Programming Assignment 3

Facebook Simulator

James Nguyen

I. Introduction

When someone mentions social media, usually the first idea that is prominent within one’s head is “Facebook”, as Facebook is one of the most popular platforms today as, according to Zephoria, the platform has a total of 2.7 billion users worldwide. Facebook is a social media platform, founded in 2004, by Mark Zuckerberg, in which users have the ability keep in contact with other people across the world through the process of adding each other as friends. Through its use of its complex data structures, such as hash table, to store and access each account, the social media platform is able to efficiently keep track of all the user that has created an account, and all the distinct accounts that are associated with the user’s account.

Image: *Facebook logo with the founder Mark Zuckerberg standing in the front. Credits to* *https://www.cnet.com/news/mark-zuckerberg-on-facebooks-2018-weve-changed-we-promise/*

This assignment will show what effectively is occurring within the background of Facebook, when a user performs actions such as creating an account and friending or defriending other users within the social media service. The main objective of this assignment is to design and implement a micro version of Facebook or a Facebook Simulator, in which users have two different types of a micro version of Facebook as each version will be separate and use different methods to implement simulation. However, both versions are able to create at least fifty accounts, deactivate an account, add and defriend other users, and view distinct types of information about a specific user. Furthermore, the assignment will be using Java Runtime Environment, since the assignment is coded through the Java programming language.

II. Design and Implementation

This section will go over the different designs and concepts of the Facebook Simulator, and how it will effectively be implemented as code.

A. Storage for Users Accounts

When a user creates an account for the Facebook Simulator, there needs to be a data structure that can store these different accounts. For both versions of the micro version of Facebook, we will be using a data structure known as hash tables to store these different accounts.

Hash tables is a data structure that stores an object based on its numerical value. Hash tables uses equations known as Hash functions to calculate the index in which the inputted object can be stored. In our case, we will have two different hash tables for the two different micro versions of Facebook, and each hash table will each have different hash functions. Using hash functions will be incredibly useful and efficient to store the different accounts as we can turn these accounts into numbers, enter them into the Hash function, and find the index in which these accounts will be put into. For both micro versions of Facebook, the size of the Hash Table will be eleven as eleven is a good number for the two different methods that we will be using.

B. Numerical Value of Users

Since these hash tables takes hash function, will need to turn these accounts into numerical values. In this assignment, we will turn these accounts into numerical values, by first turning the name of the account into all lower case. Then we will use the number representation of each character of the string known as the ascii code, and adding them all together, giving us the number representation of the account.

For instance, with the name, "James", we will first turn the name in to lower case which is "james". Then, we take the ascii code of each character and add them together which is 106 +97 + 109 + 101+115 or 528.

C. Methods to Store the Accounts

Throughout the programming assignment, there will be two different micro versions of Facebook as although they will perform, the same way, they will be implemented through different methods to store the accounts into the hash table. Since these methods will store the accounts differently through different hash functions, two hash tables are necessary.

1) Division Method – This method is a very fast operation to store accounts within the hash table as it finds the index that the account will be stored on by taking the modulo of the numerical value of the account by the length of the hash table. This method works very well with hash table with a size that is a prime number and not a power of 2 or 10. Since the hash table in our case is 11, our returned number from the function will be a number from 0 to 10 which will be the index.

   The function for the division method can be written as h(k) = k mod m in which k is the key that is inputted, and m is the size of the hash table. In our case the equation, k will be the numerical version of the account, and m will be equal to 11.
2) Multiplication method – This method is another effective approach to store the account within the hash tables through the operations of multiplipcation and modulo. To find the index of an account through the multiplication method, we can first multiply the key which is the numerical value of the account with a constant of A which is less than 1 but greater than 0. In our case A will follow Knuth's suggestion of ($\sqrt{5}$ -1)/2). We then will take the modulo of the result by 1 to get the fractional part of the result. The fractional part will then be multiplied with to the length of the table which is 11. We then take the whole number from the result, and that will be the index in which the account will be added to.

The method is very efficient as it doesn't rely on the length of the table as the constant A will give us an effective random number. As a result, we will also be using a length of 11 to make it easier for us when we create the hash tables as the length or m doesn't play a large role in within the function.

D. Handling Collisions

Since we are required to be able to handle at least fifty accounts, and the table only has 11 slots to put the accounts, there will be many collisions, as some accounts will receive the same index from the hash functions causing collisions. In order to handle this, we will be using a technique called "chaining", in which each slot of the hash table will be a linked list, and accounts which have the same index and are "colliding" with each other, will be inserted in the linked list together, fixing the collisions from occurring.

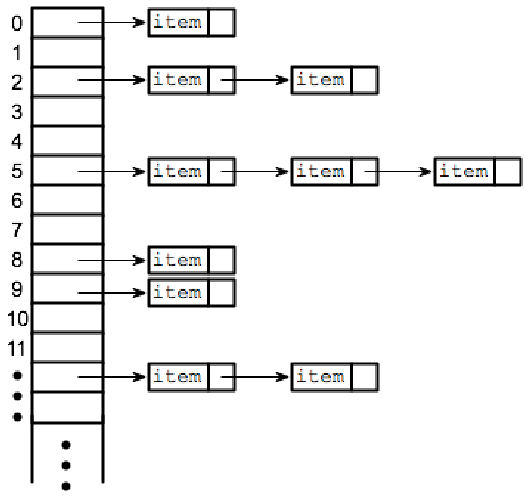


Image: *Design of Chaining in a hash table. Credits to https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/*

E. Searching and Deactivating Accounts

Throughout the program, we will be searching and deactivating accounts within the hash table. In order to search for accounts, we can use the multiplication or division method, depending on what micro version we will be using, to locate the linked list at the index, and then iterate through the linked list to find the account.

Furthermore to deactivate an account, we can delete the account the same way, as we can also use the multiplication or division method depending on what micro version that we are using, to locate the linked list at the index, and then iterate through the linked list to find the account, and then delete it.

F. Adding and Defriending Users

Each person throughout the Facebook Simulator should be able to add each other as friends or defriend each other. As a result, a data structure is needed to store an accounts friend. In order to implement this, we can have each person have a linked list in that can store their friends within. They can add a person to the linked list if they are friending the person or remove a person from the linked list if they are defriending the specific person. Furthermore, friending in the Facebook Simulator is mutual, so if an account has another person in the friend linked list, the specific person should also have the account in their friends linked list.

G. Other Data Structures

Other than the hash table and linked list, there are also other data structures that we will be using. In order to construct both the hash tables, we will be using an Array List which will contain a specific amount of linked list. Furthermore, we will also be turning the linked list for the friends into an Array List only to iterate through the linked list

H. User Interface

The programming assignment requires for users to be able to create, view, and deactivate accounts. As a result, a user interface is needed to be created to allow users to have different selections to create, view, edit the accounts. To meet these requirements, a pop-up menu with six different buttons for each different vision would suffice. The six choices will be:

1) Create an account for the Social Media
2) Choose a created account and add another created account as a friend
3) Choose a created account and defriend another created account
4) Search for a user and show their friends list
5) Check whether two users are friends
6) Deactivate and delete a created account.

