CS 146 Section 7

Programming Assignment Four Report

Claire Lin

December 13th, 2018

Facebook simulator design/implementation detail

The simulator stores Persons objects in a hash table, where each Person object consists of a name and a person's friend list. The calculation of hash code is the remainder of the length of a person's name (String) divided by the number of slots in the hash table.

Considering range of length of names and the test case of the implementation, the hash table is chosen to have 12 slots. As the results, the hash table is constructed by a 12-slot array of linked lists, implementing a hashing with chaining. Each linked list is implemented to contain a list of Nodes, and each Node contains a Person as key, a previous attribute, and a next attribute to assign ordering in the linked list. When there are two Persons with the same length of name, the two Person Nodes are put into the same slot of the hash table, with a linked list connecting the two. Each Person object also maintains a linked list of Nodes as friend list.

The linked list of Person objects implements its own deletion, insertion, and search methods, and the deletion, insertion, and search operations are based on these methods. **Insertion** in the hash table is done by finding the slot that corresponds with the hash code of the person's name, creating a Person object, creating a Node that takes the Person object, and adding the Node to head of linked list in the slot. **Searching** in the hash table is done by finding the slot that corresponds with the hash code of the person's name, searching for a Node that contains a Person with the same name as the searched person's name, and returning the node. **Deletion** in the hash table is done by finding the slot that corresponds with the hash code of the name of the person to be deleted, calling search on the slot's linked list, and remove the node returned by search.

A user interface is created to allow the user to construct person objects, add friend, unfriend, list friend list, and check friend status, based on the three operation of the hash table:

List of classes/subroutines/function calls and explanations of each purpose

Person.java: A class that has one field for the name and another field for the list of friends, containing essential information for each profile in the Facebook simulator.

Node.java: a class for storing Person object and assigning ordering relationship with other Nodes to support PersonList manipulation

PersonList.java: a class that creates a linking structure of Node objects and supplies deletion, searching, and insertion of nodes into the linking structure.

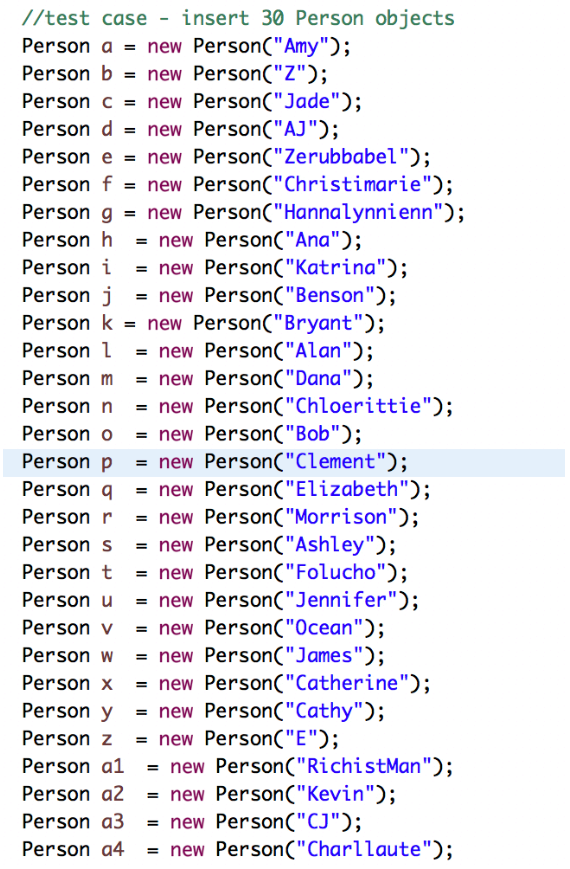
HashTable.java: a class that creates a hash table that indexes each Node object under the name. An array of PersonList (PersonList[]) is used as the structure of hash table.

Tester.java: a class that serves as a driver to create a GUI for users to perform all the hashing-by-chaining requirements specified.

Screen Shots of Results of Each Functional Requirement

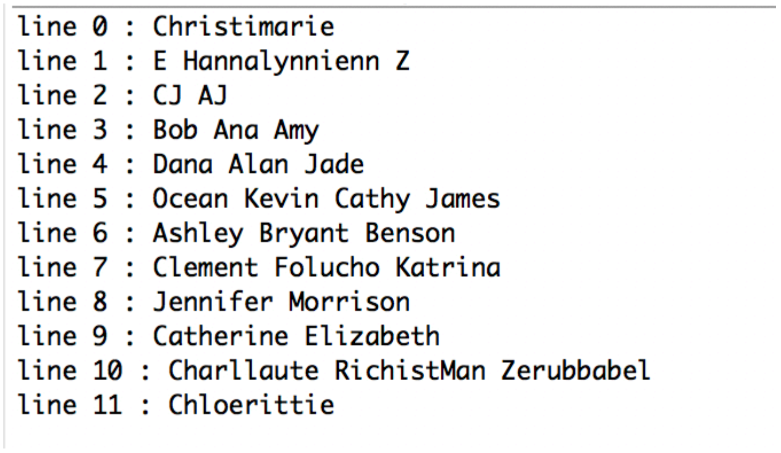
Functional Requirements 1 - Create a person record of the specified name

