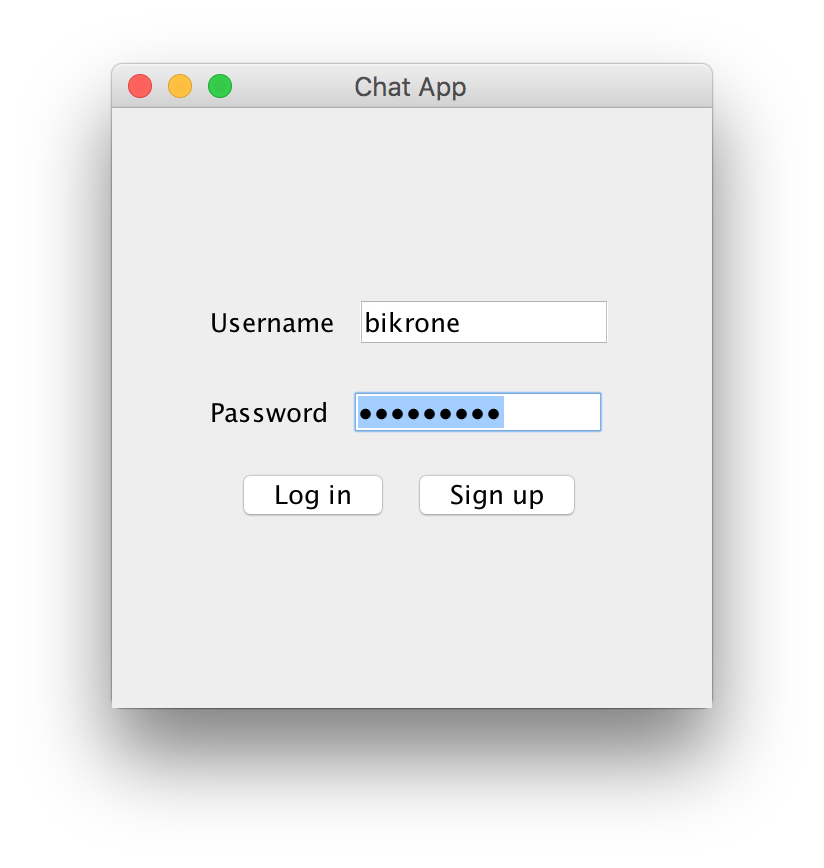
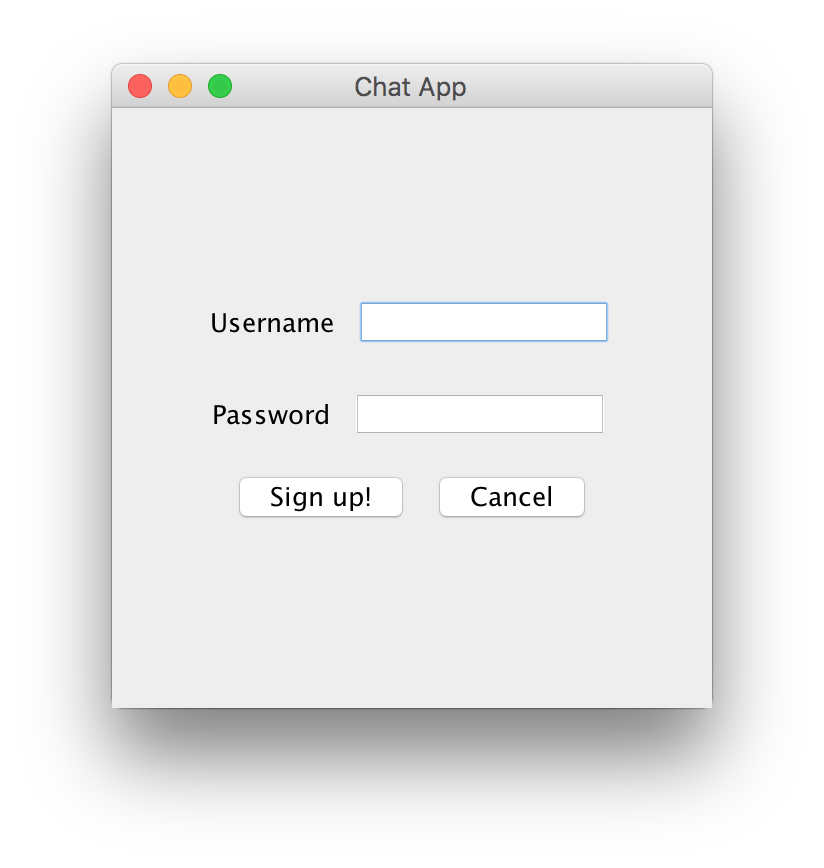
Project 1 – CS494