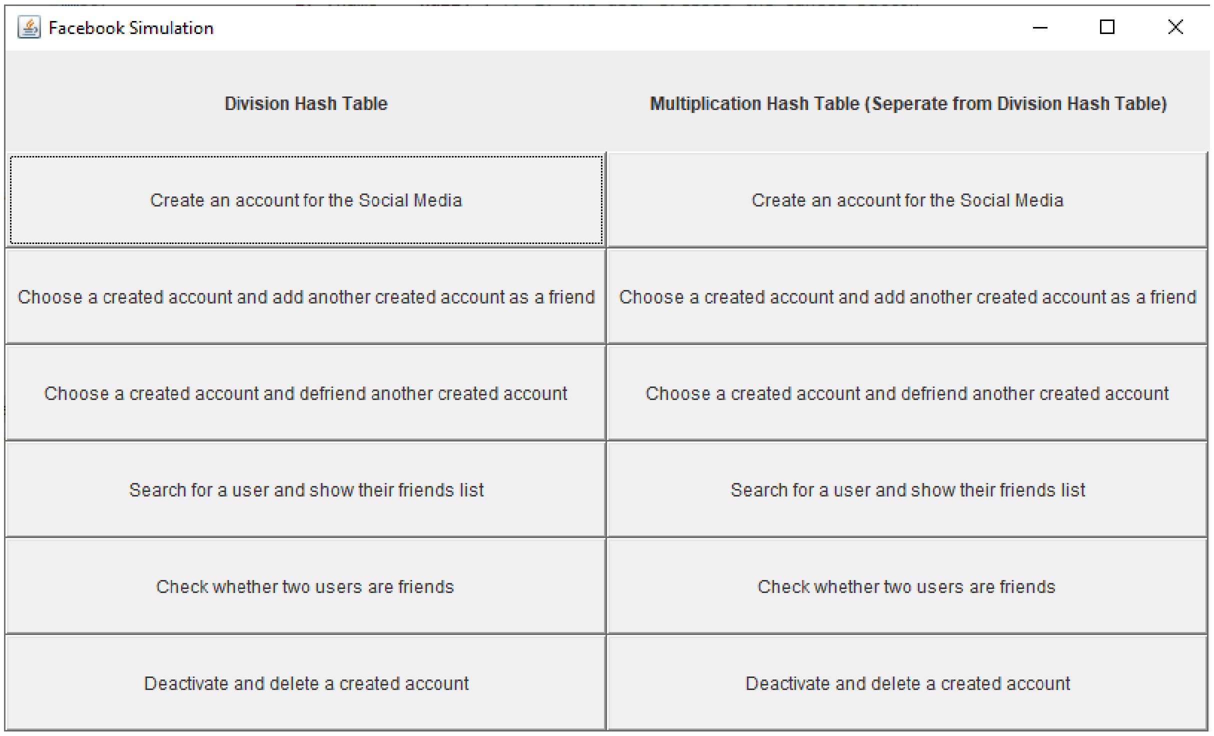

Facebook Simulation
Division Hash Table
Multiplication Hash Table (Seperate from Division Hash Table)
Create an account for the Social Media
Create an account for the Social Media
Choose a created account and add another created account as a friend
Choose a created account and add another created account as a friend
Choose a created account and defriend another created account
Choose a created account and defriend another created account
Search for a user and show their friends list
Search for a user and show their friends list
Check whether two users are friends
Check whether two users are friends
Deactivate and delete a created account
Deactivate and delete a created account

Image: *Implementation of menu with different options for users to choose from.*

With the user interface, users are efficiently able to choose many different choices to edit the data base of the different accounts within the Facebook Simulator. These choices essentially allow us to match the requirements of the programming assignment.

III. List of classes /subroutines/function calls

This section will go over the different classes as well as the different methods that are used within each class.

A. Person Class

This class allows one to create a person which contains information such as the name of the person as well as a list of friends of the person. The list of friends can also be edited through this class through methods of defriend and friending.

1) Constructor – The constructor of this class lets the Person class to initialize as it initializes the name of the person to the inputted string. It furthers initiates the friends list of the Person class.
2) getName – This method returns the name of the Person.
3) nameToInt- This method takes the name of the person, turn the name into lower case form, and through a for loop, the method will iterate through each character of the name, and add the ascii code of each character together. This gives the integer representation of the person.
4) addFriend – This void method essentially allows the class to add a friend by adding the inputted person to the linked list of friends of the Person class.
5) deFriend – This void method allows the class to defriend a friend by removing the inputted person from the linked list of friends if the Person is found in there.

B. PersonNode Class

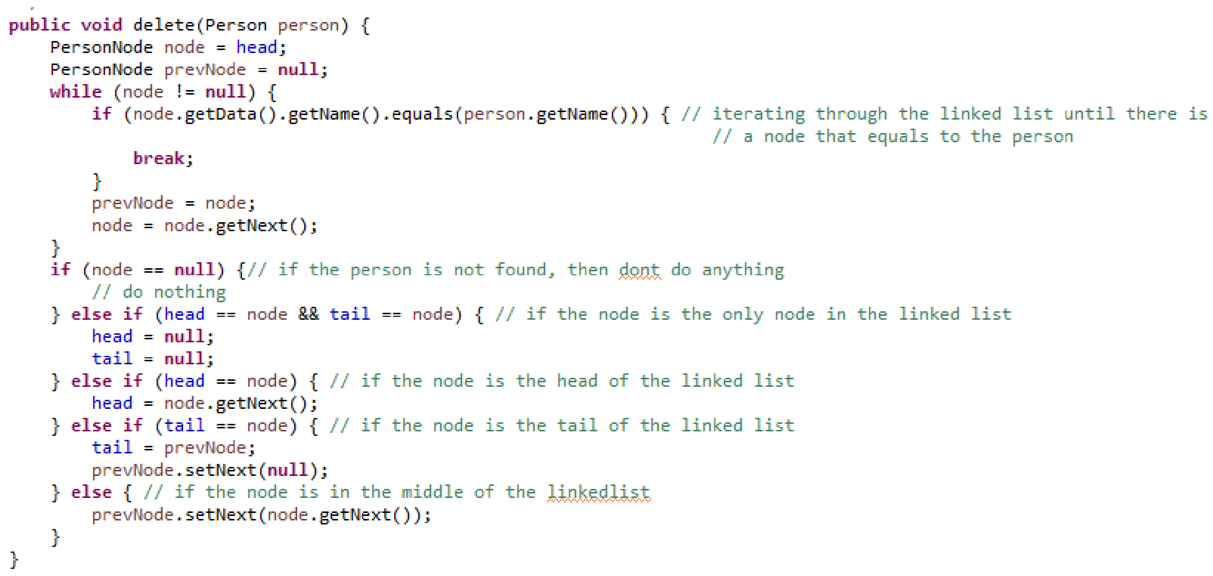
This class lets one be able to create a PersonNode which is essentially a node that has the Person Object as its data. The Node has methods to access the node as well as to set its next pointer for the next Node. The Node class will be used more by the PersonLinkedList Class.

1) Constructor – The constructor lets us initiate the PersonNode class by imitating the data which is the Person Object of the class.
2) getNext – This method will return the next node that the object is pointing to.
3) setNext – This method will set the next node that the class is point to.
4) getData – This method will return the data of the Person Node which is the Person that it contains.

C. PersonLinkedList Class

The class will use the PersonNode Class and Person Class to create a Singly Linked List in which users are able to insert to the list, remove from the list, and find a certain Person from the list. The class contains many methods which allows the linked list to be able to work and for users to access.

