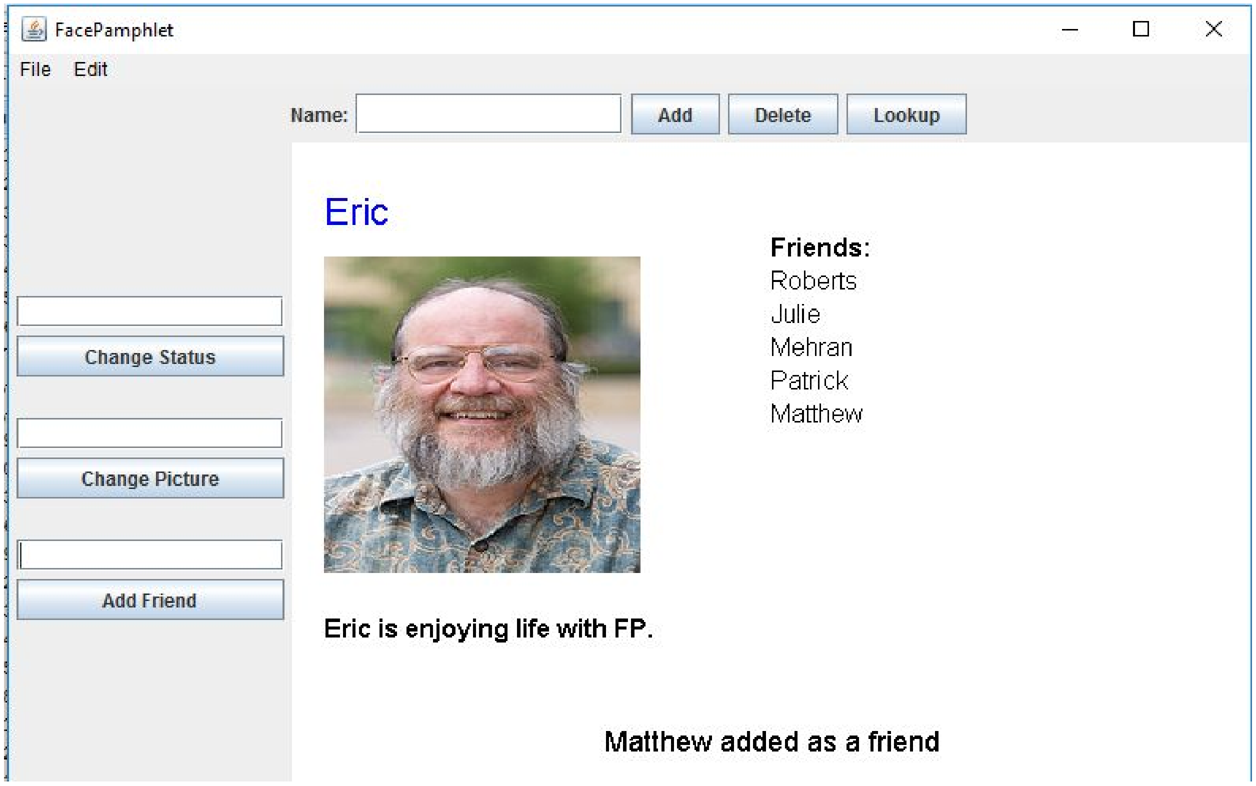
LBYCP12

Data Structures and Algorithm Analysis Laboratory



Eric

Friends:
Roberts
Julie
Mehran
Patrick
Matthew

Eric is enjoying life with FP.

Matthew added as a friend

Laboratory Activity 7

FacePamphlet

By

Chan, Patrick Matthew J, LBYCP12-EQ1

INTRODUCTION

This activity aims to further enhance the student's knowledge in utilizing various interactors (such as JLabels, JFields, and JButtons) in a GUI for the acm Program package. This program lets students create a simple social network, wherein people can play around with profiles, statuses, profile pictures, and lists of friends. So, as students are tasked with creating this social network, they are to apply their knowledge in Graph data structures, implement it, and integrate it into the linkages between the different profiles.

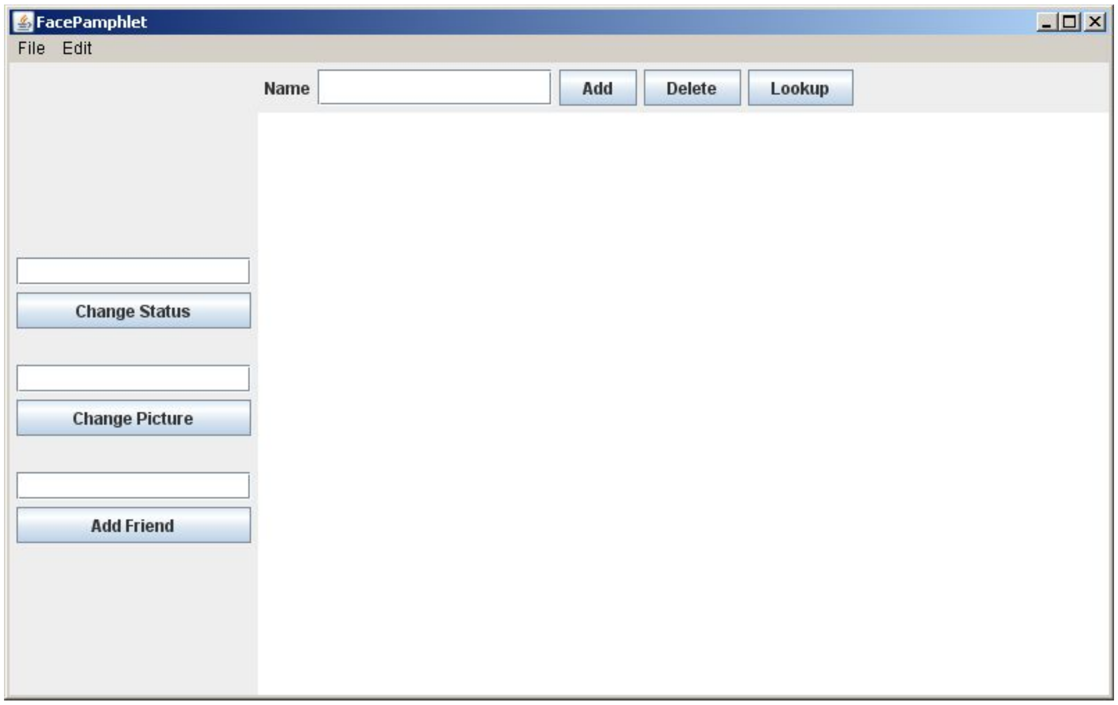
OBJECTIVES

- To further improve students knowledge on usage of various interactors in Java
- To allow the students to create their own implementation of a Graph data structure
- To integrate a graph data structure to take note of the connections between a vast collection of objects (profiles)

MATERIALS

1. Java SE Development Kit (JDK) 8
2. NetBeans integrated development environment (IDE)
3. `acm.jar` by the ACM Java Task Force
4. Provided starter kit for the FacePamphlet Assignment

PROCEDURE



1. Add the necessary interactors as shown in the picture above.
2. Add the necessary implementations for the actionPerformed method for the different interactors.
3. Temporarily set the main class to extend ConsoleProgram instead of Program, in order to test out the interactors.
4. Take note of the following when creating and implementing the interactors:
    a. All text fields are TEXT_FIELD_SIZE characters wide. TEXT_FIELD_SIZE is just a constant set in FacePamphletConstants.
    b. The Name text field in the NORTH region does not have any actionCommand associated with it. In other words, pressing the Enter key in that text field should have no effect, so you don't need to worry about that case.
    c. The three text fields in the WEST region do have actionCommands associated with them. The actionCommand associated with each respective text field should be the same as its corresponding button.

For example, pressing the Enter key in the text field next to the Change Status button should have the same effect as pressing the Change Status button.

d. If a text field is empty when its corresponding button is pressed, then nothing should happen. For example, if the Name text field in the NORTH region has nothing in it when the Add (or Delete, or Lookup) button is clicked (i.e., the text field's value is the empty string ("")), then we should simply not do anything as a result of the button click. This idea applies to all text fields in the application, and helps prevent situations such as trying to add a profile with an empty name, or trying to change the status of a profile to the empty string.

e. There are spaces between the various text field/button pairs (for example, there is space between the Change Status button and the text field associated with Change Picture). These spaces should be produced by adding a JLabel with the label text EMPTY_LABEL_TEXT (this is just a constant defined in FacePamphletConstants) at the appropriate points when adding interactors to the WEST border region. It would look something like this:

```
add(new JLabel(EMPTY_LABEL_TEXT), WEST);
```

5. Get the starter file for the FacePamhletProfile class.

6. Implement this class so that it encapsulates the information pertaining to one profile in the social network. That information consists of four parts:

    a. The name of the person with this profile, such as "Mehran Sahami" or "Julie Zelenski"

    b. The status associated with this profile. This is just a String indicating what the person associated with the profile is currently doing. Until it is explicitly set, the status should initially be the empty string.

    c. The image associated with that profile. This is a GImage. Until it is explicitly set, this field should initially be null since we don't initially have an image associated with a profile.

   d. The list of friends of this profile. The list of friends is simply a list of the names (i.e., list of Strings) that are friends with this profile. This list starts empty. The data structure you use to keep track of this list is left up to you.

7. Make a toString implementation for the FacePamphletProfile class, which contains amd divides each of this information to make it easier to test and debug later on.

8. Get the starter file for the FacePamphletDatabase class.

9. Implement this class so that it will be the class containing all of the FacePamphletProfiles in the social network whenever the program is running. The class should contain five public entries as follows:

   a. A constructor that has no parameters. You can use this to perform any initialization you may need for the database. Note: depending on how you implement the database, it is entirely possible that your constructor may not need to do anything. It's perfectly fine if that's the case.

   b. An addProfile method that is passed a FacePamphletProfile, and is responsible for adding that profile to the database. Note that profile names are unique identifiers for profiles in the database. In other words, no two profiles in the database should have the same name and the name associated with a profile will never change. If a client were to call the addProfile method with a profile that has the same name as an already existing profile in the database, then the existing profile should be replaced by the new profile. Depending on what data structure you use to keep track of the database, this behavior may actually be quite easy to implement. Please note: the behavior of replacing an existing profile with a new one that has the same name is the behavior defined for the addProfile method of the FacePamphletDatabase class (just to be precise about what should happen in that case), but your program will likely not actually make calls to the addProfile method with a profile that has the same name as an existing profile in the database. Rather, your FacePamphlet program will eventually not actually allow the user to create a new profile with the same name as an existing profile.

c. A getProfile method that takes a name, looks it up in the database of profiles, and returns the FacePamphletProfile with that name, or null if there is no profile with that name.

d. A deleteProfile method that takes a profile name, and deletes the profile with that name from the profile database. Note that when we delete a profile from the database, we not only delete the profile itself, but we also update all other profiles in the database so as to remove the deleted profile's name from any friends lists in other profiles. In this way, we ensure that someone cannot be friends with a person who does not have a profile in the database.

e. A containsProfile method that takes a profile name, and returns true if there is a profile with that name in the database. Otherwise, it returns false.

10. Try to to represent the data so that you can implement the methods above as simply and as efficiently as possible. You may choose to immediately integrate a graph from here, or later on, when the GUI has been completed.

11. Test your newly implemented classes by adding code to the FacePamphlet program so that it creates the FacePamphletDatabase; and then, change the code for the Add, Delete, and Lookup button handlers as follows:

    a. Entering a name in the Name text field and clicking the Add button looks up the current name in the database to see if a profile with that name already exists. If the name does not exist, then it adds a new profile to the database and prints out "Add: new profile: " followed by the string version of the profile (using the toString method of the FacePamphletProfile). If the profile name already exists in the database, then it prints out the fact that the profile with that name already exists followed by the string representation of the profile.

   b. Entering a name in the Name text field and clicking the Delete button looks up the current name in the database to see if it exists. If the name does exist, then it deletes the profile with that name from the database and prints out that the profile was deleted. If the profile name does not exist in the database, then it simply prints out that a profile with the given name does not exist.

   c. Entering a name in the Name text field and clicking the Lookup button looks up the current name in the database to see if it exists. If the name does exist, then prints out "Lookup: " followed by the string version of the profile. If the name does not exist, then it prints out that a profile with the given name does not exist.

12. To continue with the implementation for the functionality for the Change Status, Change Picture, and Add Friend buttons, the program should have the notion of a current profile.

13. In adding a notion of a current profile, update the code for the Add, Delete, and Lookup button handlers so that:

    a. Whenever a new profile is added, the current profile is set to be the newly added profile. If the user tried to add a profile with the name of an existing profile, then the existing profile with that name is set to be the current profile (this is similar to the case below where the users simply looks up an existing profile).

    b. Whenever a profile is deleted (whether or not the profile to be deleted exists in the database), there is no longer a current profile (regardless of what the current profile previously was).

    c. Whenever the user lookups up a profile by name, the current profile is set to be the profile that the user looked up, if it exists in the database. If a profile with that name does not exist in the database, then there is no longer a current profile (regardless of what the current profile previously was).

14. Implement Change Status so that if the user enters some text in the text field associated with the Change Status button and either presses the Change Status button or hits Enter, the application should update as follows:

    a. If there is a current profile, then the status for that profile should be updated to the text entered, and you can just print out a message to that effect.

    b. If there is no current profile, then you should simply prompt the user to select a profile to change the status of (and there should be no changes to any of the profiles in the database).

15. Implement Change Picture so that if the user enters some text in the text field associated with the Change Picture button and either presses the Change Picture button or hits Enter, the application should update as follows:

    a. If there is a current profile, then we need to see if the we can create a GImage with the filename of the text entered in the text field. Checking to see if a valid image file exists can be accomplished using the code fragment below (where filename is a String containing the name of the image file we are trying to open):

```
GImage image = null;
try {
image = new GImage(filename);
} catch (ErrorException ex) {
// Code that is executed if the filename cannot be opened.
}
```

    b. Note in the code fragment above that the variable image will still have the value null if we were unable to open the image file with the given filename. Otherwise, the value of the variable image will be a valid GImage object (whose value will not be null).

    c. If we obtained a valid GImage, then the image for the current profile should be updated to this image, and you can print out a message to that effect (although you won't be able to display the actual image for now).

    d. If there is no current profile, then you should simply prompt the user to select a profile to change the image of (and there should be no changes to any of the profiles in the database).

16. You can test this out with your own pictures, or with the provided pictures in the starter kit.

17. Implement Add Friend so that if the user enters some text in the text field associated with the Add Friend button and either presses the Add Friend button or hits Enter, the application should update as follows:

    a. If there is a current profile, then we need to see if the name entered in the text field is the name of a valid profile in the database. If it is, then we try to add the named friend to the list of friends for the current profile. If the named friend already exists in the list of friends for the current profile, then we simply write out a message that such a friend already exists. If that named friend does not previously exist in the list of friends (i.e., it was successfully added to the list of friends for the current profile), then (recalling that friendships are reciprocal) we also need to update the profile of the named friend to add the name of the current profile to its list of friends. For example, if the current profile was “Mehran” and we tried to add as a friend “Julie” (which, say, is the name of valid profile in the database, which is not already a friend of Mehran), then we should add Julie as a friend of Mehran and also add Mehran as a friend of Julie.

   b. If the name entered in the Add Friend text field is not a valid profile in the system, we should just print out a message to that effect.

   c. If there is no current profile, then you should simply prompt the user to select a profile to add a friend to (and there should be no changes to any of the profiles in the database).

18. Get the starter file for the FacePamphletCanvas class.

19. Implement the showMessage and displayProfile methods with the following parameters and behavior:

    a. A showMessage method that is passed a String, and is responsible for displaying that string as the Application Message at the bottom of the canvas.

       i. The method should display this Application Message text centered horizontally with respect to the width of the canvas, and the vertical baseline for the text should be located BOTTOM_MESSAGE_MARGIN pixels up from the bottom of the canvas.

       ii. The font for the text should be set to MESSAGE_FONT. Note that BOTTOM_MESSAGE_MARGIN and MESSAGE_FONT are simply constants defined in FacePamphletConstants.

       iii. Whenever this method is called, any previously displayed message is replaced with the new message text that is passed in.

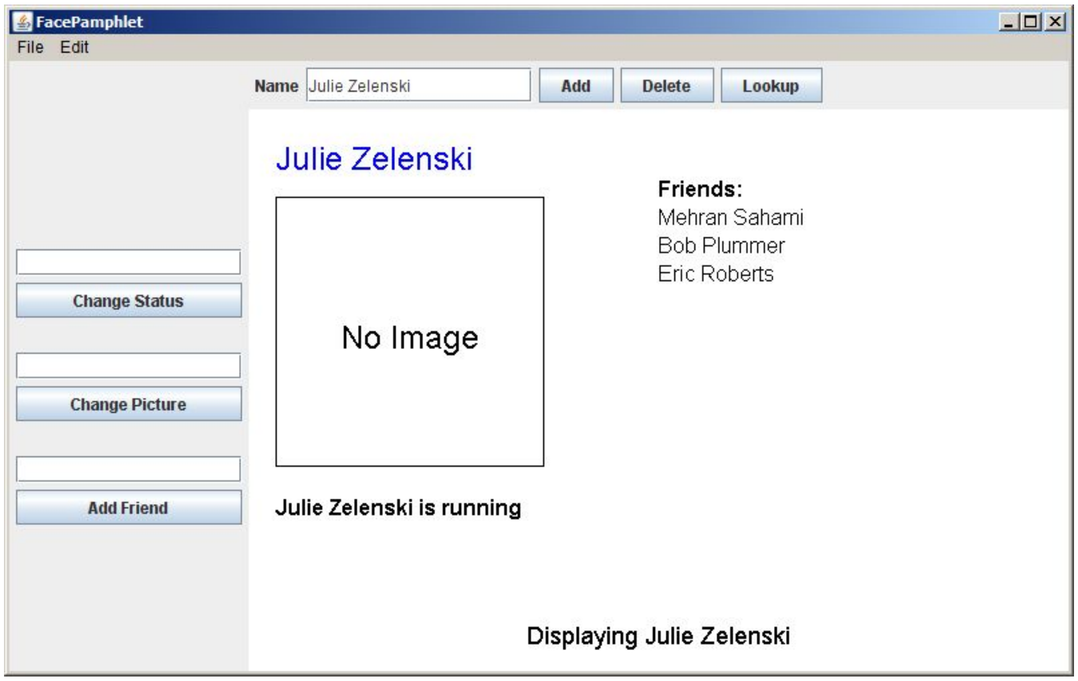
    b. A displayProfile method that is passed a FacePamphletProfile, and is responsible for displaying the contents of that profile in the canvas, including the profile's name, image (if any), the status of the profile (if any), and the list of friends (if any). Whenever this method is called, all existing contents of the canvas should be cleared (including any previously displayed profile as well as any displayed

Application Messages), and the profile passed in should be displayed.

20. Go back to the FacePamphlet class and change its definition so that it extends Program rather than the temporary expedient of extending ConsoleProgram.

21. Here, declare a FacePamphletCanvas private instance variable.

22. Change the constructor of the FacePamphlet class so that it creates a new FacePamphletCanvas object and adds that object to the display.



23. Implement displayProfile class so that it displays these components with the following conditions:

    a. Name: Near the top of the display, the name associated with the profile ("Julie Zelenski" in the example above) should be displayed in the color Blue. Horizontally, the text should located LEFT_MARGIN pixels in from the left-hand side of the canvas. Vertically, the top of the text (not its baseline) should be TOP_MARGIN pixels from the top of the canvas. The font for the text should be set to PROFILE_NAME_FONT.

    b. Image: Although there is currently no image associated with the profile above, we can see that there is space set aside to display an image immediately under the name of the profile. The space for the image will always be IMAGE_WIDTH by IMAGE_HEIGHT pixels. If no image is associated with the profile then a rectangle of the dimensions of the image size should be drawn. Horizontally, this rectangle should be located LEFT_MARGIN pixels in from the left-hand side of the canvas. Vertically, the top of the rectangle should be should be IMAGE_MARGIN pixels below the baseline of the profile name text. Centered (both horizontally and vertically) within this rectangle should be the text "No Image" in the font PROFILE_IMAGE_FONT. If an image is associated with the profile then the image should be displayed (in the same location as the rectangle described above). The image should be scaled so that it displays with IMAGE_WIDTH by IMAGE_HEIGHT pixels. The scale method of GImage should be useful to make image display with the appropriate size.

c. Status: Under the area for the image, the current status of the person with this profile should be displayed (Julie's status is "running" in the example above). If the profile currently has no status (i.e., it has an empty status string), the text "No current status" should be displayed. If the profile does have a status, the status text should have the name of the profile followed by the word "is" and then the status text for the profile. In any case, the line describing the profile's status should be located horizontally LEFT_MARGIN pixels in from the left-hand side of the canvas. Vertically, the top of the text (not its baseline) should be located STATUS_MARGIN pixels below the bottom of the image. The font for the text should be set to PROFILE_STATUS_FONT.

d. Friends: To the right of the profile's name, there is the header text "Friends:", and the names of the friends of this profile (e.g., Mehran Sahami, Bob Plummer, and Eric Roberts) are listed below. The start of the header text "Friends:" should be horizontally located at the midpoint of width of the canvas. Vertically, the baseline for this text should be the same as the top of the image area. The "Friends:"

header text should be displayed in the font PROFILE_FRIEND_LABEL_FONT. Immediately below the header, the friends of this profile should be listed sequentially, one per line, with the same horizontal location as the "Friends:" header text. You can use the getHeight() method of GLabel to determine how to vertically space out the list of friends to get one friend per line. The friend names should be displayed in the font PROFILE_FRIEND_FONT.

   e. Application Message: As described previously (but repeated here for completeness) the Application Message text ("Displaying Julie Zelenski" in the example above) should be centered with respect to the width of the canvas, and the baseline for the text should be located BOTTOM_MESSAGE_MARGIN pixels up from the bottom of the canvas. The font for the text should be set to MESSAGE_FONT.