Input: Create 30 person records and insert them to the hash table



```
//test case - insert 30 Person objects
Person a = new Person("Amy");
Person b = new Person("Z");
Person c = new Person("Jade");
Person d = new Person("AJ");
Person e = new Person("Zerubbabel");
Person f = new Person("Christimarie");
Person g = new Person("Hannalynnienn");
Person h  = new Person("Ana");
Person i  = new Person("Katrina");
Person j  = new Person("Benson");
Person k = new Person("Bryant");
Person l  = new Person("Alan");
Person m  = new Person("Dana");
Person n  = new Person("Chloerittie");
Person o  = new Person("Bob");
Person p  = new Person("Clement");
Person q  = new Person("Elizabeth");
Person r  = new Person("Morrison");
Person s  = new Person("Ashley");
Person t  = new Person("Folucho");
Person u  = new Person("Jennifer");
Person v  = new Person("Ocean");
Person w  = new Person("James");
Person x  = new Person("Catherine");
Person y  = new Person("Cathy");
Person z  = new Person("E");
Person a1  = new Person("RichistMan");
Person a2  = new Person("Kevin");
Person a3  = new Person("CJ");
Person a4  = new Person("Charllaute");
```

Output: The printout of hash table from index(line) 0 – 11

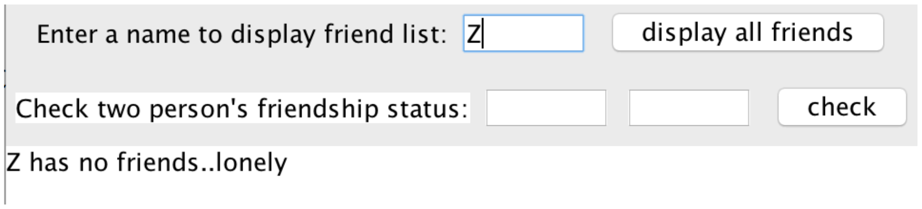


```
line 0 : Christimarie
line 1 : E Hannalynnienn Z
line 2 : CJ AJ
line 3 : Bob Ana Amy
line 4 : Dana Alan Jade
line 5 : Ocean Kevin Cathy James
line 6 : Ashley Bryant Benson
line 7 : Clement Folucho Katrina
line 8 : Jennifer Morrison
line 9 : Catherine Elizabeth
line 10 : Charllaute RichistMan Zerubbabel
line 11 : Chloerittie
```

Functional Requirements 2 - Record a person as a new friend.

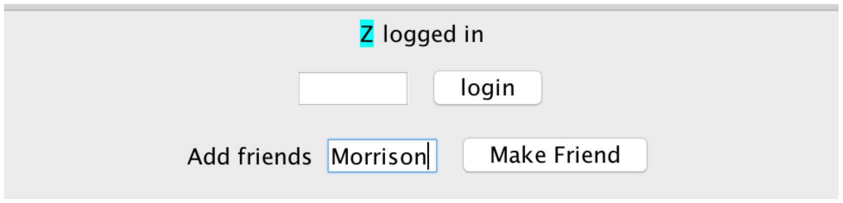
Test case 1

Before adding any friend to Person Z:


Enter a name to display friend list: Z

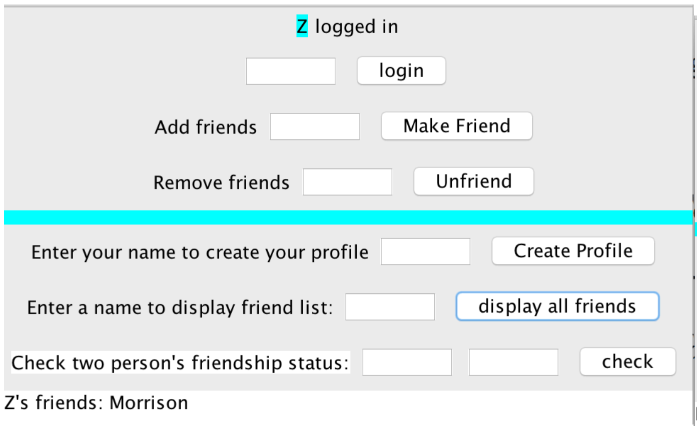
Z has no friends..lonely

Input: Adding Person Morrison as Z's friend, clicking on "Make friend"