1) insert- This method allows one to insert a node to the head of the linked list. The head and tail may change when this occurs. If the list is empty, the node will be set as the head and tail. Otherwise, the head will be set as the inserted node, and the next pointer of the node will point at the previous head.
2) linkedToArray- This method will turn the linked list into an array list of Persons, by first creating an array list. Then the method will iterate through the linked list and add the data of each node to the array list. Then the array will be returned.
3) delete- This method will delete a person node that has the key of the inputted Person from the linked list. The method will first iterate through the list to search for the node with the key of the inputted Person. While it iterates, it will have a second instance node which will keep track of the previous node that was pointed at. After iterating, if the node is null, then the method will not do anything as no node is found. If the node is the only node in the list, the head and tail are both set to null. If the found node is the head of the linked list, then the head is set to the next node of the list. If the found node is the tail of the linked list, then the previous node will be set as the tail, and the next pointer of the previous node is set to null. Otherwise, if the node is in the center, the next pointer of the previous node is set to whatever the next pointer of node is pointing to.



```java
public void delete(Person person) {
    PersonNode node = head;
    PersonNode prevNode = null;
    while (node != null) {
        if (node.getData().getName().equals(person.getName())) { // iterating through the linked list until there is
                                                              // a node that equals to the person
            break;
        }
        prevNode = node;
        node = node.getNext();
    }
    if (node == null) {// if the person is not found, then dont do anything
        // do nothing
    } else if (head == node && tail == node) { // if the node is the only node in the linked list
        head = null;
        tail = null;
    } else if (head == node) { // if the node is the head of the linked list
        head = node.getNext();
    } else if (tail == node) { // if the node is the tail of the linked list
        tail = prevNode;
        prevNode.setNext(null);
    } else { // if the node is in the middle of the linkedlist
        prevNode.setNext(node.getNext());
    }
}
```

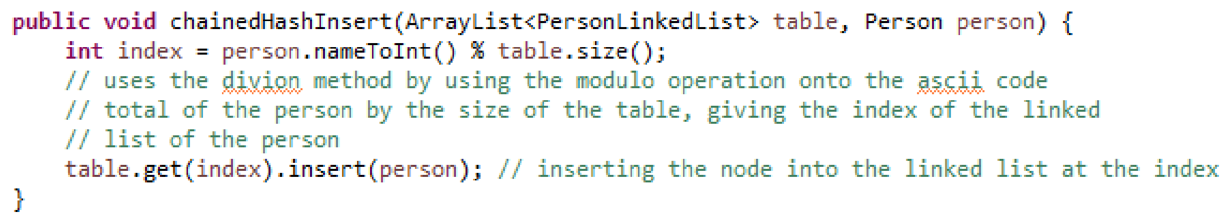
Image: *Implementation of the delete method as code.*

4) find- This method will find a person node based on a specific name. The method will iterate through the linked list and find the first-person node with the specific string. The method will then return the person in the list with the string.
5) toString- The toString method will allow the linked list to be printed as it will iterate through the linked list and print the data of the person in the Person node in order.

D. DivHashTableEditor Class

This class allows one to be able to edit a hash table as they will be able to insert and delete and search data through the Division method.

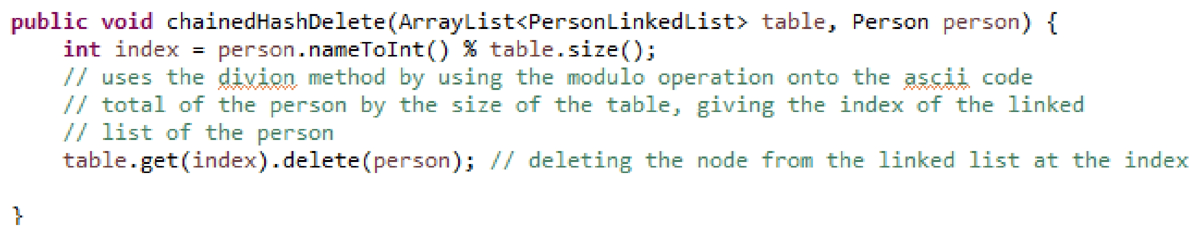
1) chainedHashInsert – This method will insert a person into the hash table by first turning the inputted person into a numerical value, and then modulo the numerical value by the size of the hash table to find the index. Then the person is inserted into the linked list at the index.



```java
public void chainedHashInsert(ArrayList<PersonLinkedList> table, Person person) {
    int index = person.nameToInt() % table.size();
    // uses the divion method by using the modulo operation onto the ascii code
    // total of the person by the size of the table, giving the index of the linked
    // list of the person
    table.get(index).insert(person); // inserting the node into the linked list at the index
}
```

***Image:** Implementation of the chainedHashInsert method as code.*

2) chainedHashDelete- This method will delete a person that matches the inputted person from the hash table, by first turning the inputted person into a numerical value, and then modulo the numerical value by the size of the hash table to find the index of the linked list of the person. Then the delete method is called from the linked list to delete the person if the object exists in the list.

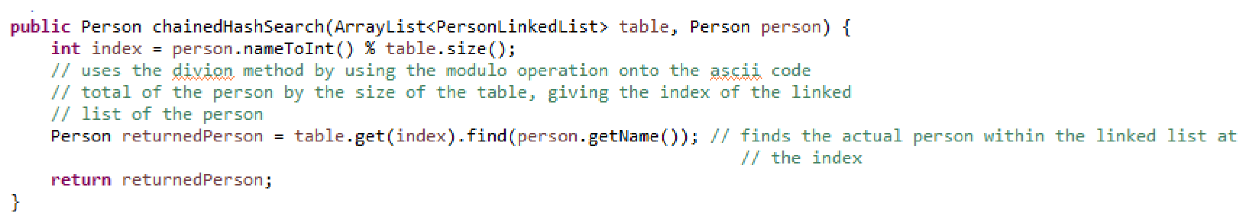


```java
public void chainedHashDelete(ArrayList<PersonLinkedList> table, Person person) {
    int index = person.nameToInt() % table.size();
    // uses the divion method by using the modulo operation onto the ascii code
    // total of the person by the size of the table, giving the index of the linked
    // list of the person
    table.get(index).delete(person); // deleting the node from the linked list at the index

}
```

***Image:** Implementation of the chainedHashDelete method as code.*

3) chainedHashSearch- This method will search a person that matches the inputted person from the has table, by by first turning the inputted person into a numerical value, and then modulo the numerical value by the size of the hash table to find the index of the linked list of the person. Then the find method is called from the linked list to find the person with the name and return the person.



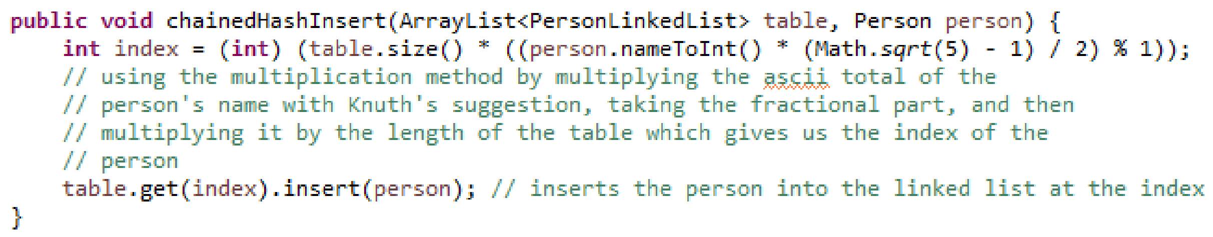
```java
public Person chainedHashSearch(ArrayList<PersonLinkedList> table, Person person) {
    int index = person.nameToInt() % table.size();
    // uses the divion method by using the modulo operation onto the ascii code
    // total of the person by the size of the table, giving the index of the linked
    // list of the person
    Person returnedPerson = table.get(index).find(person.getName()); // finds the actual person within the linked list at
                                                                     // the index
    return returnedPerson;
}
```

***Image:** Implementation of the chainedHashSearch method as code.*

E. MultHashTableEditor Class

This class allows one to be able to edit a hash table as they will be able to insert and delete and search data through the Multiplication method.