24. Test this implementation by running the program and creating a single profile.

25. Finish up the GUI implementation by finalizing the behavior of each interactor with these conditions:

    a. Adding a Profile

       i. When a new profile is being added you should see if a profile with that name already exists. If it does, you should display the existing profile and give the user the message “A profile with the name <name> already exists”. If the profile does not already exist, you should display the newly created profile and give the user the message “New profile created”.

    b. Deleting a Profile

       i. When a profile is being deleted you should see if a profile with that name exists. If it does, you should delete the profile, clear any existing profile from the display, and give the user the message “Profile of <name> deleted”. If the profile does not exist, you should clear any existing profile from the display, and give the user the message “A profile with the name <name> does not exist”.

c. Looking up a Profile

    i. When a profile is being looked up you should see if a profile with that name exists. If it does, you should display the profile, and give the user the message “Displaying <name>”. If the profile does not exist, you should clear any existing profile from the display, and give the user the message “A profile with the name <name> does not exist”.

d. Changing Status

    i. When the status for a profile is being changed, you should determine if there is a current profile. If no current profile exists, you should just give the user the message “Please select a profile to change status”. If there is a current profile, you should update its status, redisplay the profile (to show the changed status), and give the user the message “Status updated to <status>”.

e. Changing Picture

    i. When the picture for a profile is being changed, you should determine if there is a current profile. If no current profile exists, you should just give the user the message “Please select a profile to change picture”. If there is a current profile, you should see if the filename given for the picture contains a valid image, and if it does, you should add the image to the profile, redisplay the current profile (to show the new image), and give the user the message “Picture updated”. If the given filename could not be opened, you should just give the user the message “Unable to open image file: <filename>”. In that case, the image associated with the profile is unchanged.

f. Adding Friend

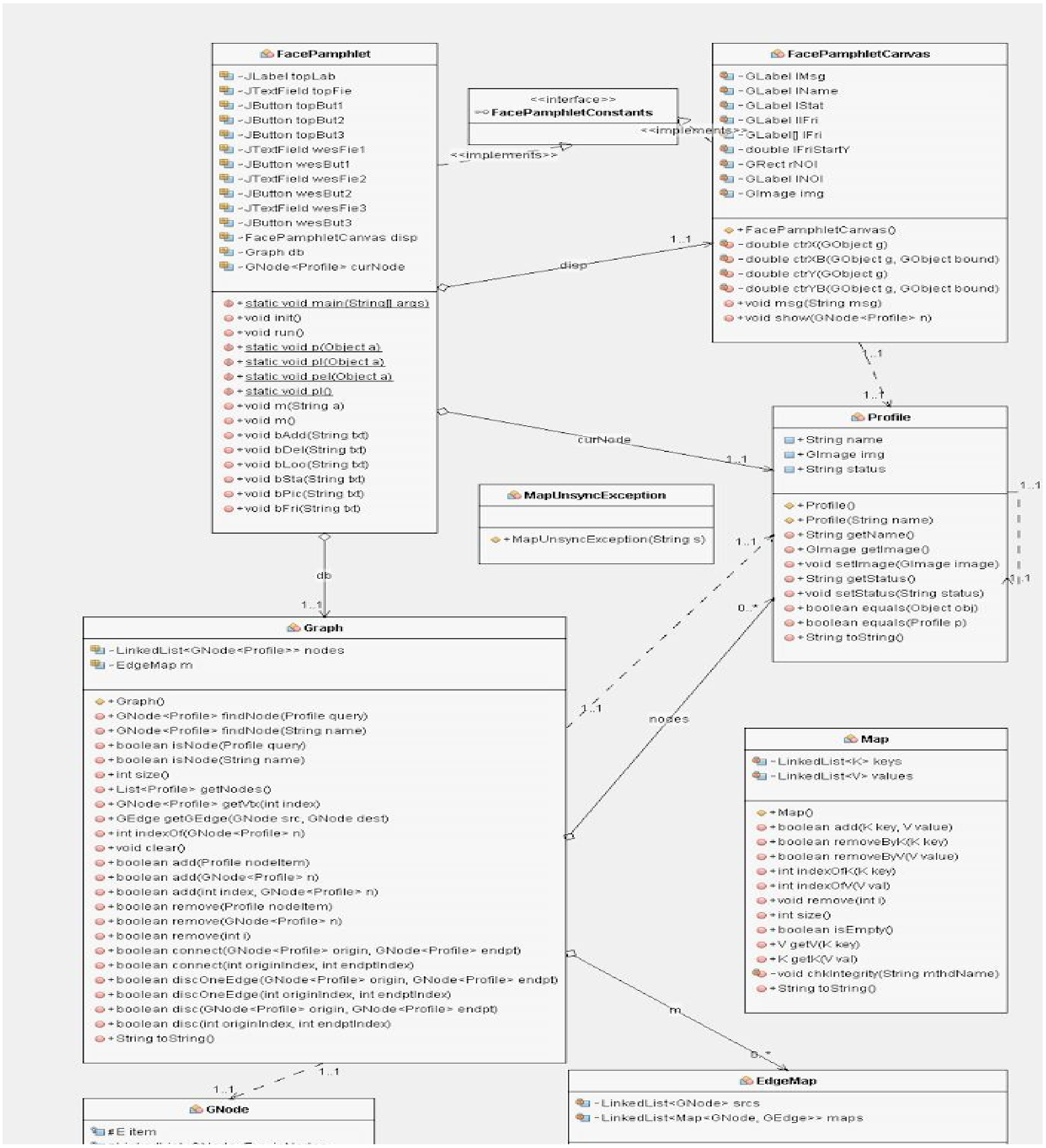
i. When a friend is being added to a profile, you should determine if there is a current profile. If no current profile exists, you should just give the user the message “Please select a profile to add friend”. If there is a current profile, you should see if the given friend name is the name for a valid profile in the social network. If the name is valid and the current profile does not already have that person as a friend, then you should update the friend list for both the current profile and the named friend, redisplay the current profile (to show the addition of the friend), and give the user the message “<friend name> added as a friend”. If the named friend is already a friend of the current profile, you should just display the message “<name of current profile> already has <friend name> as a friend.” If the named friend does not have a profile in the social network, then you should simply display the message “<friend name> does not exist.”

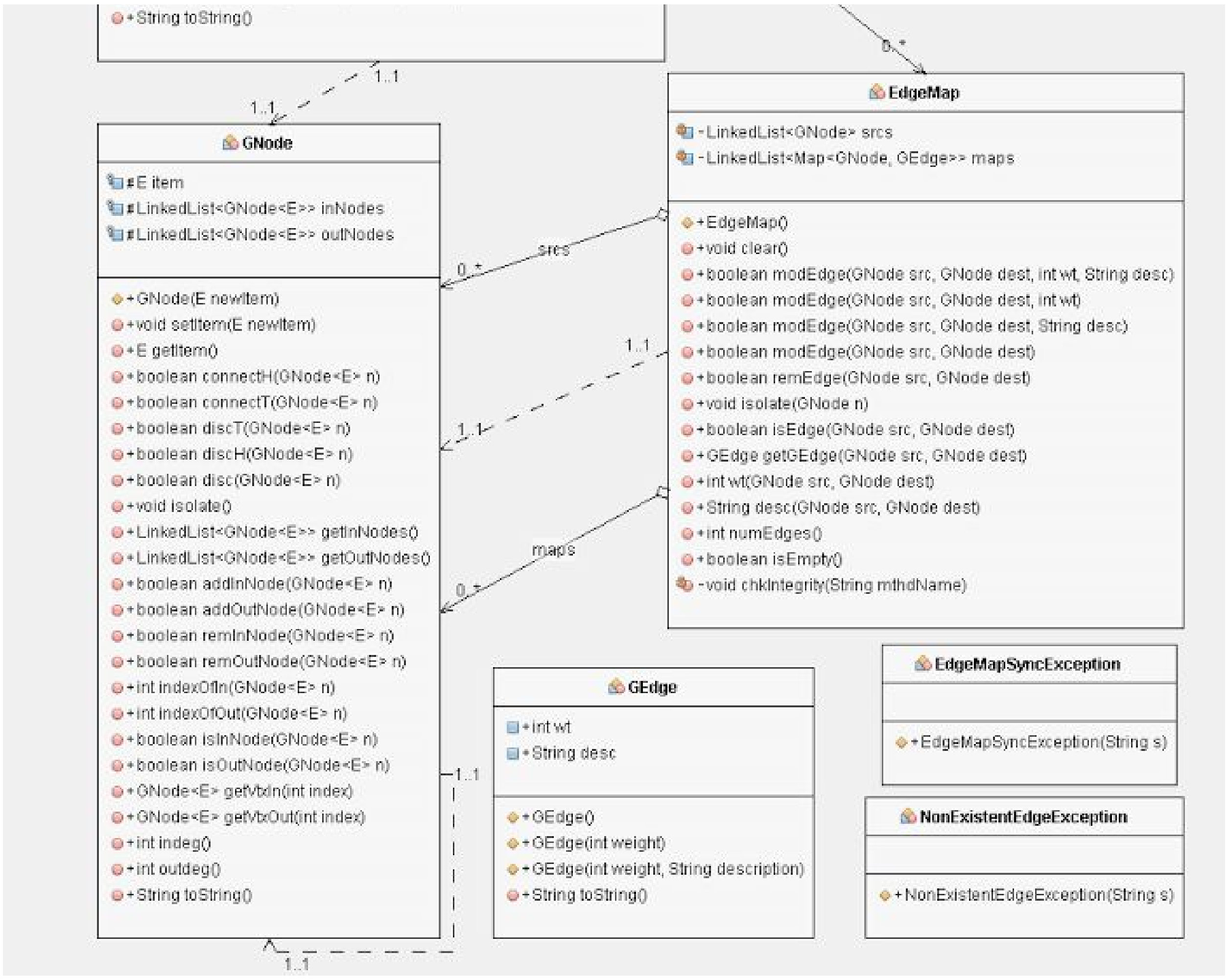
26. Make your own implementation of a Graph data structure

27. Make the necessary changes to the code of the program, so that the friend list is recorded with a Graph data structure.

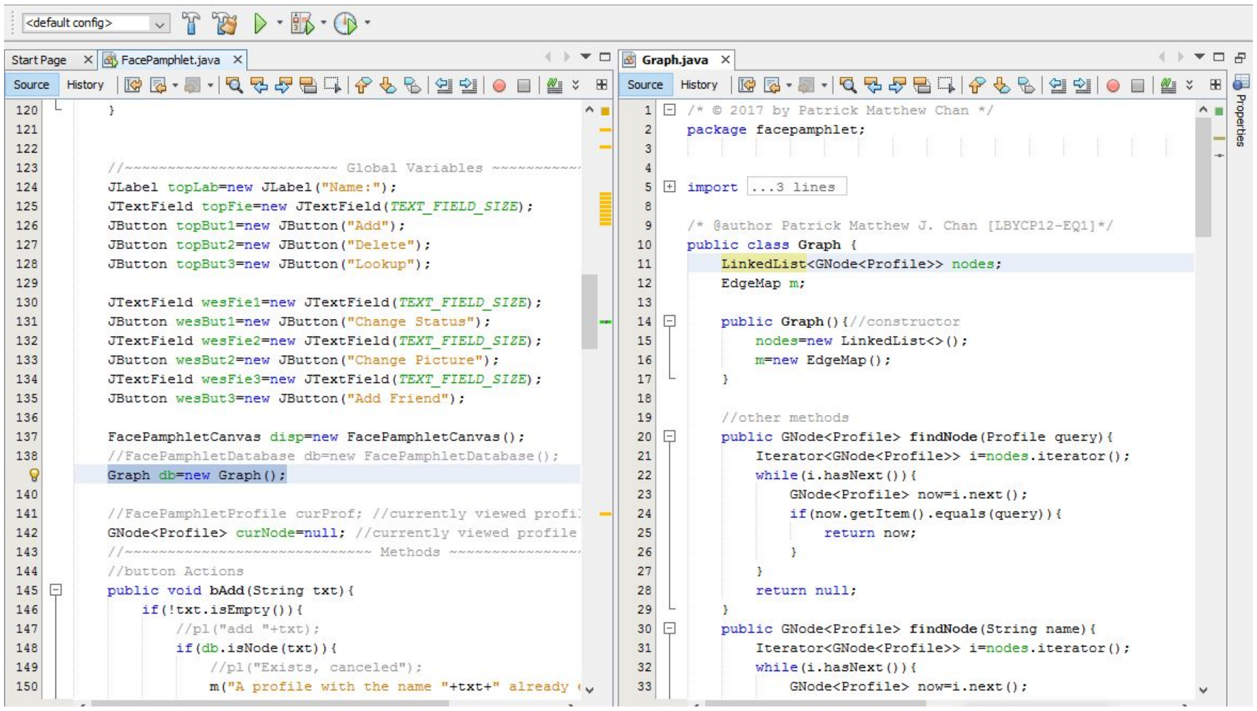
RESULTS AND DISCUSSION

UML:




+String toString()
1..1
1..1
0..*
EdgeMap
-LinkedList<GNode> srcs
-LinkedList<Map<GNode, GEdge>> maps
+EdgeMap()
+void clear()
+boolean modEdge(GNode src, GNode dest, int wt, String desc)
+boolean modEdge(GNode src, GNode dest, int wt)
+boolean modEdge(GNode src, GNode dest, String desc)
+boolean modEdge(GNode src, GNode dest)
+boolean remEdge(GNode src, GNode dest)
+void isolate(GNode n)
+boolean isEdge(GNode src, GNode dest)
+GEdge getGEdge(GNode src, GNode dest)
+int wt(GNode src, GNode dest)
+String desc(GNode src, GNode dest)
+int numEdges()
+boolean isEmpty()
-void chkIntegrity(String mthdName)
GNode
#E item
#LinkedList<GNode<E>> inNodes
#LinkedList<GNode<E>> outNodes
+GNode(E newItem)
+void setItem(E newItem)
+E getItem()
+boolean connectH(GNode<E> n)
+boolean connectT(GNode<E> n)
+boolean discT(GNode<E> n)
+boolean discH(GNode<E> n)
+boolean disc(GNode<E> n)
+void isolate()
+LinkedList<GNode<E>> getInNodes()
+LinkedList<GNode<E>> getOutNodes()
+boolean addInNode(GNode<E> n)
+boolean addOutNode(GNode<E> n)
+boolean remInNode(GNode<E> n)
+boolean remOutNode(GNode<E> n)
+int indexOfIn(GNode<E> n)
+int indexOfOut(GNode<E> n)
+boolean isInNode(GNode<E> n)
+boolean isOutNode(GNode<E> n)
+GNode<E> getVtxIn(int index)
+GNode<E> getVtxOut(int index)
+int indeg()
+int outdeg()
+String toString()
srcs
0..*
1..1
1..1
maps
0..*
1..1
1..1
GEdge
+int wt
+String desc
+GEdge()
+GEdge(int weight)
+GEdge(int weight, String description)
+String toString()
EdgeMapSyncException
+EdgeMapSyncException(String s)
NonExistentEdgeException
+NonExistentEdgeException(String s)

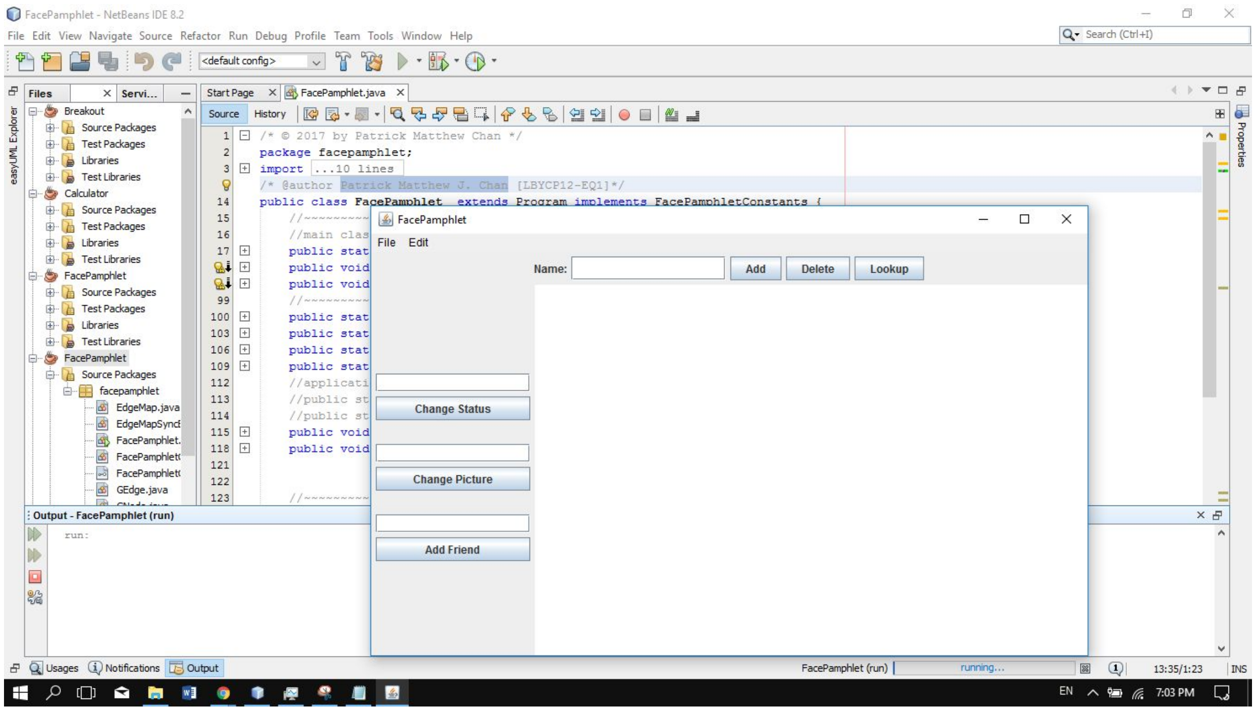
Screenshots:



```
    }


    //~~~~~~~~~~~~~~~~~~~~~~~~~ Global Variables ~~~~~~~~~~~
    JLabel topLab=new JLabel("Name:");
    JTextField topFie=new JTextField(TEXT_FIELD_SIZE);
    JButton topBut1=new JButton("Add");
    JButton topBut2=new JButton("Delete");
    JButton topBut3=new JButton("Lookup");

    JTextField wesFie1=new JTextField(TEXT_FIELD_SIZE);
    JButton wesBut1=new JButton("Change Status");
    JTextField wesFie2=new JTextField(TEXT_FIELD_SIZE);
    JButton wesBut2=new JButton("Change Picture");
    JTextField wesFie3=new JTextField(TEXT_FIELD_SIZE);
    JButton wesBut3=new JButton("Add Friend");

    FacePamphletCanvas disp=new FacePamphletCanvas();
    //FacePamphletDatabase db=new FacePamphletDatabase();
    Graph db=new Graph();

    //FacePamphletProfile curProf; //currently viewed profi
    GNode<Profile> curNode=null; //currently viewed profile
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Methods ~~~~~~~~~~~~~~~
    //button Actions
    public void bAdd(String txt){
        if(!txt.isEmpty()){
            //pl("add "+txt);
            if(db.isNode(txt)){
                //pl("Exists, canceled");
                m("A profile with the name "+txt+" already
```

```
/* © 2017 by Patrick Matthew Chan */
package facepamphlet;

import ...3 lines

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class Graph {
    LinkedList<GNode<Profile>> nodes;
    EdgeMap m;

    public Graph(){//constructor
        nodes=new LinkedList<>();
        m=new EdgeMap();
    }

    //other methods
    public GNode<Profile> findNode(Profile query){
        Iterator<GNode<Profile>> i=nodes.iterator();
        while(i.hasNext()){
            GNode<Profile> now=i.next();
            if(now.getItem().equals(query)){
                return now;
            }
        }
        return null;
    }
    public GNode<Profile> findNode(String name){
        Iterator<GNode<Profile>> i=nodes.iterator();
        while(i.hasNext()){
            GNode<Profile> now=i.next();
```

