

# **BBM 104 – Introduction to Programming II Lab.**

## **Programming Assignment 4 – Object-Oriented Programming with Java**

### **Developing Mini *Facebook* system via GUI – a prototype<sup>1</sup>**

Developed for the Spring 2017 term by Research Assistant Selman Bozkir

---

#### **Submission:**

- **Project Folder Zipped as b<StudentID>.zip**
  - **Deadline: 19.05.2017 23:59**
- 

#### **1 INTRODUCTION**

Graphical User Interfaces, or GUIs, are how users interact with programs. Further, JAVA provides a rich set of libraries to create Graphical User Interface in a platform independent way. This homework provides a great understanding on JAVA GUI Programming concepts and after completing this tutorial you will be at an intermediate level of expertise, from where you can take yourself to higher levels of expertise. In particular, we'll look at SWING GUI controls along with Window Builder GUI designer.

As is known, it was aimed to create a mini Facebook system in the previous assignment. In that assignment, you have utilized the concepts of OOP in order to design and create the required classes. So far, you have developed the logic in the backend. This time, you will find a chance to create a Graphical User Interface (GUI) based frontend for presenting an easier to use system. By the end of this assignment, you will have learnt the concepts of Java Swing based GUI programming as well as adapting the business rules to views.

#### **1.1 LEARNING OBJECTIVES**

The learning objective of this programming assignment is to let students practice the following concepts of Java programming language in particular (and GUI programming in general):

- Object-oriented programming
- Abstraction, Inheritance, Polymorphism
- Abstract Class, Java Interface
- Java Collections
- UML class diagrams
- Developing Graphical User Interfaces (GUI) via Swing components on Window Builder tool.
- Connecting the business rules with views (graphical front ends)

---

<sup>1</sup> **Covered subjects:** OOP Basics, Abstraction, Encapsulation, Inheritance, Polymorphism, Abstract Class, Interfaces, Java Collections, UML class diagrams, GUI Programming with Window Builder, Dynamic Control Creation

## 1.2 MARKING SCHEME

The marking scheme is shown below. Carefully review each mark component.

Component	Mark %
Creation of Base Classes with the requested class structure (the main class of your application must be named <b>Main.java</b> )	10%
Successful instantiation of all initial user objects from the input file	5%
Correct execution of Login and Create User Frames	10%
Correct execution of Profile Page Frame along with dynamic control creation	50%
Correct execution of Tag Friend Frame	10%
Correct execution of Add Post Frame	10%
Drawing of UML class diagram including all JFrame classes (saved as <b>uml.jpg</b> )	5%
<b>Total</b>	<b>100%</b>

**Note:** In your solutions, your algorithm and code should be as simple as possible.

## 2 USEFUL INFORMATION

In this section you can find some useful beginner's level information that you will need for this project. For more information on each subject you need to do additional research and consult other resources (e.g. lecture notes, textbook, Internet resources, etc.).

### 2.1 SHORT OVERVIEW OF SWING AND YOUR FIRST GUI APP

Java GUI programming involves two packages: the original abstract windows kit (AWT) and the newer Swing toolkit. Swing introduced a mechanism that allowed the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a pluggable look and feel allows Swing components to emulate the appearance of native components while still retaining the benefits of platform independence. Originally distributed as a separately downloadable library, Swing has been included as part of the Java Standard Edition since release 1.2. The Swing classes and components are contained in the `javax.swing` package hierarchy.

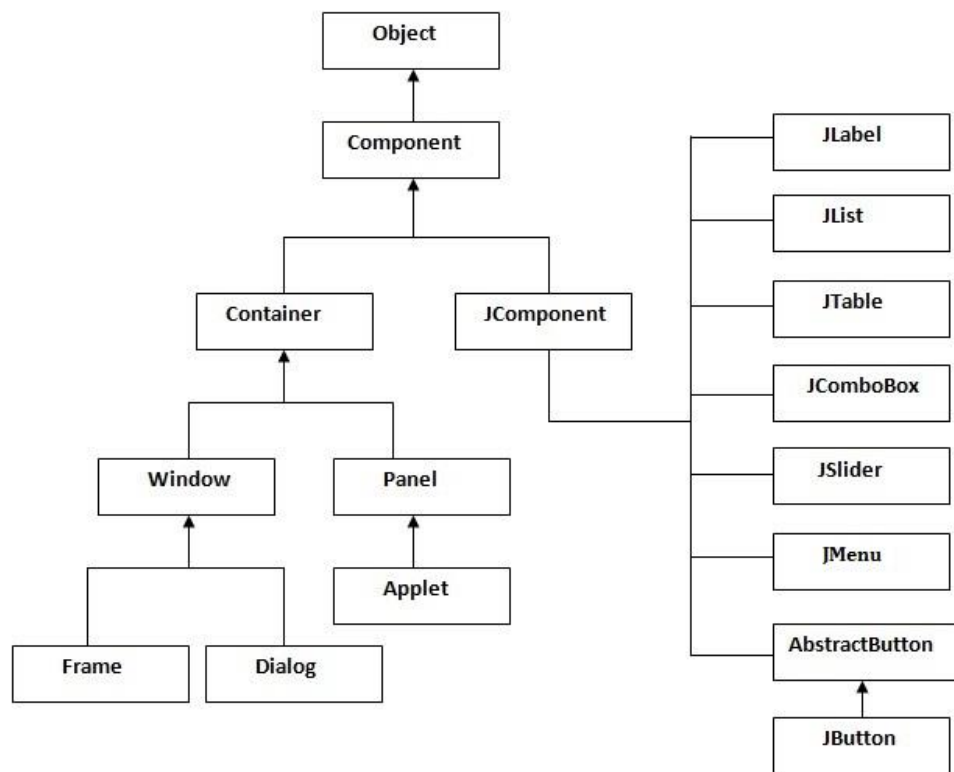
Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element.

Swing components have the prefix J to distinguish them from the original AWT ones (e.g. **JFrame** instead of **Frame**). To include Swing components and methods in your project, you must import the `java.awt.*`, `java.awt.event.*`, and `javax.swing.*` packages. Displayable frames are top-level containers such as **JFrame**, **JWindows**, **JDialog**, and **JApplet**, which interface with the operating system's window manager. Non-displaying content panes are intermediate containers such as **JPanel**, **JOptionPane**, **JScrollPane**, and **JSplitPane**. Containers are therefore widgets or GUI controls that are used to hold and group other widgets such as text boxes, check boxes, radio buttons, et al.