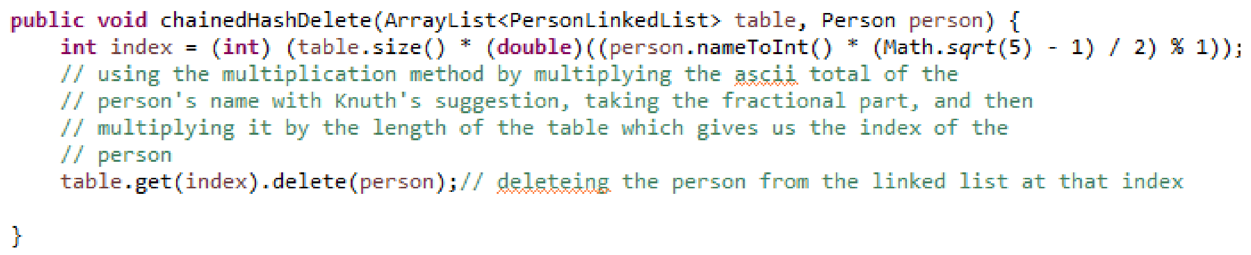
1) chainedHashInsert – This method will insert a person into the hash table by first turning the inputted person into a numerical value, and then multiply the number by Knuth’s suggestion of (√5 -1)/2). Then modulo the result by 1 to get the fractional part. We then multiply the fractional part with the length of the table, giving us the index. Then the person is inserted into the linked list at the index.



```
public void chainedHashInsert(ArrayList<PersonLinkedList> table, Person person) {
    int index = (int) (table.size() * ((person.nameToInt() * (Math.sqrt(5) - 1) / 2) % 1));
    // using the multiplication method by multiplying the ascii total of the
    // person's name with Knuth's suggestion, taking the fractional part, and then
    // multiplying it by the length of the table which gives us the index of the
    // person
    table.get(index).insert(person); // inserts the person into the linked list at the index
}
```

Image: *Implementation of the chainedHashInsert method as code.*

2) chainedHashDelete- This method will delete a person that matches the inputted person from the hash table, by first turning the inputted person into a numerical value, and then multiply the number by Knuth’s suggestion of (√5 -1)/2). Then modulo the result by 1 to get the fractional part. We then multiply the fractional part with the length of the table, giving us the index. Then the delete method is called from the linked list at the index to delete the person if the object exists in the list.

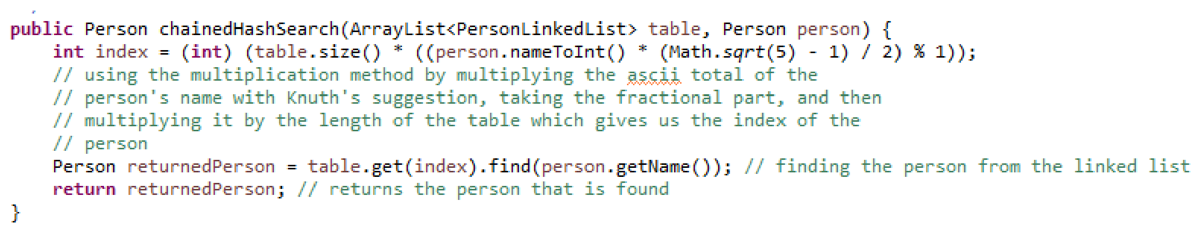


```
public void chainedHashDelete(ArrayList<PersonLinkedList> table, Person person) {
    int index = (int) (table.size() * (double)((person.nameToInt() * (Math.sqrt(5) - 1) / 2) % 1));
    // using the multiplication method by multiplying the ascii total of the
    // person's name with Knuth's suggestion, taking the fractional part, and then
    // multiplying it by the length of the table which gives us the index of the
    // person
    table.get(index).delete(person);// deleteing the person from the linked list at that index

}
```

Image: *Implementation of the chainedHashDelete method as code.*

3) chainedHashSearch- This method will search a person that matches the inputted person from the has table, by first turning the inputted person into a numerical value, and then multiply the number by Knuth’s suggestion of (√5 -1)/2). Then modulo the result by 1 to get the fractional part. We then multiply the fractional part with the length of the table, giving us the index. Then the find method is called from the linked list at the index to find the person with the name and return the person.



```java
public Person chainedHashSearch(ArrayList<PersonLinkedList> table, Person person) {
    int index = (int) (table.size() * ((person.nameToInt() * (Math.sqrt(5) - 1) / 2) % 1));
    // using the multiplication method by multiplying the ascii total of the
    // person's name with Knuth's suggestion, taking the fractional part, and then
    // multiplying it by the length of the table which gives us the index of the
    // person
    Person returnedPerson = table.get(index).find(person.getName()); // finding the person from the linked list
    return returnedPerson; // returns the person that is found
}
```

Image: *Implementation of the chainedHashSearch method as code.*

E. Facebook Simulation Class

This class allows the User to create a Facebook Simulation in which users are to have many different choices as they able to create Facebook accounts, choose other accounts to add or defriend each other, search for other accounts, and find information about the different accounts in the system. The class also extends JFrame and Action Listener in order for all the buttons of the User Interface to work. The class also effectively uses all the classes that are above.

1) Constructor – The constructor will initiate the MultHashTableEditor and DivHashTableEditor of the class. Furthermore, the class will initialize the hash tables for the multiplication and division method by initializing the array list of linked lists and adding eleven linked list into each hash table. The constructor will then add fifty accounts of the top fifty distinct names into both tables. The constructor will then create buttons and labels which will be added to a window, that will pop up to allow user to choose many different options and execute actions for the two different versions of the Facebook Simulator.
2) actionPerformed- This method causes actions to occur when a specific button from the window is pushed. Each button on the method passes and releases an action command when it is pressed which this method will detect. The buttons for both the Multiplication and Division Method Edition of the Facebook Simulator are really similar, so the actions for both editions will be explained at the same time. There are many different actions that will occur for each different button:
   a) "Create an account for the Social Media"– When this button is clicked on both editions, the commands, "divCreate" or "multCreate", is sent out. When the commands are recieved, a window from the JOptionPane is popped out which will ask the user to enter the name of the account they would like to add. The name is then received and turned into a Person object. The method then calls the chainedHashInsert from the DivHashTableEditor or the MultHashTableEditor depending on which edition it was pressed, to insert the name into the hash table.
   b) "Choose a created account and add another created account as a friend"- When this button is clicked on both editions, the commands, "divAdd" or "multAdd" is sent out. When the commands are received, a window from the JOptionPane is popped out which will ask the user to enter the name of the first account. Then another window from the JOptionPane is popped out to ask for the account that the first account would like to add as a friend. If they are not already friends of each other, they are both not the same accounts, and both accounts are able to be found through