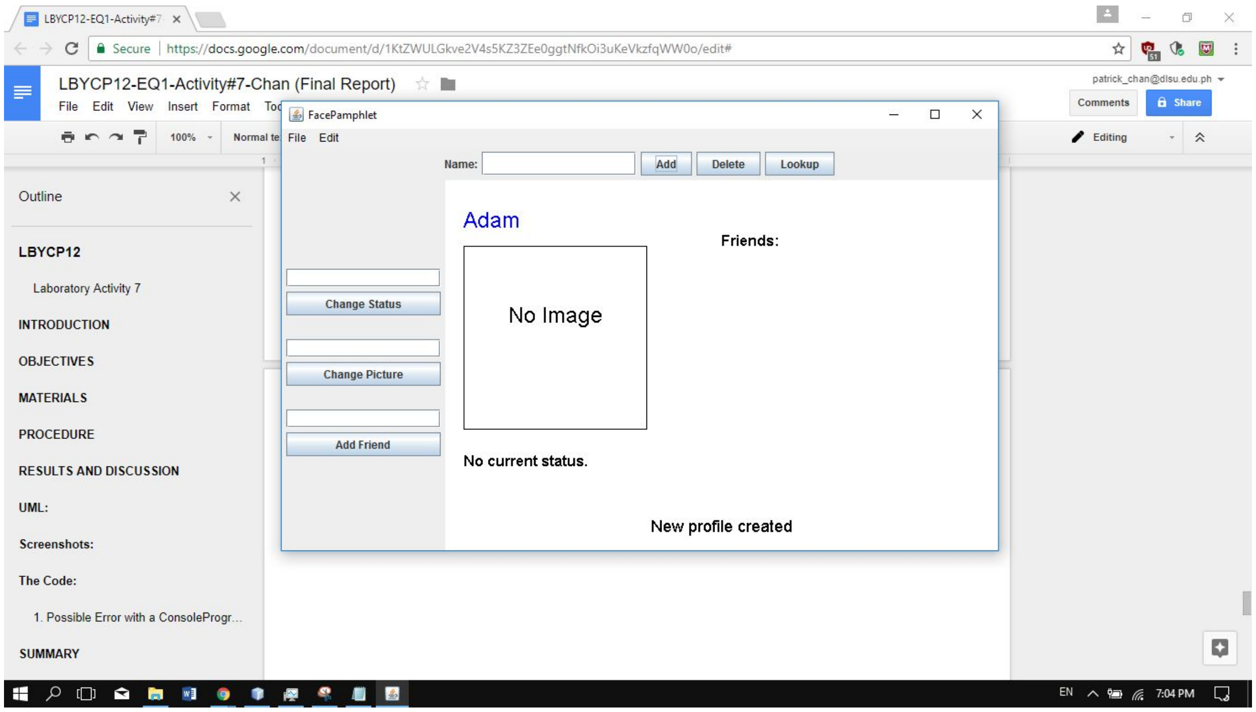
Program uses Graph Data Structure with Linked List Implementation.



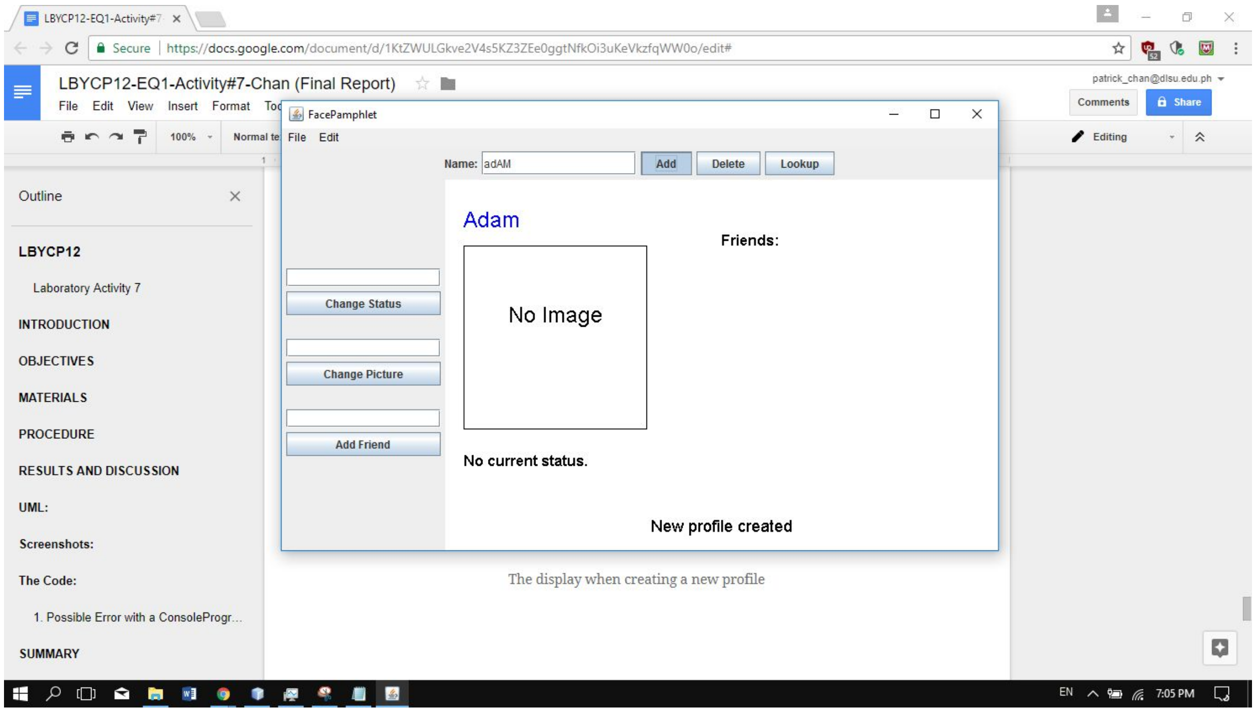
```
/* © 2017 by Patrick Matthew Chan */
package facepamphlet;
import ...10 lines
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class FacePamphlet  extends Program implements FacePamphletConstants {
    //~~~~~~~~~
    //main clas
    public stat
    public void
    public void
    //~~~~~~~~~
    public stat
    public stat
    public stat
    public stat
    //applicati
    //public st
    //public st
    public void
    public void


    //~~~~~~~~~
```

```
run:
```

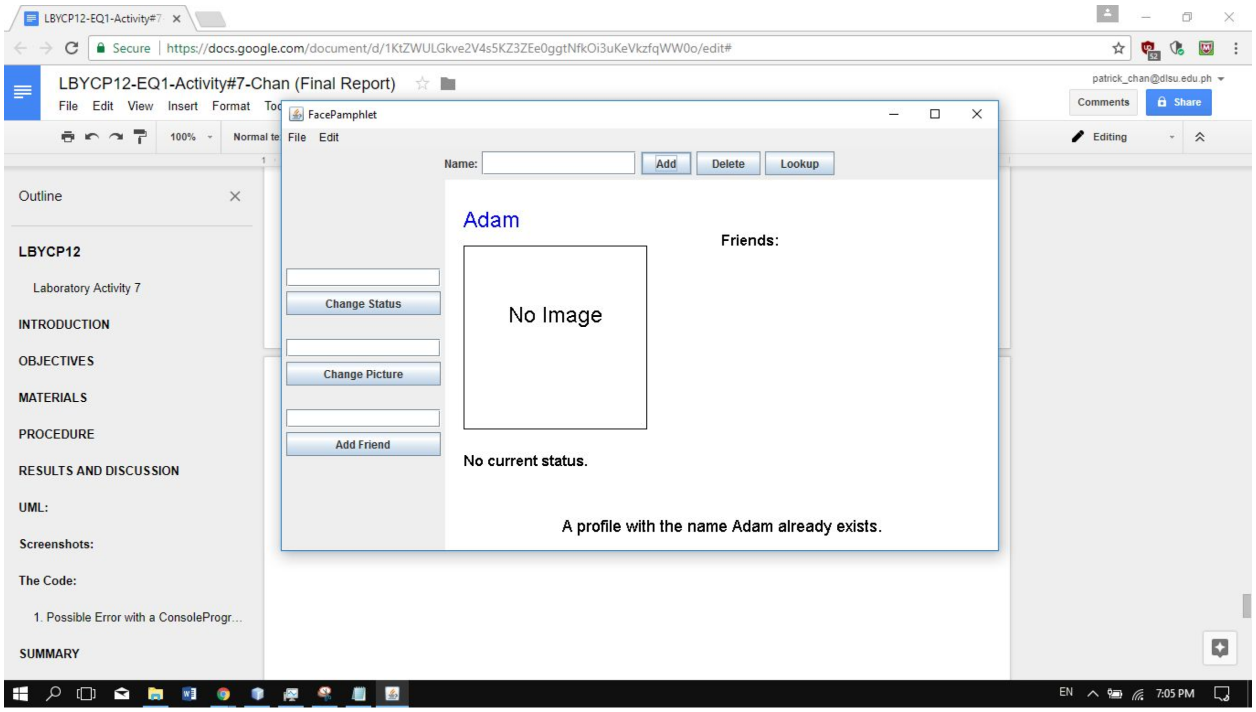
Program at startup.



The display when creating a new profile. (The user input in the field is automatically erased when the input is accepted)



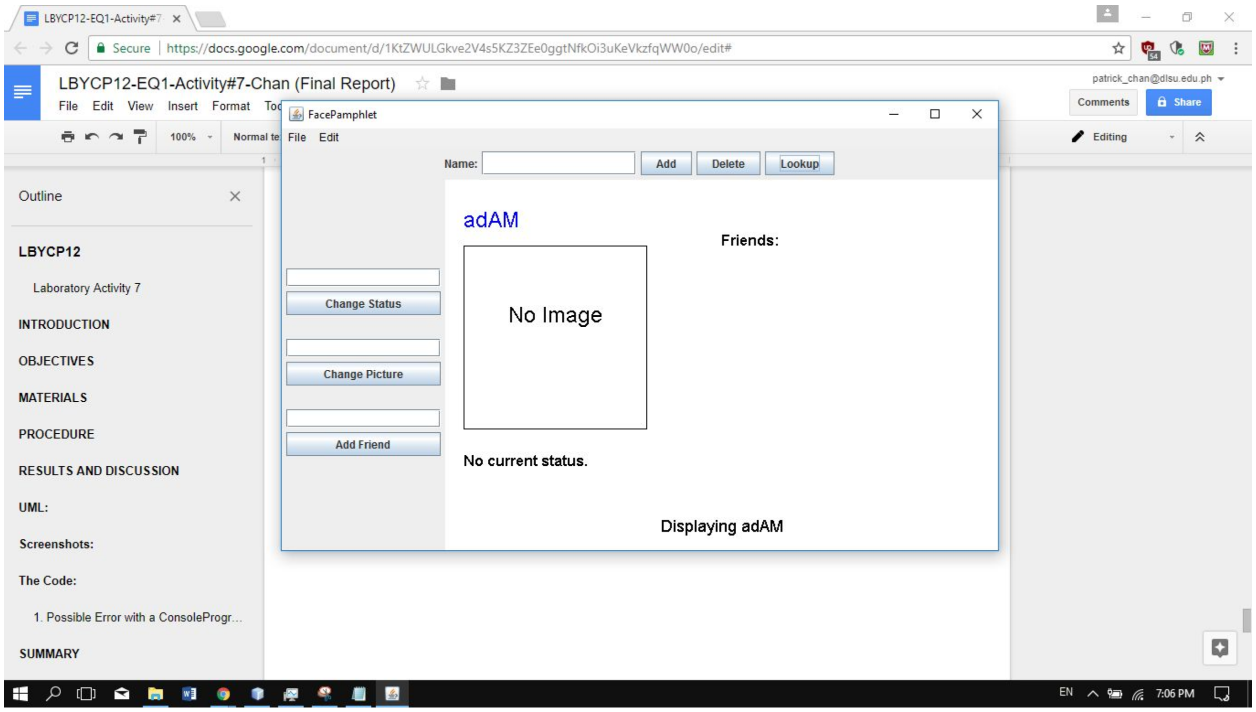
Profile names are case sensitive.



No current status.

A profile with the name Adam already exists.

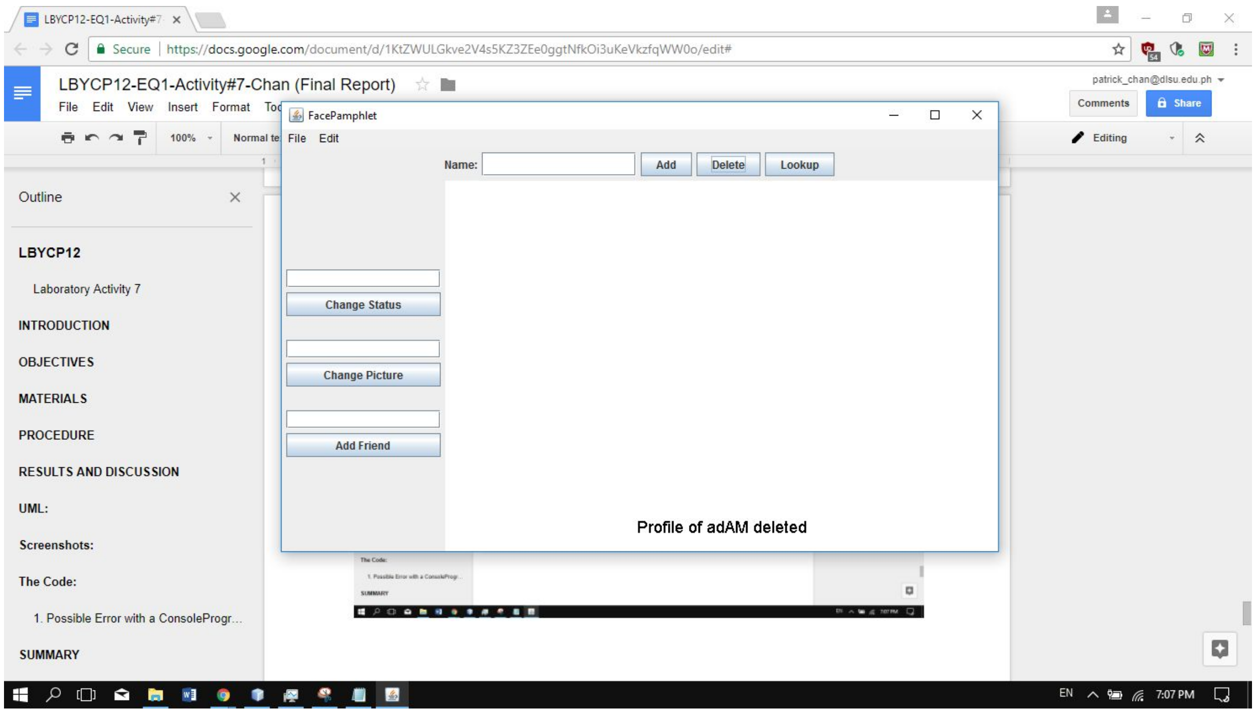
The user cannot add another profile of the same name.



No current status.

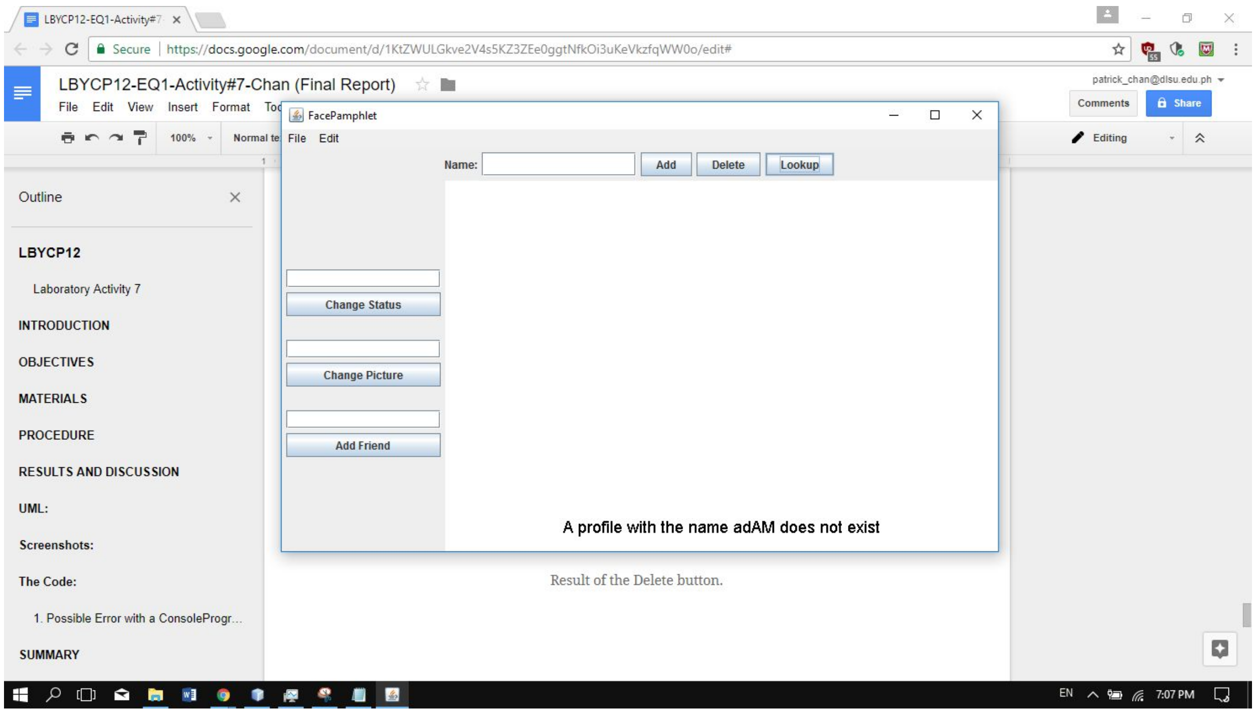
Displaying adAM

Result of the Lookup button.



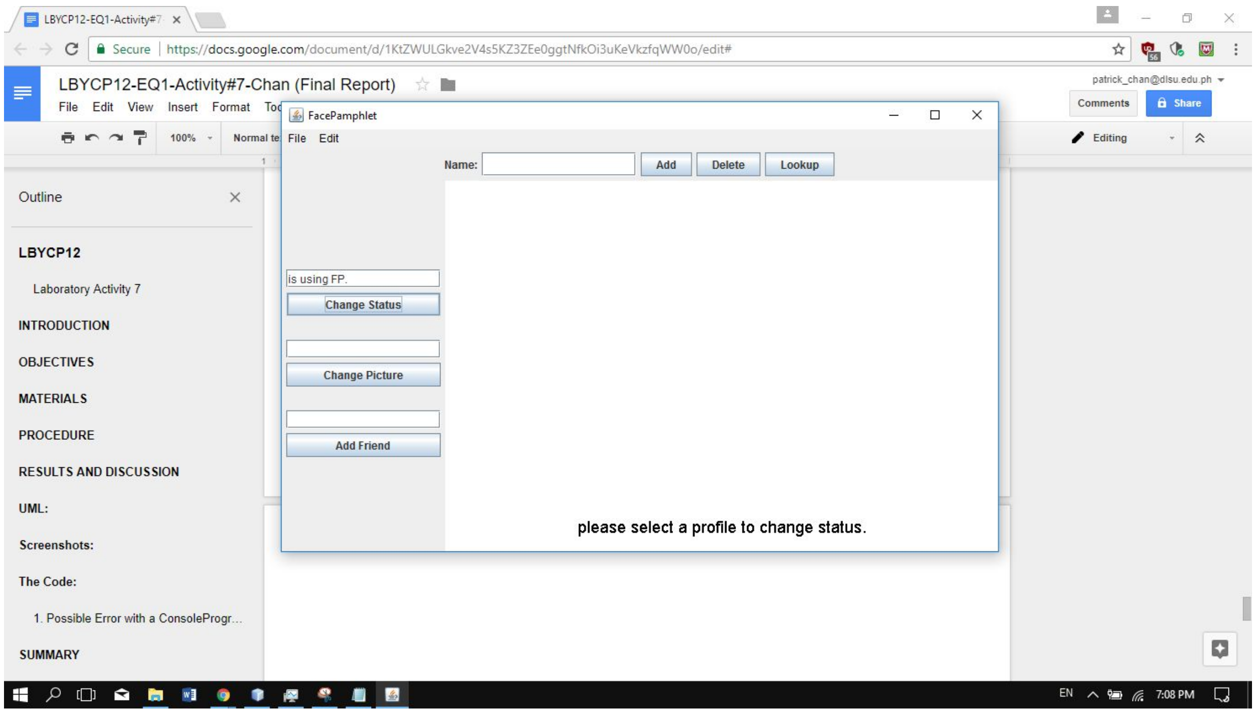
Profile of adAM deleted

Result of the Delete button.