Z logged in

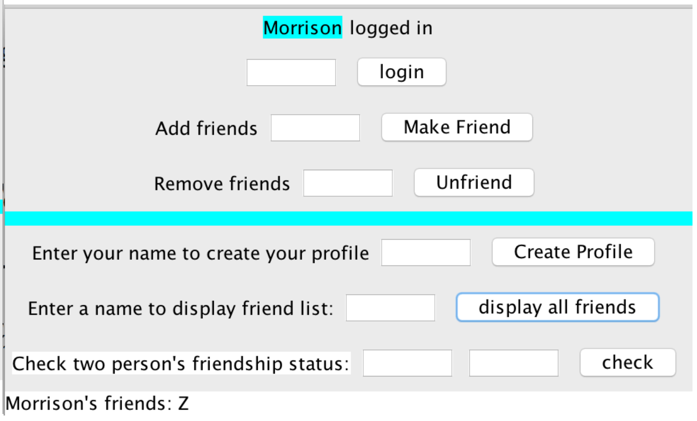
Add friends Morrison

Output: clicking "display all friends" of Morrison and Z


Z logged in

login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Z's friends: Morrison

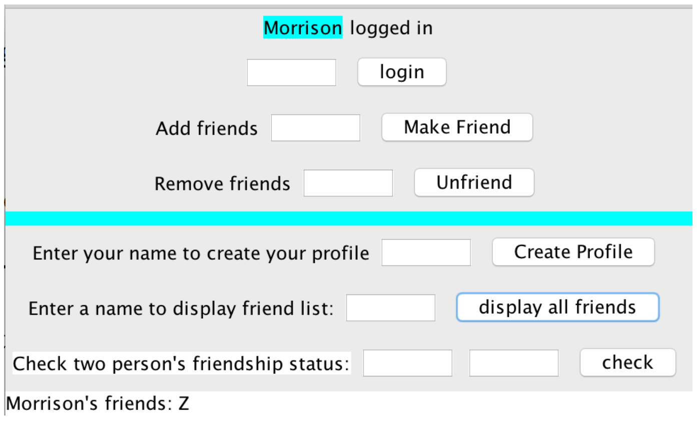

Morrison logged in

login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Morrison's friends: Z

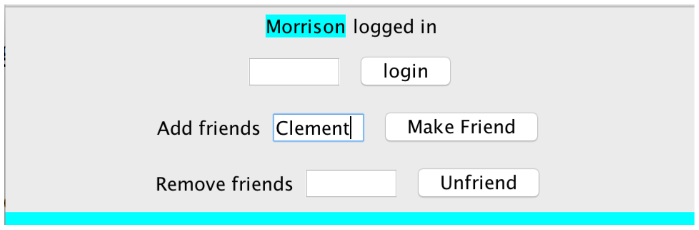
Test case 2

Before adding two more friends to Person Morrison:

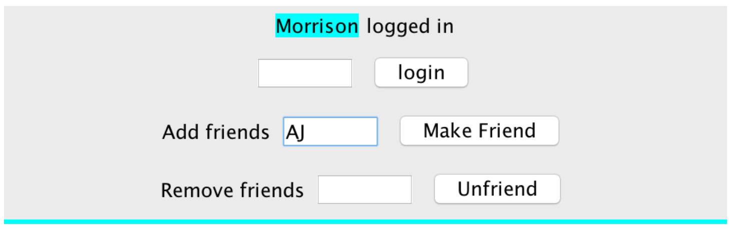

Morrison logged in
login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Morrison's friends: Z

Input: Adding 2 friends more Person Morrison

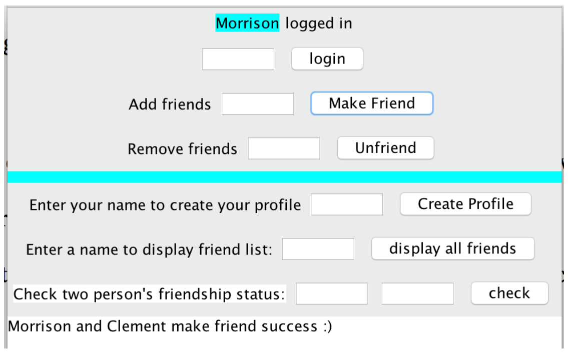


Clement

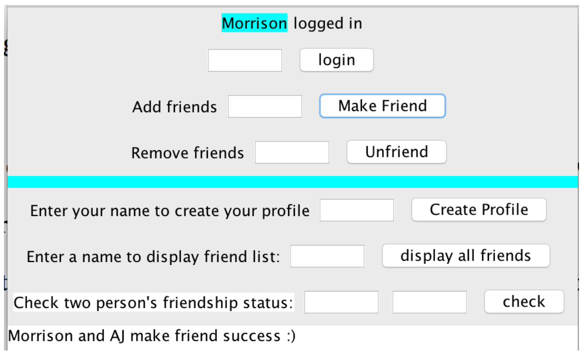


AJ

Output: Make friend was successful

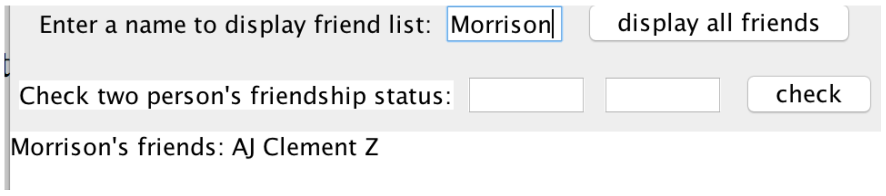

Morrison logged in
login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Morrison and Clement make friend success :)