Every GUI starts with a window meant to display things. In Swing, there are three types of windows: the *Applet*, the *Dialog*, and the *Frame*. These interface with the windows manager. In swing, a frame object is called a **JFrame**. A **JFrame** is considered the top most container. These are also called displayable frames. Non-displaying content panes are intermediate containers such as **JPanel**, **JScrollPane**, **JLayeredPane**, **JSplitPane** and **JTabbedPane** which organize the layout structure when multiple controls are being used. Stated simply, the content pane is where we place out text fields are other widgets, so to add and display GUI controls, we need to specify that it is the content pane that we are adding to. The content pane is then at the top of a containment hierarchy, in which this tree-like hierarchy has a top-

level container (in our case JFrame). Working down the tree, we would find other top level containers like **JPanel** to hold the components. The hierarchy of java swing API is given below.



**Fig. 1** A short overview of the Swing API

Here is the code that produces a simple frame upon to build on:

```

public class helloSwing extends JFrame {

    private JPanel contentPane;
    private JTextField textMass;
    private JTextField textHeight;

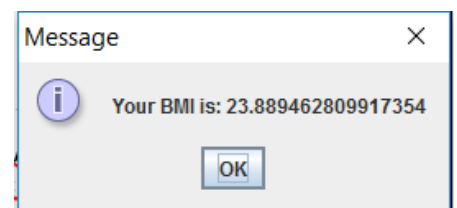
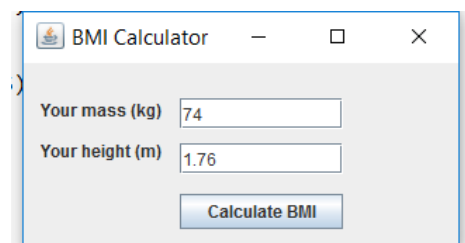
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    helloSwing frame = new helloSwing();
                    frame.setVisible(true);
                } catch (Exception e) {e.printStackTrace();}
            }
        });
    }

    /**
     * Create the frame.
     */
    public helloSwing() {
        setTitle("BMI Calculator");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 271, 177);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Your mass (kg)");
        lblNewLabel.setBounds(12, 25, 101, 16);
        contentPane.add(lblNewLabel);

        JLabel lblYourHeightm = new JLabel("Your height (m)");
        lblYourHeightm.setBounds(12, 54, 101, 16);
    }
}

```



```

        contentPane.add(lblYourHeightm);

        textMass = new JTextField();
        textMass.setBounds(111, 25, 116, 22);
        contentPane.add(textMass);
        textMass.setColumns(10);

        JButton btnCalculateBMI = new JButton("Calculate BMI");
        btnCalculateBMI.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                double mass = Double.parseDouble(textMass.getText());
                double height = Double.parseDouble(textHeight.getText());
                double BMI = mass / Math.pow(height, 2);
                JOptionPane.showMessageDialog(null, "Your BMI is: " + BMI);
            }
        });
        btnCalculateBMI.setBounds(111, 93, 116, 25);
        contentPane.add(btnCalculateBMI);
        textHeight = new JTextField();
        textHeight.setColumns(10);
        textHeight.setBounds(111, 57, 116, 22);
        contentPane.add(textHeight);
    }
}

```

## 2.2 INTRODUCTION TO “WINDOWBUILDER”

A GUI builder is a visual programming tool that lets a user build a graphical user interface by dragging and dropping elements from a palette onto a design surface. GUI builders typically contain Wizards and templates that help automate the GUI building process. GUI building tools allow developers to spend less time and money creating Java GUIs, because they can focus on creating application-specific functionality rather than coding the low-level logic required for GUIs to run.

According to Ben Galbraith, in his Successful GUI Building blog, “In my experience, the number one reason why GUI builders hold appeal is that they often lead to an order of magnitude improvement in productivity over hand coding, especially amongst average-level Swing developers. I consider myself above average, and I see improvements in productivity around 2x-10x (perhaps higher) in my own work.”

Created in 2003 by Instantiations, WindowBuilder is widely regarded as the best GUI builder in the Java world (winning the award for Best Commercial Eclipse Tool in 2009). Beyond being an exemplary example of a GUI design tool for SWT, XWT, Swing and Google’s GWT, WindowBuilder is also a highly extensible and customizable framework (with 50+ existing extension points) for creating new GUI design tools based on any UI toolkit for any language. Google re-launched WindowBuilder as a free product available to any member of the Eclipse community soon after its acquisition of Instantiations in 2010, and now wishes to contribute the framework (the WindowBuilder Engine) and associated products (SWT Designer and Swing Designer) to the Eclipse Foundation. Google will continue to invest heavily in the project, as it will continue to be the basis for its own in-house GWT Designer tool.

WindowBuilder is composed of SWT Designer and Swing Designer and makes it very easy to create Java GUI applications without spending a lot of time writing code. Use the WYSIWYG visual designer and layout tools to create simple forms to complex windows; the Java code will be generated for you. Easily add controls using drag-and-drop, add event handlers to your controls, change various properties of controls using a property editor, internationalize your app and much more.

WindowBuilder is built as a plug-in to Eclipse and the various Eclipse-based IDEs (RAD, RSA, MyEclipse, JBuilder, etc.). The plug-in builds an abstract syntax tree (AST) to navigate the source code and uses GEF to display and manage the visual presentation.

Generated code doesn't require any additional custom libraries to compile and run: all of the generated code can be used without having WindowBuilder Pro installed. WindowBuilder Pro can read and write almost any format and reverse-engineer most hand-written Java GUI code. It also supports free-form code editing (make changes anywhere...not just in special areas) and most user re-factorings (you can move, rename and subdivide methods without a problem). **WindowBuilder toolkit can be easily download from Help menu of Eclipse by searching in marketplace.**