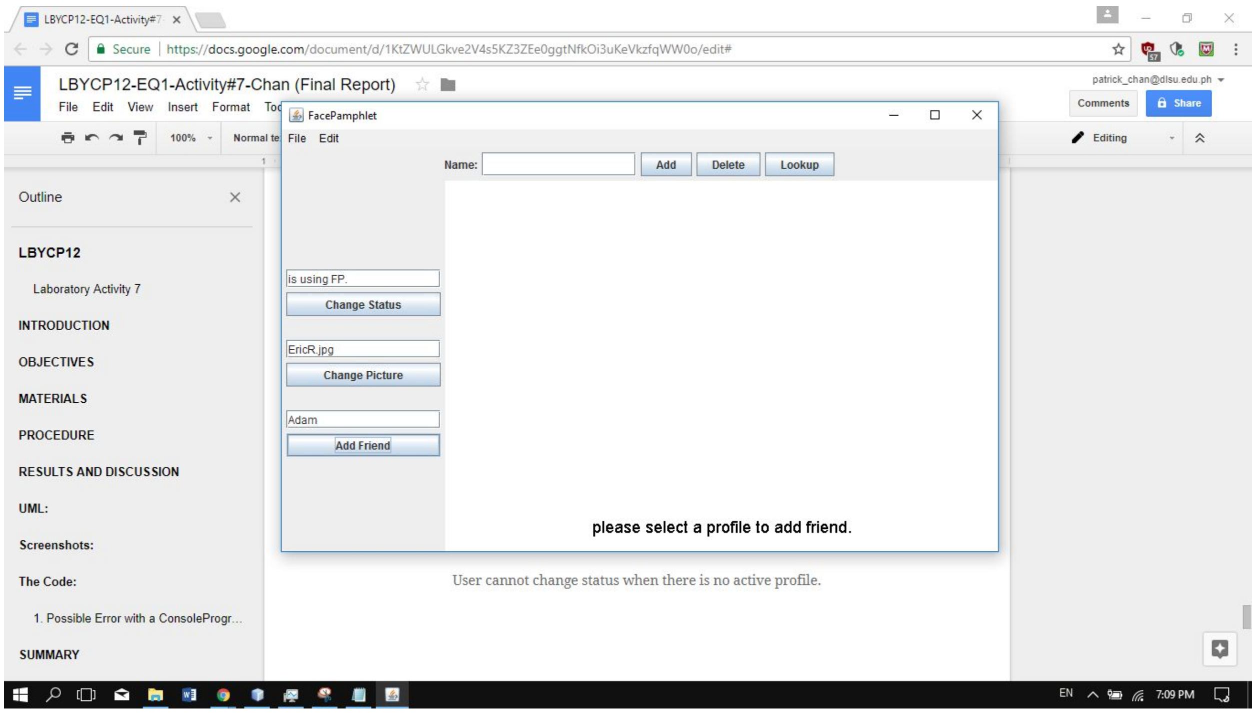
A profile with the name adAM does not exist

As a result, the deleted profile name can no longer be found.



please select a profile to change status.

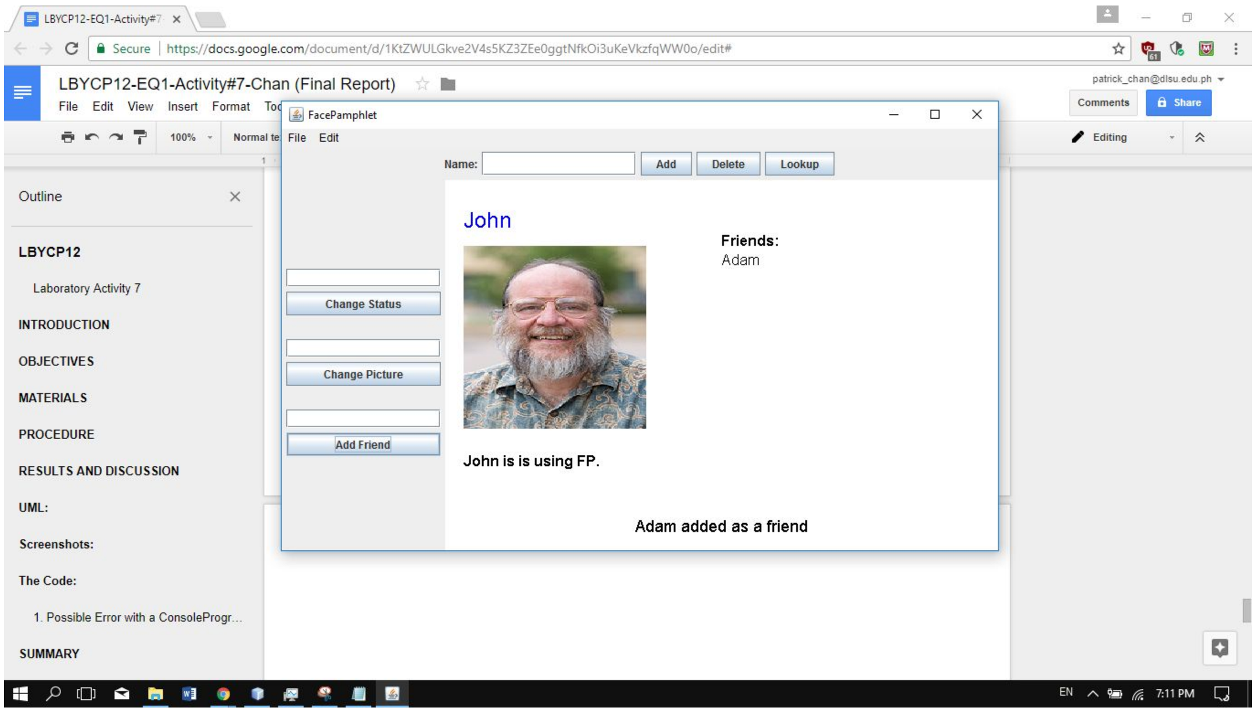
User cannot change status when there is no active profile.



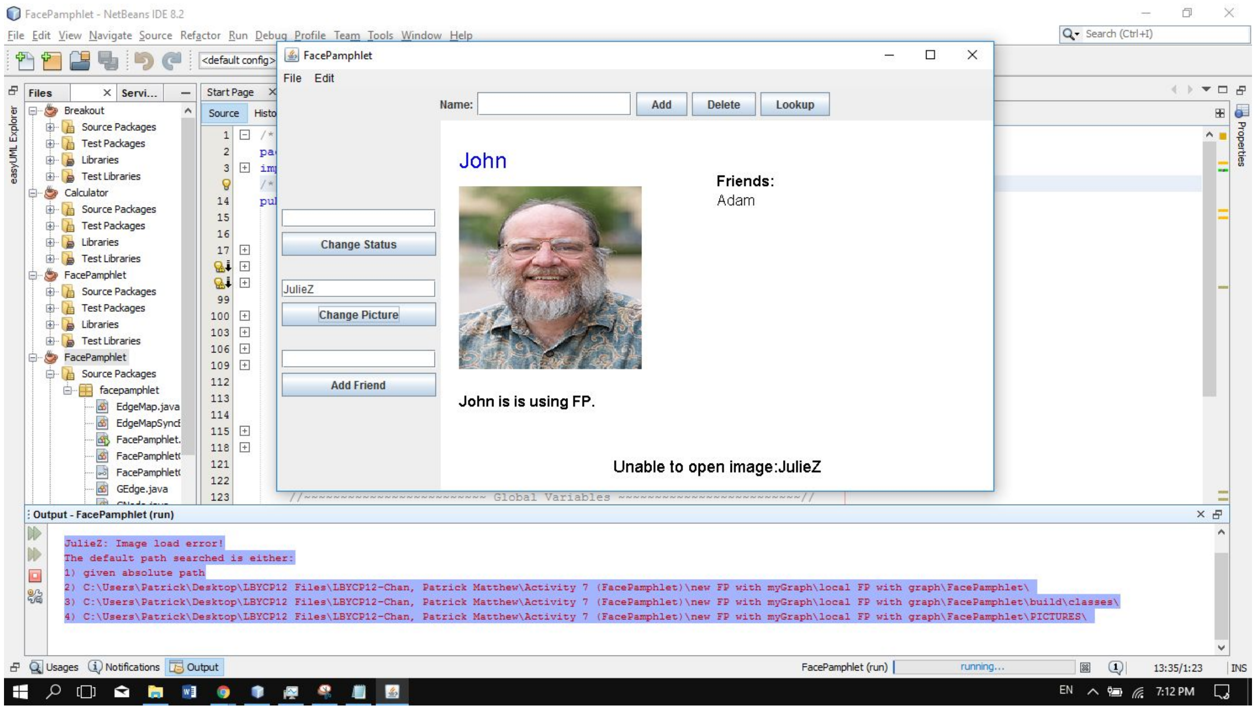
please select a profile to add friend.

User cannot change status when there is no active profile.

The entries for the three fields are not erased when no change is done.

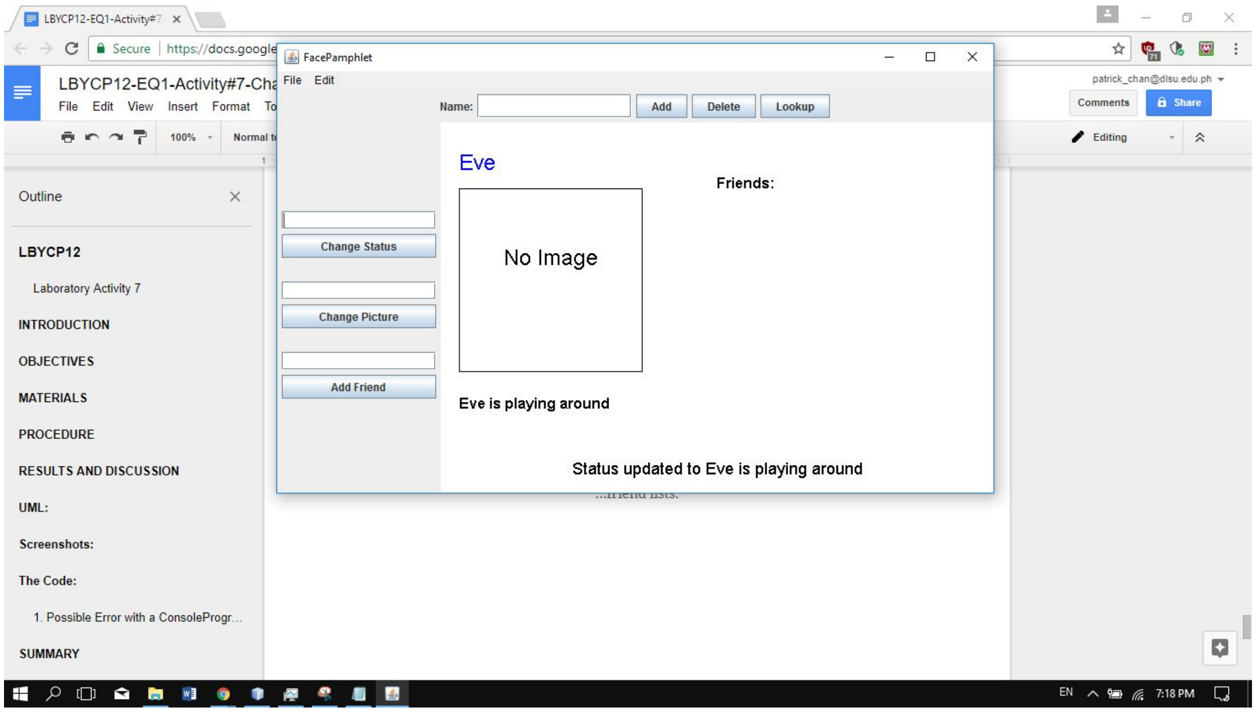


Changing status, photo, and adding friend.



```
JulieZ: Image load error!
The default path searched is either:
1) given absolute path
2) C:\Users\Patrick\Desktop\LBYCP12 Files\LBYCP12-Chan, Patrick Matthew\Activity 7 (FacePamphlet)\new FP with myGraph\local FP with graph\FacePamphlet\
3) C:\Users\Patrick\Desktop\LBYCP12 Files\LBYCP12-Chan, Patrick Matthew\Activity 7 (FacePamphlet)\new FP with myGraph\local FP with graph\FacePamphlet\build\classes\
4) C:\Users\Patrick\Desktop\LBYCP12 Files\LBYCP12-Chan, Patrick Matthew\Activity 7 (FacePamphlet)\new FP with myGraph\local FP with graph\FacePamphlet\PICTURES\
```

When the photo is not found. Note that the program also prints all the paths where it can look for the picture.



Eve

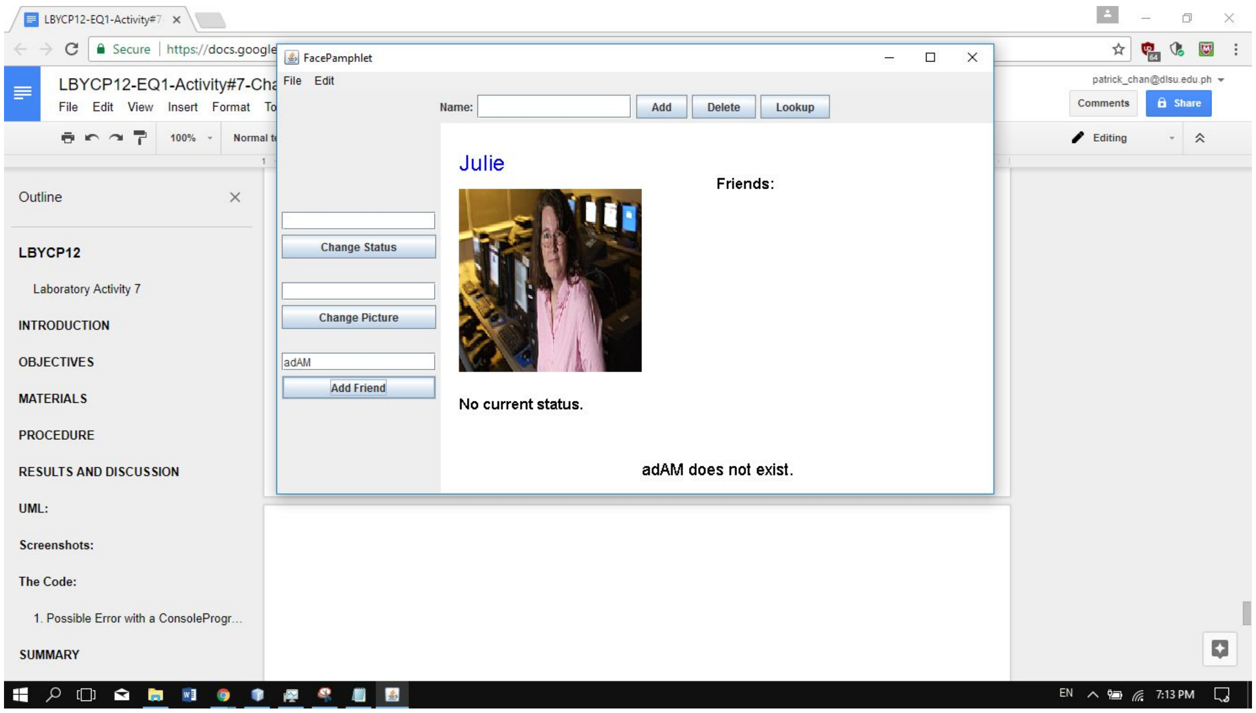
Friends:

No Image

Eve is playing around

Status updated to Eve is playing around

Changing the status



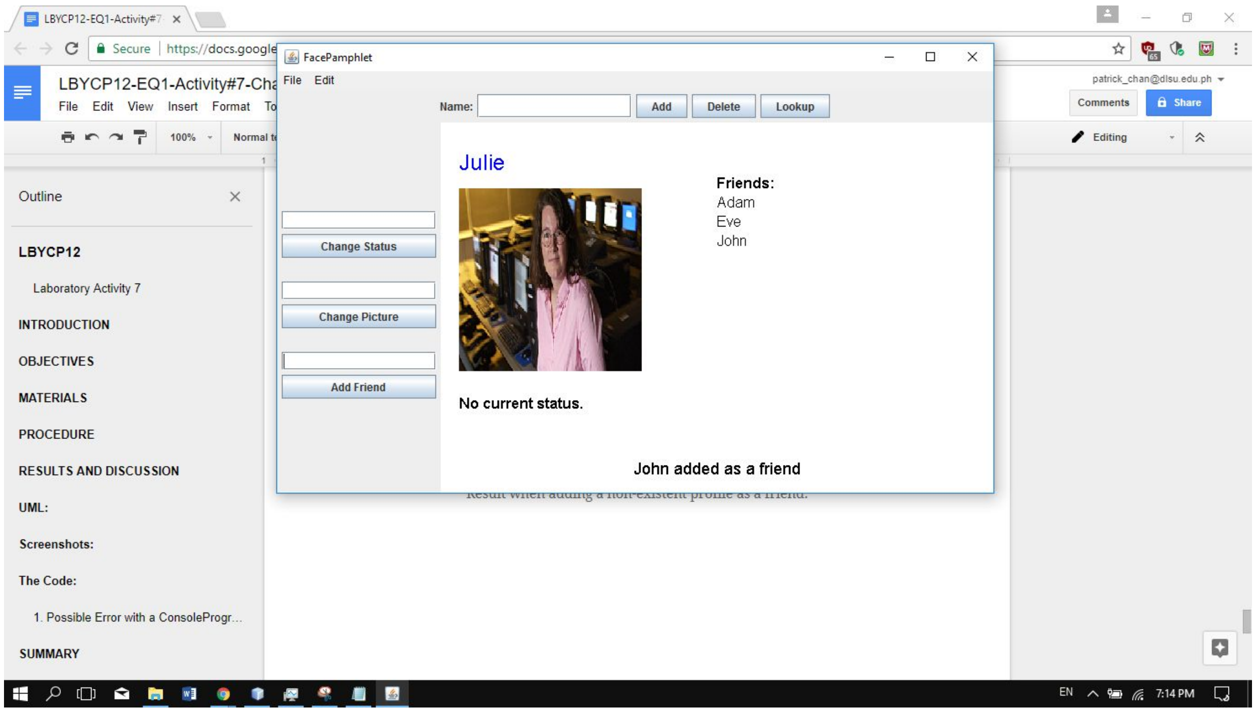
Julie

Friends:

No current status.

adAM does not exist.

Result when adding a non-existent profile as a friend.



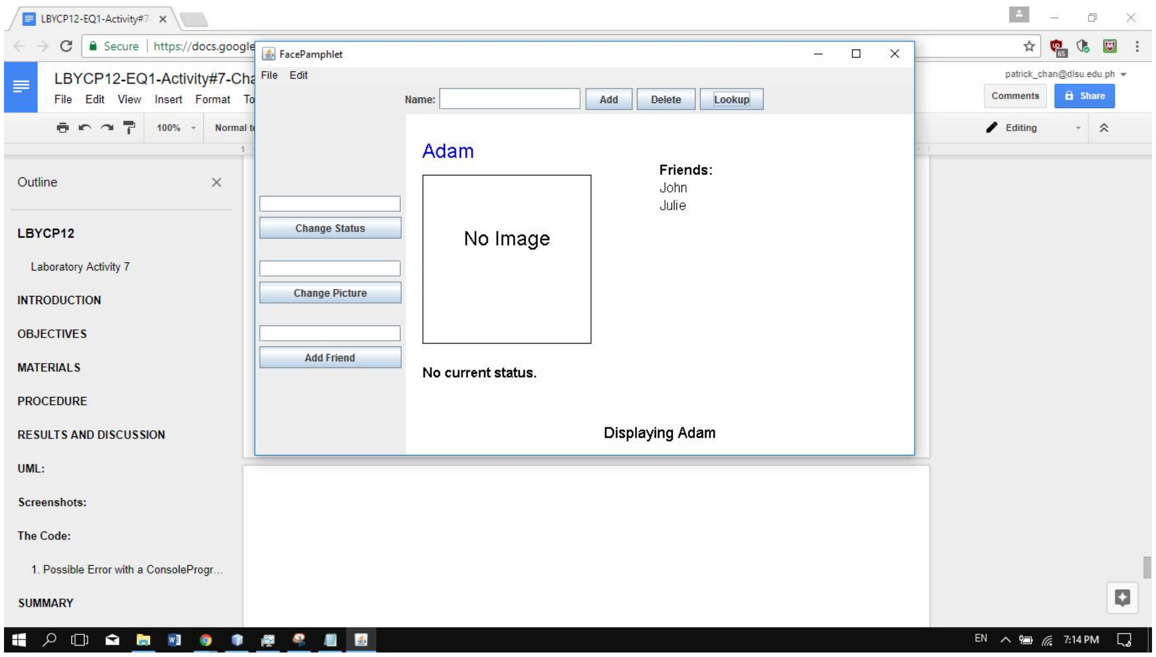
Julie

Friends:
Adam
Eve
John

No current status.