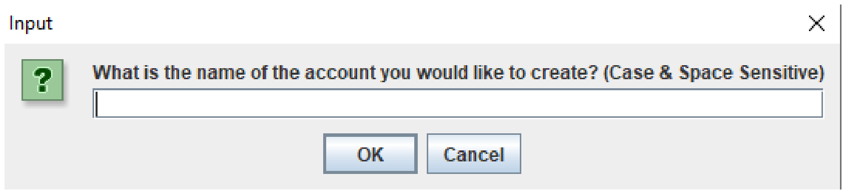
the chainedHashSearch, they can be added by calling on to the friends list of each user and calling the friend method. A success window will then pop out telling the user, that the action is successful.

c) "Choose a created account and defriend another created account" – When this button is clicked, the command, "divDef" or "multDef", is sent out. When the commands are received, a window from the JOptionPane is popped out which will ask the user to enter the name of the first account. Then another window from the JOptionPane is popped out to ask for the account that the first account would like to defriend. If they are friends of each other, they are not both the same account, and both accounts are able to be found through the chainedHashSearch, they can defriend each other. A success window will then pop out telling the user, that the action is successful.
d) "Search for a user and show their friends list" – When this button is clicked, the command, "divSearch" or "multSearch", is sent out. When the commands are received, a window from the JOptionPane is popped out which will ask the user to enter the name of the account. If the account is able to be found, the method will then turn the accounts friends list into an array list, and then print it onto a window for the users to see.
e) "Check whether two users are friends"- – When this button is clicked, the command, "divCheck" or "multCheck", is sent out. When the commands are received, a window from the JOptionPane is popped out which will ask the user to enter the name of the first account to check. Then another window from the JOptionPane is popped out to ask for the account that would be check. If both accounts are able to be found, the method will show a window for the user that will tell whether both accounts are friends with each other or not.
f) "Deactivate and delete a created account." -When this button is clicked, the command "divDeac" and "multDeac" is sent out. When the commands are received this method, a window from the JOptionPane is popped out which will ask the user to enter the name of the account they would like to deactivate. If the account is found, then the chainedHashDelete is called to delete the account. Furthermore, the hash table will be iterated to delete the account from all the other account's friend list if they are friends.

IV. Self -Testing Screen Shots

This section will display screenshots of testing the program to show that it has met the Functional Requirements of the programming assignment. In this case, we will be using the top 50 distinct popular names to fill up our hash tables.

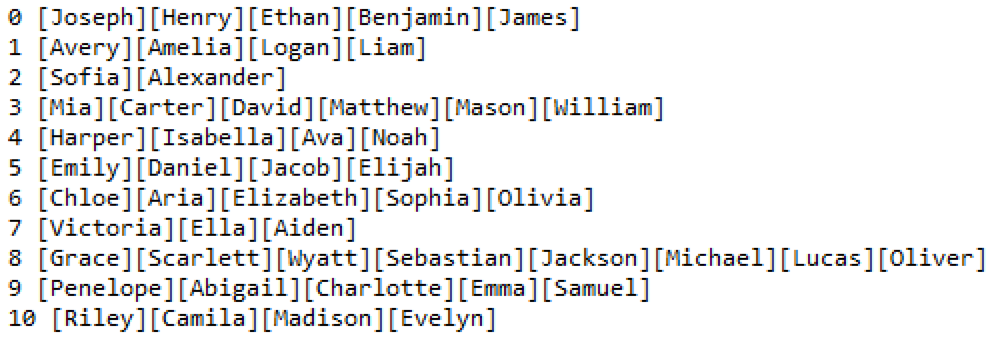
A. Creating an account with a specified name


Input

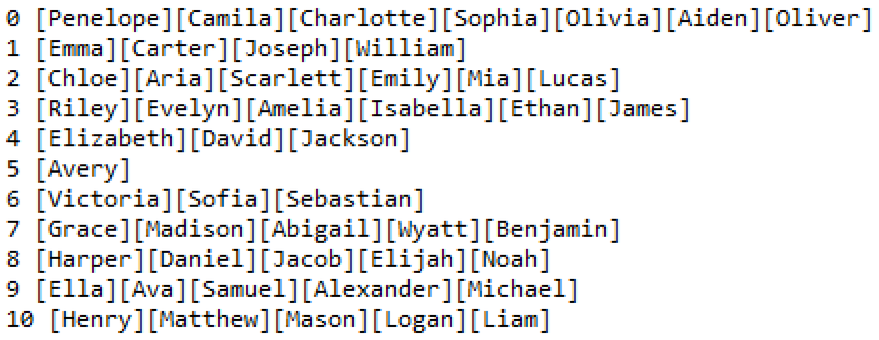
What is the name of the account you would like to create? (Case & Space Sensitive)

Users are able to enter a name, and a person object is created based on the name and added to the either the Multiplication or Division Hash Tables.

B. Adding the top 50 distinct names into the Hash table


```
0 [Joseph][Henry][Ethan][Benjamin][James]
1 [Avery][Amelia][Logan][Liam]
2 [Sofia][Alexander]
3 [Mia][Carter][David][Matthew][Mason][William]
4 [Harper][Isabella][Ava][Noah]
5 [Emily][Daniel][Jacob][Elijah]
6 [Chloe][Aria][Elizabeth][Sophia][Olivia]
7 [Victoria][Ella][Aiden]
8 [Grace][Scarlett][Wyatt][Sebastian][Jackson][Michael][Lucas][Oliver]
9 [Penelope][Abigail][Charlotte][Emma][Samuel]
10 [Riley][Camila][Madison][Evelyn]
```

After adding all fifty distinct names and printing the hash table, we get the result above for the division method.

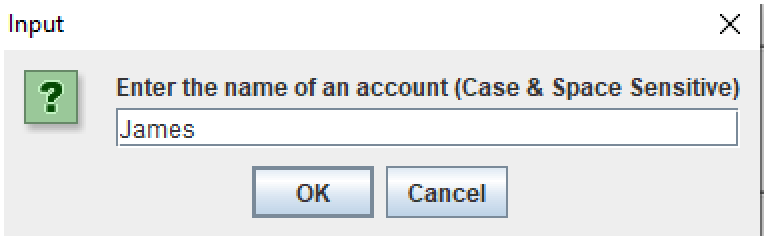

```
0 [Penelope][Camila][Charlotte][Sophia][Olivia][Aiden][Oliver]
1 [Emma][Carter][Joseph][William]
2 [Chloe][Aria][Scarlett][Emily][Mia][Lucas]
3 [Riley][Evelyn][Amelia][Isabella][Ethan][James]
4 [Elizabeth][David][Jackson]
5 [Avery]
6 [Victoria][Sofia][Sebastian]
7 [Grace][Madison][Abigail][Wyatt][Benjamin]
8 [Harper][Daniel][Jacob][Elijah][Noah]
9 [Ella][Ava][Samuel][Alexander][Michael]
10 [Henry][Matthew][Mason][Logan][Liam]
```

After adding all fifty distinct names and printing the hash table, we get the result above for the multiplication method.

B. Record a person as a new friend

Recording a person a new friend is practically the same for the multiplication and division method.

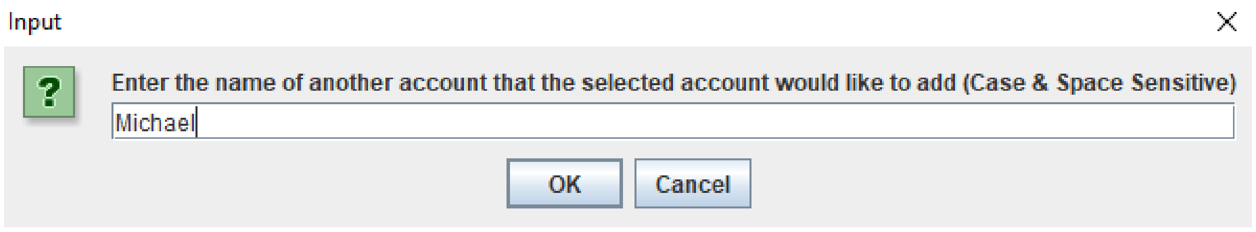
In our case we will be adding James and Michael.