Morrison logged in
login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Morrison and AJ make friend success :)

Display Morrison's friend list

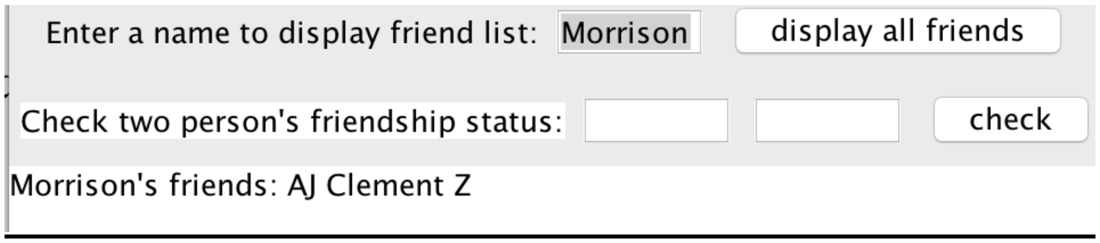

Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Morrison

Morrison's friends: AJ Clement Z

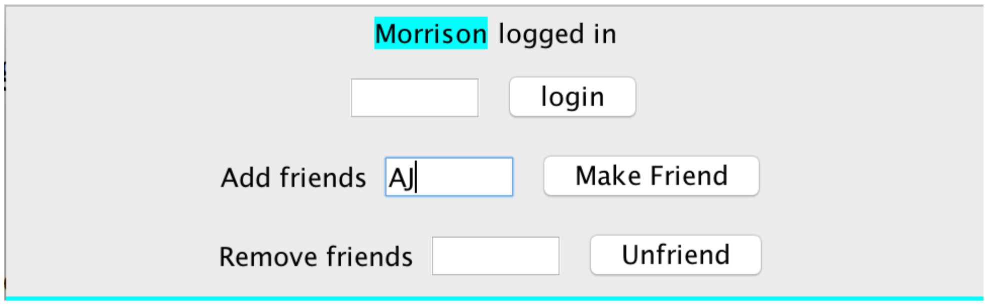
<u>Test case 3</u>

Before adding another friend to Morrison that is already in Morrison's friend list

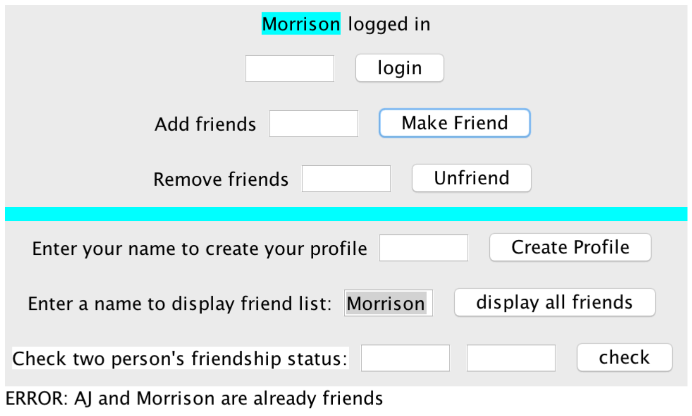

Enter a name to display friend list:
Morrison
display all friends
Check two person's friendship status:
check