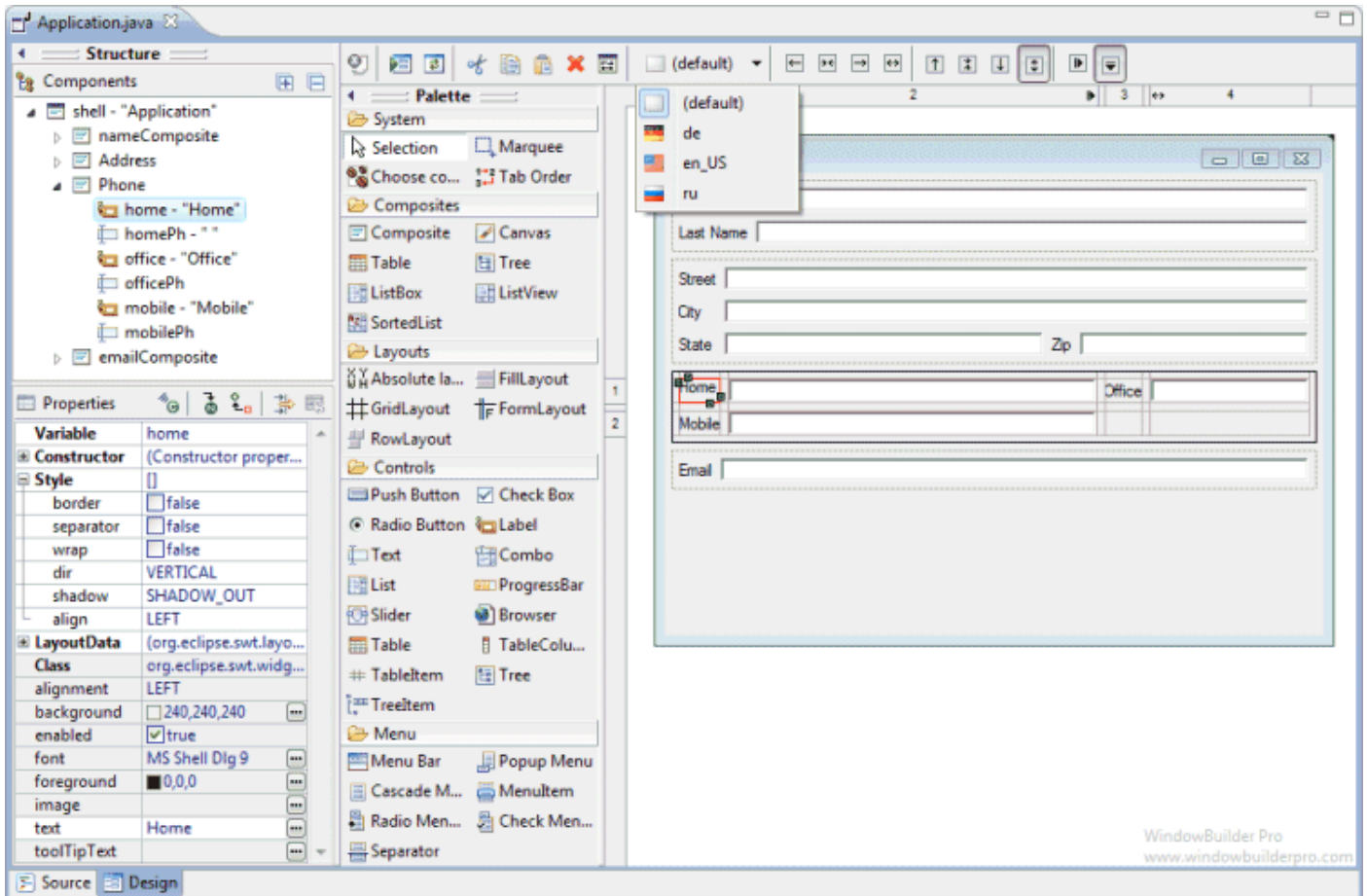


Fig. 2 A screenshot of the WindowBuilder

### 3 AN OVERVIEW OF THE APPLICATION

In this programming assignment, you will develop a very simplified version of *Facebook*, well-known social media platform. We will call it Mini-Facebook. Note that Programming Assignments 3 and 4 are consecutive and you can use your classes which you have already created in Assignment 3 with few modifications.

#### 3.1 PROBLEM DEFINITION

In this assignment you are supposed to create a GUI based front end by using Swing components. By using this GUI, a user must interact with your Mini-Facebook system.

First, you will be supplied with two text files ("users.txt" and "commands.txt") which you will use them to initialize the system. To be precise, all users, their friendship/blocked users lists will be built by use of these two input files. Moreover, all the initial posts will be created and assigned to related users. You can get the name of these files from **args** parameter in static main function. Their names and order will be users.txt and commands.txt respectively.

In this assignment, the user class and business logic of the system have been changed a bit. In the following list, these changes have been listed

1. In this assignment it is enabled all users to login by providing his/her **username** and **password** whereas the 3<sup>rd</sup> assignment focuses on single user login. Note that, to let another user login, the previously logged in user must log out first. This is because your system is capable of handling only one active logged in user.
2. **User** class now contains a new property "**relationship\_status**" which is designed to store one of the following values {"In a relationship", "Divorced", "Complicated", "Single"}.
3. The *userid* property of the User class is no longer needed since username field is a unique value property. Therefore, you can use *username* property as a key in collections.
4. Some functions (e.g. change password) of the User class have been removed. Please read the Section 3.1.1 for new rules of User class.

5. In the 3<sup>rd</sup> assignment, there was a requirement of the **UserCollection** class which states “Displaying users’ posts”. However, within this assignment, the responsibility of the Posts has been moved to User class itself. In other words, some functions such as “add post” or “show post(s)” will be handled by User class since Posts of the user must be stored in a collection at User class.

Note that, user collection class still exists and it is responsible for storing each user in its collection. In the following sub sections the details of the business classes are presented.

### 3.1.1 Users

Users are people who have an account on our Mini-Facebook system. Users will not have different roles (e.g. admin or system manager vs. other users); that is, they will all be peers. User’s attributes and behaviors are explained below:

- Every **User** in the system has a unique **username**, a **name**, a **password**, a **date of birth**, information about **school from which the user graduated**, a **last log-in date**, a **collection of friends**, a **collection of blocked users**, and a **collection of posts**.
- Users’ dates of birth and last log-in dates should be stored as Date or Calendar variable. However both the input and GUI rendering format must be “dd/mm/yyyy” (e.g. 03/11/1997)
- All collections must be implemented as Java Collections.
- Users must be able to
  - **Sign in,**
  - **Sign out,**
  - **Update their user profile info (name, date of birth, school graduated and relationship status),**
  - **Add friends to their friend lists,**
  - **Remove friends from their friend lists,**
  - **Add posts,**
  - **Block users,**
  - **Unblock users,**
  - **List their friends,**
  - **List all available users** (*here the term available is used to refer the users that you did not block and the users that you have not been blocked by them*),
  - **List blocked users.**
- When a user is not logged-in, he/she should be allowed only to perform a sign-in. After user log-in, other actions should be permitted.
- Note that you are required to hide sensitive personal user info and implement setter and getter methods as necessary.

### 3.1.2 Posts

Posts represent textual or multimedia content shared by users. You are expected to design your post classes in a hierarchical structure. The necessary information about posts is as follows:

- First of all, you need to implement an **interface** which will specify behavior (methods) that the Post class will implement.
- Secondly, you need to implement an **abstract Post class** which will implement the post interface and be a superclass to all other post classes.
- This abstract Post class should define post attributes, implement the methods defined in the interface, and define two additional abstract methods that will **show tagged users** and **show post location**. These methods should be implemented in the immediate subclasses of the Post class.
- Posts are categorized as either **TextPosts**, **ImagePosts**, or **VideoPosts**. But **be careful**: all posts have a textual part (all posts are actually text posts), and can optionally contain image or video (but not both!).
- All posts have a unique **postID** which should be assigned as a random [UUID \(immutable universally unique identifier\)](#), a **text**, a **date** when the post originated, a **location**, and a **collection of tagged friends**.
- **Location** should be implemented as a separate class with the following attributes: **latitude** and **longitude** (stored as double values).
- **TextPost** class should implement abstract methods from the abstract Post class. To show the tagged friends, it is enough to print their names (if any). To show the post location, both latitude and longitude should be printed.
- **ImagePosts** have the **image filename**, and the **image resolution** (width and height in pixels).
- **VideoPosts** have the **video filename**, **video duration**, and a constant attribute that specifies the **maximum video length** in minutes (the maximum allowed video duration is 10 minutes).



