

Computer Science Internal Assessment
School Clubs and Activities Database

Carmel Kozlov
000007-034
Frankfurt International School
May 2007

Table of Contents

Part A

Investigation and analysis	
The problem (overview) ...	A1
Existing system	A1-A2
Sample data	A2-A3
Prototype and End User discussion	A4-A12
Goals (criteria for success)	A13-A14

Part B

Data Structures	B1-B4
Modular Organization	B5-B7
Hierarchical Classes and Algorithm Pseudocode	B8-B15
Mastery Factors	B16

Part C

Program Listing	C1-C55
Main Class	C1-C6
Teacher Class	C7-C13
Student Class	C14-C18
New Club Class	C19-C27
Search Class	C28-C41
Linklist Class	C42-C46
Admin Class	C47-C55
Usability	C56-C58
Handling Errors	C59-C72
Goals with reference numbers to documentation	C73

Part D

Documentation	D1-D23
Evaluating Solutions	D24-D26
User Documentation	D27-D33
Master Factors with Code Lines	D34-D35

Part A - School Clubs and Organization Database

Investigation and Analysis

The Problem (overview)

Our school has a lot of after school activities, service projects and entertainment clubs. It is a good that there are so many options and students have a variety of clubs to choose from. Nevertheless, I always sensed that there is a problem as students don't tend to find out what clubs they can join in a very organized manner. This discourages many students from joining clubs; as well all know that high school students are always busy and lazy. Many student doing things outside of the classroom, but because no one is supplying them with the information in an organized matter, many clubs stay unpopular since they are new, or because no one heard about them. If perhaps the students had a system in which they could search all possible clubs in the school and perhaps even sign up onto straight though that system, they might be more encouraged to join clubs.

Existing System

There are a lot of ways in which the school informs the students of the clubs there are. But this is exactly the problem – there are a lot of different places in which information about activities are posted, but there is not one specific place in which all this information is presented.. The information never seems to get to people who need it, and a lot of people will not take care of papers that are posted around the school about a new club.

For example, once a trimester a PDF file is uploaded onto the Parent portal where parents can look at some (but not all, and in fact a very small amount) of the after school activities available. We can see a fundamental problem here – the PDF file is uploaded onto the PARENT portal, not the student portal, since there isn't something like that (only on the school servers, which students can not access from home). Indeed, there are students who do check the parent portal and the Tuesday folder in the parent portal (where the PDF file is uploaded onto), but usually in higher grades. The smaller students usually do not bother checking the parent portal. Since this PDF file is uploaded onto the parent portal, not a lot of students get to see this. Not all parents check the parent portal, and therefore not all parents can inform their children about the after school activities – therefore there is a communication problem.

Another way students hear about clubs and activities is through the daily bulletin. Every day in the morning there are announcements, and sometimes clubs like the Kalahari project will post an announcement saying that students can join. This is done frequently. New clubs that start in the beginning of the year will randomly post announcements on the bulletin. This is usually confusing for most students, and they don't seem to take great care for this. Other ways clubs usually get students to hear about them is post papers around the school campus. But this way is not useful. It causes problems as student's don't notice the posted information on the walls of the school, and do not take great notice to the bulletin, and most students definitely do not check the parent portal.

If I could create a program through which students could look up all the clubs organization in the school, which will provide them with all the necessary information they need (like times in which the meetings take place, what grades can join and so on), it will encourage students to look through the program and perhaps even help them join some clubs. But most importantly, it will provide all the clubs' most important information in just a few clicks away. Today, our school already has a state-of-the-art technology facilities, and students seem to take full advantage of these facilities. Therefore, they wouldn't mind using a program to find information about clubs. This therefore is a simple and practical solution.

Sample Data



This is a screen print of the parent portal, where we can see two clubs being advertised. The first is after school drama and dance lessons, and the second is an announcement for times for the Upper School Choirs (highlighted in black). This could easily be put into the program.

The following is a flyer that was posted around the school in order to advertise this club.

Quantity of Data

The system will consist of one major data type

- club index listing
 - o This will be a list of all the clubs. It will be stored in a random access file and it will contain all the information on the club.

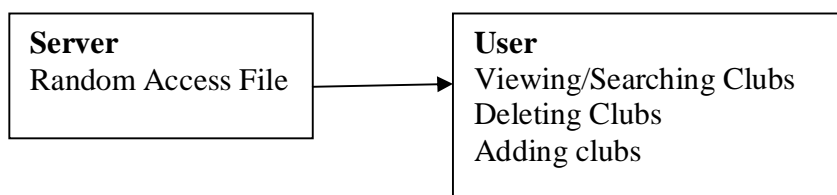
Initial Ideas from the intended User

After an initial talk with the intended user, a few points were brought up

- teachers should be able to create new clubs
- students should be able to search for clubs easily and efficiently
- teachers should be able to search for clubs as well
- The interface should be simple, but effective.
- The clubs information will be saved on random access file, which will be stored on the server, so that the program will be useable on different computers.

Initial System Design

The initial system design incorporates the ideas from the intended user.

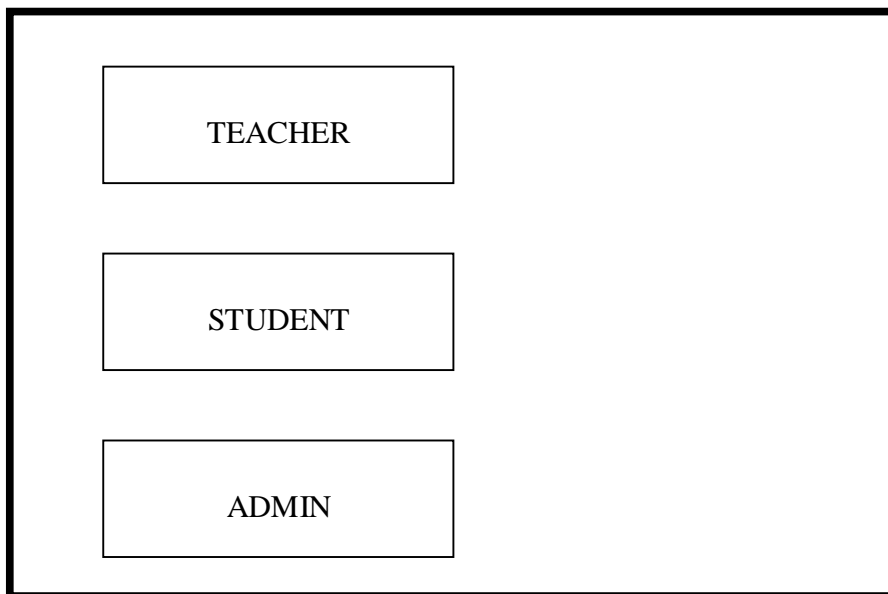


Feasibility Prototype

After the initial ideas the intended user and I came up with, I had a better idea of how I was going to do the prototype. The design should be simple and effective and therefore shouldn't have any unnecessary buttons.

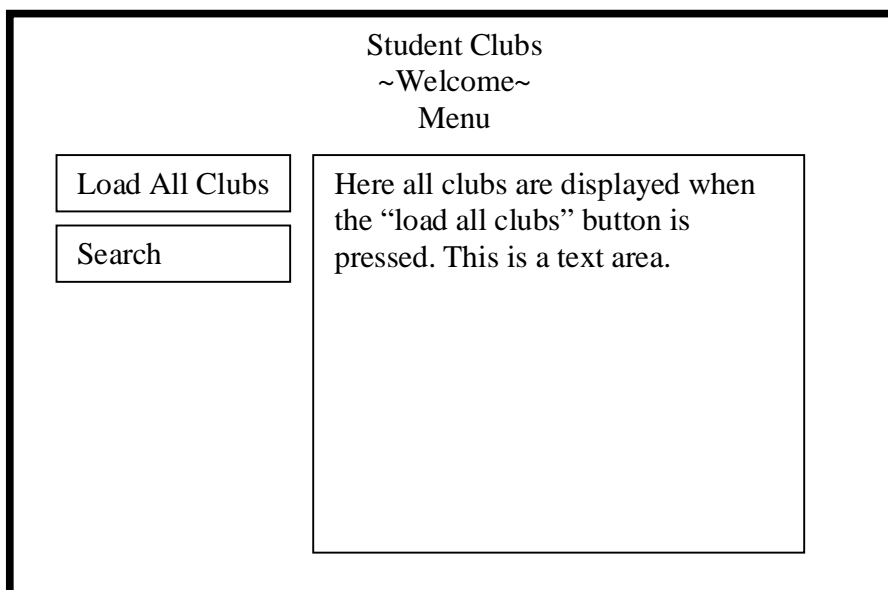
The initial idea was as follows:

The Splash Screen:



When a user presses "Teacher" an input for password and name will appear, and only when the password is corresponding with the right name they will be able to proceed to the teacher screen. Students can access it without a password, nevertheless, they will not have the amount of rights teachers and administrator has. The admin will only have one password and should be appointed to a teacher that will take responsibility over the managing of the program.