Morrison's friends: AJ Clement Z

Input: Adding AJ as Morrison's friend



Output: ERROR message is displayed

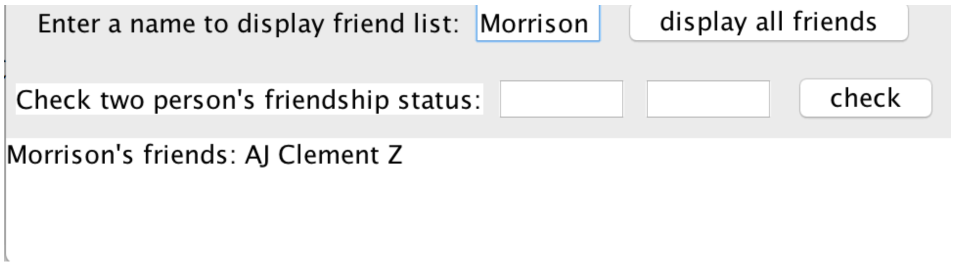


ERROR: AJ and Morrison are already friends

Functional Requirements 3 – Unfriend

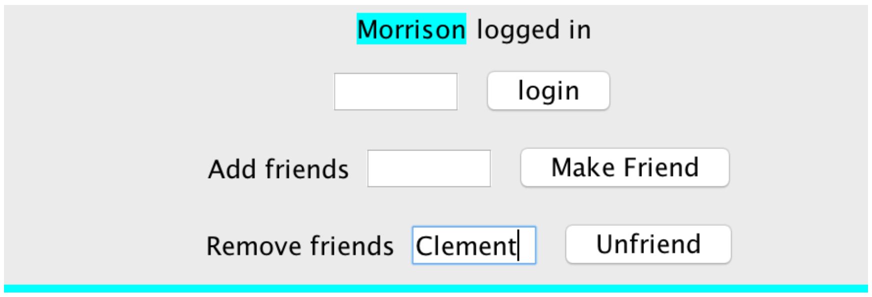
Test case 1

Morrison’s friend list before unfriending a friend

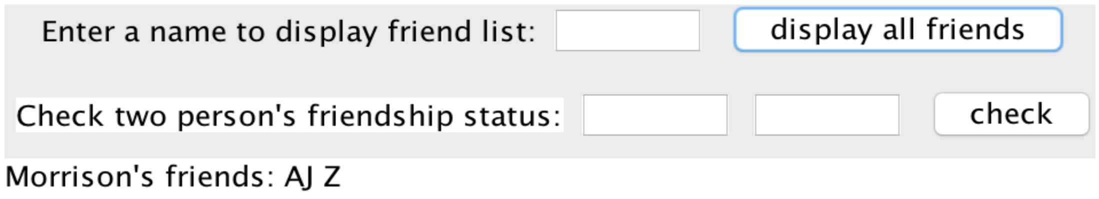

Enter a name to display friend list: Morrison
display all friends
Check two person's friendship status:
check

```
Morrison's friends: AJ Clement Z
```

Input: Removing Clement from Morrison’s friend list

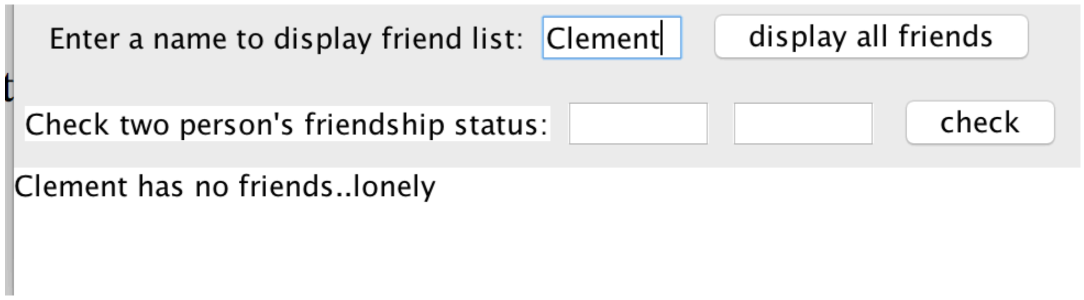

Morrison logged in
login
Add friends
Make Friend
Remove friends Clement
Unfriend

Output: Morrison’s friend list after unfriending


Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

```
Morrison's friends: AJ Z
```

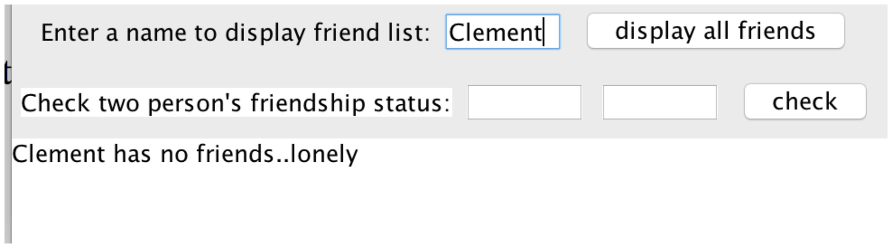
Clement’s friend list after unfriending


Enter a name to display friend list: Clement
display all friends
Check two person's friendship status:
check

```
Clement has no friends..lonely
```

Test case 2

Before removing a friend that is not in Morrison's friend list



Enter a name to display friend list: Clement

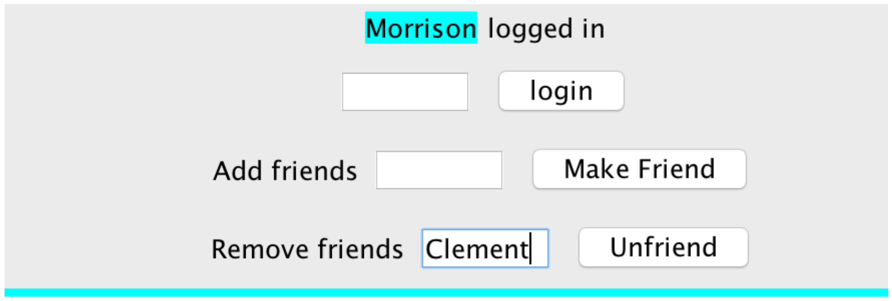
display all friends

Check two person's friendship status:

check

Clement has no friends..lonely

Input: Removing Clement again from Morrison's friend list



Morrison logged in

login

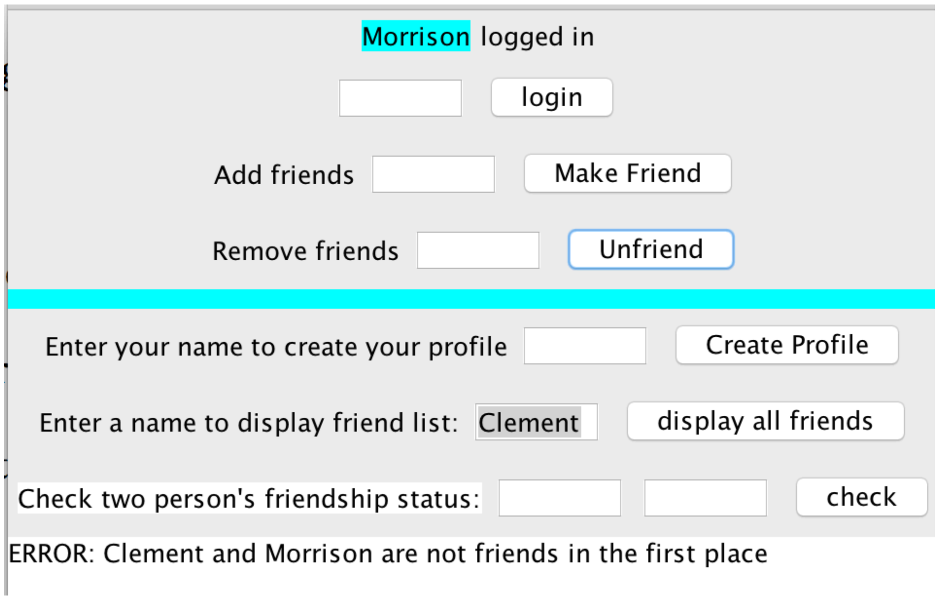
Add friends

Make Friend

Remove friends Clement

Unfriend

Output: Error message is displayed



Morrison logged in

login

Add friends

Make Friend

Remove friends

Unfriend

Enter your name to create your profile

Create Profile

Enter a name to display friend list: Clement

display all friends

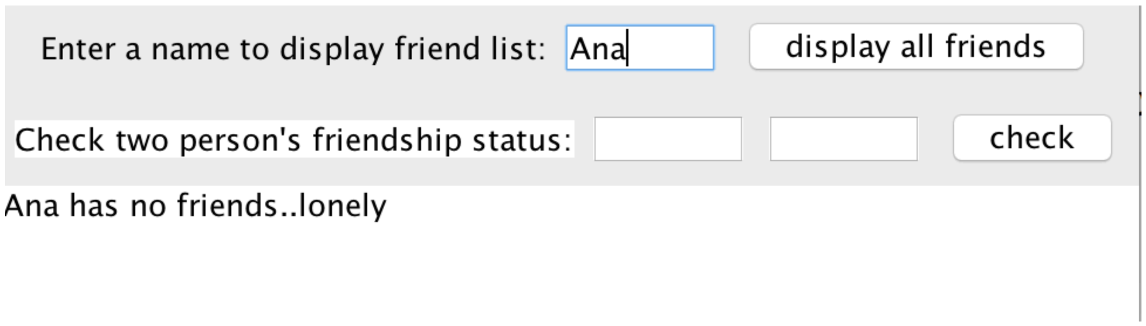
Check two person's friendship status:

check

ERROR: Clement and Morrison are not friends in the first place

Functional Requirements 4, 5 – List friends of a person.

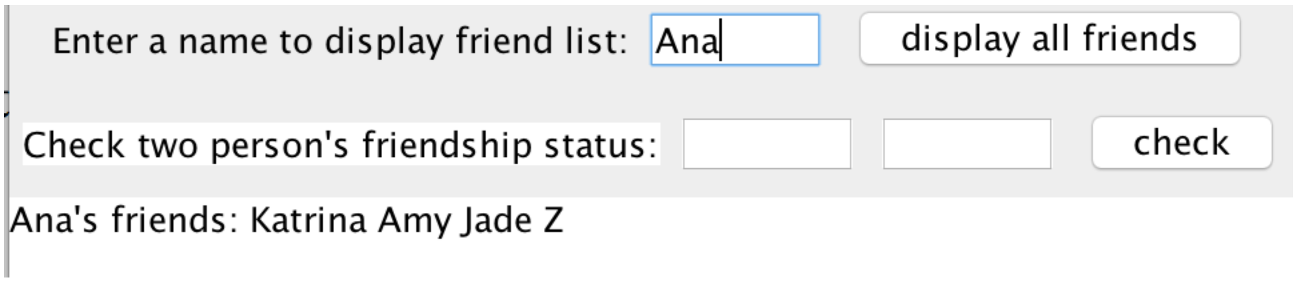
Amy's friend list before adding more friends