Input

Enter the name of an account (Case & Space Sensitive)

James

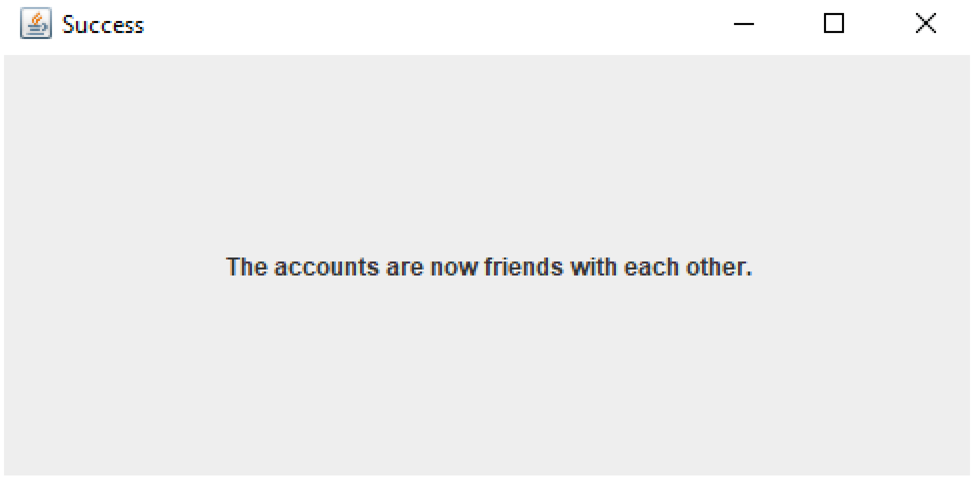
Inputting the first account.


Input

Enter the name of another account that the selected account would like to add (Case & Space Sensitive)

Michael

Inputting the second account

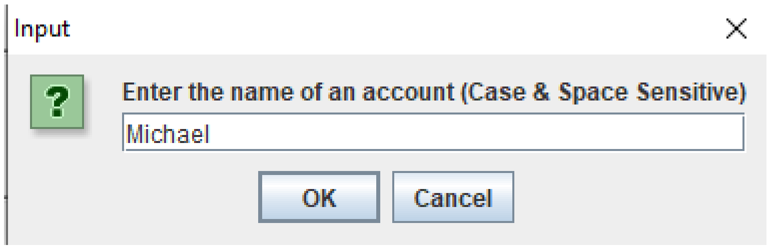

Success

The accounts are now friends with each other.

Success window shown.

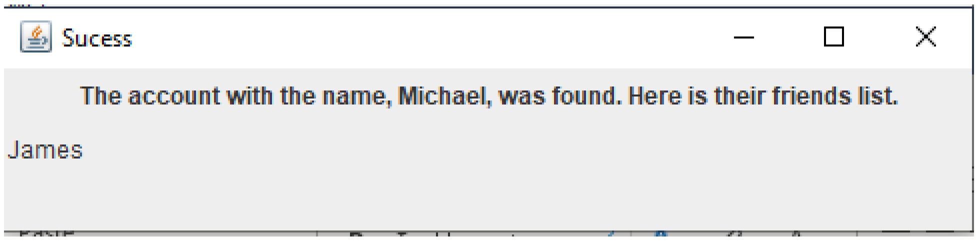
C. Search a person to list his/her friend list

Now that James is friends with Michael, we can search for Michael, and should see James under his friends list.


Input

Enter the name of an account (Case & Space Sensitive)

Michael

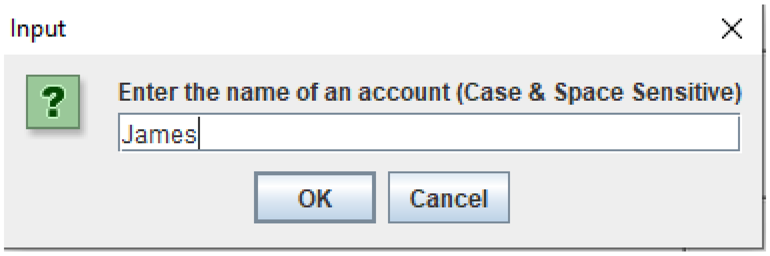

Sucess

The account with the name, Michael, was found. Here is their friends list.

James

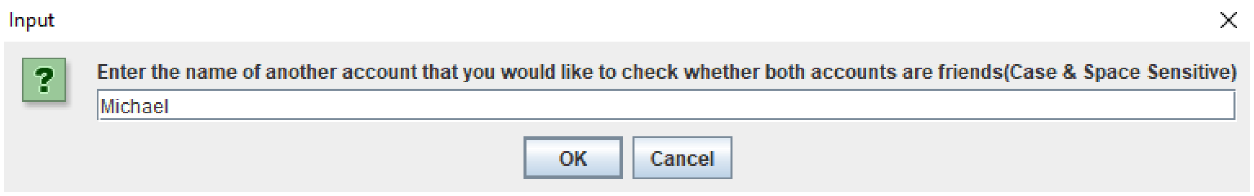
D. Enter two persons name and check whether the two people are friends.

We will search for James and Michael since they are already friends.


Input

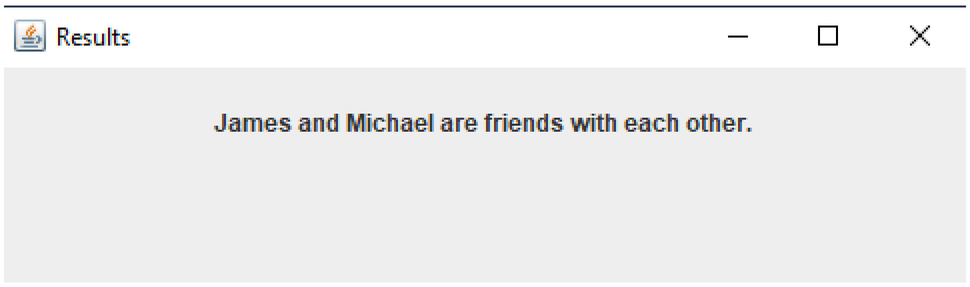
Enter the name of an account (Case & Space Sensitive)

James


Input

Enter the name of another account that you would like to check whether both accounts are friends(Case & Space Sensitive)

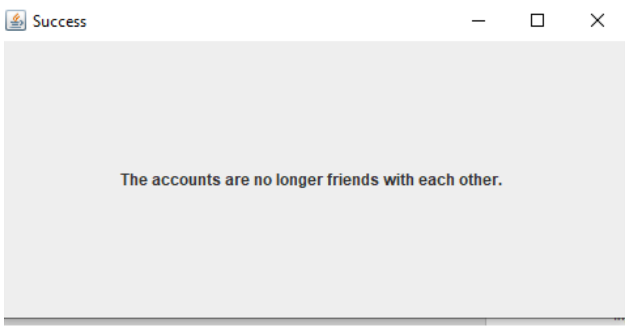
Michael


Results

James and Michael are friends with each other.

The result is correct.