When a student logs in, they will see a screen as follows:



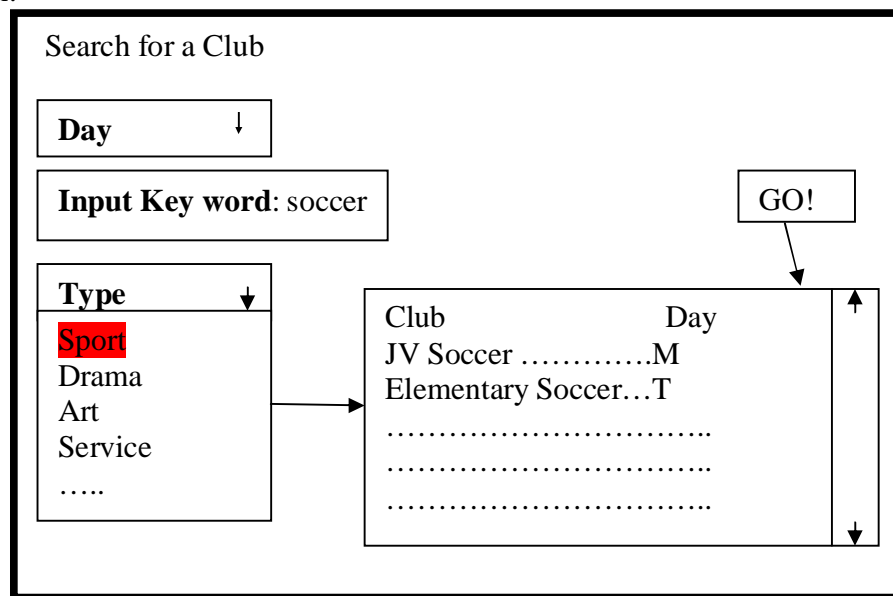
With the “load” button all clubs will be displayed on the screen. Pressing Search will upload a search option in which the students and/or teachers will be able to search for specific clubs.

The clubs database is saved on the server in the form of a random access file.

There were also ideas for different searching methods for the clubs. These were presented to the intended user. After a brief discussion, the intended user suggested there should be two ways of searching for clubs:

- by types of clubs
- by a word search
- by a day search

While discussing this with the intended user, the following ideas came up for the first suggestion:



This interface provides with the user with an easy and simple search option.

As we can see in the picture an option is to press the “type” option list, which will reveal a list full of the different types of activities. When for example the option “Sport” will be pressed, the list will show all sport clubs available, along with the rest of the information. Days will be shown with their letter, M = Monday, T = Tuesday, W = Wednesday, Th = Thursday, F = Friday, S = Saturday. This was discussed with the intended user, and was done in order to save space. Nevertheless, when a specific club from the list is pressed, a pop-up appears with further detail of the club.

The other option would be a word search. The word search will look for the key word, and if it can find a club with that key word in its description, it will show the club on the list.

Further Discussion with the user

Further discussions were taken place with the intended user, in which we came up with more options and ideas for the program.

- **User:** since the teachers will be able to add clubs, could a notification appear for every time a student opens the program to notify of new clubs?
Me: I don't think that would be possible. Besides, how do we know if the next person who will use the program on the computer will have already seen the notification or not? I would have to create user-accounts and that would serve no real purpose.

- **User:** there is no button on every screen that will allow the user to go back to the menu at any given point, can we have one?
Me: Yes, I could do that

- **User:** in the example you showed me, the list wasn't alphabetically ordered. Whenever a user will look at a club list, can it be alphabetically ordered?
Me: yes.

- **User:** what if a teacher doesn't want to log in as a teacher? Will it matter if they press the student button?
Me: not at all. The student login doesn't require a password, and is can be used just as browsing. If a teacher wants to log in and create a new club, they will have to get the password. The functions they will have as a teacher is the same as the students have, except they can also add clubs. Students on the other hand should not be able to log in as teachers.

- **User:** can we have teachers delete their clubs?
Me: Yes we can

- **User:** what will happen if a user will search using the keyword function, but the Computer will not find anything that corresponds?
Me: nothing will be displayed..

- **User:** will the teachers be allowed to write how much they want to in the description on the club?
Me: no, since I am using random access files. I could make an error message appear if they wrote too much in the description part.

- **User:** can we have an administrator?
Me: yes, we can
User: I would like the administrator to manage teacher account. But I do not want the administrators to delete specific clubs, only the whole file, so that mistakes are not made and only teachers that have created a club can do so.
Me: that is possible.

User stories

The boxes below summarize user stories based on the investigation above.

<Angle bracket> indicate significant user actions

Asterisk indicate automated computer processes.

(Round parentheses) indicate data-storage (file).

Stories

Task: Search for club

Interface: Search Menu

Input: Key Word

Output: computer *searches* for the key words in the (file), then *displays* the right clubs on the screen

Actions: user <scrolls> up and down, looks for the club that most interests them

Task: Add Club

Interface: Add club menu

Input: teacher <fills out> necessary information, then <saves> it

Automation: The computer *saves* the information at the end of the (file).

Task: Contact Club

Interface: Club description

Input: student <presses> the “Contact” button

Output: computer *opens* the “contact” interface

Input: student <inputs> the information into the contact form

Automation: The computer *generates* an automatic email and *sends* it to the necessary teacher.

Task: Log In

Interface: Splash screen

Input: user <clicks> the correct button

Output: if student is pressed, the main menu is *loaded*, otherwise, the password filed for the teacher is *loaded

Input: teacher <inputs> password

Automation: computer *checks* if password is correct (if it corresponds with the file in the server), if it is, *loads* the main menu screen, if not, it *re-loads* the password screen

Task: Load All Clubs

Interface: main menu

Input: user <clicks> the “load all clubs” button

Automation: computer *loads* all clubs from (file) and *displays* them in an alphabetical order.

Task: Delete Club

Interface: main menu, teacher login

Input: teacher <searches> for their specific club.

Output: computer *finds* the right random access (file) and *loads* it.

Input: teacher <clicks> the “delete” button

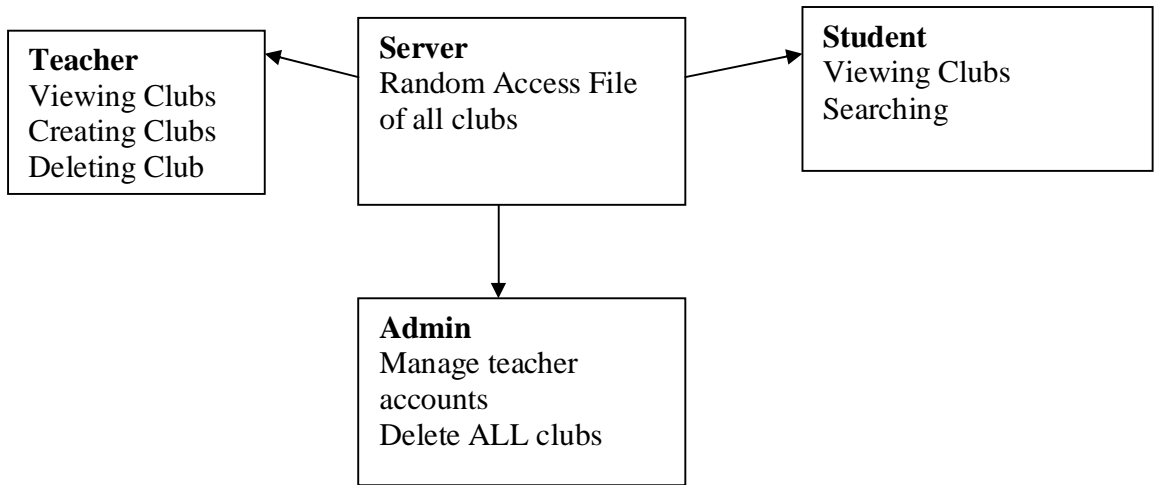
Output: computer <verifies> that the teacher is sure he wants to delete the club

Input: teacher <clicks> on yes

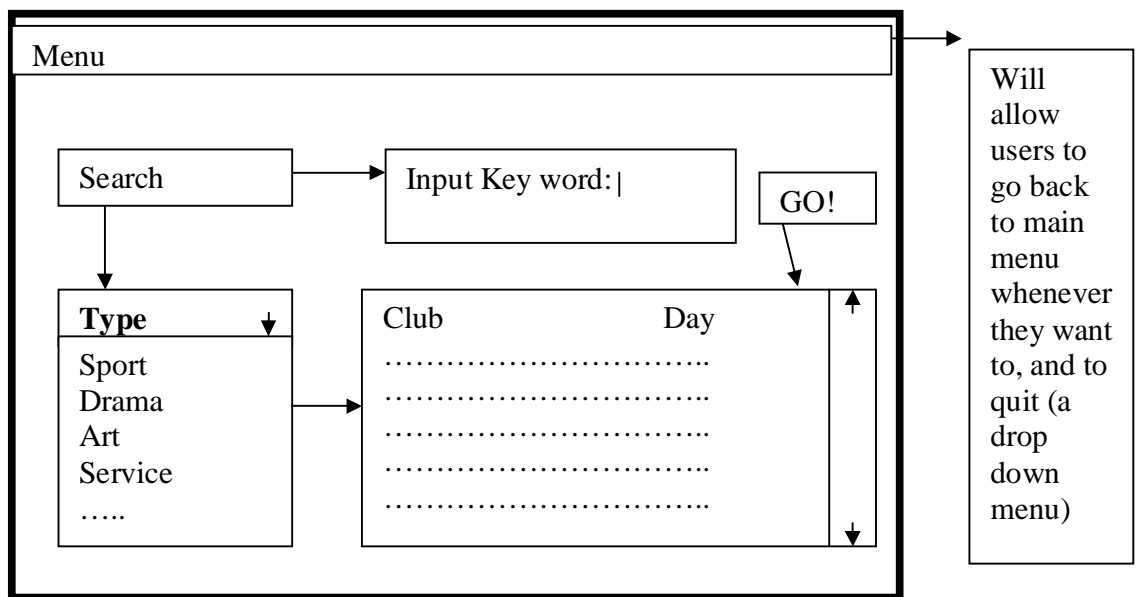
Automation: computer *deletes* the random access (file).

