct: Aaron

The remaining users in this task are:

Jim

Aza

Correct:

Jim

Aza

New position for users in this task are:

Aza

Jim

Correct:

Aza

Jim

Removing Aaron who was already removed

Output: false

Correct: false

**Appendix User:**

**Code:**

package testDrivers;

import framework.User;

import framework.Group;

import framework.Task;

public class DriverUser

{

public static void main(String args[])

{

User user = new User("Aaron Garcia", "agarc75", "agarc75@yahoo.com", "p@SSworD");

User friend = new User("friend1", null, null, null);

Group group = new Group();

Task task = new Task("task1", user);

//User method testing

System.out.println("User method testing\n");

System.out.println("addFriend(Friend friend): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.addFriend(friend)) + "\n");

System.out.println("getFriends(): Expected output -- friend1");

System.out.println("Output: " + user.getFriends().get(0).getName() + "\n");

System.out.println("removeFriend(Friend friend): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.removeFriend(friend)) + "\n");

System.out.println("addTask(Task task): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.addTask(task)) + "\n");

System.out.println("getTasks(): Expected output -- task1");

System.out.println("Output: " + user.getTasks().get(0).getName() + "\n");

System.out.println("removeTask(Task task): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.removeTask(task)) + "\n");

System.out.println("addGroup(Group group): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.addGroup(group)) + "\n");

System.out.println("getGroups(): Expected output -- Unknown");

System.out.println("Output: " + user.getGroups().get(0).getName() + "\n");

System.out.println("removeGroup(Group group): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.removeGroup(group)) + "\n");

System.out.println("setPassword(String password): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.setPassword("1234")) + "\n");

System.out.println("validate(String password): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.validatePassword("1234")) + "\n");

System.out.println("getName(): Expected output -- Aaron Garcia");

System.out.println("Output: " + user.getName() + "\n");

System.out.println("setEmail(String email): Expected output -- true");

System.out.println("Output: " + Boolean.toString(user.setEmail("agarc75@email.arizona.edu")) + "\n");

System.out.println("getEmail(): Expected output -- agarc75@email.arizona.edu");

System.out.println("Output: " + user.getEmail() + "\n");

System.out.println("getUsername(): Expected output -- agarc75");

System.out.println("Output: " + user.getUsername() + "\n");

}

}

**Output:**

User method testing

addFriend(Friend friend): Expected output -- true

Output: true

getFriends(): Expected output -- friend1

Output: friend1

removeFriend(Friend friend): Expected output -- true

Output: true

addTask(Task task): Expected output -- true

Output: true

getTasks(): Expected output -- task1

Output: task1

removeTask(Task task): Expected output -- true

Output: true

addGroup(Group group): Expected output -- true

Output: true

getGroups(): Expected output -- Unknown

Output:

removeGroup(Group group): Expected output -- true

Output: true

setPassword(String password): Expected output -- true

Output: true

validate(String password): Expected output -- true

Output: true

getName(): Expected output -- Aaron Garcia

Output: Aaron Garcia

setEmail(String email): Expected output -- true

Output: true

getEmail(): Expected output -- agarc75@email.arizona.edu

Output: agarc75@email.arizona.edu

getUsername(): Expected output -- agarc75

Output: agarc75