John added as a friend

Adding various friends…



Adam

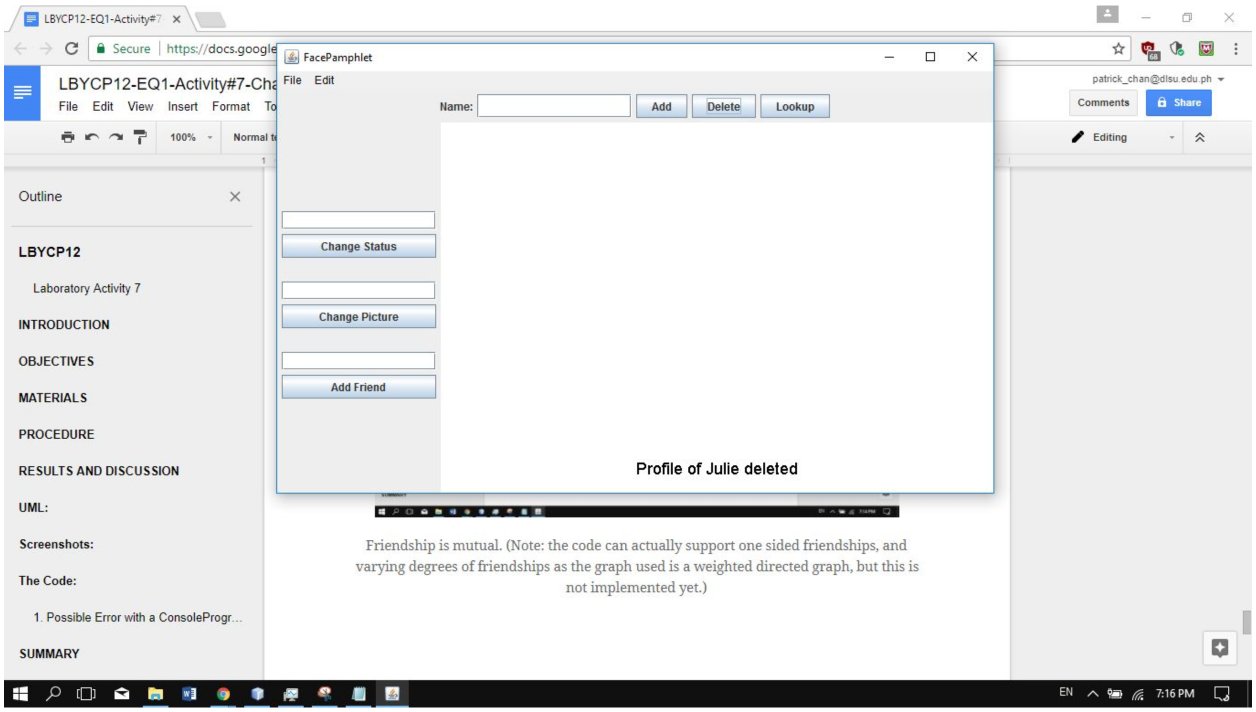
Friends:
John
Julie

No Image

No current status.

Displaying Adam

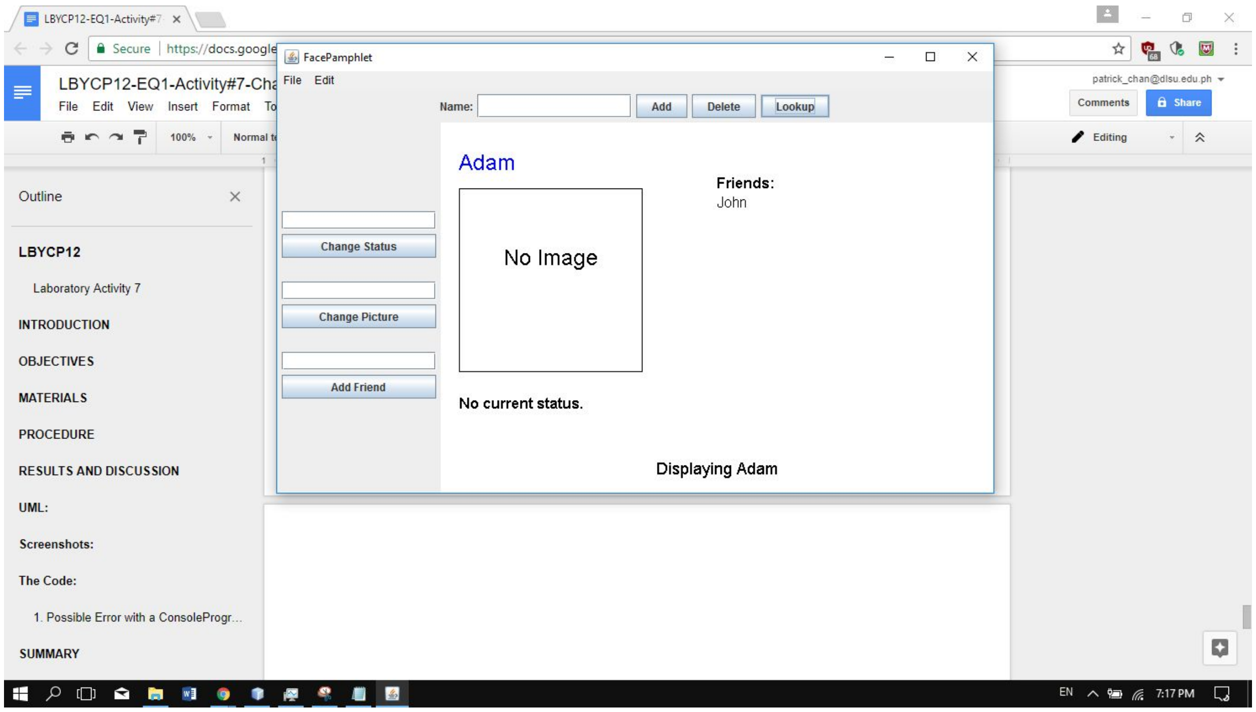
Friendship is mutual. (Note: the code can actually support one sided friendships, and varying degrees of friendships as the graph used is a weighted directed graph, but this is not implemented yet.)



Profile of Julie deleted

Friendship is mutual. (Note: the code can actually support one sided friendships, and varying degrees of friendships as the graph used is a weighted directed graph, but this is not implemented yet.)

Deleting a profile...



Adam

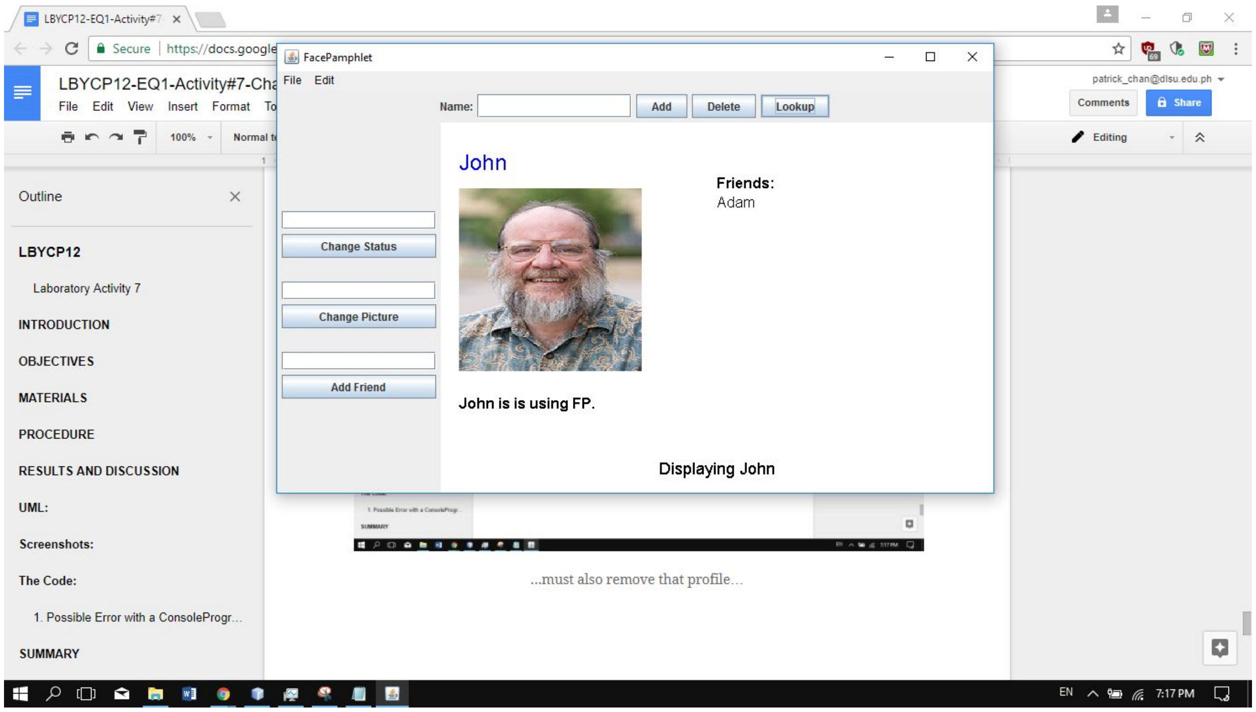
No Image

Friends:
John

No current status.

Displaying Adam

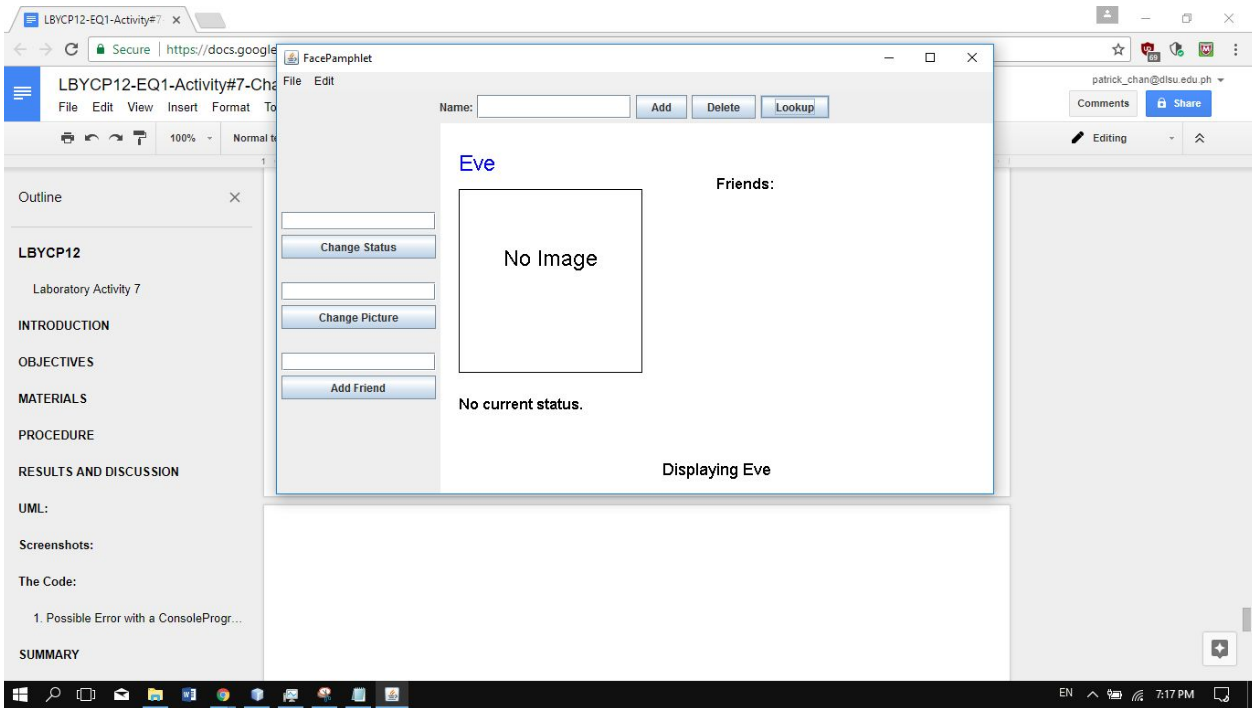
...must also remove that profile...



```
John
Friends:
Adam
John is is using FP.
Displaying John
```

...must also remove that profile...

...from everyone's...



```
Eve
Friends:
No Image
No current status.
Displaying Eve
```

...friend lists.

SUMMARY

In this activity, I have recalled both the way of adding interactors to a Java program, and creating and positioning various objects on a GCanvas. In fact, this activity is the first time that I was able to utilize GImg in my program, and I saw how the process of adding a photo to the GCanvas was not far from the process of adding any other graphical objects to the GCanvas.

Even though setting up the GUI for the FacePamphlet can be quite laborious, implementing and integrating a Graph data structure to the program was actually the more difficult task to do. As I was thinking on how to implement the graph, I had two choices, which is, whether I should use arrays or linked list. Thinking about it, I saw that using arrays would actually make the task much easier, as I could simply represent the adjacency matrix of a graph as a two-dimensional matrix, assign which of the indices pertain to the head of an edge, and assign the value of the edge as the weights. By following these steps, I could then make a weighted directed graph implementation. But then, I had one problem, which was the thing that I disliked the idea of resizing the array every time a new node is added, as this meant that I had to keep creating a new two dimensional array whenever a profile is added, and exceeds the size of the maximum number of nodes. This is why I chose to use a Linked List implementation instead.

Aside from implementing the graph, another difficult task was to integrate the graph into the FacePamphlet program, as the code already has its own class for a Profile, and for a Database, so I had no idea where I could simply squeeze in the Graph into these. In the end, I ended up replacing the database with the graph itself, which consists of different nodes of type profile, so that each profile would contain the information of each entity, while each node would contain the pointers to its connections in the graph.

APPENDIX

The Code

A) Facepamphlet.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
import acm.graphics.*;
import acm.io.*;
import acm.program.*;
import acm.util.*;
import java.io.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class FacePamphlet  extends Program implements FacePamphletConstants {
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Main Classes ~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
   //main classes
   public static void main(String[] args) {
      new FacePamphlet().start(args);
   }
   public void init(){
      System.out.println("Running: FacePamphlet...");
      //ADD--north
      add(topLab,NORTH);
      add(topFie,NORTH);
      add(topBut1,NORTH);
      add(topBut2,NORTH);
      add(topBut3,NORTH);
      //west
      add(wesFie1,WEST);
      add(wesBut1,WEST);
      add(new JLabel(EMPTY_LABEL_TEXT),WEST);
      add(wesFie2,WEST);
      add(wesBut2,WEST);
      add(new JLabel(EMPTY_LABEL_TEXT),WEST);
      add(wesFie3,WEST);
      add(wesBut3,WEST);
      add(new JLabel(EMPTY_LABEL_TEXT),WEST);
      add(wesFie4,WEST);
      add(wesBut4,WEST);
      //disp
      add(disp);

      //listeners--north
      topBut1.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            bAdd(topFie.getText());
            topFie.requestFocusInWindow();
         }
```

```
});
topBut2.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bDel(topFie.getText());
      topFie.requestFocusInWindow();
   }
});
topBut3.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bLoo(topFie.getText());
      topFie.requestFocusInWindow();
   }
});
//west
wesFie1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bSta(wesFie1.getText());
      wesFie1.requestFocusInWindow();
   }
});
wesBut1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bSta(wesFie1.getText());
      wesFie1.requestFocusInWindow();
   }
});
wesFie2.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bPic(wesFie2.getText());
      wesFie2.requestFocusInWindow();
   }
});
wesBut2.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      bPic(wesFie2.getText());
      wesFie2.requestFocusInWindow();
   }
});
wesFie3.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
        bFri(wesFie3.getText());
        wesFie3.requestFocusInWindow();
    }
});
wesBut3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        bFri(wesFie3.getText());
        wesFie3.requestFocusInWindow();
    }
});
wesFie4.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        bUnf(wesFie4.getText());
        wesFie4.requestFocusInWindow();
    }
});
wesBut4.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        bUnf(wesFie4.getText());
        wesFie4.requestFocusInWindow();
    }
});

//persistence
    //note:delete the .ser file whenever you edit your code
String filePath=new File("").getAbsolutePath();
try {
    File a=new File(filePath+"/CurrentGraph.ser");
    if(!a.createNewFile()){
        System.out.println("file:"+a.getPath());
        FileInputStream fis = new FileInputStream(filePath+"/CurrentGraph.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        db = (Graph) ois.readObject();
        ois.close();
    } else {
        db=new Graph();
        System.out.println("new file created, new Graph() created.");
    }
} catch (IOException ex) {
    System.out.println("\nIOException encountered:");
    ex.printStackTrace();
    if(db==null){
        db=new Graph();
        System.out.println("new Graph object created.");
```

```
            }
        } catch (ClassNotFoundException ex) {
            System.out.println("\nClassNotFoundException encountered:");
            ex.printStackTrace();
            if(db==null){
                db=new Graph();
                System.out.println("new Graph object created.");
            }
        }
    }
    public void run(){
        double prevModCount=db.accessCount;
        Thread savT=new Thread();
        Runnable savR=new Runnable() {
            @Override
            public void run() {
                String filePath=new File("").getAbsolutePath();
                try {
                    FileOutputStream fos = new FileOutputStream(filePath+"/CurrentGraph.ser");
                    ObjectOutputStream oos = new ObjectOutputStream(fos);
                    oos.writeObject(db);
                    oos.close();
                    System.out.println("--->Saving complete");
                } catch (IOException ex) {
                    System.out.println("\nIOException encountered:");
                    ex.printStackTrace();
                }
            }
        };

        while(true){
            pause(0);
            if(db.accessCount>prevModCount+1.5){
                if(!savT.isAlive()){
                    savT=new Thread(savR);
                    System.out.print("Saving...");
                    savT.start();
                    db.accessCount=prevModCount;
                }

            }

            if(disp.isJumping && disp.recentView!=curNode){
                curNode=disp.recentView;
                disp.isJumping=false;
            }
        }
```

```
}
@Override
public void exit() {
   String filePath=new File("").getAbsolutePath();
   System.out.print("Close--Saving...");
   try {
      FileOutputStream fos = new FileOutputStream(filePath+"/CurrentGraph.ser");
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(db);
      oos.close();
      System.out.println("--->Saving complete");
   } catch (IOException ex) {
      System.out.println("\nIOException encountered:");
      ex.printStackTrace();
   }
   super.exit(); //To change body of generated methods, choose Tools | Templates.
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~ Debugging & Misc ~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
public static void p(Object a){//debug
   System.out.print(a+"");
}
public static void pl(Object a){//debug
   System.out.println(a+"");
}
public static void pel(Object a){//debug
   System.err.println(a+"");
}
public static void pl(){//debug
   System.out.println();
}
//application size  //alternative method is setSize(x,y)
//public static final int APPLICATION_WIDTH = 400;
//public static final int APPLICATION_HEIGHT = 650;
public void m(String a){
   disp.msg(a);
}
public void m(){
   disp.msg("");
}

public static final boolean ALLOW_SELF_FRIENDSHIP=false;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~ Global Variables ~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
JLabel topLab=new JLabel("Name:");
JTextField topFie=new JTextField(TEXT_FIELD_SIZE);
```

```
JButton topBut1=new JButton("Add");
JButton topBut2=new JButton("Delete");
JButton topBut3=new JButton("Lookup");

JTextField wesFie1=new JTextField(TEXT_FIELD_SIZE);
JButton wesBut1=new JButton("Change Status");
JTextField wesFie2=new JTextField(TEXT_FIELD_SIZE);
JButton wesBut2=new JButton("Change Picture");
JTextField wesFie3=new JTextField(TEXT_FIELD_SIZE);
JButton wesBut3=new JButton("Add Friend");
//tryyy
JTextField wesFie4=new JTextField(TEXT_FIELD_SIZE);
JButton wesBut4=new JButton("Unfriend");