Revisions

After the discussions with the intended user, I came up with a few changes and additions.

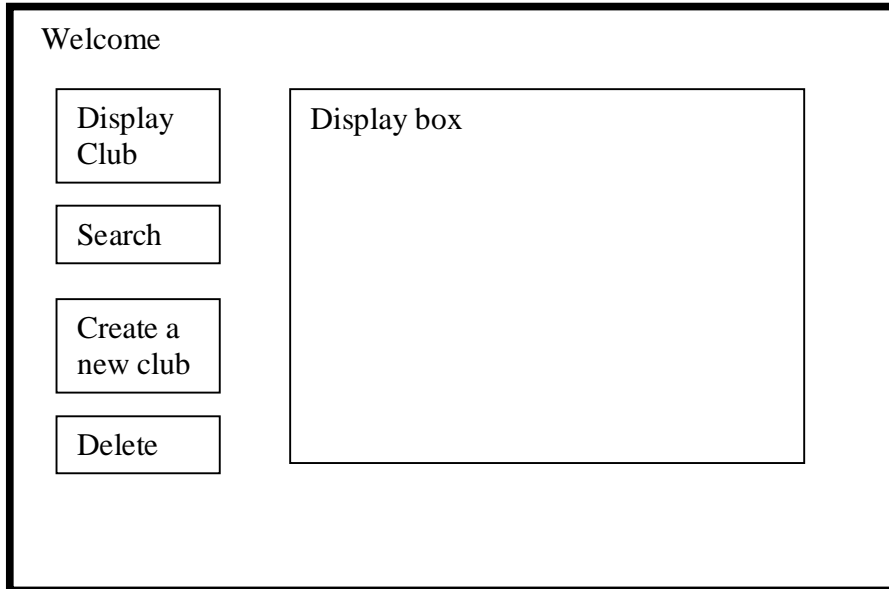


The following screen is the search screen, both for the students and the teacher. It is the revised version:



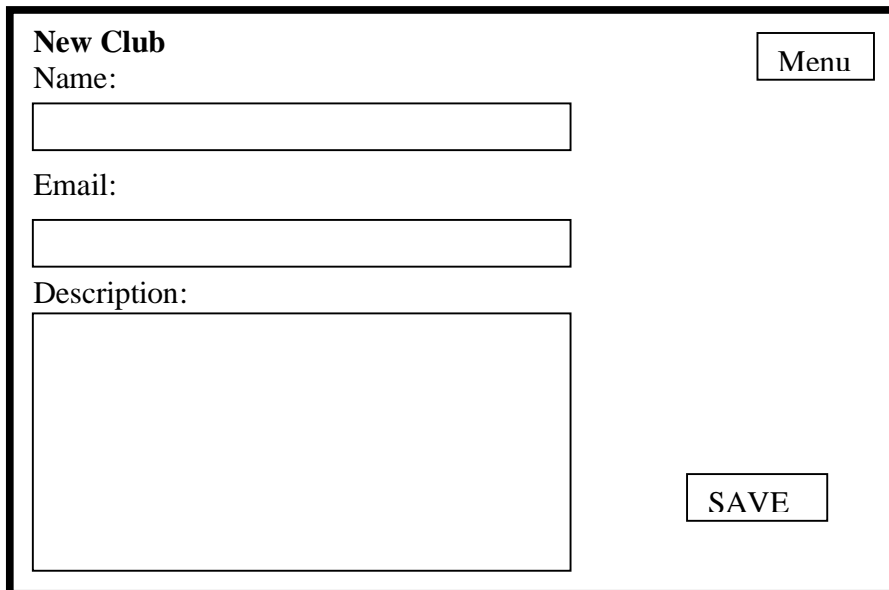
The teacher will have an interface of their own. Same with the administrators. This can be shown below.

This following screen is what the interface of the teacher will look like, more or less.



A screenshot of a web interface for a teacher. At the top left, it says "Welcome". Below this, there are four buttons stacked vertically: "Display Club", "Search", "Create a new club", and "Delete". To the right of these buttons is a large rectangular area labeled "Display box".

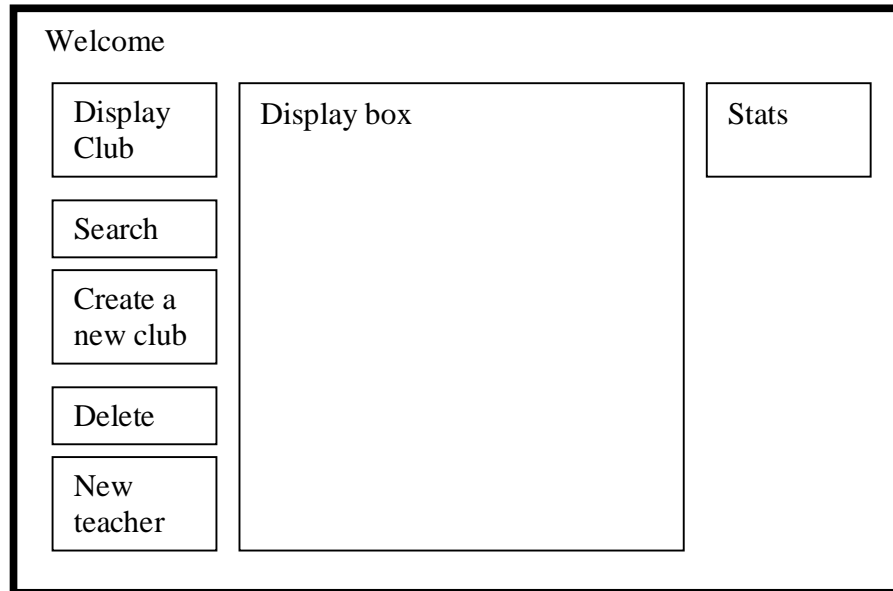
Pressing the create a new club button will pop up a new window which will look something like this:



A screenshot of a "New Club" form. The title "New Club" is at the top left. In the top right corner, there is a "Menu" button. Below the title, there are three input fields: "Name:" with a single-line text box, "Email:" with a single-line text box, and "Description:" with a larger multi-line text box. In the bottom right corner of the form, there is a "SAVE" button.

The empty boxes will be used in order for the user to input the correct information. Pressing the save button will save the information on the random access file.

The admin will have a lot more functions than the average user, and the admin screen will look something like this:



Goals

A condense and effective database of all clubs and organizations in our school

The effective use of storage in order to make sure all the clubs and organizations in our school will be stored in the file, in a condense format.

Making sure all clubs and organizations are actually listed in the program will be difficult, as not all teachers prefer to work with computers. The transition between the current messy system, to the condense database on the server might take a while.

Effective access and use of program in order to find out information about clubs and organizations

A simplistic use of GUI, that is appropriate to the use of students and teachers in that it doesn't distract the user. The user should be able to use the program and find the necessary information in just a few clicks.

The program can only work on the school intranet, as the files are stored on a local school server. Students will not have to search in the bulletin, parent portal or around the school in order to find the information they are looking for. It will all be stored on the files, and will only need to enter a keyword in order to search for something, the computer will do everything else automatically (the searching algorithm).

Useful search options, which will allow users to find clubs uncomplicatedly

The users will be able to either view clubs in their different categories, or search using a key word, or even search regarding to the day in which the club takes place.

When a key word is put in, the program has to search through each random access file in order to find clubs with the given key word. In addition, the keyword that is used by the user might not even be anywhere in the files, even though what they are looking for does exist. There might be some clubs that do not really belong to any specific category and they will be put in the "others" category. This might hinder the student from finding what they are looking for.

Teachers will be able to add their own new clubs

Teachers who start their own clubs will be able to sign into the program, providing they ask for the password, and create their own club entry.

A problem this might cause is the fact that it is open to all teachers, and doesn't limit the amount of clubs entries they will be able to put in. This might cause some teachers to perhaps create double entries, or even spam the program. There will be nothing to stop them from doing this, as the time limitation I am under doesn't allow me to create some sort of filter. In addition, the amount of information the teachers will be able to put into the description of their file will be limited, as this is a random access file, and is chronically organized. A counter will show the teacher how much space they have left to write in the description of their club.

Teachers are able to delete their club

If a club isn't operating anymore for various reasons (it's only during first half of the year, or it's closed down) the teacher might choose to delete the entry so that students do

not get misinformed. The teacher accounts will allow each teacher to delete only clubs they have created.

Administration of the program

The administration of the program is important. There will be the option to delete all the clubs, and to add new teachers to the program.

This is a beneficial idea.

Other Limitation

Besides the specific limitations already mentioned, the following limitations must be kept in mind.

There is a limited amount of equipment on which the program can be installed. In addition, programs will rely on the fact that Java is installed on the computer.. In addition, computers should be connected to the local server in order to have access to the random access files. Teachers will be able to use the program on their computer only if it is installed on it. Programs will only be able to run in the school environment, and therefore home use of the program is not available. Users should be aware of the fact that sometimes the server fails and the program will not be able to access the random access files.