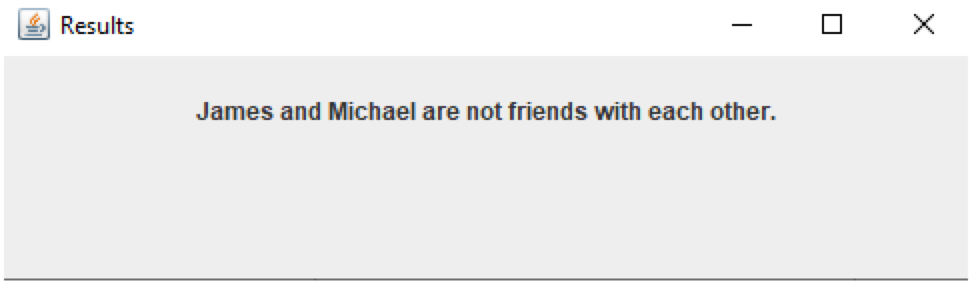
E. Remove a person as a friend


Success

The accounts are no longer friends with each other.

Window show when Michael and James are removed as friends

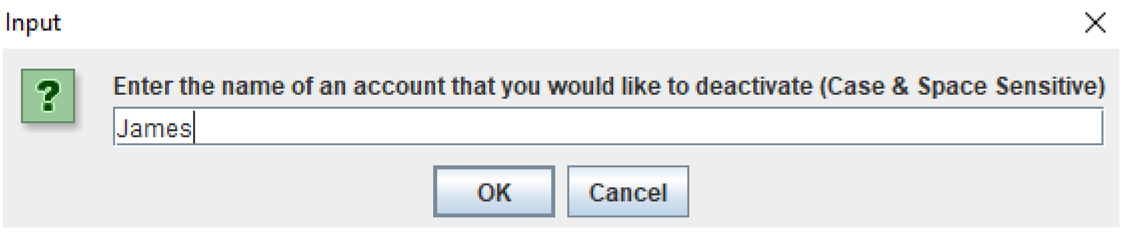
We can then use the checking button again to see if they are indeed no longer friends.


Results

James and Michael are not friends with each other.

E. Deactivating an account

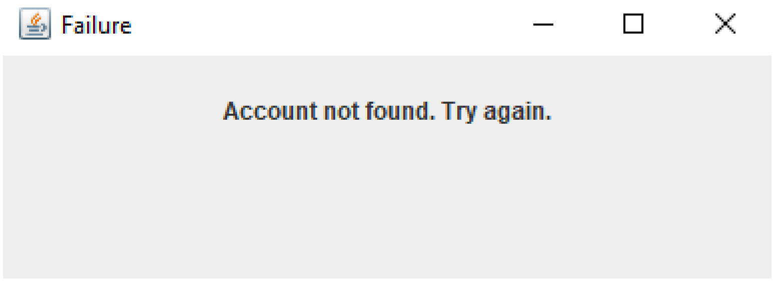
In this case, we will be deactivating James's account.


Input

Enter the name of an account that you would like to deactivate (Case & Space Sensitive)

James

Now when we try to search for the account, we get this.


Failure

Account not found. Try again.

V. How to Install the Application and Test the Codes

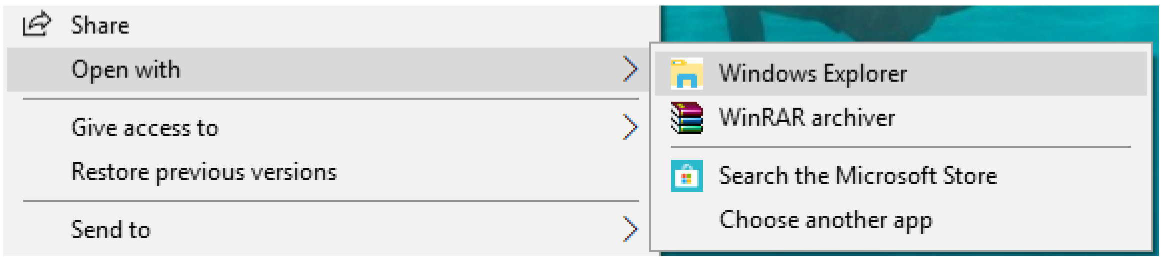
This section will demonstrate how to install the application and test the code of the programming assignment.

A. Required Programs

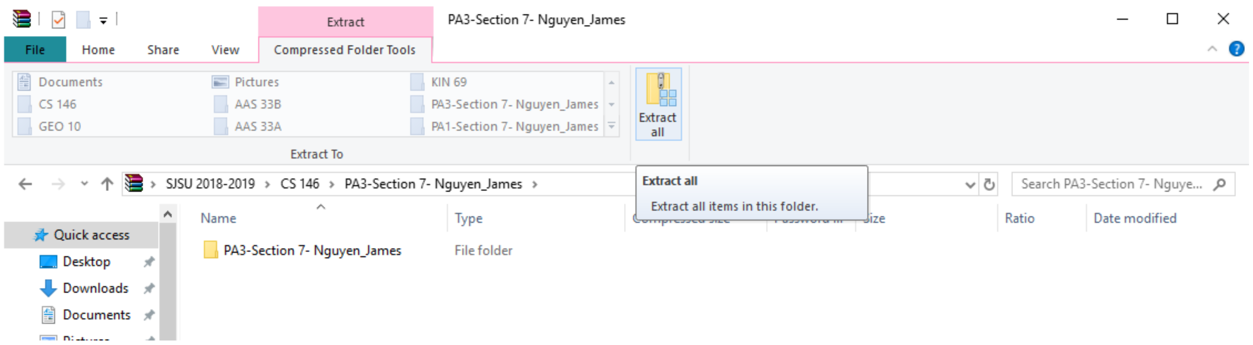
In order to test the application and the code, you will need to download Java which can be found at https://www.java.com/en/download/. This will allow you to run the executable jar file that comes in the zip folder. Furthermore, you will also need to have Eclipse downloaded which can be found at https://www.eclipse.org/downloads/.

B. Extracting the Folder

In order to run the application you will need to extract the zipped folder first. In order to do this, you first have to left click on the folder, and then under the drop-down menu, you will select open with, and then Windows Explorer.



Once you have opened it with Windows Explorer, you can then click the extract all button on the top of the window.


Extract
PA3-Section 7- Nguyen_James
File
Home
Share
View
Compressed Folder Tools
Documents
CS 146
GEO 10
Pictures
AAS 33B
AAS 33A
KIN 69
PA3-Section 7- Nguyen_James
PA1-Section 7- Nguyen_James
Extract all
Extract To
SJSU 2018-2019 › CS 146 › PA3-Section 7- Nguyen_James ›
Extract all
Extract all items in this folder.
Search PA3-Section 7- Nguye...
Quick access
Desktop
Downloads
Documents
Name
Type
Size
Ratio
Date modified
PA3-Section 7- Nguyen_James
File folder

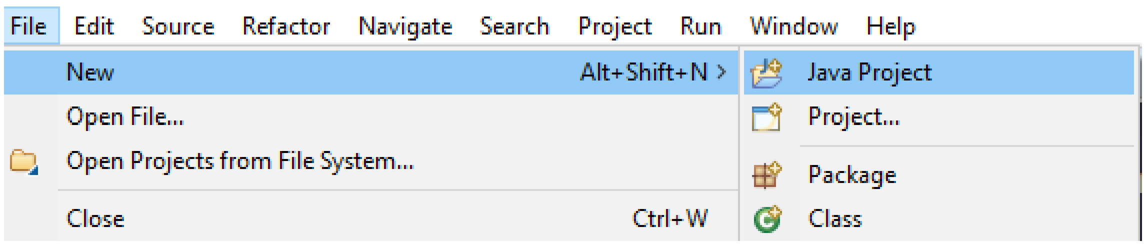
Then a pop-up window will ask you to select a destination window to extract the files to. Once it has been extracted, a folder containing the extracted files will pop up.