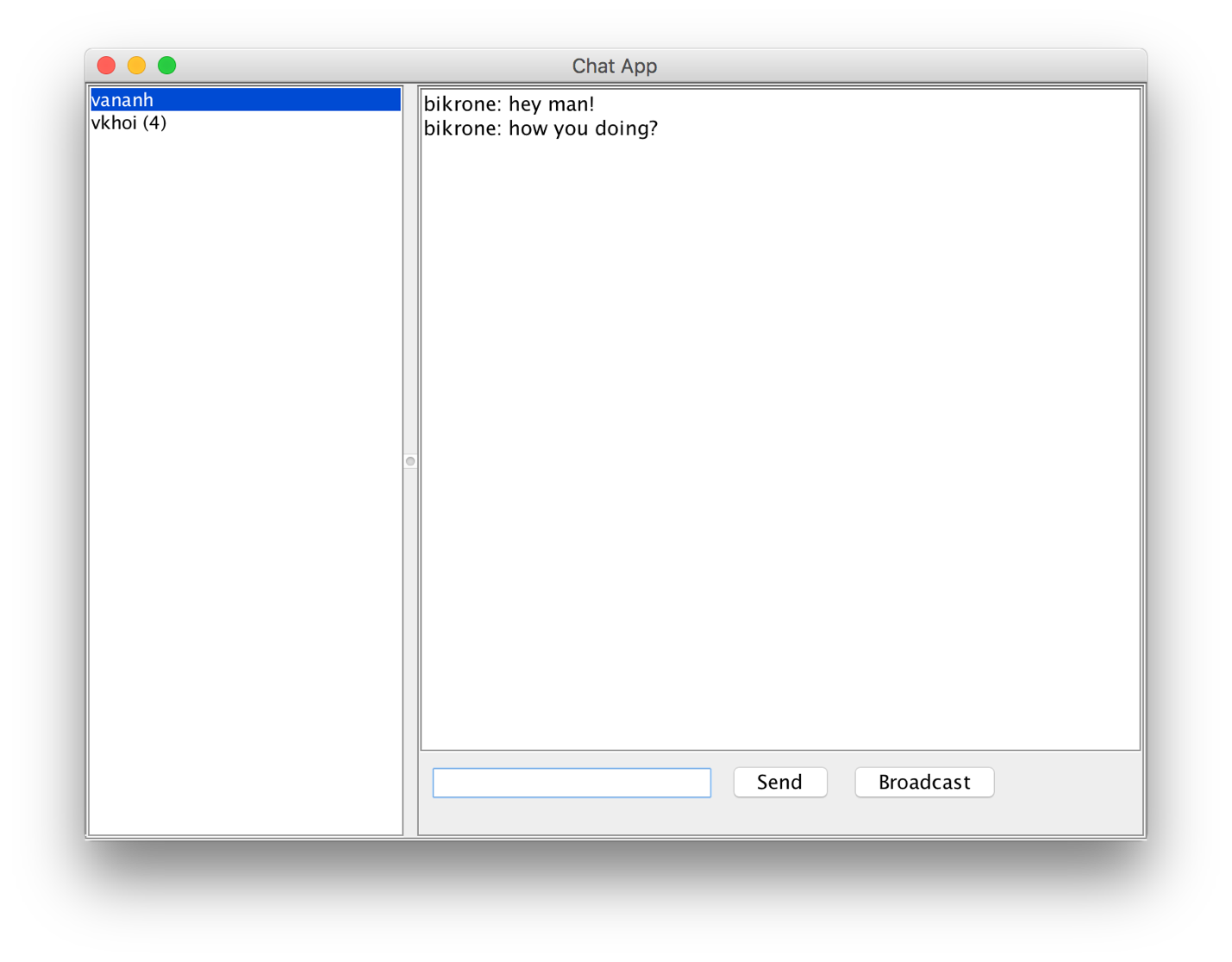
Nguyen Hai Khanh – 1251019 | Nguyen Minh Tu – 1251050