Students do not have the same access as teachers do to the program, and can only search for clubs. Teachers have to input their name and password if they want to delete or create clubs.

Finally, it will be an effort of ALL SIDES in order to actually use the program successfully. That is, if teachers will not bother to put in their clubs into the program, the whole point of the program will be lost. And if students do want to find out about clubs, they WILL have to actually search for them and not be lazy about it. Therefore, while the program might end up being very good, it is up to the school community to take the advantage of the program and decide to switch from the annoying bulletin, parent portal and wall advertisements to a simple yet effective local intranet program.

Part B – Student Club and Organizations Database

Data Structures

My program will have three data structures:

- Text File
- Random Access File
- Linked List

Text File

This will be a normal text file in which I will store my teacher accounts. The teacher accounts consist of their name and passwords. The addition of teacher accounts will only be able to be done by the administrator of the program. This will be stored on a server meaning every teacher can log in whenever they want to from whatever machine as long as the program is installed and they have an account. Text files are easy to use, nevertheless, at the same time, they are not very useful. As they are sequential and for big amounts of data are pretty much useless. While I could have used a random access file for this, I decided, for the sake of simplicity, to go with a text file, as the amount of data I am dealing here with is very small. This does mean though that the text file could be abused really easily by any user, which is not a good thing. Nevertheless, since I already had a random access file, I decided to go with a text file.

This is an example of a part of my text file.

```
George Taylor
wackwackduck
Lauren Sinta
ger2448tth
Daniel Radcliffe
quidditch2
...
...
```

The first two lines à George Taylor, wackwackduck are one teacher profile. The teacher George Taylor has the password wackwackduck.

It goes on, repeating name and password, name and password.

There is nothing special about the text file, as it is easy to use it, add and remove lines from it and to read it. When the teacher logs-in, they will have to enter their name and password. If they do not have a teacher account, or if they have typed in their name or password incorrectly, they will not be able to sign in as the program will not be able to verify what they have entered with the existing teacher profiles inside of the text file.

The program checks the name of the teacher and checks in the text file. If it finds a corresponding name, it will read the next line and then see if the password the teacher has entered corresponds with the password in the text file. The administrator will be the one adding the teacher profile to the text file. No one else will have access to it.

Random Access File

The random access file will be used to store the clubs data.

Each record will consist of 300 bytes, with the following breakdown:

Name of club – 50 bytes

Day – 10 bytes

Organizer – 30 bytes

Email – 60 bytes

Grade – 10 bytes

Location – 20 bytes

Type - 20 bytes

Description – 100 bytes

All fields are stored at UTF strings.

Yoga M Roger Smith Roger.smit@fis.edu 6-12 Gym Sport Yoga for beginners.	Record 1
Art Th Yaniv Levi Levi@fis.edu 9-12 Art Studio Art for beginners	Record 2

The random access file will be stored on the server, meaning all the users can use it (the programs will access it). When a teacher updates a club, since there is only one random access file which sits on the server, all of the programs (that are installed on different computers) will use it, and therefore will be affected.

Random access made the most sense in this case. I do not see what other data type I could have used. Text files are not good as I am dealing with quite a bit of information here. The difficult thing about random access file is not the adding, as that will be done at the end of the file at every case, but the deleting of data from the file. As I need to keep track of the records. Some records might not be exactly 300 bytes, and some might exceed (in which case I need to make sure they do not otherwise my whole system is ruined and there will be an error). While deleting from the random access file I need to make sure I move a record up to replace the empty place I have created.

Link List

The link list will be used for searches. That is, when a search is being made, the data will be stuck inside of a linked list and only then will be displayed on the screen. I've done this in this way since I thought it was a good idea. Not only does it allow me to do more things with the results, but it means that if in the future I would want to somehow add more functions to the search algorithms I will not have to change a lot of code. The link list is my ADT.

Link lists are also not very hard to code. Keeping track of the head and the tail is important, but they allow a lot of flexibility, considering the fact that they are not permanent.

Link lists work on the idea that there is a node, with data in it, which points to a new node. This allows me to add in data whenever I want to, in whatever order – since it is not sequential, but random access. This is useful for programming a search, as can be seen in my examples. If someone decides to make a search I would just display it, I would display a result, and then display another one. This way, I have a link list and all I need to do is display it. This is good in case I have more algorithms that my link list might go through in the future. As I mentioned, if I wouldn't have used a link list I would have not user a data type at all for my searches, since I couldn't think of any other data type to fit this problem.

The link list usually looks like this:

Node 1: Pointers: First Node, Last Node

If added:

Node 1: Pointers: First node, next node

Node 2: Pointers: Last node

If added:

Node 1: Pointers: First node, next node

Node 2: Pointers: next node

Node 3: Pointers: Last node

Etc...

Nevertheless, the nodes include data, and therefore an example of this might be:

Node 1: First node, String name = Art, String organizer = Smith Robert, String location = art studio, String grade = 6, String day = M, String description = art for 6th grade, next node
Node 2: String name = Indoor Soccer, String organizer = Paul Thomson, String location = gym, String grade = 10-12, String day = S, String description = indoor soccer for beginners, next node
Node 3: String name = Yoga, String organizer = Ohm Levi, String location = foyer, String grade = adults, String day = T, String description = yoga for adults.
Last node.

Adding to a link list is not that hard. I just need to take the last pointed and point it to my new node. Nevertheless the tricky part about this is that I always need to make sure my tail pointer is actually pointing as my tail. Keeping track of the head pointer is easy, as it rarely changes, and if it does, all I need to do is make sure the head pointer points to the next node after the current head node, which is only 2 or 3 code lines.

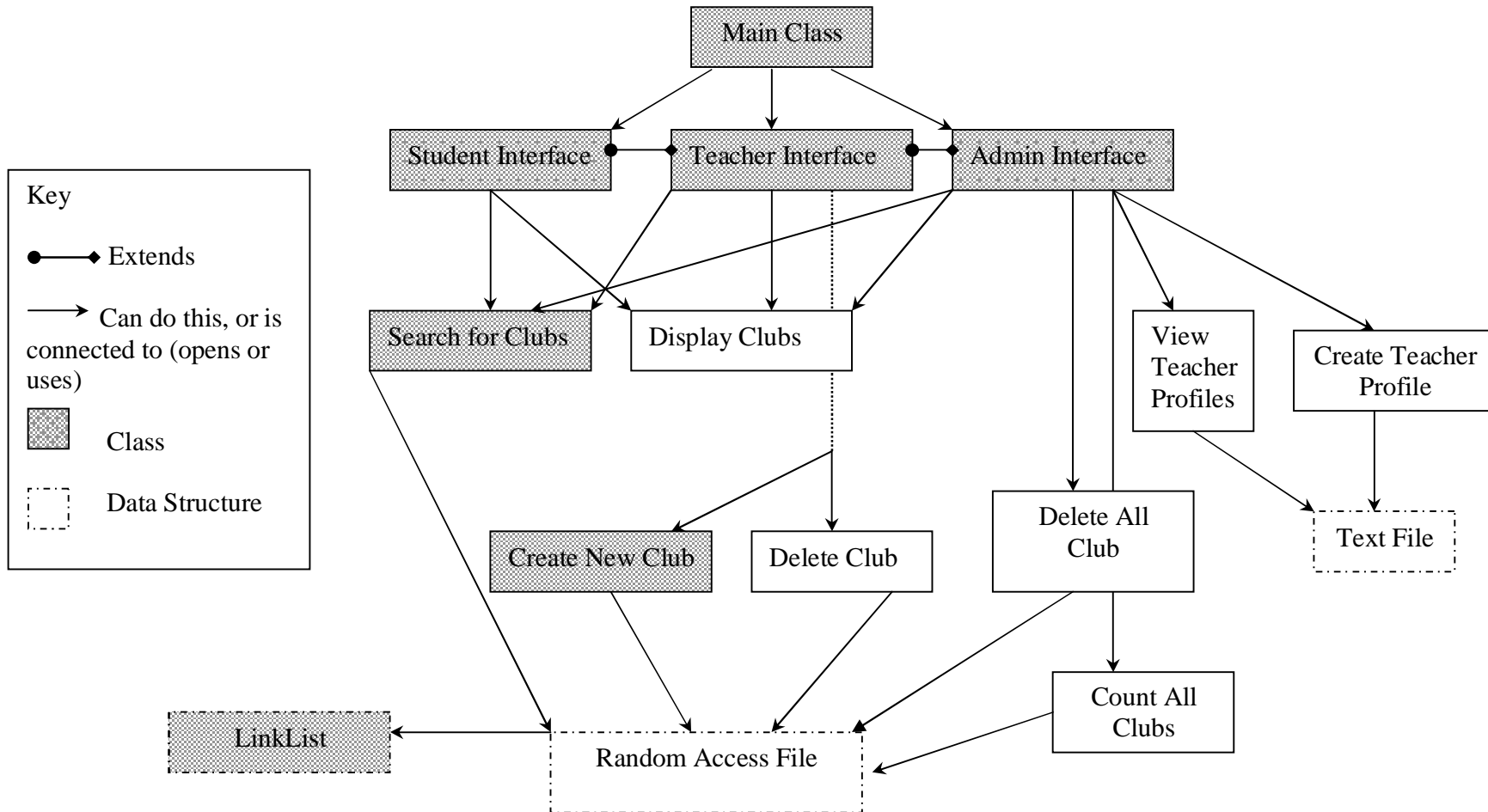
Deleting from a link list is also quite tricky as I have to make sure I'm keeping track of the correct nodes. The good thing about this though is that once I actually know where my pointers are pointing it, all I need to do is change the pointers and whatever I did not need it deleted.