- Note that you are required to implement setter and getter methods as necessary.

### 3.1.3 User Collection

The main class of *UserCollection* needs to allow for the following actions:

- Keeping track of all users,
- Adding new users to the system (by providing new user information),
- Removing existing users from the system (by providing the username),
- User sign-in (with username and password),
- In the 3<sup>rd</sup> assignment, there was a requirement which states “Displaying users’ posts”. However, the responsibility of Posts has been moved to User class itself.

### 3.1.4 System Initialization

By using “users.txt” and “commands.txt” you should initialize the system. The command structure and details are presented in the following lines. Further, you will neither generate a warning nor an output during the initialization stage since the commands will be error-free and do not contain any Turkish character. As you may realize, the number of commands in the input files have been reduced to 5 because of moving to GUI environment.

- **Adding New Users (user.txt):** At the initial stage, your program should retrieve the user information from the file **users.txt** and add them to the system. Syntax of the command is as provided below. Every line starts with the name of the user.

```
name<TAB>userName<TAB>password<TAB>dateOfBirth<TAB>schoolGraduated<TAB>relationshipstatus
```

*Usage Illustration:*

```
Adnan Yuksel<TAB>adnan<TAB>adnan1<TAB>01/01/1991<TAB>ODTU<TAB>Single
```

- **Creation of Friendship (commands.txt):** In this command, you build a friendship between two users. As a result, two users add each other as a friend Syntax of the command is as provided below.

```
ADDFRIEND<TAB>username_of_the_first_user<TAB>username_of_the_second_user
```

*Usage Illustration:*

```
ADDFRIEND<TAB>adnan<TAB>selman
```

- **Blocking the Users (commands.txt):** In most of the real world social platforms, a user might want to block another user due to several reasons (disturbing posts, etc.). In this assignment, your implementation should also allow users to block other users given a username as the argument. As a result of this command, two users should block each other. In real world, blocking operation is one directional. However, in this assignment, blocking operation has been defined as bi-directional for the sake of simplicity. To be precise, if the user x blocks user y, this will lead user y to block user x simultaneously. Similarly, unblocking operation holds the same bi-directional behavior that is two users will unblock each other at the same time.

```
BLOCKFRIEND<TAB>username_of_the_first_user<TAB>username_of_the_second_user
```

*Usage Illustration:*

```
BLOCK<TAB>ahmet<TAB>mehmet
```

- **Adding Text Post (commands.txt):** User may create a new text post by specifying the text content, location (longitude & latitude), and username of the friends who will be tagged to this post, each of which is separated by TAB. Furthermore, the entity username of the post owner has been added to denote whose the post is.

**ADDPOST-**

```
TEXT<TAB>username_of_the_post_owner<TAB>textContent<TAB>longitude<TAB>latitude<TAB>B>userName1<:>userName2<:>...<:>userNameN
```

```
ADDPOST-TEXT<TAB>selman<TAB>This is my 1st text
post<TAB>39.8833431<TAB>32.7381663<TAB>ahmet:demet:adnan
```

- **Adding Image Posts (commands.txt):** Not only can a user create text post, but he/she can also create an image or a video post. In this scenario, the image path and resolution information should be provided to the system as well. Similar to ADDPOST-TEXT command, here the username of the post owner has been added to parameter list.

```
ADDPOST-IMAGE<TAB>username_of_the_post_owner
<TAB>textContent<TAB>longitude<TAB>latitude<TAB>
<TAB>userName1<:>userName2<:>...<:>userNameN<TAB>filePath<TAB>resolution
```

```
ADDPOST-IMAGE<TAB>selman<TAB>This is my 1st image
post<TAB>42.8833431<TAB>32.638153<TAB>demet:gizem<TAB>image.png<TAB>135<x>250
```

- **Add Video Post:** User should only provide the file path of the video and the duration of the video. Similar to ADDPOST-TEXT and ADDPOST-IMAGE commands, here the username of the post owner has been added to parameter list.

```
ADDPOST-VIDEO<TAB>username_of_the_post_owner
<TAB>textContent<TAB>longitude<TAB>latitude
<TAB>userName1<:>userName2<:>...<:>userNameN<TAB>filePath<TAB>videoDuration
```

```
ADDPOST-VIDEO<TAB>selman<TAB>This is my 1st video
post<TAB>39.8833431<TAB>32.638133<TAB>utku:gizem<TAB>myvideo.avi<TAB>8
```

## 4 INPUT FILES

You will be provided with two (tab-separated) text input files:

- **Users** in the system,
- **Commands** that you need to execute in order to test your application.

### 4.1 USERS INPUT FILE

List of all users in the system will be given in the (tab-separated) `users.txt` file:

The format of user's data will be given as follows:

```
name<TAB>username<TAB>password<TAB>dateofBirth<TAB>graduatedSchool<TAB>relationship_status
```

A sample `users.txt` input file is given below:

Ahmet Utku	ahmet	ahmet12	04/25/2001	Meram Anadolu Lisesi	Single
Demet Akbag	demet	demet00	01/16/1999	Ankara Fen Lisesi	In relationship
Zeki Öztürk	zeki	zeki01	08/16/1987	Kadikoy Lisesi	Divorced
Gizem Kocadag	gizem	gizem1	12/09/1997	Hacettepe Universitesi	Single
Utku Anil	utku	utku99	10/06/1999	Bilkent Universitesi	Complicated
Hakan Kocakulak	hakan	hakan81	03/01/1981	Orta Dogu Teknik Universitesi	Single
Selman Bozkir	selman	selo12	11/12/1983	Hacettepe Universitesi	Single
Hamdi Karayagiz	hamdi	xy253	04/02/1987	Osmangazi Universitesi	Divorced
Aylin Yuksel	aylin	45ay12	03/12/1998	Besiktas Lisesi	Single
Sevda Karaca	sevda	g4256	19/01/1980	Marmara Universitesi	Complicated
Murat Özbayir	murat	murat2	06/08/1974	Ankara Universitesi	Single
Necdet Emin	necdet	neco34	11/10/1982	Ege Universitesi	In relationship

You are expected to create an instance for each user in this file.