Enter a name to display friend list:
Ana
display all friends
Check two person's friendship status:
check

Ana has no friends..lonely

Input: Adding Amy, Jade, Katrina, and Z to Ana's friend list

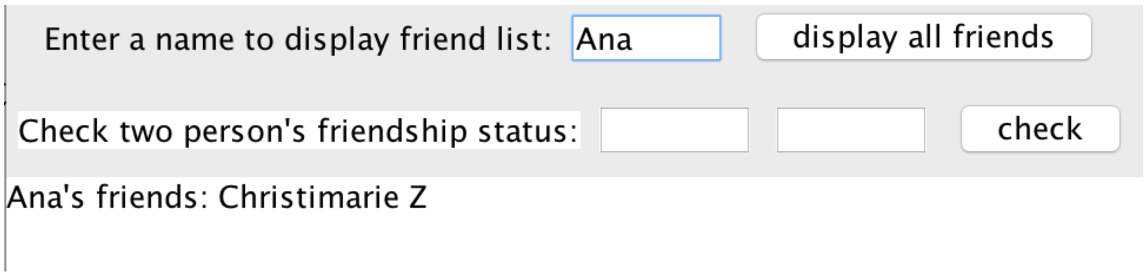
Output: Displaying Ana's friend list


Enter a name to display friend list:
Ana
display all friends
Check two person's friendship status:
check

Ana's friends: Katrina Amy Jade Z

Functional Requirements 6 – Enter two person's names to check whether the two people are friends.

Ana's current friend list before any changes

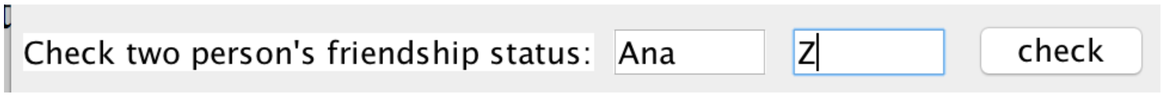


Enter a name to display friend list: Ana [display all friends]

Check two person's friendship status: [check]

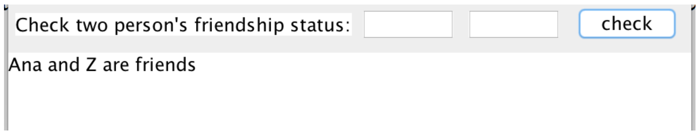
Ana's friends: Christimarie Z

Input: Checking if Ana and Z are friends



Check two person's friendship status: Ana Z [check]

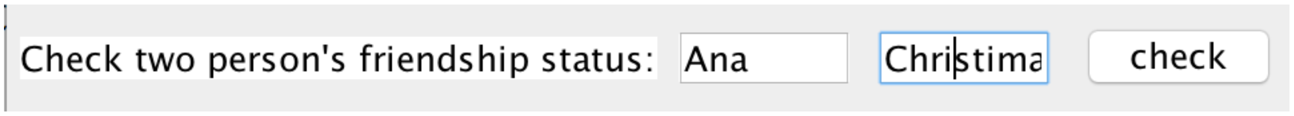
Output: result of Ana and Z's friendship status



Check two person's friendship status: [check]

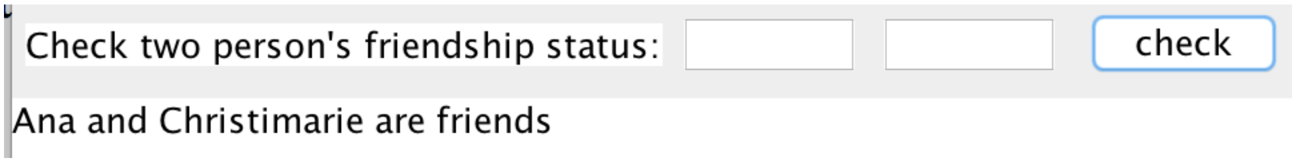
Ana and Z are friends

Input: Checking if Ana and Christimarie are friends



Check two person's friendship status: Ana Christima [check]

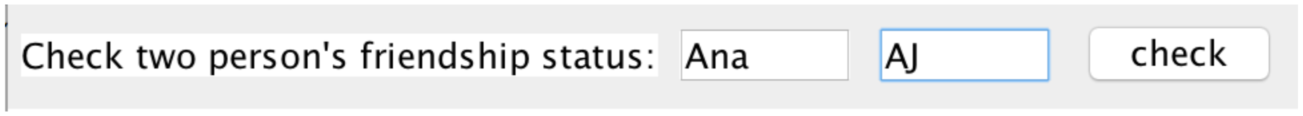
Output: Ana and Christimarie are friends