Modular Organization

On the following page is the chart for my modular organization. Notice how it begins with the classes and ends with their primal functions. The boxes marked as grey are the classes. The chart outlines how the classes are connected to each other, and which classes are extending others. All of the methods are obviously not outlined; nevertheless, the important functions such as deleting clubs or adding them are a part of the modular organization and are therefore included. Data structures are also outlined.

When looking at the modular organization chart, there is a key that might be of use to fully understand the chart.

Modular Organization



Classes

My classes consist of:

Student, teacher, admin, search, new club and link list.

My admin class extends my teacher class which extends my student class. This makes a lot of sense if you think about it. The admin has all the rights the teacher does, nevertheless the teacher has fewer rights than the admin and the student has the least rights out of the three. This is reflected by how they extend each other, meaning that the admin class inherits all of the student and teacher objects, but I can add things to it that the other two classes cannot do, which is very appropriate to my program.

My new club class was done separately because it consists of lots of objects and also has to access the random access file and therefore I thought it would be best to create its own separate class.

My search class is done as a class so that my three types of user's à admin, teachers and students can all access the same class as the search functions do not defer for them.

My link list is done as its own separate class as it is an abstract data type.

Data Types

The three data types are outlined in my modular organization. It is now possible to see what parts of my program access and use each of the different data type.

Notice how my random access file is connected to the link list. This is because only by a search it is possible to get a linked list in my program. That is, when a search is being done, it will access the random access file through which the link list is created.

As outlined, the text file can only be accessed by the admin class.

Methods

It is not possible to outline every single method in my program; nevertheless, the methods outlined in my modular organization are the major methods that create the major parts of my program. For example, my delete methods and my display methods. The display and delete method consist of an important part of my program that actually affect what the user sees and how the user interacts with the program. My display method is not in any way connected to my link list method as the display method is not the display method for the search. The link list class has a display method which displays it separately from the normal display method, which can not be used in my search class.

Hierarchical Classes and Algorithm Pseudocode

Main Class (extends EasyApp)

This is the main class. Will allow the user to enter into the splash screen, where the options for student, teacher or admin is.

Algorithms that will be used here:

Sign(String valid)

This method will validate the master password (which doesn't change and is programmed into the program)

Before: splash screen is present

After: password is validated for teacher, creates new teacher class and disposes the splash screen

Parameters: password entered

Pseudocode

Compares the valid string (parameter) to the "master password"

If it is the same, creates new teacher class

If it is not the same, gives error to user

signT(String name, String password)

The signT(teacher) method is the signing in method for the teachers. A teacher can only sign in if they have

An account. This can be created in the Admin section of the program.

When a teacher signs in they have to put in their password and they have to put in their name.

This method checks whether or not their name/password combination is correct.

Before: teacher hasn't signed in

After: teacher has been signed in, or has been denied access to the programmed

Parameters: name and password

Pseudocode:

Open the file.

Loop through the file, look for name of teacher

If found, compare the two passwords

If they are the same, sign the teacher if

If they are incorrect, do nothing.

Student Class (extends EasyApp)

The student class is a simple class on which the admin class and teacher class are relied upon.

Display()

This method is a method to display the clubs from the random access file.
It will display the clubs onto an Area Field

Before: the area field is empty

After: the area field has all the clubs displayed

Pseudocode:

Open random access file

Loop through file, seeking to all of the names of the clubs

Read the name of clubs

Displays the name of club

Go to next line, display next name of club

Close random access file

Teacher Class (extends User)

Teachers can delete and create clubs.

Delete(String name,String organizer)

This method will allow the user to delete a club. For this, they will need to enter the name of the club
exactly like it appears in the random access file.

parameter: name of club

before: the club is still in the random access file, there is a random access file

after: the club is erased, the random access file exists.

Loop through file

read the name of each record

if its the right name

move all subsequent records up one position

 go to position x(record) + 1

 read the stuff from there

 copy to position x - 1

 repeat until there is nothing to copy anymore

if club is at last position

delete the club at last position

Admin Class (extends Teacher)

Administrators have a lot more rights than students and teacher.

DisplayTeacher()

The display Teachers method will be used by the admin to find the passwords of all the teachers.

This could be used if the teacher has forgotten their password.

Before: there must be a text file with teachers and passwords

After: the passwords and the teachers are displayed.

Pseudocode:

read the file

read name of teacher

read password

display them

go on until file is done

deleteAll()

The delete all method will allow the administrator to delete all of the content of the random

access file in one click. This is in case the year is over and there is definitely not going to be any use for the current list anymore.

Before: the random access file has content in it

After: the random access file has no content in it

Pseudocode:

Open up random access file

Set length to zero

CountAllClubs()

This method is a very simple method to count all the clubs in the random access file.

The administrator might use it in order to quickly check how many clubs were added since the last time

he checked the program.

Pseudocode:

Open random access file

Count all the clubs

Display the count

CheckDuplicate(String name, String password)

This method checks whether or not the administrator is trying to add in a teacher profile that already exists.

Before: teacher has not been checked for duplicated

After: teacher has been checked for duplicated

Parameters: name, password

Pseudocode:

Open up file

Compare all the teacher names to the name in the parameter

If it is the same

 Do not let them add it

If it is different

 Execute the add nprofile method

Nprofil(String name, String password)

This method writes in a new profile for the teachers.

Before: there is no new teacher profile

After: there is a new teacher profile

Parameters: Name of profile and password

Pseudocode:

Open file

Write in the name, password

Close file

New Club Class

The Clubs will have their own Class.

This class will include the following:

String name – name of club

String time – **String** for the time and date of the club

String organizer – **String** for the name of the organizer of the club

String email – **String** for the email of the club organizer

String grade – **String** for the grade and class of the potential students

String location – **String** for the location of the club

String description – **String** for the description of the club

String type – **String** for the type of club

All of this will be stored in the random access file. Each **String** will be a field in the whole record. Each club will be one record.

validate(String name, String day, String organizer, String email, String grade, String location, String type, String description)

I need to make a method in order to validate the email. I'm doing this because it is easy to make

a mistake when it comes to typing in the day. The day has to be in a special format (in this case it has to be a letter/number to represent the day it is on

Before: the user puts in the day of the club

After: the program checks whether the day is correct

Parameters: day of clubs

Pseudocode:

user inputs the day of clubs in the form of a string

program checks if the length of the string is smaller than 1

if it is smaller than one, it tells user to try again

deletes the last node

else, the program checks if the length is bigger than one

if it is bigger than one, it tells user to try again

deletes the last nodes

addNewClub(String name, String day, String organizer, String email, String grade, String location, String type, String description)

This creates a new club

There are around 100-200 clubs in the school.

Parameters:

Each club has the following fields in their record:

Name – 50 bytes

Day – 10 bytes

Organizer – 30 bytes

Email – 60 bytes

Grade – 10 bytes

Location – 20 bytes

Type - 20 bytes

Description – 100 bytes

Therefore, one club will have a total of 300 bytes. If there are around 100-200 clubs, the random access file will need to be around 6000 bytes. $+ 200 * 2 = 400$
 $6000 + 400 = 6400$

Before: There is a random access file, main program has decided that this is not a duplicate

After: There is a random access file with a new club

Pseudocode

seek to the end of file

write in data - add new club

LinkList Class

Data structures:

Link list: in order to search through all clubs, a link list of all the clubs will be made. This will allow an easy search for the search. The nodes will include name of the club and their location in the file (so that when a club is pressed the program will seek to that specific position in order to retrieve all the information and display it on the screen).

add(String Lclub,String Lday,String Lorganizer,String Lemail,String Lgrade,String Llocation,String Ltype,String Ldescription)

The add method adds another node to the linklist

Parameters: String Lclub

String Lday

String Lorganizer

String Lemail

String Lgrade

String Llocation

String Ltype

String Ldescription

Pseudocode:

if the head is equals to null, add at beginning

else, add at end.

removeHead()

The remove head method removes the head from the linklist

Pseudocode:

if the head is equals to null, print error message

else, head = head.next.

removeTail()

This method reomves the tail from the linklist.

If the head is equal to null,

give error message

else if head.next is equal to null, both head and tail are null.

else, if head.next.next = null, temp.next = null.

displayNextNode()

This displayNextNode method displays the next node.

Pseudocode:

if the node is equals to null, it is the head

otherwise, node is equal to node.next.

return the node.

Search Class

Display(LinkList nameList)

This display method is used to display the nodes from the link list.

Before: there is a link list

After: linklist is displayed

Parameters: linklist

Pseudocode

as long as the next node is not equals to zero,
get the node, display it
go on to next node

search(String keyword, int amount)

The search method for a keyword is a bit more complicated.

Since it also comes with an int.

Before: there must be a random access file

After: the correct clubs are displayed

Parameters: int for the amount of maximum results, and the string for the keyword

Pseudocode:

if amount == 0,
search for all the possible results
seek to each record
make one big string out of it
check is the keyword matches any part of it
if this is true, stick it in the link list
else
seek to each record
make one big string out of it
check is the keyword matches any part of it
if this is true, stick it in the link list
only display up to the given maximum (int).

searchType(String type)

This method will be used in order to search for a club through the use of the type based search.