### 4.2 COMMANDS INPUT FILE

List of all commands, which will be given to test your program and which you are expected to execute correctly and in order, will be given in the (tab-separated) `commands.txt` file.



ADDFRIEND	<u>ahmet</u>	<u>demet</u>
ADDFRIEND	<u>ahmet</u>	<u>selman</u>
ADDFRIEND	<u>ahmet</u>	<u>murat</u>
ADDFRIEND	<u>demet</u>	<u>necdet</u>
ADDFRIEND	<u>zeki</u>	<u>demet</u>
ADDFRIEND	<u>hakan</u>	<u>murat</u>
ADDFRIEND	<u>selman</u>	<u>gizem</u>
ADDFRIEND	<u>sevda</u>	<u>murat</u>
ADDFRIEND	<u>murat</u>	<u>necdet</u>
ADDFRIEND	<u>murat</u>	<u>aylin</u>
ADDFRIEND	<u>necdet</u>	<u>murat</u>
BLOCKFRIEND	<u>selman</u>	<u>ahmet</u>
BLOCKFRIEND	<u>gizem</u>	<u>utku</u>
BLOCKFRIEND	<u>utku</u>	<u>hakan</u>
BLOCKFRIEND	<u>gizem</u>	<u>hamdi</u>
BLOCKFRIEND	<u>aylin</u>	<u>selman</u>
BLOCKFRIEND	<u>demet</u>	<u>necdet</u>
ADDPST-IMAGE	<u>selman</u>	Thanks Tram Tran for this amazing photo 12.1831 223.6153 gizem:hakan image.png 135<x>250
ADDPST-VIDEO	<u>selman</u>	Doctor at center of USA Gymnastics scandal 82.7331 51.7331 gizem:hakan video1.avi 67
ADDPST-TEXT	<u>selman</u>	My nieces and nephew in my mind... 39.8833431 32.7381663 <u>necdet</u>
ADDPST-TEXT	<u>demet</u>	Aurora hanging out with Pro York 39.8833431 32.7381663 hamdi:necdet
ADDPST-TEXT	<u>demet</u>	In Oklahoma homeboy killed 3 with AR-15. 9.83431 32.7663 hamdi:necdet
ADDPST-IMAGE	<u>demet</u>	Thanks for this amazing photo 12.183431 223.631 utku:sevda image.png 135<x>250

**Important note: DO NOT use Turkish characters in your code (anywhere, not even comments)!**

## 5 GUI BASED SCENARIOS

In this section, usage scenarios and the points that you must pay attention are listed. Each window has been captioned for ease of understanding. As a general rule, all JFrames must be shown at the center of the screen and their dimensions must not be resized.

### 1. Login window (Main.java)

This window is the startup project and the **JFrame** class of this window must be named with “Main.java”. The static void main call must only be implemented in this class. Further, all initialization stages must be implemented in this class. Thus, the **JList** control must be filled with usernames of the registered users. When user clicks on a user the **JList** component at below the appropriate username and password must be typed in username and password (**JTextField**) controls. When the system user clicks on the **JButton** “Login” system must check the credentials and let the user log in or warn for the incorrect username/password combination by showing a warning message. All messages throughout the homework must be shown by utilizing the **JOptionPane.showMessageDialog** function.

This window must contain three buttons (Create User, Remove User and Login). If user clicks on “Remove User” button, then the system must ask to the system admin whether he/she is sure about operation. You can again use **JOptionPane** dialogs for this confirmation. If user clicks on the “Yes” button of the Dialog then all the entries related to deleted user must be removed from the system. Note that, this operation must also delete the removed user from all post tags.

If system user clicks on the button of “Create User” then the Create User JFrame must be loaded and shown.

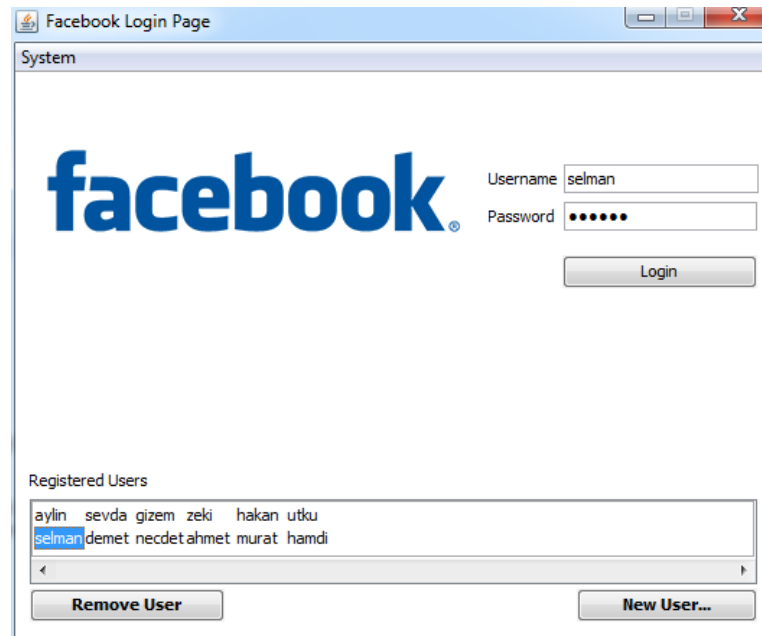


Fig. 3 System startup JFrame.

## 2. Create User window (CreateUser.java)

In this JFrame, a new user can be added by providing all the required information depicted in the Fig 4. All text fields must be checked against empty string values and password values must be doubly checked. If the required conditions are not met your application must warn the user via *JOptionPane.showMessageDialog* function. If all values are valid, then a new user must be created and added to User Collection object via a showing a "Success!" dialog.



Fig. 4 Create User JFrame.

## 3. Profile Page window (ProfilePage.java)

This window is the core part of the assignment and it has been used to show profile pages of both logged in user and visited friends' profile pages. In this regard, it serves a polymorphic behavior ☺ As can be seen in Fig. 5, profile page form has a header, a user icon (it will be static and does not change according to the user), a panel for presenting user information, a panel for listing both friends and blocked users and a right panel for visualizing posts and their respective tagged users. Header part contains a hidden Home button, a textfield for searching

“available” people and the buttons of “Create Post” and “Log out”. The operation rules belonging to this window are listed below:

- Profile page window is a polymorphic window. In other words, it should be used to show both logged user and a friend of logged user upon user searching action. Hence, it has two view modes: (a) logged in user view mode (See Fig. 5a and 5b), (b) visiting mode (See Fig. 6a and 6b)
- If the logged in user searches someone and double clicks on a name of one available person, profile page must be refreshed and the all information on the profile page must be updated according to the searched user.
- In the logged in user view mode, the Home button must not be visible since the window is at the page of logged in user. Therefore, the “Home” button is redundant. In visiting mode, the “Home” button must be visible in order to allow user to come back to his/her original home page.
- If the logged in user types some characters into the “Search Friend” text field, the available users must be listed on the search friend list just below the text field. Note that, this list is always hidden and it must only be shown when user types some characters in the search friend text field. This **JList** must return to hidden state if the user clicks anywhere else than the search list on the screen.
- The posts panel located at the right side of the JFrame is a tabbed panel (it also contains scrollable JScrollPanels) and it comprises two post lists: (a) active user’s own posts (left tab), (b) all posts belonging to the friends of the active user (right tab). If the profile page is in logged in user view mode, each post of logged in user must contain a “Tag User” button for tagging the friends. In contrast, the posts in the right tab will not contain such a button since those posts belong to other people (See Fig. 5b). On the other hand, if the profile page is in visiting mode, all posts in both tabs will not contain any “Tag User” because of the page mode (See Fig. 6a and 6b).
- Every post must denote its type by locating the initial character of its kind to the left of the post that is ‘T’ stands for text posts while ‘I’ denotes image posts. Similarly, video posts must be shown by the character of ‘V’
- Logged in user posts will be shown by presenting the post text and tagged users. On the other hand, friends’ posts must contain a text on top of the post (e.g. “Ayse Tunc has shared”) to denote the owner of the post. This feature is unnecessary for the logged in user posts. This discrimination is also valid for the visiting mode.

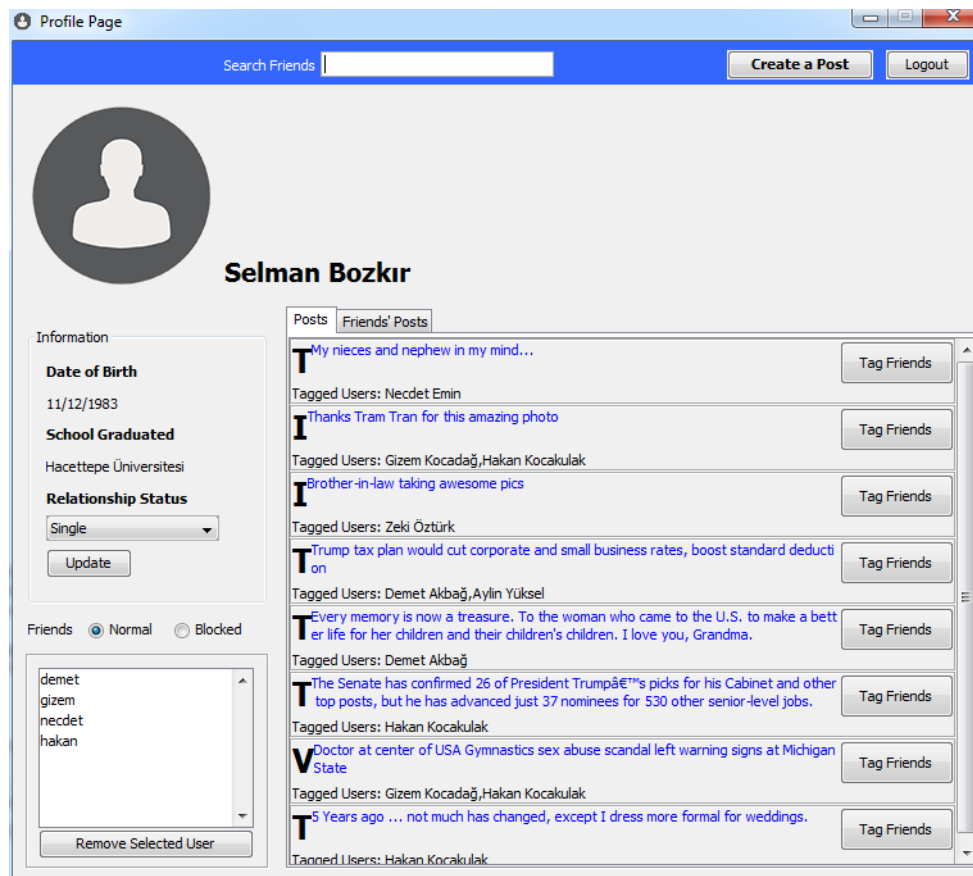


Fig. 5a Logged in user view mode at profile page window (see the posts of the logged in user)

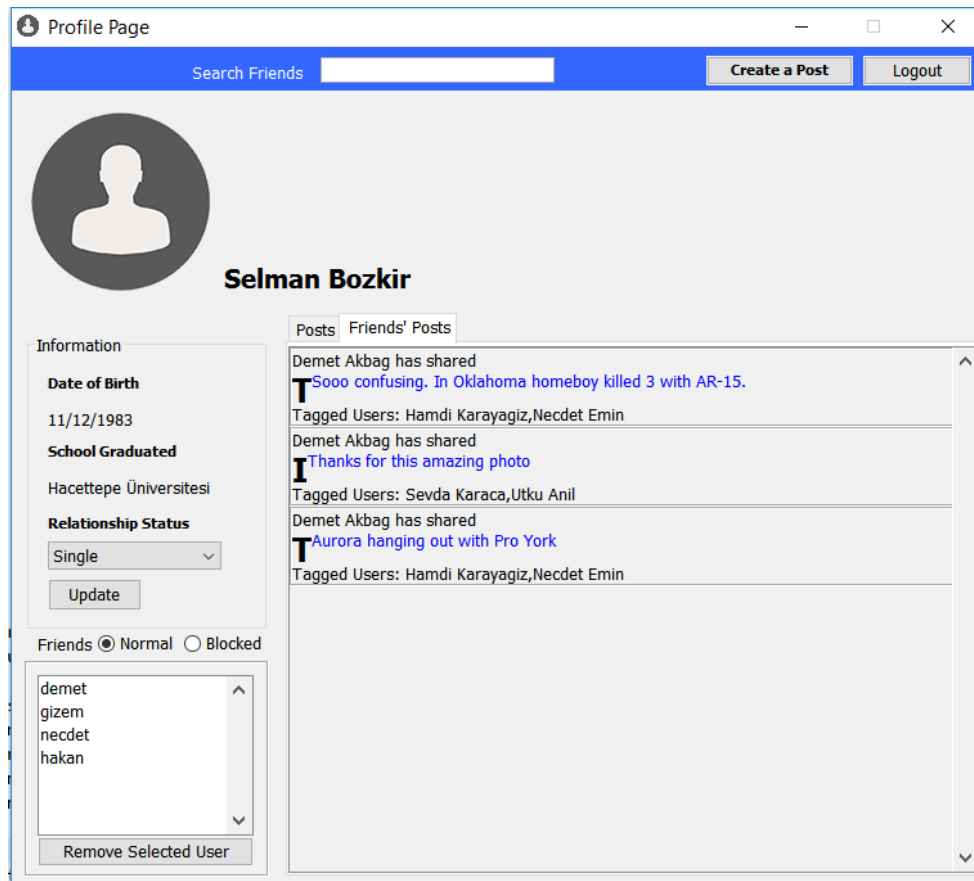


Fig. 5b Logged in user view mode at profile page window (see the posts of the friends of the logged in user)