FacePamphletCanvas disp=new FacePamphletCanvas();
//FacePamphletDatabase db=new FacePamphletDatabase();
//Graph db=new Graph();
Graph db;

//FacePamphletProfile curProf; //currently viewed profile
GNode<Profile> curNode=null; //currently viewed profile

//tryyy
   //---none/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Methods ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
//button Actions
public void bAdd(String txt){
   if(!txt.isEmpty()){
      //pl("add "+txt);
      if(db.isNode(txt)){
         //pl("Exists, canceled");
         m("A profile with the name "+txt+" already exists.");
         curNode=db.findNode(txt);
      } else {
         GNode<Profile> bago=new GNode<>(new Profile(txt));
         db.add(bago);
         //pl("Added: "+bago);
         m("New profile created");
         curNode=bago;
      }
      //pl("curProf:"+curProf);
      disp.show(curNode);
      topFie.setText("");
   }
}
public void bDel(String txt){
   if(!txt.isEmpty()){
```

```
            //pl("del "+txt);
            if(db.isNode(txt)){
                //System.out.println("graph= "+db);
                db.remove(db.findNode(txt));
                //pl("deleted success"+txt);
                m("Profile of "+txt+" deleted");
            } else {
                //pl("not Found:"+txt);
                m("A profile with the name "+txt+" does not exist");
            }
            curNode=null;
            //pl("curProf:"+curProf);
            disp.show(curNode);
            topFie.setText("");
        }
    }
    public void bLoo(String txt){
        if(!txt.isEmpty()){
            //pl("lkup "+txt);
            if(db.isNode(txt)){
                //pl("Lookup--Profile found:"+txt);
                m("Displaying "+txt);
                curNode=db.findNode(txt);
            } else {
                //pl("Lookup--not Found:"+txt);
                m("A profile with the name "+txt+" does not exist");
                curNode=null;
            }
            //pl("curProf:"+curProf);
            disp.show(curNode);
            topFie.setText("");
        }
    }
    public void bSta(String txt){
        if(!txt.isEmpty()){
            //pl("stat upd "+txt);
            if(curNode==null){
                //pl("please select a profile to change status.");
                m("please select a profile to change status.");
            } else {
                curNode.getItem().setStatus(curNode.getItem().getName()+
                        " is "+txt);
                disp.show(curNode);
                m("Status updated to "+curNode.getItem().getStatus());
                wesFie1.setText("");
            }
        }
```

```
}
public void bPic(String txt){
    if(!txt.isEmpty()){
        //pl("pic "+txt);
        if(curNode==null){
            //pl("please select a profile to change picture.");
            m("please select a profile to change picture.");
        } else {
            GImage img=null;
            String path="";
            try{
                path=txt;
                img=new GImage(txt);
            } catch (ErrorException e){
                String rel=new File("").getAbsolutePath();
                try{
                    path=rel+"\\PICTURES\\"+txt;
                    img=new GImage(rel+"\\PICTURES\\"+txt);
                } catch (ErrorException e2){
                    pel(txt+": Image load error!");
                    pel("The default path searched is either:\n" +
                        "1) given absolute path\n" +
                        "2) " + rel + "\\ \n" +
                        "3) " + rel + "\\build\\classes\\\n"+
                        "4) " + rel + "\\PICTURES\\ \n");
                    pl();
                    m("Unable to open image:"+txt);
                    //e.printStackTrace(); or e2's
                    return;
                }
            }
            curNode.getItem().setImage(img,path);
            disp.show(curNode);
            m("Picture updated");
            wesFie2.setText("");
        }
    }
}
public void bFri(String txt){
    if(!txt.isEmpty()){
        //pl("friend "+txt);
        if(curNode==null){
            //pl("please select a profile to add friend.");
            m("please select a profile to add friend.");
        } else {
            if(db.isNode(txt)){
                if(curNode.getItem().getName().equals(txt)){
                    if(ALLOW_SELF_FRIENDSHIP){
                        curNode.connectH(curNode);
                        m("Congratulations! You are now friends with: yourself...");
                        disp.show(curNode);
                        wesFie3.setText("");
                    } else {
                        m("You know, being friends with yourself is pointless on a social network.");
                    }
                } else if(curNode.isInNode(db.findNode(txt))){
                    if(curNode.isOutNode(db.findNode(txt))){
                        m(curNode.getItem().getName()+ " already has "+txt
                            +" as a friend");
                    } else {
                        db.findNode(txt).connectH(curNode);
                        disp.show(curNode);
                        m(txt+ ": friend request confirmed.");
                        wesFie3.setText("");
                    }
                } else if(curNode.isOutNode(db.findNode(txt))){
                    m(txt+": request already sent!");
                } else {
                    db.findNode(txt).connectH(curNode);
                    disp.show(curNode);
                    m(txt+": friend request sent!");//m(txt+" added as a friend");
                    wesFie3.setText("");
                }
            } else {
                //pl(txt+"'s profile not found!");
                m(txt+" does not exist.");
            }
        }
    }
}
public void bUnf(String txt){
    if(!txt.isEmpty()){
        //pl("friend "+txt);
        if(curNode==null){
            //pl("please select a profile to add friend.");
            m("please select a profile to remove friend from.");
        } else {
            if(db.isNode(txt)){
                if(true&&curNode.getItem().getName().equals(txt)){
                    if(curNode.isInNode(curNode)){//in or out, doesn't matter
                        curNode.discH(curNode);//H or T, doesn't matter
                        disp.show(curNode);
                        wesFie4.setText("");
                    }
                    m("You hate yourself that much?");
                } else if(curNode.isOutNode(db.findNode(txt))){
                    if(curNode.isInNode(db.findNode(txt))){//mutually conected
                        db.findNode(txt).disc(curNode);
                        disp.show(curNode);
                        m(txt+" removed from friend list");
                    } else {//friend request only
                        db.findNode(txt).discH(curNode);
                        disp.show(curNode);
                        m("Friend request canceled:"+txt);
                    }
                    wesFie4.setText("");
                } else if(curNode.isInNode(db.findNode(txt))){
                    pl("success?"+db.findNode(txt).discT(curNode));
                    disp.show(curNode);
                    m("Friend request deleted:"+txt);
                    wesFie4.setText("");
                } else {
                    m(curNode.getItem().getName()+ " does not have "+txt
                            +" as a friend");
                }
            } else {
                //pl(txt+"'s profile not found!");
                m(txt+" does not exist.");
            }
        }
    }
}

}
```

B) BinaryTree.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
//import java.lang.reflect.Field;//optional,for toString shortcut

import java.util.ArrayList;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class BinaryTree<E> {
    private BinaryTree<E> parent;
    private E root;
    private BinaryTree<E> left;
private GEdge leftEdge;
private BinaryTree<E> right;
private GEdge rightEdge;


public BinaryTree(E rootItem){//constructor
   root=rootItem;
   parent=null;
   left=null;
   leftEdge=null;
   right=null;
   rightEdge=null;
}

//other methods
/**
 * only use this within the tree ADT.
 */
public void setParent(BinaryTree<E> parentTree){
   if((parentTree.left!=null && parentTree.left.equals(this))||
         (parentTree.right!=null && parentTree.right.equals(this))){
      parent=parentTree;
   } else {
      throw new TreeOperationException("setParent(bin):this"
            + " is not the parent's child");
   }
}
public BinaryTree<E> getParent(){
   return parent;
}

public void setLeft(BinaryTree<E> leftTree,GEdge edge){
   left=leftTree;
   leftEdge=edge;
   leftTree.setParent(this);
}
public void setLeft(BinaryTree<E> leftTree,int wt,String desc){
   setLeft(leftTree,new GEdge(wt, desc));
}
public void setLeft(BinaryTree<E> leftTree,int wt){
   setLeft(leftTree,new GEdge(wt));
}
public void setLeft(BinaryTree<E> leftTree,String desc){
   setLeft(leftTree,new GEdge( desc));
}
public void setLeft(BinaryTree<E> leftTree){
   setLeft(leftTree,new GEdge());
```

```
}
public void setLeft(E leftTreeItem,GEdge edge){
   left=new BinaryTree<>(leftTreeItem);
   leftEdge=edge;
   left.setParent(this);
}
public void setLeft(E leftTreeItem,int wt,String desc){
   setLeft(leftTreeItem,new GEdge(wt, desc));
}
public void setLeft(E leftTreeItem,String desc){
   setLeft(leftTreeItem,new GEdge(desc));
}
public void setLeft(E leftTreeItem,int wt){
   setLeft(leftTreeItem,new GEdge(wt));
}
public void setLeft(E leftTreeItem){
   setLeft(leftTreeItem,new GEdge());
}
public void setRight(BinaryTree<E> rightTree,GEdge edge){
   right=rightTree;
   rightEdge=edge;
   rightTree.setParent(this);
}
public void setRight(BinaryTree<E> rightTree,int wt,String desc){
   setRight(rightTree,new GEdge(wt, desc));
}
public void setRight(BinaryTree<E> rightTree,int wt){
   setRight(rightTree,new GEdge(wt));
}
public void setRight(BinaryTree<E> rightTree,String desc){
   setRight(rightTree,new GEdge( desc));
}
public void setRight(BinaryTree<E> rightTree){
   setRight(rightTree,new GEdge());
}
public void setRight(E rightTreeItem,GEdge edge){
   right=new BinaryTree<>(rightTreeItem);
   rightEdge=edge;
   right.setParent(this);
}
public void setRight(E rightTreeItem,int wt,String desc){
   setRight(rightTreeItem,new GEdge(wt, desc));
}
public void setRight(E rightTreeItem,String desc){
   setRight(rightTreeItem,new GEdge(desc));
}
public void setRight(E rightTreeItem,int wt){
    setRight(rightTreeItem,new GEdge(wt));
}
public void setRight(E rightTreeItem){
    setRight(rightTreeItem,new GEdge());
}


public BinaryTree<E> getLeft(){
    return left;
}
public GEdge getLeftEdge() {
    return leftEdge;
}
public BinaryTree<E> getRight() {
    return right;
}
public GEdge getRightEdge() {
    return rightEdge;
}


public void preOrder(ArrayList<BinaryTree<E>> output){
    output.add(this);
    left.preOrder(output);
    right.preOrder(output);
}
public void postOrder(ArrayList<BinaryTree<E>> output){
    left.postOrder(output);
    right.postOrder(output);
    output.add(this);
}
public void inOrder(ArrayList<BinaryTree<E>> output){
    left.inOrder(output);
    output.add(this);
    right.inOrder(output);
}


public BinaryTree<E> inOrderSearch(E query){
    ArrayList<BinaryTree<E>> list=new ArrayList<>();
    inOrder(list);
    for(BinaryTree<E> t:list){
        if(t.root.equals(query)){
            return t;
        }
    }
    return null; //not found
    }



    // <editor-fold defaultstate="collapsed" desc="toString">

    @Override
    public String toString() {
        StringBuilder s=new StringBuilder("[parent:");
        if(parent==null){
            s.append("(null)");
        } else {
            s.append(parent.root);
        }
        s.append("],(BinTree) Root:"+root+"\n ");
        if(left!=null){
            s.append("left="+left.root+"("+leftEdge.wt+","+
                    leftEdge.desc+");");
        }
        if(right!=null){
            s.append("right="+right.root+"("+rightEdge.wt+","+
                    rightEdge.desc+") ");
        }
        if(left!=null){
            s.append("\b\nLEFT:\n"+left);
        }
        if(right!=null){
            s.append("\b\nRIGHT:\n"+right);
        }
        if(left==null && right==null){
            s.append("THIS IS A LEAF\n\n\n");
        }
        return s.toString();
    }
    // </editor-fold>
}
```

C) EdgeMap.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
//import java.lang.reflect.Field;//optional,for toString shortcut

import java.io.Serializable;
import java.util.Iterator;
import java.util.LinkedList;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class EdgeMap implements Serializable {
    protected static final long serialVersionUID = 2L;
    protected LinkedList<GNode> srcs=new LinkedList<>();
    protected LinkedList<Map<GNode,GEdge>> maps=new LinkedList<>();


    public EdgeMap(){//constructor
        srcs=new LinkedList<>();
        maps=new LinkedList<>();
        chkIntegrity("edgemap");
    }

    //other methods
    public void clear(){
        srcs.clear();
        maps.clear();
    }

    /**
     * creates an edge if it does not exist, and modifies it if it does. (in map)
     * @param src head node
     * @param dest tail node
     * @param wt weight
     * @param desc description
     * @return true if edge already exists prior to modification, false
     *  if it creates a new edge (in map)
     */
    public boolean modEdge(GNode src,GNode dest,int wt,String desc){
        int i=srcs.indexOf(src);
        if(i==-1){
            srcs.add(src);
            Map<GNode,GEdge> temp=new Map();
            temp.add(dest,new GEdge(wt, desc));
            maps.add(temp);
            chkIntegrity("modedge");
            return false;
        } else {
            Map<GNode,GEdge> curmap=maps.get(i);
            int j=curmap.indexOfK(dest);
            if(j==-1){
                curmap.add(dest,new GEdge(wt, desc));
                chkIntegrity("modedge");
                return false;
            } else {//edge already exists
                GEdge curedge=curmap.getV(dest);
                curedge.desc=desc;
                curedge.wt=wt;
                chkIntegrity("modedge");
                return true;
            }
        }
    }
    public boolean modEdge(GNode src,GNode dest,int wt){
        String d="";
        try{
            d=desc(src, dest);
        } catch (NonExistentEdgeException e){}
        chkIntegrity("modedge");
        return modEdge(src, dest, wt, d);
    }
    public boolean modEdge(GNode src,GNode dest,String desc){
        int i=1;
        try{
            i=wt(src, dest);
        } catch (NonExistentEdgeException e){}
        chkIntegrity("modedge");
        return modEdge(src, dest, i, desc);
    }
    public boolean modEdge(GNode src,GNode dest){//basically add edge if not exist
        if(isEdge(src, dest)){
            chkIntegrity("modedge");
            return false;
        } else {
            chkIntegrity("modedge");
            return modEdge(src, dest, 1, "");
        }
    }

    public boolean remEdge(GNode src, GNode dest){
        int i=srcs.indexOf(src);
        if(i==-1){
            chkIntegrity("remedge");
            return false;
        } else {
            Map<GNode,GEdge> curmap=maps.get(i);
            int j=curmap.indexOfK(dest);
            if(j==-1){
                chkIntegrity("remedge");
                return false;
            } else {//edge already exists
                curmap.removeByK(dest);
                if(curmap.isEmpty()){
                    maps.remove(curmap);
                    srcs.remove(src);
                }
                chkIntegrity("remedge");
                return true;
            }
        }
    }
    public void isolate(GNode n){
        int i=srcs.indexOf(n);
        if(i!=-1){
            maps.remove(i);
            srcs.remove(i);
        }

        for(int j=0;j<maps.size();j++){
            maps.get(j).removeByK(n);
        }

        chkIntegrity("isolate");
    }

    public boolean isEdge(GNode src,GNode dest){
        int i=srcs.indexOf(src);
        if(i==-1){
            chkIntegrity("isedge");
            return false;
        } else {
            Map<GNode,GEdge> curmap=maps.get(i);
            int j=curmap.indexOfK(dest);
            if(j==-1){
                chkIntegrity("isedge");
                return false;
            } else {//edge already exists
                chkIntegrity("isedge");
                return true;
            }
        }
    }
    public GEdge getGEdge(GNode src,GNode dest){
        int i=srcs.indexOf(src);
        if(i==-1){
            chkIntegrity("getGEdge");
            return null;
        } else {
            Map<GNode,GEdge> curmap=maps.get(i);
            int j=curmap.indexOfK(dest);
            if(j==-1){
                chkIntegrity("getGEdge");
                return null;
            } else {//edge already exists
                chkIntegrity("getGEdge");
                return curmap.getV(dest);
            }
        }
    }

    public int wt(GNode src,GNode dest){
        chkIntegrity("wt");
        if(isEdge(src, dest)){
            return maps.get(srcs.indexOf(src)).getV(dest).wt;
        } else {
            throw new NonExistentEdgeException("wt method");
        }
    }
    public String desc(GNode src,GNode dest){
        chkIntegrity("desc");
        if(isEdge(src, dest)){
            return maps.get(srcs.indexOf(src)).getV(dest).desc;
        } else {
            //return null;//comment out, so that value can be null
            throw new NonExistentEdgeException("desc method");
        }
    }


    /*this is based on map, so no edit here yet
    public int indexOfK(K key){
        chkIntegrity("indexofk");
        return keys.indexOf(key);
    }
    public int indexOfV(V val){
        chkIntegrity("indexofv");
        return values.indexOf(val);
    }*/

    public int numEdges(){
        chkIntegrity("numEdges");
        return srcs.size();
    }
    public boolean isEmpty(){
        chkIntegrity("isempty");
        return numEdges()==0;
    }


    protected final void chkIntegrity(String mthdName){
        if(srcs.size()!=maps.size()){
            throw new EdgeMapSyncException("list sizes unsycnhronized:\n"
                    +"srcs:"+ srcs.size()+"maps:"+maps.size()+
                    "\n last method: "+mthdName+"\n");
        }
    }


    // <editor-fold defaultstate="collapsed" desc="toString shortcut">
    /*//++toString shortcut
    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (Field f: getClass().getDeclaredFields()) {
            try {
            result
            .append(f.getName())
            .append(" : ")
            .append(f.get(this))
            .append(System.getProperty("line.separator"));
            }
            catch (IllegalStateException ise) {
                result
                .append(f.getName())
                .append(" : ")
                .append("[cannot retrieve value]")
                .append(System.getProperty("line.separator"));
            }
            // nope
            catch (IllegalAccessException iae) {}
        }
        return result.toString();
    }*/
    // </editor-fold>
}
```

D) EdgeMapSyncException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
```

```
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class EdgeMapSyncException extends RuntimeException{
    public EdgeMapSyncException(String s){
        super(s);
    }//end constructor
}
```

E) FacePamphletCanvas.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;

import acm.graphics.*;
import acm.util.JTFTools;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.LinkedList;
import javax.swing.JButton;
//import java.util.Iterator;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class FacePamphletCanvas extends GCanvas implements FacePamphletConstants {
    //global vars
    private GLabel lMsg=new GLabel("");
    private GLabel lName=new GLabel("");
    private GLabel lStat=new GLabel("");
    private GLabel llFri=new GLabel("Friends:");
    private GLabel[] lFri;
    private JButton[] lRCon;
    private JButton[] lRDel;
    private double lFriStartY=0;
    //image or NOI=no image
    private GRect rNOI=new GRect(IMAGE_WIDTH, IMAGE_HEIGHT);
    private GLabel lNOI=new GLabel("No Image");
    private GImage img=null;
    GNode<Profile> recentView=null;
    public boolean isJumping=false;//profile jumping
```

```
//utility methods
private double ctrX(GObject g){
   return (getWidth()-g.getWidth())/2;
}
private double ctrXB(GObject g,GObject bound){
   return bound.getX()+(bound.getWidth()-g.getWidth())/2;
}
private double ctrY(GObject g){
   return (getHeight()-g.getHeight())/2;
}
private double ctrYB(GObject g,GObject bound){
   return bound.getY()+(bound.getHeight()-g.getHeight())/2;
}

/**
 * Constructor
 * This method takes care of any initialization needed for
 * the display
 */
public FacePamphletCanvas() {
   lMsg.setFont(MESSAGE_FONT);
   lName.setColor(Color.BLUE);
   lName.setFont(PROFILE_NAME_FONT);
   lStat.setFont(PROFILE_STATUS_FONT);
   llFri.setFont(PROFILE_FRIEND_LABEL_FONT);
   lNOI.setFont(PROFILE_IMAGE_FONT);
   addComponentListener(new ComponentAdapter() {
      @Override
      public void componentResized(ComponentEvent e) {
         show(recentView);
      }
   });
}
/**
 * This method displays a message string near the bottom of the
 * canvas.  Every time this method is called, the previously
 * displayed message (if any) is replaced by the new message text
 * passed in.
 */
public void msg(String msg) {
   remove(lMsg);
   lMsg.setLabel(msg);
   add(lMsg,ctrX(lMsg),getHeight()-BOTTOM_MESSAGE_MARGIN);
}
/**
 * This method displays the given profile on the canvas.  The
```