Before: There is a random access file with data in it

After: the user has searched and all clubs are displayed

Parameter: type

Pseudocode:

Opens the random access file

Searches through file and compares the type given in the parameter with the type of each record
If they are equals
Displays that club in the area field

Search(String Day)

This method will be used in order to search for a club through the use of the day based search.

Before: The user hasn't searched for anything

After: the user has searched and all clubs are displayed

Parameter: day

Pseudocode:

Opens the random access file

Searches through file and compares the day with anything written in the file

If it finds something that looks the same

Displays that club in the area field

Mastery Factors

Adding to Random Access File	My random access file, "clubs.dat", has a method to add clubs to it in the New Club class, called addNewClub().
Deleting from Random Access File	Teachers can delete clubs from the file using the delete() method in the teacher class.
Searching in Random Access File	Searching in the random access file is done in the Search file, and can be done with three methods, search(String day), search(String word, int amount) and searchType(String type).
Recursion	None
Merging two sorted data-structures	None
Polymorphism	In my search class, I have two search methods, search(String day) and search(String word, int amount).
Inheritance	My admin class extends my teacher class which extends my student class.
Encapsulation	In my linklist method, there are private nodes that can only be accessed using get and set.
Parsing a text file	None
Hierarchical composite data structure	I've got linked list, in which each node has more than one data record in it. To be more precise, each node in my linked list has 6 strings in it.
Five SL mastery factors	Searching in both random access file and normal text file, use of additional libraries, user-defined objects, simple selection, loops, nested loops, user defined methods with parameters.
ADT # 1 – Add and Retrieve data	This will be in my Link List class, displayNextNode() and add().
ADT # 2 – Handeling Ends	Is also in my LinkList class, can be seen in removeTail()
ADT # 3 – Many error-handeling	Spread throughout my LinkList class, in nearly all of the methods.
ADT # 4 – All methods, Robust	None

Part C – Student Clubs and Organizations Database

Program Listing

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.io.*;
4  import java.util.*;
5
6  /**
7
8  ~~~~~
9  ~~~~~
10 Student Clubs and Organization Database
11 Carmel Kozlov, March 2007, Frankfurt International School
12 Microsoft PC, SCEditor, the purpose of this program is to organize and create a good
database for the school clubs and organizations
13 ~~~~~
14 ~~~~~
15
16 =====
17 =====
18
19 MAIN CLASS extends EASYAPP
20
21 =====
22 =====
23 ***/
24     public class Main extends EasyApp
25     {
```

```

26     public static void main(String[] args)
27     { new Main(); }
28
29     Menu mMenu = addMenu("Help|Quit");
30
31     Label bMain = addLabel("Main Menu",200,50,100,50,this);
32
33     Button bTeacher = addButton("Teacher",50,100,100,50,this);
34     Button bStudent = addButton("Student",50,150,100,50,this);
35     Button bAdmin = addButton("Admin",50,200,100,50,this);
36
37     public Main()
38     {
39         setBackground(new Color(11,118,53));
40         bMain.setBackground(new Color(11,118,53));
41         bMain.setFont(new Font("Arial",1,16));
42     }
43     public void actions(Object source,String command)
44     {
45
46         if(command.equals("Help|Quit"))
47         {
48             System.exit(0);
49         }
50         if(source == bTeacher)
51         {
52             try{ //error handling
53                 String name = input("NAME: (Firstname Lastname)");
54                 String mpassword = input("PASSWORD");
55                 if(name.length() < 1)
56                     { output("No name typed, try again");}

```

```

57         else if(mpassword.length() < 1)
58             {}
59         else
60             {
61                 signT(name,mpassword);
62             }}
63         catch(Exception ex)
64             { output("Login failed"); }
65     }
66     if(source == bStudent)
67     {
68         this.dispose();
69         new Student();
70     }
71     if(source == bAdmin)
72     {
73         String mpassword = input("PASSWORD");
74         sign(mpassword);
75     }
76
77
78     }
79
80
81     public void sign(String valid)
82     /****
83     This method will validate the master password (which doesn't change and is
programmed into the program)
84     Before: splash screen is present
85     After: passsword is validated for teacher, creates new teacher class and
disposes the splash screen

```

```

86     Parametes: password entered
87
88
89     Pseudocode
90     Compares the valid string (parameter) to the "master password"
91     If it is the same, creats new teacher class
92     If it is not the same, gives error to user
93     ****/
94     {
95         try
96         {
97             if(valid.equals("xyz12345")) //the given password
98             { this.dispose();
99               new Admin();
100            }
101            else
102            {
103                output("Password is incorrect, please try again"); //error handling
104            }
105        }
106        catch(Exception ex)
107        { output("Login failed for Admin"); }
108    }
109
110
111     public void signT(String name, String password) //sign teacher
112     /**
113     The signT method is the signing in method for the teachers. A teacher can only
sign in if they have
114     an account. This can be created in the Admin section of the program.
115     When a teacher signs in they have to put in their password and they have to

```

put in their name.

```
116     This method checks whether or not their name/password combination is correct.
117
118     Before: teacher hasn't signed in
119     After: teacher has been signed in, or has been denied access to the programmed
120     Parameters: name and password
121
122     Pseudocode:
123     Open the file.
124     Loop through the file, look for name of teacher
125     If found, compare the two passwords
126         If they are the same, sign the teacher if
127         If they are incorrect, do nothing.
128
129     ***/
130     {
131         try
132         {
133             BufferedReader file = new BufferedReader(new
134             FileReader("f:\\teacherprofiles.txt"));
135             while(file.ready())
136             {
137                 String Cname = file.readLine();
138                 String Cpassword = file.readLine();
139                 if(name.equals(Cname))
140                 {
141                     if(Cpassword.equals(password))
142                     {
143                         this.dispose();
144                         new Teacher(name);
145                         return;
146                     }
147                 }
148             }
149         }
150     }
```

```
145         }
146         else if(!Cpassword.equals(password))
147         {
148             output("Password incorrect");
149             return;
150         }
151
152
153     }
154
155 }
156
157 output("Please type in the correct name and password");
158 }
159     catch(IOException e)
160     { output("Could not sign teacher in, problem with text file"); }
161 }
162
163
164 }
165
```

```

166 /**
167 =====
168
169 TEACHER CLASS extends STUDENT
170
171 =====
172 **/
173
174     class Teacher extends Student
175     {
176         static String name1;
177         Button bNew = addButton("Create New Club",50,190,100,50,this);
178         Button bDelete = addButton("Delete club",50,240,100,50,this);
179
180         public Teacher(String Aname)
181         {
182             setSize(700,400);
183             name1 = Aname;
184         }
185
186
187         public Teacher()
188         {
189
190         }
191
192
193         public void actions(Object source, String command)
194         /**
195         The action method allows the user to choose what action or command to doing
196         Parameter: object source, string command

```

```

197     Before: Nothing is happening
198     After: The action takes place
199
200     Pseudocode:
201     method checks what option the user has chosen
202     executes the commands in that option
203
204     ***/
205     {
206         if(source == bNew)
207         {
208             new nClub();
209         }
210         if(source == bDelete)
211         {
212             String organizername = name1;
213             String noc = input("Name of club you wish to delete");
214             delete(noc,organizername);
215         }
216         if(source == bDisplay)
217         {
218             display();
219         }
220         if(command.equals("Help|Quit"))
221         {
222             System.exit(0);
223         }
224         if(command.equals("Help|Help"))
225         {
226             tDisplay.setText("Welcome," + "\n" + "If you would like to make a
search," + "\n" + "press the search button." + "\n" + "Otherwise, press the display all

```

```

button" + "\n" + "to see a list of all the clubs that teachers" + "\n" + "have put into
the system." + "\n" + "If you would like to add" + "\n" + "another club, press the
'create new club' button." + "\n"+ "Or 'delete club' to delete one.");
227     }
228     if(command.equals("Help|Back to Main Menu"))
229     {
230         this.dispose();
231         new Main();
232     }
233     if(source == bSearch)
234     {
235         new Search();
236     }
237 }
238
239
240
241
242     public void delete(String name, String organizerB)
243     /**
244     This method will allow the user to delete a club. For this, they will need to
enter the name of the club
245     exactly like it appears in the random access file.
246
247     parameter: name of club
248     before: the club is still in the random access file
249     after: the club is erased
250
251     Loop through file
252     read the name of each record
253     if its the right name

```

```

254     move all subsequent records up one position
255         go to position x(record) + 1
256         read the stuff from there
257         copy to position x - 1
258         repeat until there is nothing to copy anymore
259     if club is at last position
260     delete the club at last position
261     **/
262     { long x;
263       try
264         {
265           RandomAccessFile data = new RandomAccessFile("clubs.dat","rw");
266           long records = (data.length()+299)/300; //gives us the number of records
267           for(x = 0;x < records-1; x = x+1) // goes up one record
268             {
269               data.seek(x*300); //seeks to name
270               String clubN = data.readUTF();
271               if(clubN.equals(name)) //compared the two names of clubs
272                 { data.seek((x*300)+60);
273                   String organizerA = data.readUTF();
274                   if(organizerA.equals(organizerB)) //compares the organizers (club
can only be deleted by whoever created it
275                     {
276
277                       for(long d = x; d < records-1; d = d+1)
278                         {
279                           data.seek((x+1)*300); //seeks to position of next club
(after the one we want delete)
280                           String pname = data.readUTF(); // copies the name
281                           data.seek(x*300);
282                           data.writeUTF(pname);

```