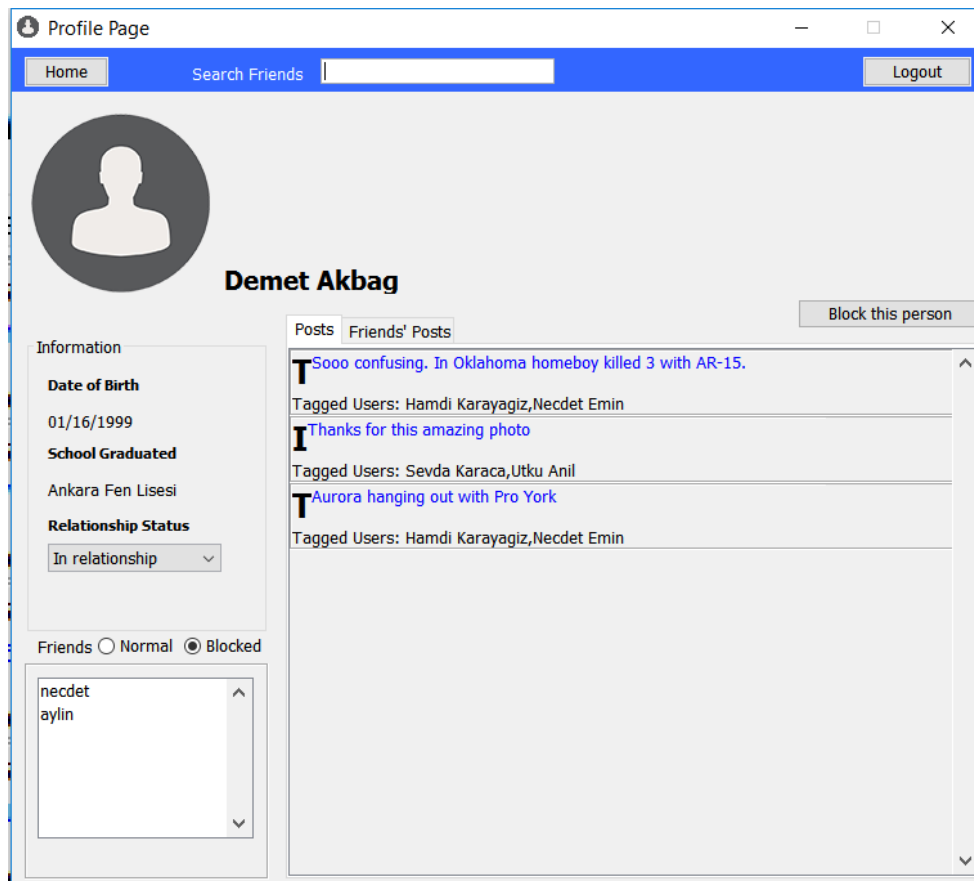
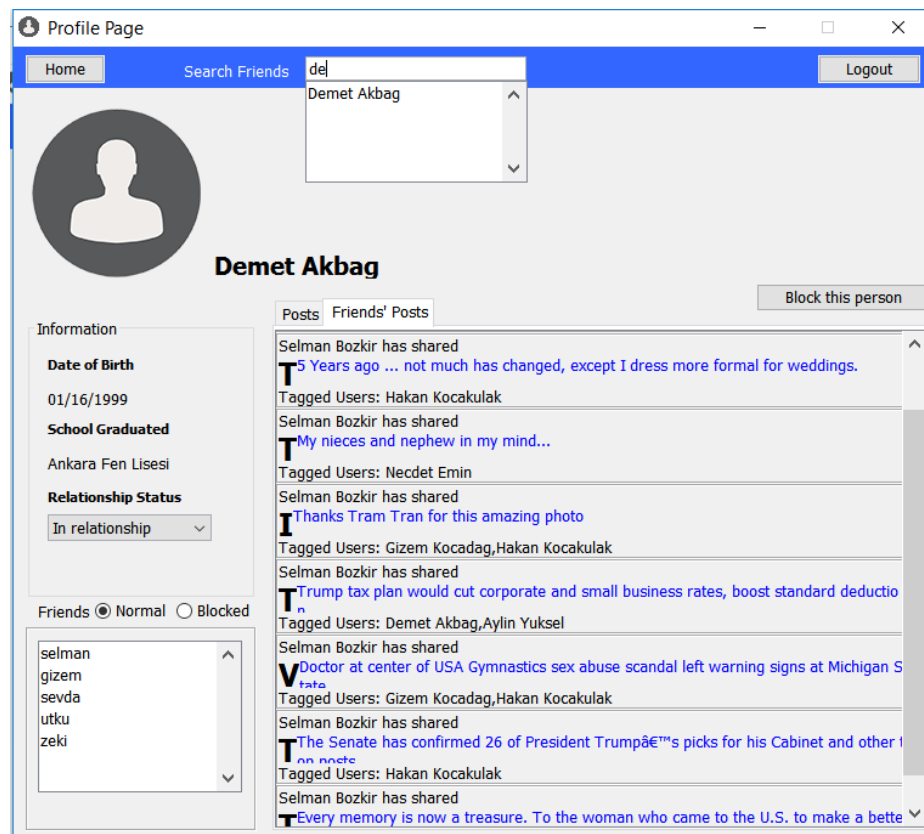


Fig. 6a Visiting mode at profile page window (see the posts of the logged in user)



**Fig. 6b** Visiting mode at profile page window (see the posts of the friends of the logged in user's friend)

- In visiting mode, "Block Friend", "Remove Friend" and "Create Post" buttons must not be visible since logged in user is visiting another user. Once he/she comes back at his/her own page then these buttons must be visible again.
- In visiting mode, two additional buttons: (a) "Add Friend" (b) "Block this person" must be visible depending on the friendship status. If the visited user is a friend of logged in user, then only the button of "Block this person" must be visible. In contrast, if visited user is not a friend of the logged in user then "Add Friend" must also be visible above the button of "Block this person". Note that, for the sake of simplicity, "Add Friend" button accepts the friendship of two people (logged user and visited user) immediately by prompting (e.g. "You added yyy as a friend")
- If the logged in user visits another profile and clicks on "Block this person" then system must block two people in bi-directional way and window must return to logged in user view mode immediately following the prompt message (e.g. " You blocked yyy").
- If the logged in user clicks on the button of "Create Post" then the AddPost JFrame must be shown.
- The button "Log out" makes the logged in user sign out from the system immediately (there is no need to prompt such as "Are you sure?" for log out operation)

#### 4. Adding Posts window (AddPost.java)

This window is a popup window that be shown following the user click on "Create Post" button. Post type must be picked at a Combo Box (JComboBox) from the values (a) Text Post, (b) Image Post and (c) Video Post. Different options must be enabled depending on the post type. See Fig. 7a, 7b and 7c for details.