# How to use

* Start the ChatServer FIRST:
  + Using your command line, run “java –jar ChatServer.jar”.
* Start the ChatClient SECOND:
  + Using your command line, run “javar –jar ChatClient.jar”.
* Login form:

- Sign up form:

- Chat form:



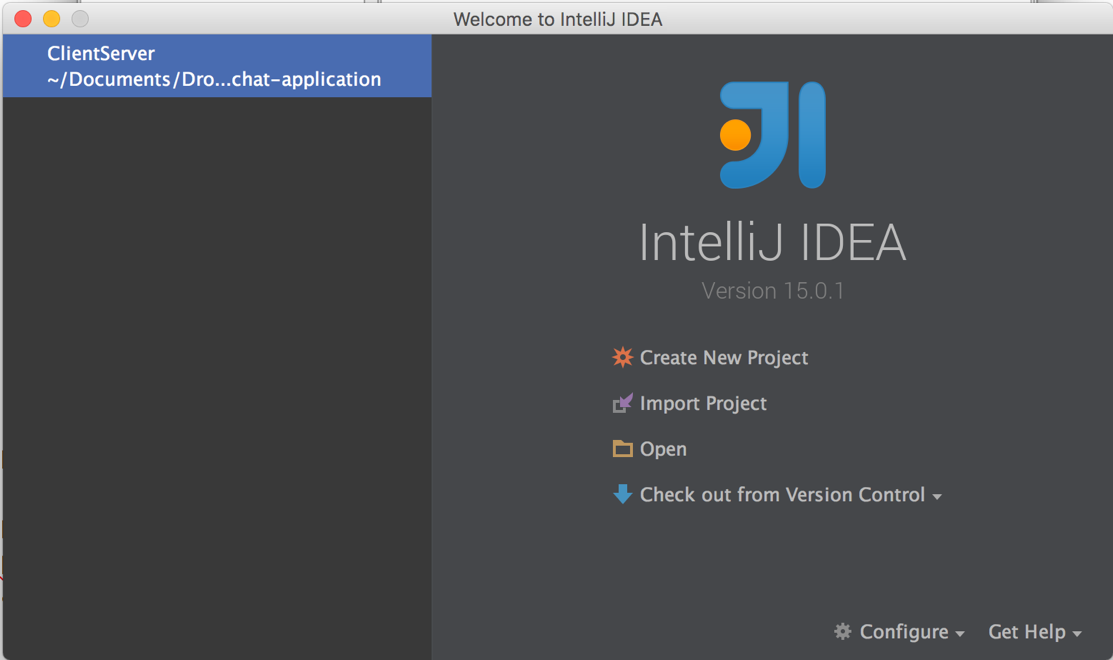
Notice that the **“vkhoi (4)”** means that there are 4 unread messages from vkhoi.