```
283
284         data.seek(((x+1)*300)+50); //seeks to club and copies day,
then goes back and copies it on the club before
285         String pday = data.readUTF();
286         data.seek((x*300)+50);
287         data.writeUTF(pday);
288
289         data.seek(((x+1)*300)+60);
290         String porganizer = data.readUTF();
291         data.seek((x*300)+60);
292         data.writeUTF(porganizer);
293
294         data.seek(((x+1)*300)+90);
295         String pemail = data.readUTF();
296         data.seek((x*300)+90);
297         data.writeUTF(pday);
298
299         data.seek(((x+1)*300)+150);
300         String pgrade = data.readUTF();
301         data.seek((x*300)+150);
302         data.writeUTF(pgrade);
303
304         data.seek(((x+1)*300)+160);
305         String plocation = data.readUTF();
306         data.seek((x*300)+160);
307         data.writeUTF(plocation);
308
309         data.seek(((x+1)*300)+180);
310         String ptype = data.readUTF();
311         data.seek((x*300)+180);
312         data.writeUTF(ptype);
```

```

313
314         data.seek(((x+1)*300)+200);
315         String pdescription = data.readUTF();
316         data.seek((x*300)+200);
317         data.writeUTF(pdescription);
318
319         long l = data.length();
320         data.setLength(l-300);
321         return;
322     }
323
324         output("Your club has been deleted");
325
326     }
327
328
329     else{output("not the same teacher"); }
330 }
331 }
332 }
333 if(x == records-1) //if it is the last club in the file...there is a
different method of deleting
334 { data.seek(x*300);
335   String clubName = data.readUTF(); //checking that they fit...
336   data.seek((x*300)+60);
337   String organizerA = data.readUTF();
338   if(clubName.equals(name))
339   {
340     if(organizerA.equals(organizerB)) //if they match
341     {
342       data.setLength(x*300);

```

```
343         output("Your club has been deleted");
344     }
345     else
346     {output("not the same teacher");}
347 }
348 }
349 data.close();
350 }
351
352
353
354     catch (IOException e)
355     { output("Could not erase club, please try again"); }
356
357
358 }
359 }
360 }
361
```

```

362 /**
363 =====
364 =====
365
366 Student CLASS extends EASYAPP
367
368 =====
369 =====
370 ***/
371
372 class Student extends EasyApp
373 {
374     Label welcome = addLabel("Welcome",50,30,75,30,this);
375
376     Button bDisplay = addButton("Display Clubs",50,70,100,50,this);
377     Button bSearch = addButton("Search",50,120,100,50,this);
378     TextArea tDisplay = addTextArea("",170,70,500,300,this);
379
380     Menu mMenu = addMenu("Help|Help|Back to Main Menu|Quit");
381
382
383     public Student()
384     {
385         setBackground(new Color(11,118,53));
386         welcome.setBackground(new Color(11,118,53));
387         welcome.setFont(new Font("Arial",1,16));
388         setSize(700,400);
389     }
390     public void actions(Object source, String command)
391     {
392         if(source == bDisplay)

```

```

393     {
394         display();
395     }
396     if(source == bSearch)
397     {
398         new Search();
399     }
400     if(command.equals("Help|Quit"))
401     {
402         System.exit(0);
403     }
404     if(command.equals("Help|Help"))
405     {
406         tDisplay.setText("Welcome," + "\n" + "If you would like to make a
search," + "\n" + "press the search button." + "\n" + "Otherwise, press the display all
button" + "\n" + "to see a list of all the clubs that teachers" + "\n" + "have put into
the system.");
407     }
408     if(command.equals("Help|Back to Main Menu"))
409     {
410         this.dispose();
411         new Main();
412     }
413 }
414 }
415
416 public void display()
417 /**
418 This method is a method to display the clubs from the random access file.
419 It will display the clubs onto an Area Field
420

```

```

421     Before: the area field is empty
422     After: the area field has all the clubs displayed
423
424     Pseudocode:
425     Open random access file
426     Loop through file, seeking to all of the names of the clubs
427     Read the name of clubs
428     Displays the name of club
429     Go to next line, display next name of club
430     Close random access file
431
432     **/
433     {
434         tDisplay.setText("");
435         try
436         {
437             RandomAccessFile data = new RandomAccessFile("clubs.dat", "rw");
438             long records = (data.length()+299)/300;
439             for(long z = 0; z<records; z = z+1) //goes through all the random access
file to display all the clubs
440             {
441                 data.seek(z*300);
442                 String dname = data.readUTF();
443
444                 data.seek((z*300)+50);
445                 String dday = data.readUTF();
446
447                 data.seek((z*300)+60);
448                 String dorganizer = data.readUTF();
449
450

```

```

451         data.seek((z*300)+90);
452         String demail = data.readUTF();
453
454         data.seek((z*300)+150);
455         String dgrade = data.readUTF();
456
457         data.seek((z*300)+160);
458         String dlocation = data.readUTF();
459
460         data.seek((z*300)+180);
461         String dtype = data.readUTF();
462
463         data.seek((z*300)+200);
464         String ddescription = data.readUTF();
465
466         String t = tDisplay.getText() + "\n" + "Name of Club: "+dname + "\n"
+ "Day: " + dday + "\n" + "Organizer: " +dorganizer + "\n" + "Contact Email: " + demail +
"\n" + "For ages: " + dgrade + "\n" + "Location: " +dlocation + "\n" + "Type of Club: "
+ dtype + "\n" + "Description: "+ ddescription + "\n" + "\n";
467         tDisplay.setText(t);
468
469
470
471     }
472     data.close();
473 }
474     catch(IOException e)
475     {output("error in displaying clubs"+e.toString()); }
476
477
478

```

479 }
480 }
481 }
482 }

```

483  /**
484  =====
485  =====
486
487  New Club CLASS extends EASYAPP
488
489  =====
490  =====
491  ***/
492      class nClub extends EasyApp
493      {
494          Label lMain = addLabel("Creating a New Club",200,50,200,50,this);
495          Label lday = addLabel("Day:",30,100,70,30,this);
496          Label lname = addLabel("Name:",30,130,70,30,this);
497          Label lemail = addLabel("Email:",30,160,70,30,this);
498          Label lgrade = addLabel("Grades:",30,190,70,30,this);
499          Label llocation = addLabel("Location:",30,220,70,30,this);
500          Label ltype = addLabel("Type:",30,250,70,30,this);
501          Label ldescription = addLabel("Description:",30,280,70,30,this);
502
503
504          Choice tday = addChoice("- DAY -|M|T|W|Th|F|S",110,100,300,30, this);
505          TextField tname = addTextField("", 110, 130, 300, 30, this);
506          TextField temail = addTextField("", 110, 160, 300, 30, this);
507          TextField tgrade = addTextField("", 110, 190, 300, 30, this);
508          TextField tlocation = addTextField("", 110, 220, 300, 30, this);
509          Choice ttype = addChoice("- TYPE -
|Art|Sport|Drama|Service|Intellectual|Music|Other",110,250,300,30,this);
510          TextField tdescription = addTextField("", 110, 280, 300, 30, this);
511
512          Button bsave = addButton("SAVE!", 450,200,70,50,this);

```

```

513 Button bquit = addButton("Cancel",450,250,70,50,this);
514
515 public nClub()
516 {
517     setBackground(new Color(11,118,53));
518     lday.setBackground(new Color(11,118,53));
519     lname.setBackground(new Color(11,118,53));
520     lemail.setBackground(new Color(11,118,53));
521     lgrade.setBackground(new Color(11,118,53));
522     llocation.setBackground(new Color(11,118,53));
523     ldescription.setBackground(new Color(11,118,53));
524     ltype.setBackground(new Color(11,118,53));
525     lMain.setBackground(new Color(11,118,53));
526     lMain.setFont(new Font("Arial",1,16));
527 }
528
529 public void actions(Object source, String command)
530 {
531     if(source == bsave)//the following code checks that all of the strings
actually make some sort of sense, rather than just
532     //copying in to the random access file an empty string
533     {
534         String name = tname.getText();
535         String day = tday.getSelectedItem();
536         String organizer = Teacher.name1;
537         String email = temail.getText();
538         String grade = tgrade.getText();
539         String location = tlocation.getText();
540         String type = ttype.getSelectedItem();
541         String description = tdescription.getText();
542         if(description.length() > 98)

```

```

543         {
544             String answer = input("Description is too long. If you proceed, your
description will be forced to be shorter. Do you want to proceed?");
545             if(answer.equalsIgnoreCase("yes") || answer.equalsIgnoreCase("y") )
546                 {
547                     description = description.substring(0,98);
548                     validate(name,day,organizer, email, grade, location,type,
description);
549                     this.dispose();
550                 }
551             }
552         }
553         if(name.length() < 1)
554             {
555                 output("Please write a name for the club");
556             }
557         if(day.equals("-DAY-"))
558             {
559                 output("Please select a day");
560             }
561         if(grade.length() < 1)
562             {
563                 output("Please type in a valid grade (e.g. 9-12)");
564             }
565         if(location.length() < 1)
566             {
567                 output("Please type in a location");
568             }
569         if(type.equals("- TYPE -"))
570             {
571                 output("please select type");

```