C. Running the Application

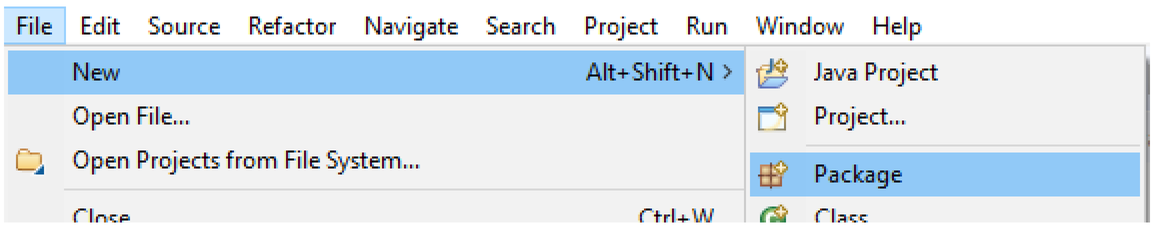
To run on the application, double click on the “Facebook Simulation .jar” application. A text file named “Random_SSN.txt” will then appear on your desktop, and a pop-up menu will show up in which you can pick a choice in how you would like to sort the file.

D. Testing the Code in Eclipse

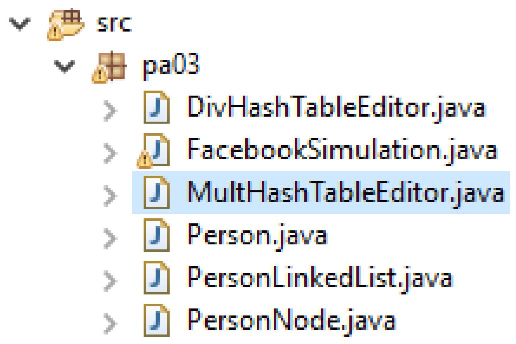
In order to test and view the code, you will have to open Eclipse first. After selecting a workspace in Eclipse, you will first press ‘File’ on the top left corner, then select new, and then Java Project.



You can name this new Java Project ‘Tester’. Next you will then go back to ‘File’, select new, and this time, you will select ‘Package’



You will then name this Package ‘pa03’. Once this has finished, you can then drag and drop the *.java files into the package.



Then, you can double click the files, and the source codes for both the java files will pop up. If you would like to run the code in eclipse, you can then press the green play button to run the main method of the Facebook Simulation java file. When the Simulation runs, you can see the hash tables of the multiplication and division method print as you insert more accounts.

VI. Problems Encountered during the Implementation

During the implementation of the Facebook Simulator, there were many different problems that occurred during the coding and execution of the program, and these problems needed to be fixed or debugged.

A. Implementation of Linked List

It has been a while since I have created my own linked list as I’ve been using the Linked List class of Java most of the time now. As a result, when I was implementing my own linked list, many different bugs occurred.

One of the bugs that occurred is that fact that I forget to set the head again if I were to delete a node or insert a node at the head or forget to reset the tail if I delete or inset node within the Linked List. I forgot about the different cases that occurs when deleting and inserting which led to many bugs in which Nodes were actually not being deleted and nodes were disappearing for no reason from the linked list.

Another bug was forgetting to call node.next within a loop to iterate through the linked list. As a result, the loop would never end, and cause my program to keep running.

B. Failure to Initialize

Another bug that occurred was forgetting to initialize instances within the code. There were many different instances that I needed to initialize as the program used many different Data Structures. Due to the large amount, I would sometimes forget to initialize a Linked List or Array List, causing the list to not actually do anything when call methods on it. As a result, I would be left with an empty list when I expected for something to occur.

C. Using JFrame and JOptionPane

This was my third time using the classes of JFrame and JOptionPane. Even though I was starting to get used to the implementation of the two different classes, I still experienced many problems.

The first problem was forgetting to set the close button of the window to end the program. As a result, for a long time, I didn't know that the program was running in the background of my computer, after pressing the close button. This heavily slowed my computer which finally caused me to notice. In order to fix this, I would set the button to exit on close which ends the program.

Another problem that was occurring was the incorrect sizes for the different JFrame windows. There were points where the windows were too large, and other times the windows were too small that you are not able to see all the text.

D. Failure to Have Mutual Friendships

While using the program I would forget that when you friend someone, the other person must also friend you back. As a result, the friendships in my program at one weren't mutual as one person was friends with another person, but it didn't go both ways.

This was also the case when I was defriending, as when an account defriended someone, I forgot to defriend the account from the defriended account as well cause the friendships to be non-mutual.

VII. Lessons Learned

Throughout this programming assignment, I learned many new concepts as well as gained many new skills and am further inspired for my enjoyment of Computer Science.

A. How Facebook Works

By completing this programming assignment, I was able to see exactly what Facebook does in the background. I use Facebook very often and almost daily, so it's very fascinating to know what processes are occurring when I friend or defriend someone on the internet, create an account, or want to deactivate my account. Through building my own micro version of Facebook, my eyes were opened as I now know the different data structures such as hash tables that Facebook uses, and how they are able to store so many accounts into hash tables. Next time when I got on Facebook, I will definitely remember what is occurring in the background!

B. Further Confidence in a User Interface

Although I faced many bugs while creating the User Interface, I definitely felt much more confident in my skills of the JFrame User Interface in Java. I found myself remember more information about the User Interface within Java, and that I no longer needed to refer to the API page to find methods to use. Furthermore, I also realized that the time to create the User Interface was drastically faster than the first time I did it. As a result, I am growing much more confident in building a User Interface in Java due to finishing this programing assignment.

C. Refresher on Linked List

I have not built a Linked List on my own since CS 46B. After completing this programming assignment, I am more refreshed about linked lists. Furthermore, in CS 46B we were guided by a Professor through comments to create the linked list, but in this assignment, I had to create the linked list all on my own. By creating it on my own, I was able to learn much more about the structure of the linked list and how to make the linked list my own.

D. Hash Tables

With this programming assignment, I was able to have a better understanding of hash tables as I am able to implement the data structure into a real-life example. I was able to see how useful hash tables as they are able to carry so much data such as in Facebook. The assignment also made me understand how exactly to use a hash table, and how the indexes of inputted items are dependent on the numerical value of the inputted item.

Furthermore, I was able to see the different hash functions one can use, specifically the hash functions of the Division Method and Multiplication Method. I was able to speculate the different ways items can be inserted into a hash table and have a better understanding to see what cases I would use what method to have a good performance for my applications in the future.

E. Inspiration for Computer Science and Career

By completing the programming assignment, I am again inspired for my computer science major as I was able to see that I am able to create a program all on my own without any directions like in CS 46B. Although I faced many bugs while programming this assignment, I was patient and continued to debug the program until everything was perfect. This further inspired my career goal of a Software Engineer as these traits of being patient and calm will effectively help me in the near future. This was also the third time in which I created an application all on my own, and without following any specific instruction which showed me that I can continue my endeavors in Computer Science. This programming assignment furthered opened my eyes and showed how Computer Science is something meant for me!