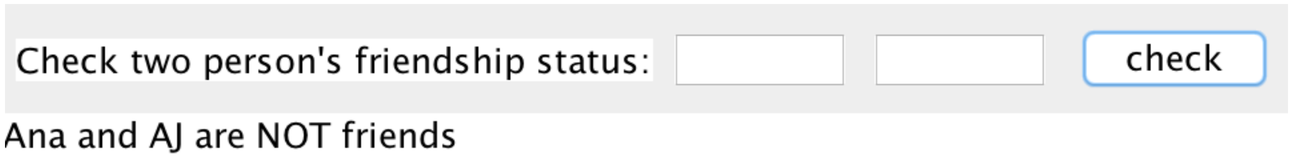
Check two person's friendship status: [check]

Ana and Christimarie are friends

Input: Checking if Ana and AJ are friends



Output: Ana and AJ are not friends



Ana and AJ are NOT friends

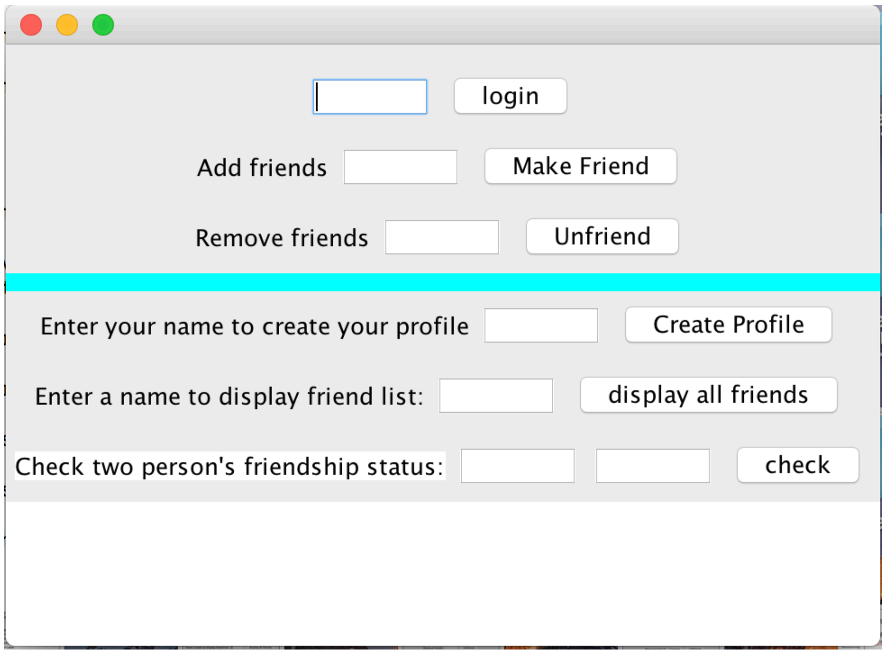
Procedure to unzip your files, install your application, and run/test your codes.

Step 1. Navigate to the directory that stores the Claire_Lin-PA4.zip file

Step 2. Unzip the file

Step 3. Run PA4.jar using the command line " Java –jar PA4.jar"

Step 4. A user interface will pop up as shown below


login
Add friends
Make Friend
Remove friends
Unfriend
Enter your name to create your profile
Create Profile
Enter a name to display friend list:
display all friends
Check two person's friendship status:
check

Step 5. Create new person profile by entering name and clicking "Create Profile"

Step 6. Login to your profile by entering name and clicking "login" to access add friends and remove friends features

Step 7. Display a person's friend list by entering a name and clicking "display all friends" friend list will be shown at the bottom white text area

Step 8. Check two people's friendship status by enter two names and clicking "check"

Step 9. Any addition messages, including massages will be displayed at the bottom white text area upon execution

Problems encountered during the implementation

There were several problems encountered during the implementation process. The first one is the realization of needing two linked list, one as a friend list for each Person object and the other as the linked list inside each slot of the hash table array. The self-implemented linked list, PersonList, turned out to be usable for both purposes as both use linked lists that contains the same object, the person object.

Another problem encountered during the implementation process was adding button action-listener with lambda expression. The addFriend() and unfriend functions require a user to input a correct Person name that is already in the hash table, and storing the input name as a new variable. With the use of lambda expression, however, I was not able to assign the input name as a new variable and brought it out of the lambda expression as everything in the expression must be effectively final. One way to solve it was to create a Node class outside of the lambda expression, and assign the name to the Node class using a method inside the expression, but not assigning by creating a new variable.

Lastly, there is one problem that I was not able to fix. My program experienced InvocationTargetException every time the program is run. I have read my codes closely and still was not able to fix the problem. The exception, however, does not affect the execution of the program. The program was able to executes all functions correctly.

Lessons Learned

One most important lesson I learned was the importance of choosing a good hash function. A good hash function has to be able to distribute data in a fairly uniform way and being independent of the data. My hash function is not the best since it is based on the length of the name, and a lot of names have length from five characters to eight characters, and less with three or less or ten or more characters. Therefore, my program can improve its efficiently by changing to a better hash function.