Note that, tagging is independent from post adding operation. A post must be first created and then tagged later on. As mentioned before, "Tag User" button must be used to tag only your friends.

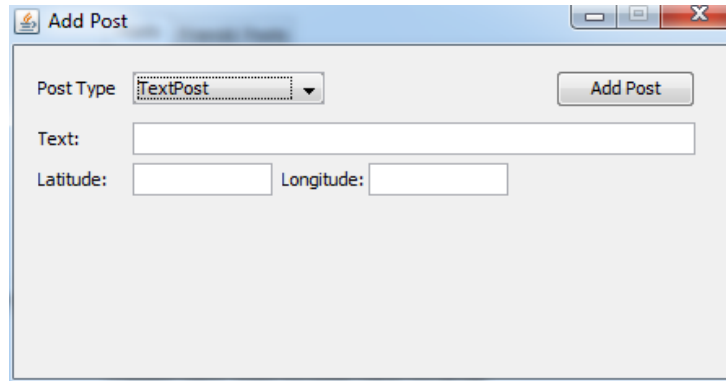


Fig. 7a Adding of Text Post

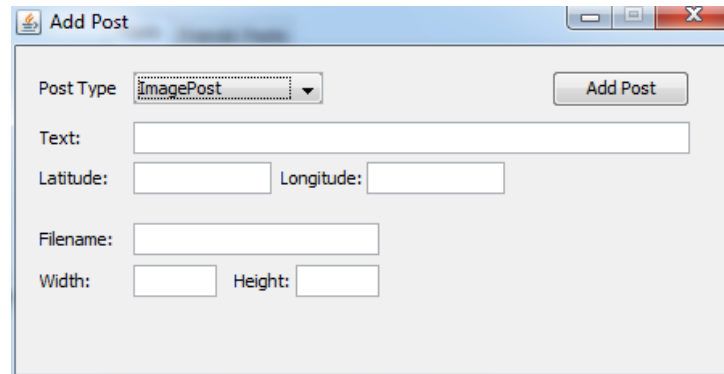


Fig. 7b Adding of Image Post (see the additional properties and their related visual controls)

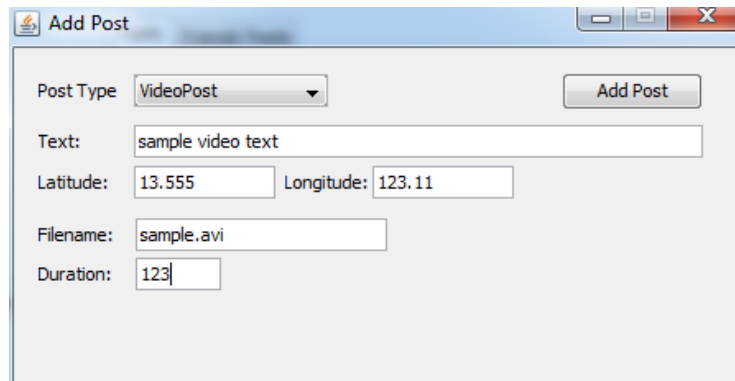


Fig. 7c Adding of Video Post (see the additional properties and their related visual controls)

In order to add a post all the relevant information should be provided. Otherwise, system must warn the user by prompting a message box to inform user. If all the values are valid the Add Post window must be closed and the create post must be added to the post list of the logged in user.

##### 5. Tagging User Window (TagUser.java)

When the logged in user clicks on the “Tag User” button of a specific post, this window must be shown. Tag User window is a relatively simple window. However, it has a special property that is retrieving only the untagged friends. Suppose that there exist 5 friends of the logged in user. If there is no tagged user for the Post P then all the friends must be listed by adding their names (not user names). As it can be predicted, the Tag User window’s responsibility is listing the untagged friends for a specific post. If a user has already been tagged for a specific post then she/he must not be listed anymore for that post. See Fig. 8 for a Tag User window example

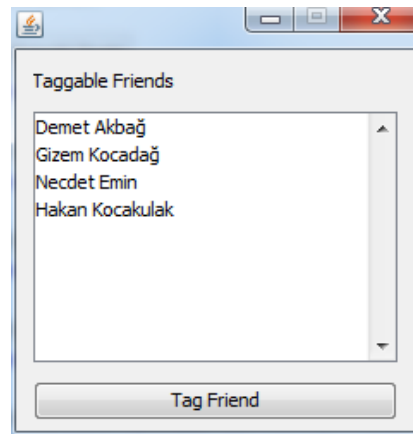


Fig. 8 Tag User Window (the list must always contain all friends that can be tagged)

## 6 CONSTRAINTS

- **Add New Friend:** The system should not allow users to add a friend who is already in his/her friend or does not exist in the system.
- **Blocking a User:** If the logged in user (LU) blocks another user (AU) then AU must be immediately added to blocked user list of LU. If they were friend, this friendship must also be removed. Furthermore, LU must not be able to see AU's name in person search pane anymore and the tags of AU in the posts of LU must be deleted completely. On the other hand, although AU is not directly visible to LU, LU can still see AU's name in the post tags lists. For the sake of simplicity, this exception is allowed.
- **General constraints:**
  - When a user is not logged-in, he/she should be allowed only to perform a sign-in. Only after the user signs in, other actions should be permitted.
  - Users should not be able to tag other users in their posts if they are not their friends.
  - It is assumed that only one user can log in to the system at a particular time, so all actions related to users should be performed by the user who is currently logged in.
  - Input files will be error free. Thus, please be ensure that your code is working and system can be initialized without any failure.

## 7 SUBMISSION

- Submissions will be accepted only electronically via [submit.cs.hacettepe.edu.tr](http://submit.cs.hacettepe.edu.tr).
- The deadline for submission is **19.05.2017 until 23:59**.
- The submission format is:
  - b<student id>.zip
    - src.zip (your all project folder)
    - uml.jpg (In this assignment I am expecting a more detailed UML class diagram including not only the logical classes but also JFrames and their place in this ecosystem)
- **Your main class must be named Main.java!** All of your classes should be in the **src** folder (they may, however, be further classified into other subfolders in case you decide to make packages). Note that only zipped folders can be submitted.
- **All work must be individual!** Plagiarism check will be performed! Duplicate submissions will be graded with 0 and disciplinary action will be taken.

## 8 COMPILE & RUN

```
javac Main.java
```

```
java Main users.txt commands.txt
```