# Building the project

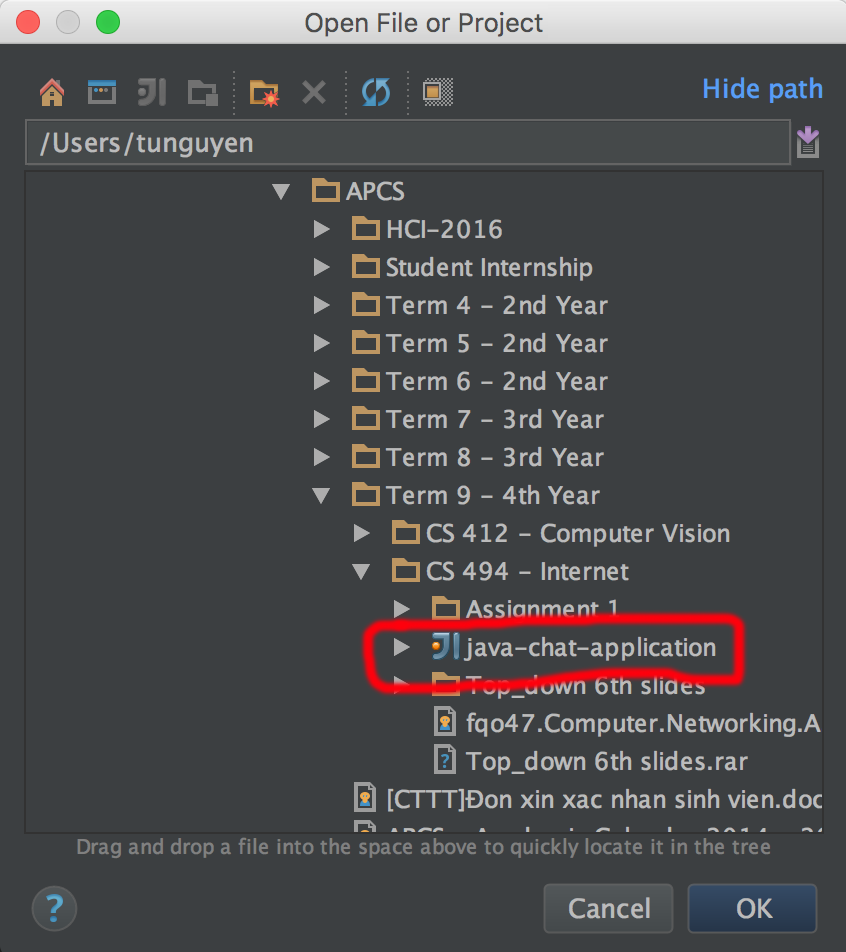
* Download and install the latest version of IntelliJ at:

<https://www.jetbrains.com/idea/>

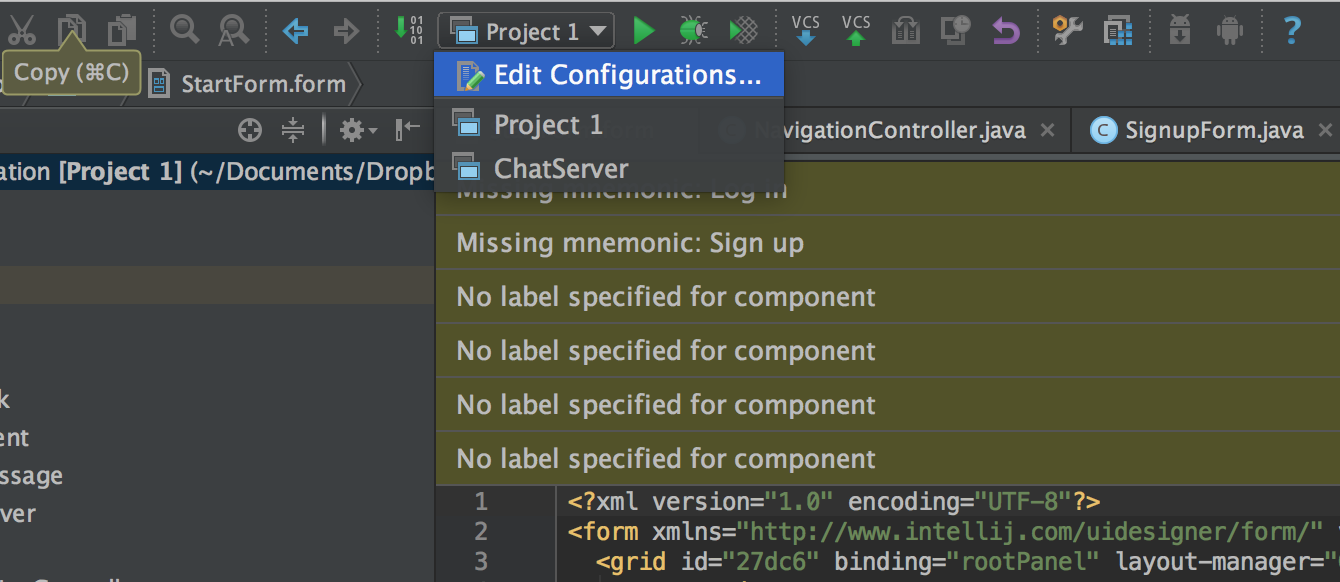
* Select “Open” on start screen:



* Select the project IntelliJ project folder



* Turn on toolbar if it’s off: View -> Toolbar.
* On the toolbar, select the project to build.



* “Project 1” is the client.
* “ChatServer” is the server.

# Project Structure & Algorithm

## a. ChatServer

This is the main class responsible for the server side.

* Extends Thread class. Create 1 instance responsible for each client connected
* When getting requests from clients:
  + Send/broadcast it to other clients
  + Return result (SUCCESS/DENY) to sender
* Request type:
  + MESSAGE: send request to other/ broadcast
  + SIGN IN: sign in with username and password
  + SIGN UP: sign up with username and password
  + SIGN OUT
  + NEW USER: one user just online
  + REMOVE USER: one user just offline
  + GET USERS: get list of all users
* The out writer of each client is stored in HashMap<String, PrintWriter> users
* The list of users with password is stored in HashMap<String, String> userPasswords

## b. ChatClient

This is the main class responsible for the client side

* It extends Thread class. Including 2 instance, itself as a sender (send request to the server), and a receiver (continuously reading response from the server)
* There are methods for each request type (MESSAGE, SIGN IN, SIGN UP, etc.)
* Method for sending request and getting response:
  + Whenever a request is sent, its callback function will be pushed in to a ConcurerentLinkedQueue called requestCallbacks (a type of queue that can be synchronized multithreaded)
  + Whenever a response is received, we poll (pop) the callback from the queue and call it

## c. Callback

This is a interface for Callback function, we can create it as a lambda expression.

For example, our method in ChatClient is signIn(username, password, callback) can be called like this:

Main.*client*.signIn(username, password, (obj) -> {  
 if (Main.*isSuccess*(obj)) {  
 Main.*username* = username;  
 goToChat();  
 } else {  
 JOptionPane.*showMessageDialog*(null, Main.*getContent*(obj), "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
});

## d. SignupForm, StartForm, ChatForm, NavigationController, UserCell, etc

Those other classes are used for GUI only. For example, NavigationController is used in push the frame when we open another window, or pop the frame when we close a window and jump back.

# Server – Client Communication Example

Example

* Step 1: Client connect to the server
* Step 2: Server create an instance responsible for that client
* Step 3: Client create a receiver instance, responsible for receiving message from Server, while itself responsible for sending message to the Server
* Step 4: Client send JSONString {“type”:”SIGN IN”, “from”: “bikrone”, “password”: “password” }, with the Callback of calling “System.out.println(“DONE”);”
* Step 5: Client push the callback into the callback queue (requestCallbacks)
* Step 5: Server receive the message, parse the JSONString to get username, password and type of request, return the SUCCESS object {“type”: “SUCCESS”}, while simultaneously add the username & its client output writer into correspond hashmap
* Step 6: The receiver instance of the client read the message from the Server, parse the String to get the JSONObject response
* Step 7: The receiver instance of the client pop the callback form the callback queue, call it with the argument of the response => The system print out “DONE”.
* Step 8: The client disconnect with the server
* Step 9: The server instance for the message break out, release the user from the map of client printwriter.