```
    * canvas is first cleared of all existing items (including
    * messages displayed near the bottom of the screen) and then the
    * given profile is displayed.  The profile display includes the
    * name of the user from the profile, the corresponding image
    * (or an indication that an image does not exist), the status of
    * the user, and a list of the user's friends in the social network.
    */
   public void show(GNode<Profile> n) {
      recentView=n;
      String curMsg=lMsg.getLabel();
      double curY=0;//baseline of prev obj
      removeAll();
      if(n==null){
         msg(curMsg);//msg("No Profile Available to Display.");
      }else {
         Profile p=n.getItem();
         //name
         lName.setLabel(p.getName());
         add(lName,LEFT_MARGIN,TOP_MARGIN+lName.getHeight());
         curY=lName.getY();
         //img
         curY+=IMAGE_MARGIN;
         if(p.getImage()==null){
            add(rNOI,LEFT_MARGIN,curY);
            add(lNOI,ctrXB(lNOI,rNOI),ctrYB(lNOI,rNOI));
            curY+=IMAGE_HEIGHT;
         } else {
            img=p.getImage();
            img.setSize(IMAGE_WIDTH,IMAGE_HEIGHT);
            add(img,LEFT_MARGIN,curY);
            curY+=IMAGE_HEIGHT;
         }
         //stat
         String stat=p.getStatus();
         if(stat==null || stat.equals("")){
            stat="No current status.";
         }
         lStat.setLabel(stat);
         add(lStat,LEFT_MARGIN,curY+STATUS_MARGIN+lStat.getHeight());//if err, try add
twice for getht
         curY=lName.getY();//name baseline
         //friends
         add(llFri,getWidth()/2,curY+IMAGE_MARGIN);
         curY=llFri.getY()+1.25*FRIENDS_MARGIN;//for friends listing
         LinkedList<GNode<Profile>> out=n.getOutNodes();//outgoing
         LinkedList<GNode<Profile>> in=n.getInNodes();//incoming
         LinkedList<GNode<Profile>> f=new LinkedList<>();
```

```
LinkedList<GNode<Profile>> req=new LinkedList<>();//incoming requests
for(int in_index=0;in_index<in.size();in_index++){//for every incoming connection
   if(out.contains(in.get(in_index))){//if connection is mutual
      f.add(in.get(in_index));//only if mutual then friend
   } else {//if user has not accepted request
      req.add(in.get(in_index));//list of requests
   }
}
lFri=new GLabel[f.size()+req.size()];
lRCon=new JButton[req.size()];
lRDel=new JButton[req.size()];
for(int i=0;i<f.size();i++){
   GNode<Profile> nodeNow=f.get(i);
   lFri[i]=new GLabel(f.get(i).getItem().getName());
   lFri[i].setFont(PROFILE_FRIEND_FONT);
   lFri[i].addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
         isJumping=true;
         show(nodeNow);
         msg("Currently showing profile:" + nodeNow.getItem().getName());
      }
   });
   add(lFri[i],getWidth()/2,curY+lFri[i].getHeight());
   curY=lFri[i].getY()+FRIENDS_MARGIN;
}
for(int i=f.size();i<f.size()+req.size();i++){
   int j=i-f.size();
   GNode<Profile> nodeNow=req.get(j);
   String nameNow=req.get(j).getItem().getName();
   lFri[i]=new GLabel(nameNow);
   lFri[i].setFont(PROFILE_FRIEND_FONT);
   lFri[i].addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
         isJumping=true;
         show(nodeNow);
         msg("Currently showing profile:" + nodeNow.getItem().getName());
      }
   });
   add(lFri[i],getWidth()/2-10,curY+lFri[i].getHeight());

   lRCon[j]=new JButton();
   lRCon[j].setToolTipText("Confirm Friend: "+nameNow);
   lRCon[j].setPreferredSize(new Dimension(45,(int)lFri[i].getHeight()-2));
   lRCon[j].setText("✓");
   lRCon[j].addActionListener(new ActionListener() {
```

```
                @Override
                public void actionPerformed(ActionEvent e) {
                    n.connectT(req.get(j));
                    show(n);
                }
            });
            add(lRCon[j],lFri[i].getX()+lFri[i].getWidth()+
            REQ_NAME_TO_BUTTON_SPACING,curY);

            lRDel[j]=new JButton();
            lRDel[j].setSize(45,(int)lFri[i].getHeight()-2);
            lRDel[j].setToolTipText("Delete Friend: "+nameNow);
            lRDel[j].setText("X");
            lRDel[j].addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    n.discH(req.get(j));
                    show(n);
                }
            });
            add(lRDel[j],lRCon[j].getX()+lRCon[j].getWidth()+
            REQ_BUTTON_SPACING,lRCon[j].getY());
            curY=lFri[i].getY()+FRIENDS_MARGIN;
        }
        //msg
        if(curMsg.length()==0){
            msg("Displaying "+p.getName());
        }else{
            msg(curMsg);
        }
    }
  }


}
```

F) FacePamphletConstants.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
/*
 * File: FacePamphletConstants.java
 * --------------------------------
 * This file declares several constants that are shared by the
 * different modules in the FacePamphlet application.  Any class
 * that implements this interface can use these constants.
 */

public interface FacePamphletConstants {

	/** The width of the application window */
	public static final int APPLICATION_WIDTH = 800;

	/** The height of the application window */
	public static final int APPLICATION_HEIGHT = 500;

	/** Number of characters for each of the text input fields */
	public static final int TEXT_FIELD_SIZE = 15;

	/** Text to be used to create an "empty" label to put space
	 *  between interactors on EAST border of application.  Note this
	 *  label is not actually the empty string, but rather a single space */
	public static final String EMPTY_LABEL_TEXT = " ";

	/** Name of font used to display the application message at the
	 *  bottom of the display canvas */
	public static final String MESSAGE_FONT = "Dialog-18";

	/** Name of font used to display the name in a user's profile */
	public static final String PROFILE_NAME_FONT = "Dialog-24";

	/** Name of font used to display the text "No Image" in user
	 *  profiles that do not contain an actual image */
	public static final String PROFILE_IMAGE_FONT = "Dialog-24";

	/** Name of font used to display the status in a user's profile */
	public static final String PROFILE_STATUS_FONT = "Dialog-16-bold";

	/** Name of font used to display the label "Friends" above the
	 *  user's list of friends in a profile */
	public static final String PROFILE_FRIEND_LABEL_FONT = "Dialog-16-bold";

	/** Name of font used to display the names from the user's list
	 *  of friends in a profile */
	public static final String PROFILE_FRIEND_FONT = "Dialog-16";

	/** The width (in pixels) that profile images should be displayed */
	public static final double IMAGE_WIDTH = 200;

	/** The height (in pixels) that profile images should be displayed */
	public static final double IMAGE_HEIGHT = 200;
```

```
        /** The number of pixels in the vertical margin between the bottom
         *  of the canvas display area and the baseline for the message
         *  text that appears near the bottom of the display */
        public static final double BOTTOM_MESSAGE_MARGIN = 20;

        /** The number of pixels in the hortizontal margin between the
         *  left side of the canvas display area and the Name, Image, and
         *  Status components that are display in the profile */
        public static final double LEFT_MARGIN = 20;

        /** The number of pixels in the vertical margin between the top
         *  of the canvas display area and the top (NOT the baseline) of
         *  the Name component that is displayed in the profile */
        public static final double TOP_MARGIN = 20;

        /** The number of pixels in the vertical margin between the
         *  baseline of the Name component and the top of the Image
         *  displayed in the profile */
        public static final double IMAGE_MARGIN = 20;

        /** The number of vertical pixels in the vertical margin between
         *  the bottom of the Image and the top of the Status component
         *  in the profile */
        public static final double STATUS_MARGIN = 20;

      /** Spacing bet friends in canvas (PMC)*/
        public static final double FRIENDS_MARGIN = 0;
      public static final double REQ_NAME_TO_BUTTON_SPACING = 10;
      public static final double REQ_BUTTON_SPACING = 2;
}
```

G) GEdge.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
import java.io.Serializable;
import java.lang.reflect.Field;//optional,for toString shortcut
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class GEdge implements Serializable {
   private static final long serialVersionUID = 1L;
   public int wt=1;//weight
   public String desc="";
   public GEdge(){//constructor
      wt=1;//unweighted default 1
      desc="";
   }
   public GEdge(int weight){//constructor
      wt=weight;
      desc="";
   }
   public GEdge(String description){//constructor
      wt=1;
      desc=description;
   }
   public GEdge(int weight,String description){//constructor
      wt=weight;
      desc=description;
   }

   //other methods




   // <editor-fold defaultstate="collapsed" desc="toString shortcut">
   //++toString shortcut
   @Override
   public String toString() {
      StringBuilder result = new StringBuilder();
      for (Field f: getClass().getDeclaredFields()) {
         try {
         result
         .append(f.getName())
         .append(" : ")
         .append(f.get(this))
         .append(System.getProperty("line.separator"));
         }
         catch (IllegalStateException ise) {
            result
            .append(f.getName())
            .append(" : ")
            .append("[cannot retrieve value]")
            .append(System.getProperty("line.separator"));
         }
         // nope
         catch (IllegalAccessException iae) {}
      }
      return result.toString();
```

```
    }
    // </editor-fold>
}
```

H) GNode.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
import java.io.Serializable;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class GNode<E>  implements Serializable {
    private static final long serialVersionUID = 1L;
    protected E item;
    protected LinkedList<GNode<E>> inNodes=new LinkedList<>();
    protected LinkedList<GNode<E>> outNodes=new LinkedList<>();


    //constructor
    public GNode(E newItem){
        item = newItem;
        inNodes.clear();
        outNodes.clear();
    }

    //other methods
    public void setItem(E newItem){
        item = newItem;
    }
    public E getItem(){
        return item;
    }

    //also handles the other node
    /**
     * connects a tail to the node (adds to outDegree)
     * (THIS)------->(n)
     */
    public boolean connectT(GNode<E> dest){
        if(outNodes.contains(dest)){
            System.out.println("already connected");
            return false;
        } else {
            LinkedList<GNode<E>> temp=dest.getInNodes();
            if(temp.contains(this)){
                System.out.println("Warning: other node is already half connected to this prior.");
                return outNodes.add(dest);
            } else {
                return outNodes.add(dest) && dest.addInNode(this);
            }
        }
    }
    /**
     * connects a head (ARROWHEAD) to the node (adds to inDegree)
     * (n)-------->(THIS)
     */
    public boolean connectH(GNode<E> src){
        if(inNodes.contains(src)){
            System.out.println("already connected");
            return false;
        } else {
            LinkedList<GNode<E>> temp=src.getOutNodes();
            if(temp.contains(this)){
                System.out.println("Warning: other node is already half connected to this prior.");
                return inNodes.add(src);
            } else {
                return inNodes.add(src) && src.addOutNode(this);
            }
        }
    }

    /**
     * disconnects a tail from the node (subtract from outDegree)
     * this node is the tail of the edge
     * (THIS)---/ /--->(n)
     */
    public boolean discT(GNode<E> n){
        LinkedList<GNode<E>> temp=n.getInNodes();
        if(!temp.contains(this)){
            System.out.println("Warning: discT--other node is already half "
                    + "disconnected to this prior.\n Ignore if using disc()");
            return outNodes.remove(n);
        } else {
            return outNodes.remove(n) && n.remInNode(this);
        }
    }
    /**
```

```
 * disconnects a head from the node (subtract from inDegree)
 * this node is the head of the edge
 * (n)---/ /--->(THIS)
 */
public boolean discH(GNode<E> n){
    LinkedList<GNode<E>> temp=n.getOutNodes();
    if(!temp.contains(this)){
        System.out.println("Warning: discH--other node is already half "
                + "disconnected to this prior.\n Ignore if using disc()");
        return inNodes.remove(n);
    } else {
        return inNodes.remove(n) && n.remOutNode(this);
    }
}
public boolean disc(GNode<E> n){
    boolean a=discH(n);
    boolean b=discT(n);
    return a||b;
}
public void isolate(){
    while(!inNodes.isEmpty()){
        discH(inNodes.peek());
        //System.out.println("to remove (IN):"+inNodes.size());//debug
    }
    while(!outNodes.isEmpty()){
        discT(outNodes.peek());
        //System.out.println("to remove (OUT):"+outNodes.size());//debug
    }
}

/**
 * @return clone list of inNodes
 */
public LinkedList<GNode<E>> getInNodes(){
    return (LinkedList<GNode<E>>) inNodes.clone();
}
/**
 * @return clone list of outNodes
 */
public LinkedList<GNode<E>> getOutNodes(){
    return (LinkedList<GNode<E>>) outNodes.clone();
}
/**
 * one sided add
 */
public boolean addInNode(GNode<E> n){
    return inNodes.add(n);
}
/**
 * one sided add
 */
public boolean addOutNode(GNode<E> n){
   return outNodes.add(n);
}
/**
 * one sided remove
 */
public boolean remInNode(GNode<E> n){
   return inNodes.remove(n);
}
/**
 * one sided remove
 */
public boolean remOutNode(GNode<E> n){
   return outNodes.remove(n);
}
public int indexOfIn(GNode<E> n){
   return inNodes.indexOf(n);
}
public int indexOfOut(GNode<E> n){
   return outNodes.indexOf(n);
}
public boolean isInNode(GNode<E> n){
   return inNodes.contains(n);
}
public boolean isOutNode(GNode<E> n){
   return outNodes.contains(n);
}
public GNode<E> getVtxIn(int index){
   return inNodes.get(index);
}
public GNode<E> getVtxOut(int index){
   return outNodes.get(index);
}

public int indeg(){
   return inNodes.size();
}
public int outdeg(){
   return outNodes.size();
}
```

```
    // <editor-fold defaultstate="collapsed" desc="toString">
    //++toString shortcut
    @Override
    public String toString() {
        StringBuilder s = new StringBuilder("item: "+item+"\n inNodes: [ ");
        Iterator<GNode<E>> it = inNodes.iterator();
        while(it.hasNext()){
            s.append(it.next().getItem());
            s.append(",");
        }
        s.append("\b]\n outNodes: [ ");
        Iterator<GNode<E>> it2 = outNodes.iterator();
        while(it2.hasNext()){
            s.append(it2.next().getItem());
            s.append(",");
        }
        s.append("\b]\n");
        return s.toString();
    }
    // </editor-fold>
}
```

l) Graph.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
//import java.lang.reflect.Field;//optional,for toString shortcut

import java.io.Serializable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class Graph extends GraphGeneral<Profile> implements Serializable{
    private static final long serialVersionUID = 1L;
    //EdgeMap m;


    public Graph(){//constructor
        //edgemap
        srcs=new LinkedList<>();
        maps=new LinkedList<>();
        chkIntegrity("edgemap");

        nodes=new LinkedList<>();
        //m=new EdgeMap();
    }

    //other methods
    public GNode<Profile> findNode(Profile query){
        accessCount+=0.1;
        Iterator<GNode<Profile>> i=nodes.iterator();
        while(i.hasNext()){
            GNode<Profile> now=i.next();
            if(now.getItem().equals(query)){
                return now;
            }
        }
        return null;
    }
    public GNode<Profile> findNode(String name){
        accessCount+=0.1;
        Iterator<GNode<Profile>> i=nodes.iterator();
        while(i.hasNext()){
            GNode<Profile> now=i.next();
            if(now.getItem().getName().equals(name)){
                return now;
            }
        }
        return null;
    }
    public boolean isNode(Profile query){
        return findNode(query)!=null;
    }
    public boolean isNode(String name){
        return findNode(name)!=null;
    }

    public int size(){
        return nodes.size();
    }
    public GNode<Profile> getVtx(int index){
        accessCount+=0.1;
        return nodes.get(index);
    }
    public GEdge getGEdge(GNode src,GNode dest){
        accessCount+=0.1;
    return this.getGEdge(src, dest);//edgemap
}
public int indexOf(GNode<Profile> n){
    accessCount+=0.1;
    return nodes.indexOf(n);
}
public void clear(){
    accessCount++;
    nodes.clear();
    this.clear();//edgemap
}

public boolean add(Profile nodeItem){
    accessCount++;
    if(findNode(nodeItem)!=null){//no duplicates allowed
        return false;
    }
    return nodes.add(new GNode<>(nodeItem));
}
public boolean add(GNode<Profile> n){
    accessCount++;
    if(nodes.contains(n)){//no duplicates allowed
        return false;
    }
    return nodes.add(n);
}
public boolean add(int index,GNode<Profile> n){
    accessCount++;
    if(nodes.contains(n)){//no duplicates allowed
        return false;
    }
    nodes.add(index, n);
    return true;
}
public boolean remove (Profile nodeItem){
    accessCount++;
    GNode<Profile> n=findNode(nodeItem);
    if(n==null){
        System.out.println("node not found");
        return false;
    } else {
        return remove(n);
    }
}
public boolean remove(GNode<Profile> n){
    accessCount++;
    n.isolate();
    this.isolate(n);//edgemap
    return nodes.remove(n);
}
public boolean remove(int i){
    accessCount++;
    return remove(nodes.get(i));
}

public boolean connect(GNode<Profile> origin,GNode<Profile> endpt){
    accessCount++;
    if(origin.connectT(endpt)){//automatic
        this.modEdge(origin, endpt);//edgemap
        return true;
    }
    return false;
}
public boolean connect(int originIndex,int endptIndex){
    accessCount++;
    return connect(nodes.get(originIndex),nodes.get(endptIndex));
}
public boolean discOneEdge(GNode<Profile> origin,GNode<Profile> endpt){
    accessCount++;
    if(origin.discH(endpt)){//automatic
        this.remEdge(origin, endpt);//edgemap
        return true;
    }
    return false;
}
public boolean discOneEdge(int originIndex,int endptIndex){
    accessCount++;
    return discOneEdge(nodes.get(originIndex), nodes.get(endptIndex));
}
public boolean disc(GNode<Profile> origin,GNode<Profile> endpt){
    accessCount++;
    boolean a=discOneEdge(origin, endpt);
    boolean b=discOneEdge(endpt,origin);
    return a||b;
}
public boolean disc(int originIndex,int endptIndex){
    accessCount++;
    return disc(nodes.get(originIndex), nodes.get(endptIndex));
}



// <editor-fold defaultstate="collapsed" desc="toString">
//++toString
```

```
    @Override
    public String toString() {
        Iterator<GNode<Profile>> it = nodes.iterator();
        if (! it.hasNext())
            return "[]";

        StringBuilder sb = new StringBuilder();
        sb.append('[');
        for (;;) {
            GNode<Profile> e = it.next();
            sb.append(e);
            if (! it.hasNext())
                return sb.append(']').toString();
            sb.append(";;").append(System.getProperty("line.separator"));
        }
    }
    // </editor-fold>
}
```

J) GraphGeneral.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
//import java.lang.reflect.Field;//optional,for toString shortcut

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class GraphGeneral<NodeItemType> extends EdgeMap implements Serializable{
    private static final long serialVersionUID = 1L;
    protected LinkedList<GNode<NodeItemType>> nodes;
    //EdgeMap m;
    protected double accessCount=0;//+0.1 for findNode, +1 for node adding & etc.


    public GraphGeneral(){//constructor
        //edgemap
        srcs=new LinkedList<>();
        maps=new LinkedList<>();
        chkIntegrity("edgemap");

        nodes=new LinkedList<>();
        //m=new EdgeMap();
    }

    //other methods
    public GNode<NodeItemType> findNode(NodeItemType query){
        accessCount+=0.1;
        Iterator<GNode<NodeItemType>> i=nodes.iterator();
        while(i.hasNext()){
            GNode<NodeItemType> now=i.next();
            if(now.getItem().equals(query)){
                return now;
            }
        }
        return null;
    }
    public boolean isNode(NodeItemType query){
        return findNode(query)!=null;
    }

    public int size(){
        return nodes.size();
    }
    public List<GNode<NodeItemType>> getNodes(){
        accessCount+=0.1;
        return (List<GNode<NodeItemType>>) nodes.clone();
    }
    public GNode<NodeItemType> getVtx(int index){
        accessCount+=0.1;
        return nodes.get(index);
    }
    public GEdge getGEdge(GNode src,GNode dest){
        accessCount+=0.1;
        return this.getGEdge(src, dest);//edgemap
    }
    public int indexOf(GNode<NodeItemType> n){
        accessCount+=0.1;
        return nodes.indexOf(n);
    }
    public void clear(){
        accessCount++;
        nodes.clear();
        this.clear();//edgemap
    }

    public boolean add(NodeItemType nodeItem){
        accessCount++;
        if(findNode(nodeItem)!=null){//no duplicates allowed
            return false;
        }
        return nodes.add(new GNode<>(nodeItem));
    }
    public boolean add(GNode<NodeItemType> n){
        accessCount++;
        if(nodes.contains(n)){//no duplicates allowed
            return false;
        }
        return nodes.add(n);
    }
    public boolean add(int index,GNode<NodeItemType> n){
        accessCount++;
        if(nodes.contains(n)){//no duplicates allowed
            return false;
        }
        nodes.add(index, n);
        return true;
    }
    public boolean remove (NodeItemType nodeItem){
        accessCount++;
        GNode<NodeItemType> n=findNode(nodeItem);
        if(n==null){
            System.out.println("node not found");
            return false;
        } else {
            return remove(n);
        }
    }
    public boolean remove(GNode<NodeItemType> n){
        accessCount++;
        n.isolate();
        this.isolate(n);//edgemap
        return nodes.remove(n);
    }
    public boolean remove(int i){
        accessCount++;
        return remove(nodes.get(i));
    }

    public boolean connect(GNode<NodeItemType> origin,GNode<NodeItemType> endpt){
        accessCount++;
        if(origin.connectT(endpt)){//automatic
            this.modEdge(origin, endpt);//edgemap
            return true;
        }
        return false;
    }
    /**
     * IMPT: only use this if there are NO DUPLICATES in your data set
     * @param origin
     * @param endpt
     * @return
     */
    public boolean connect(NodeItemType origin,NodeItemType endpt){
        GNode<NodeItemType> or=findNode(origin);
        GNode<NodeItemType> en=findNode(endpt);
        if(or==null || en==null){
            return false;
        }
        return connect(or,en);
    }
    public boolean connect(int originIndex,int endptIndex){
        accessCount++;
        return connect(nodes.get(originIndex),nodes.get(endptIndex));
    }
    public boolean discOneEdge(GNode<NodeItemType> origin,GNode<NodeItemType>
endpt){
        accessCount++;
        if(origin.discH(endpt)){//automatic
            this.remEdge(origin, endpt);//edgemap
            return true;
        }
        return false;
    }
    public boolean discOneEdge(int originIndex,int endptIndex){
        accessCount++;
        return discOneEdge(nodes.get(originIndex), nodes.get(endptIndex));
    }
    public boolean disc(GNode<NodeItemType> origin,GNode<NodeItemType> endpt){
        accessCount++;
        boolean a=discOneEdge(origin, endpt);
        boolean b=discOneEdge(endpt,origin);
        return a||b;
    }
    public boolean disc(int originIndex,int endptIndex){
        accessCount++;
        return disc(nodes.get(originIndex), nodes.get(endptIndex));
    }

    ////////////////////////////////////////////////////////////////////
    //||--||--||oooooooooooooooooo A L G O oooooooooooooooooo||--||--||//
    ////////////////////////////////////////////////////////////////////
```

```
public Tree<GNode<NodeItemType>> Dijkstra(GNode<NodeItemType> source){
   Map<GNode<NodeItemType>,GNode<NodeItemType>> parentMap=new Map<>();
   Map<GNode<NodeItemType>,Integer> distMap=new Map<>();
   ArrayList<GNode<NodeItemType>> toVisit=new ArrayList<>(nodes.size());
   Tree<GNode<NodeItemType>> result=new Tree<>(source);
   ArrayList<Tree<GNode<NodeItemType>>> forest=new ArrayList<>();
   //init
   forest.add(result);
   for(GNode<NodeItemType> n:nodes){
      if(n.equals(source)){//src
         parentMap.add(n,null);
         distMap.add(n,0);
         toVisit.add(n);
      } else {//others
         parentMap.add(n,null);
         distMap.add(n, Integer.MAX_VALUE);
         toVisit.add(n);
         //all orphans are added to the forest until they are adopted
         forest.add(new Tree<>(n));
      }
   }
   //body
   while(!toVisit.isEmpty()){
      GNode<NodeItemType> minNode=findMinDistDijk(toVisit, distMap);
      toVisit.remove(minNode);
      for(GNode<NodeItemType> u:minNode.getOutNodes()){
         if(toVisit.contains(u)){
            if(distMap.getV(u)>distMap.getV(minNode)+wt(minNode, u)){
               //(source will be first minNode, never the u)
               //##tree
               //we look for former parent's tree
               Tree<GNode<NodeItemType>> parentTree=
                     lookForTreeInForestOfForests(parentMap.getV(u),forest);
               //get u's tree
               Tree<GNode<NodeItemType>> uTree=
                        lookForTreeInForestOfForests(u,forest);
               //get minNode's tree
               Tree<GNode<NodeItemType>> minNodeTree=
                        lookForTreeInForestOfForests(minNode,forest);
               if(parentTree==null){//orphan-->about to be adopted
                  forest.remove(uTree);
               } else {//transfer of parenthood
                  //we disconnect old parent
                  parentTree.remChildByItem(u);
               }
               //connect new parent
               minNodeTree.addChild(uTree,wt(minNode, u));
```

```
                    //##map
                    distMap.setV(u,distMap.getV(minNode)+wt(minNode, u));
                    parentMap.setV(u, minNode);
                }
            }
        }
    }
    return result;
}

private GNode<NodeItemType> findMinDistDijk(
        ArrayList<GNode<NodeItemType>> toVisit,
        Map<GNode<NodeItemType>,Integer> distMap){
    Integer min=distMap.getV(toVisit.get(0));
    GNode<NodeItemType> minNode=toVisit.get(0);

    for(GNode<NodeItemType> n:toVisit){
        if(min>distMap.getV(n)){
            min=distMap.getV(n);
            minNode=n;
        }
    }
    return minNode;
}
private Tree<GNode<NodeItemType>> lookForTreeInForestOfForests(
        GNode<NodeItemType> query,
        ArrayList<Tree<GNode<NodeItemType>>> forest){
    //we look for query's tree
    //Tree<GNode<NodeItemType>> queryTree=null;
    for(Tree<GNode<NodeItemType>> t:forest){
        Tree<GNode<NodeItemType>> tempo=t.preOrderSearch(query);
        if(tempo!=null){
            return tempo;
            //queryTree=tempo;
            //return queryTree;
        }
    }
    return null;//not found
}



// <editor-fold defaultstate="collapsed" desc="toString">
//++toString
@Override
public String toString() {
        Iterator<GNode<NodeItemType>> it = nodes.iterator();
        if (! it.hasNext())
            return "[]";

        StringBuilder sb = new StringBuilder();
        sb.append('[');
        for (;;) {
            GNode<NodeItemType> e = it.next();
            sb.append(e);
            if (! it.hasNext())
                return sb.append(']').toString();
            sb.append(";;").append(System.getProperty("line.separator"));
        }
    }
    // </editor-fold>
}
```

K) Map.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
import java.io.Serializable;
import java.lang.reflect.Field;//optional,for toString shortcut

import java.util.LinkedList;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
//Fp, this assumes each key appears only once
public class Map<K,V> implements Serializable{//one dimensional mapper
    private static final long serialVersionUID = 1L;
    private LinkedList<K> keys=new LinkedList<>();
    private LinkedList<V> values=new LinkedList<>();


    public Map(){//constructor
        keys=new LinkedList<>();
        values=new LinkedList<>();
        chkIntegrity("map");
    }

    //other methods
    public boolean add(K key, V value){
        if(keys.contains(key)){
            return false;
        }
        boolean a=keys.add(key);
        boolean b=values.add(value);
        chkIntegrity("add");
        return a&&b;
    }
    public boolean removeByK(K key){
        int i=keys.indexOf(key);
        if(i==-1){
            chkIntegrity("removebyk");
            return false;
        } else {
            keys.remove(i);
            values.remove(i);
            chkIntegrity("removebyk");
            return true;
        }
    }
    public boolean removeByV(V value){
        int i=values.indexOf(value);
        if(i==-1){
            chkIntegrity("removebyv");
            return false;
        } else {
            keys.remove(i);
            values.remove(i);
            chkIntegrity("removebyv");
            return true;
        }
    }
    public int indexOfK(K key){
        chkIntegrity("indexofk");
        return keys.indexOf(key);
    }
    public int indexOfV(V val){
        chkIntegrity("indexofv");
        return values.indexOf(val);
    }
    public void remove(int i){
        keys.remove(i);
        values.remove(i);
        chkIntegrity("remove");
    }
    public int size(){
        chkIntegrity("size");
        return keys.size();
    }
    public boolean isEmpty(){
        chkIntegrity("isempty");
        return size()==0;
    }

    public V getV(K key){
        int i=keys.indexOf(key);
        chkIntegrity("getv");
        if(i==-1){
            return null;
        }
        return values.get(i);
    }
    public K getK(V val){
        int i=values.indexOf(val);
        chkIntegrity("getk");
        if(i==-1){
            return null;
        }
        return keys.get(i);
    }

    public boolean setV(K key,V newValue){
        int i=keys.indexOf(key);
        chkIntegrity("getv");
        if(i==-1){
            return false;
        }
        values.remove(i);
        values.add(i,newValue);
        return true;
    }
    public boolean setK(V val,K newKey){
        int i=values.indexOf(val);
        chkIntegrity("getk");
        if(i==-1){
            return false;
        }
        keys.remove(i);
        keys.add(i,newKey);
        return true;
    }


    private void chkIntegrity(String mthdName){
        if(keys.size()!=values.size()){
            throw new MapUnsyncException("list sizes unsycnhronized:\n"
                    +"keys:"+ keys.size()+"values:"+values.size()+
                "\n last method: "+mthdName+"\n"+this);
        }
    }

    // <editor-fold defaultstate="collapsed" desc="toString shortcut">
    //++toString shortcut
    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (Field f: getClass().getDeclaredFields()) {
            try {
            result
            .append(f.getName())
            .append(" : ")
            .append(f.get(this))
            .append(System.getProperty("line.separator"));
            }
            catch (IllegalStateException ise) {
                result
                .append(f.getName())
                .append(" : ")
                .append("[cannot retrieve value]")
                .append(System.getProperty("line.separator"));
            }
            // nope
            catch (IllegalAccessException iae) {}
        }
        return result.toString();
    }
    // </editor-fold>
}
```

L) MapUnsyncException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class MapUnsyncException extends RuntimeException{
    public MapUnsyncException(String s){
        super(s);
    }//end constructor
}
```

M) NonExistentEdgeException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class NonExistentEdgeException extends RuntimeException{
    public NonExistentEdgeException(String s){
        super(s);
    }//end constructor
}
```

N) Profile.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
import acm.graphics.GImage;
import acm.util.ErrorException;
import java.awt.Image;
import java.io.Serializable;
import java.lang.reflect.Field;//optional,for toString shortcut
import java.util.Arrays;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class Profile  implements Serializable {
    private static final long serialVersionUID = 1L;
    public String name="";
    public transient GImage img=null;//this cannot be serialized
    public int[][] imgArr;//only this is serialzed instead
    public String imgPath="";//this is also serialized
    public String status="";



    public Profile(){//constructor
        this("");
    }
    public Profile(String name){//constructor
        this.name=name;
        status="";
        img=getImage();
    }

    //other methods
    //only for compatibility
    /** This method returns the name associated with the profile. */
```

```
public String getName() {
    return name;
}
/**
 * This method returns the image associated with the profile.
 * If there is no image associated with the profile, the method
 * returns null. */
public GImage getImage() {
    if(img==null){
        if(imgPath.length()>0){
            try{
                img=new GImage(imgPath);
            } catch (ErrorException e){
                img=null;
                System.out.println("image not found in:\n"+imgPath);
            }
            if(imgArr!=null){
                if(img==null || !Arrays.deepEquals(img.getPixelArray(), imgArr)){
                    if(img==null){
                        System.out.println("null kasi");
                    } else {
                        System.out.println("unequal kasi");
                    }
                    img=new GImage(imgArr);
                    System.out.println("image recreated.");
                }
            }
        } else if (imgArr!=null){
            img=new GImage(imgArr);
            System.out.println("path blank. image recreated.");
        }
    }
    return img;
}
/** This method sets the image associated with the profile. */
public void setImage(GImage image,String path) {
    img=image;
    if(image==null){
        imgArr=null;
        imgPath="";
    } else {
        imgArr=img.getPixelArray();
        System.out.println("image pixel-set (just in case).");
        if(path==null){
            imgPath="";
        } else {
            imgPath=path;
            }
        }

    }
    public void setImage(GImage image) {
        setImage(image,null);
    }
    /**
     * This method returns the status associated with the profile.
     * If there is no status associated with the profile, the method
     * returns the empty string ("").
     */
    public String getStatus() {
        return status;
    }
    /** This method sets the status associated with the profile. */
    public void setStatus(String status) {
        this.status=status;
    }

    @Override
    public boolean equals(Object obj) {//names by itself cannot be the same.
        if(obj instanceof Profile){
            return equals((Profile) obj);
        }
        return super.equals(obj); //ata
    }
    public boolean equals(Profile p){
        return getName().equals(p.getName());
    }



    // <editor-fold defaultstate="collapsed" desc="toString shortcut">
    //++toString shortcut
    @Override
    public String toString() {
        StringBuilder s=new StringBuilder(getName()+" ("+getStatus()+"):");
        if(getImage()!=null){
            s.append("[has image] ");
        }
        return s.toString();
    }
    // </editor-fold>
}
```

O) Tree.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;

import java.util.ArrayList;

/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class Tree<E>{
   //here, tree has trees as children
   public Tree<E> parent;
   public E root=null;
   //these two should be in sync
   private ArrayList<Tree<E>> children=new ArrayList<>();
   private ArrayList<GEdge> edges=new ArrayList<>();

   public Tree(E rootItem){//constructor
      root=rootItem;
      parent=null;
      children=new ArrayList<>();
      edges=new ArrayList<>();
      checkSync("Tree");
   }

   //other methods
   public int getNumChildren(){
      checkSync("getNumChildren");
      return children.size();
   }


   public void addChild(Tree<E> childTree,GEdge e){
      if(children.contains(childTree)){
         System.out.println("cannot have two connections "
               + "to the same child!\nchild:"+childTree.root);
         throw new TreeOperationException("");
      } else if (childTree.equals(this)){
         System.out.println("cannot connect a tree to itself");
         checkSync("addChild");
         throw new TreeOperationException("");
      }
      children.add(childTree);
      edges.add(e);
      childTree.setParent(this);
   }
   public void addChild(Tree<E> childTree, int weight, String desc){
    addChild(childTree, new GEdge(weight, desc));
}
public void addChild(Tree<E> childTree, int weight){
    addChild(childTree, new GEdge(weight));
}
public void addChild(Tree<E> childTree,  String desc){
    addChild(childTree, new GEdge(desc));
}
public void addChild(Tree<E> childTree){
    addChild(childTree, new GEdge());
}
public void addChild(E childItem,GEdge e){
    addChild(new Tree<E>(childItem), e);
}
public void addChild(E childItem, int weight, String desc){
    addChild(new Tree<E>(childItem), new GEdge(weight, desc));
}
public void addChild(E childItem, int weight){
    addChild(new Tree<E>(childItem), new GEdge(weight));
}
public void addChild(E childItem,  String desc){
    addChild(new Tree<E>(childItem), new GEdge(desc));
}
public void addChild(E childItem){
    addChild(new Tree<E>(childItem), new GEdge());
}


//false if not found
public boolean remChild(Tree<E> childTree){
    int i=children.indexOf(childTree);
    if(i==-1){
        System.out.println("remove: child not found");
        checkSync("remChild");
        return false;
    } else {
        children.remove(i);
        edges.remove(i);
        checkSync("remChild");
        return true;
    }
}
public Tree<E> remChildByIndex(int index){
    Tree<E> temp=null;
    try{
        temp=children.remove(index);
        edges.remove(index);
    } catch (IndexOutOfBoundsException e){
        System.out.println("incorrect index, removal failed.");
        e.printStackTrace();
    }
    checkSync("remChildByIndex");
    return temp;
}
public Tree<E> remChildByItem(E childItem){
    int index=indexOfChildItem(childItem);
    if(index==-1){
        System.out.println("child item not found");
        checkSync("remChildByItem");
        return null;
    } else {
        return remChildByIndex(index);
    }
}


public int indexOfChildItem(E query){
    checkSync("indexOfChildItem");
    for(int i=0;i<children.size();i++){
        Tree<E> t=children.get(i);
        if(t.root.equals(query)){
            return i;
        }
    }
    return -1;
}
public int indexOf(Tree<E> query){
    checkSync("indexOf");
    return children.indexOf(query);
}


public boolean isChild(Tree<E> query){
    return indexOf(query)!=-1;
}
public boolean isChildByItem(E query){
    return indexOfChildItem(query)!=1;
}


public GEdge getGEdgeOfChild(Tree<E> child){
    int index=indexOf(child);
    if(index==-1){
        System.out.println("getGEdgeOfChild:not a child");
```

```
            checkSync("getGEdgeOfChild");
            return null;
        } else {
            checkSync("getGEdgeOfChild");
            return edges.get(index);
        }
    }
    public GEdge getGEdgeOfChildItem(E childItem){
        int index=indexOfChildItem(childItem);
        if(index==-1){
            System.out.println("getGEdgeOfChild:not a child");
            checkSync("getGEdgeOfChildItem");
            return null;
        } else {
            checkSync("getGEdgeOfChildItem");
            return edges.get(index);
        }
    }

    public Tree<E> getChild(int index){
        checkSync("getChild");
        return children.get(index);
    }
    public Tree<E> getChildByItem(E item){
        int index=indexOfChildItem(item);
        if(index==-1){
            System.out.println("getChildByItem:not a child");
            checkSync("getChildByItem");
            return null;
        } else {
            checkSync("getChildByItem");
            return children.get(index);
        }
    }

    public ArrayList<Tree<E>> getChildren(){
        ArrayList<Tree<E>> temp=new ArrayList<>((int)(1.5*children.size()));
        for(Tree<E> t:children){
            temp.add(t);
        }
            checkSync("getChildren");
        return temp;
    }

    public void setParent(Tree<E> parentTree){//only use this within the tree
        if(parentTree!=null){
            if(parentTree.isChild(this)){
```

```
            parent=parentTree;
        } else {
            checkSync("Tree");
            throw new TreeOperationException("parent in "
                    + "constructor does not have this as a child.");
        }
    } else {
        parent=null;
    }
    checkSync("Tree");
}

public Tree<E> getParent(){
    checkSync("getParent");
    return parent;
}
public boolean isOrphan(){
    checkSync("isOrphan");
    return parent==null;
}


//////////////////////////////////////////////////////////////////
//||--||--||ooooooooooooooooo A L G O ooooooooooooooooo||--||--||//
//////////////////////////////////////////////////////////////////
public void preOrder(ArrayList<Tree<E>> output){
    output.add(this);
    for(Tree<E> t:children){
        t.preOrder(output);
    }
}
public void postOrder(ArrayList<Tree<E>> output){
    for(Tree<E> t:children){
        t.postOrder(output);
    }
    output.add(this);
}
//in order here uses 2nd passage
public void inOrder(ArrayList<Tree<E>> output){
    children.get(0).inOrder(output);
    output.add(this);
    for(int i=1;i<children.size();i++){
        children.get(i).inOrder(output);
    }
}
```

```
public Tree<E> preOrderSearch(E query){
    ArrayList<Tree<E>> list=new ArrayList<>();
    preOrder(list);
    for(Tree<E> t:list){
        if(t.root.equals(query)){
            return t;
        }
    }
    return null; //not found
}




private void checkSync(String methodName){
    if(children.size()!=edges.size()){
        System.err.println("Sync Error");
        System.out.println("children.size() = " + children.size());
        System.out.println("edges.size() = " + edges.size());
        throw new TreeSyncException("last method call:"+methodName);
    }
}
private void checkSync(){
    if(children.size()!=edges.size()){
        System.err.println("Sync Error");
        System.out.println("children.size() = " + children.size());
        System.out.println("edges.size() = " + edges.size());
        throw new TreeSyncException("last method call: <see stacktrace>");
    }
}

// <editor-fold defaultstate="collapsed" desc="toString">

@Override
public String toString() {
    StringBuilder s=new StringBuilder("[parent:");
    if(parent==null){
        s.append("(null)");
    } else {
        s.append(parent.root);
    }
    s.append("],(tree) Root:"+root+"\n");
    if(children.isEmpty()){
        s.append("THIS IS A LEAF.\n\n");
        return s.toString();
    } else {
        s.append("children:[ ");
```

```
            for(Tree<E> t:children){
                s.append(t.root+",");
            }
            s.append("\b]\nedges:[ ");
            for(GEdge e:edges){
                s.append(e.wt+"("+e.desc+");");
            }

            s.append("\b]\n\nSubtrees:\n");
            for(Tree<E> t:children){
                s.append(t+"\n");
            }
            return s.toString();
        }
    }
    // </editor-fold>
}
```

P) TreeOperationException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class TreeOperationException extends RuntimeException{
    public TreeOperationException(String s){
        super(s);
    }//end constructor
}
```

Q) TreeSyncException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.facepamphlet;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class TreeSyncException extends RuntimeException{
    public TreeSyncException(String s){
        super(s);
    }//end constructor
}
```

REFERENCES

1. E Roberts. *Art and Science of Java*. Pearson; 2013.
2. E Roberts, M Sahami, and M Stepp, *CS 106A: Programming Methodology (Java) Handouts*, Stanford University.