```

572         }
573         else
574         {
575
576             validate(name,day,organizer, email, grade, location,type,
description);
577             this.dispose();
578         }
579     }
580     }
581     if(source == bquit)
582     { this.dispose(); }
583
584     }
585
586     public void validate(String name, String day, String organizer, String email,
String grade, String location, String type, String description)
587     /**
588         I need to make a method in order to validate the email. I'm doing this because
it is easy to make
589         a mistake when it comes to typing in the day. The day has to be in a special
format (in this case it has to be a letter/number to represent the day it is on
590
591         Before: the user puts in the day of the club
592         After: the program checks whether the day is correct
593         Parameters: day of clubs
594
595         Pseudocode:
596             user inputs the day of clubs in the form of a string
597             program checks if the length of the string is smaller than 1
598             if it is smaller than one, it tells user to try again

```

```

599         deletes the last node
600     else, the program checks if the length is bigger than one
601         if it is bigger than one, it tells user to try again
602         deletes the last nodes
603     **/
604     {
605
606         if(email.indexOf(" ") > -1)
607             {output("There can't be any blank spaces in the email address");}
608         else if(email.indexOf("@") < 2)
609             {output("Email is not valide, please try again");}
610         else //if the email is validated, it will add it
611             {addNewClub(name,day,organizer,email,grade,location,type,description);}
612     }
613
614     public void addNewClub(String name, String day, String organizer, String
email, String grade, String location, String type, String description)
615     /****
616     This creates a new club
617     There are around 100-200 clubs in the school.
618     Parameters:
619     Each club has the following fields in their record:
620     Name - 50 bytes
621     Day - 10 bytes
622     Organizer - 30 bytes
623     Email - 60 bytes
624     Grade - 10 bytes
625     Location - 20 bytes
626     Type - 20 bytes
627     Description - 100 bytes
628     Therefore, one club will have a total of 300 bytes. If there are around 100-

```

```

200
629     clubs, the random access file will need to be around 6000 bytes.  + 200*2 =
400
630     6000+400 = 6400
631
632     Before: There is a random access file, main program has decided that this is
not a duplicate
633     After:  There is a random access file with a new club
634
635     Pseudocode
636     seek to the end of file
637     write in data - add new club
638     *****/
639     {
640         try
641         {
642
643             RandomAccessFile data = new RandomAccessFile("clubs.dat", "rw");
644             long records = (data.length()+299)/300;
645             long lengthOfFile = data.length();
646             if(lengthOfFile > 0)
647             {
648                 for(int z = 0; z < records; z = z+1)
649                 {
650                     data.seek(z*300);
651                     String nameofexistingclub = data.readUTF();
652                     if(name.equalsIgnoreCase(nameofexistingclub))
653                     {
654                         output("This club exists");
655                     }
656                     else

```

```

657         {
658             data.seek(records*300); //writes the name of the club
659             data.writeUTF(name);
660
661             data.seek((records*300)+50); // writes the day
662             data.writeUTF(day);
663
664             data.seek((records*300)+60); // writes the name of the
organizer
665             data.writeUTF(organizer);
666
667             data.seek((records*300)+90); //writes the email address of the
organizer
668             data.writeUTF(email);
669
670             data.seek((records*300)+150); //writes the grade
671             data.writeUTF(grade);
672
673             data.seek((records*300)+160); //writes the location
674             data.writeUTF(location);
675
676             data.seek((records*300)+180); //writes type
677             data.writeUTF(type);
678
679             data.seek((records*300)+200); //writes the description
680             data.writeUTF(description);
681
682             output("Your club has been added");
683         }
684     }}
685 else

```

```

686     {
687         data.seek(records*300); //writes the name of the club
688         data.writeUTF(name);
689
690         data.seek((records*300)+50); // writes the day
691         data.writeUTF(day);
692
693         data.seek((records*300)+60); // writes the name of the organizer
694         data.writeUTF(organizer);
695
696         data.seek((records*300)+90); //writes the email address of the
organizer
697         data.writeUTF(email);
698
699         data.seek((records*300)+150); //writes the grade
700         data.writeUTF(grade);
701
702         data.seek((records*300)+160); //writes the location
703         data.writeUTF(location);
704
705         data.seek((records*300)+180); //writes type
706         data.writeUTF(type);
707
708         data.seek((records*300)+200); //writes the description
709         data.writeUTF(description);
710
711         output("Your club has been added");
712     }
713
714
715

```

```
716         data.close();
717     }
718         catch (IOException e)
719     { System.out.println("Error while trying to add new club"); }
720
721     }
722
723 }
```

```

724  /**
725  =====
726  =====
727
728  SEARCH CLASS extends EASYAPP
729
730  =====
731  =====
732  ***/
733
734      class Search extends EasyApp
735      {
736          public Search()
737          {
738              setBackground(new Color(11,118,53));
739              lHello.setBackground(new Color(11,118,53));
740              lHello.setFont(new Font("Arial",1,16));
741              Search1.setBackground(new Color(11,118,53));
742              Search2.setBackground(new Color(11,118,53));
743              Search3.setBackground(new Color(11,118,53));
744              amount.setBackground(new Color(11,118,53));
745              setSize(700,400);
746          }
747
748
749          Label lHello = addLabel("Search for Clubs",200,50,200,50,this);
750          Label Search1 = addLabel("Search by KeyWord:",50,100,120,30,this);
751          Label Search2 = addLabel("Search by Type:",50,200,120,30,this);
752          Label Search3 = addLabel("Search by Day:",50,265,120,30,this);
753
754          Button bSearch = addButton("Enter KeyWord",50,135,100,30,this);

```

```

755     Button bQuit = addButton("Exit", 610, 300, 70, 30, this);
756     Button bGo1 = addButton("!", 125, 237, 20, 20, this);
757     Button bGo2 = addButton("!", 125, 302, 20, 20, this);
758
759     TextArea tMain = addTextArea("", 200, 100, 400, 250, this);
760
761     Choice tChoice = addChoice("- TYPE -
|Art|Sport|Drama|Service|Intellectual|Other", 50, 235, 70, 50, this);
762     Choice tChoicea = addChoice("- DAY - |M|T|W|Th|F|S", 50, 300, 70, 50, this);
763     Choice tChoiceb = addChoice("-ALL- |1|5|10|20|100|", 100, 165, 70, 50, this);
764
765     Label amount = addLabel("results:", 50, 165, 120, 30, this);
766     Menu mMenu = addMenu("Help|Help|Back to Main Menu|Quit");
767
768
769
770
771     public void actions(Object source, String command)
772     {
773         if(source == bSearch)
774         {   try
775             {
776                 String searchWord = input("KeyWord:");
777                 if(searchWord.length() < 1)
778                 {
779                     output("Please type in a KeyWord");
780                 }
781                 else
782                 {
783                     int sAmount;
784                     String amount = tChoiceb.getSelectedItem();

```

```

785         if(amount.equals("-ALL-"))
786             {sAmount = 0;}
787         else
788             {sAmount = Integer.parseInt(amount);}
789         search(searchWord,sAmount);
790     }
791 }
792 catch(Exception ex)
793 {
794     output("KeyWord search failed");
795 }
796 }
797 if(source == bQuit)
798 { this.dispose();}
799 if(source == bGo1)
800 {
801     String type1 = tChoice.getSelectedItem();
802     if(type1.equals("- TYPE -"))
803     {
804         tMain.setText("Please select a type");
805     }
806     else
807     {searchType(type1);}
808 }
809 if(source == bGo2)
810 {
811     String day1 = tChoicea.getSelectedItem();
812     if(day1.equals("- DAY -"))
813     { tMain.setText("Please select a day" ) ; }
814     else
815     {

```

```

816         search(day1);
817     }
818 }
819 if(command.equals("Help|Quit"))
820 {
821     System.exit(0);
822 }
823 if(command.equals("Help|Help"))
824 {
825     tMain.setText("Search by day, keyword or type of club." + "\n" + "Choose
the maximum amount of results" + "\n" + "when conducting a keyword based search.");
826 }
827 if(command.equals("Help|Back to Main Menu"))
828 {
829     this.dispose();
830     new Main();
831 }
832 }
833 }
834
835 public void display(LinkList nameList)
836 /**
837 This display method is used to display the nodes from the link list.
838 Before: there is a link list
839 After: linklist is displayed
840 Parameters: linklist
841
842 Pseudocode
843 as long as the next node is not equals to zero,
844 get the node, display it
845 go on to next node

```

```

846     **/
847     {
848         tMain.setText("");
849         Node info = nameList.displayNextNode();
850         while(info != null)
851             {
852                 tMain.setText(tMain.getText()+"Name of club: "+info.name+"\n" + "Day:
"+info.day+"\n"+"Organizer: "+info.organizer +"\n"+"Email: "+info.email+"\n" + "Grade:
"+info.grade+"\n"+"Location: "+info.location+"\n"+"Type: "+info.type+"\n"+"Description:
"+info.description+"\n"+"");
853                 info = nameList.displayNextNode();
854             }
855     }
856
857     public void search(String keyword, int amount)
858     /**
859     The search method for a keyword is a bit more complicated.
860     Since it also comes with an int.
861     Before: there must be a random access file
862     After: the correct clubs are displayed
863     Parameters: int for the amount of maximum results, and the string for the
keyword
864
865     Pseudocode:
866     if amount == 0,
867     search for all the possible results
868     seek to each record
869     make one big string out of it
870     check is the keyword matches any part of it
871     if this is true, stick it in the link list
872     else

```

```

873     seek to each record
874     make one big string out of it
875     check is the keyword matches any part of it
876     if this is true, stick it in the link list
877     only display up to the given maximum (int).
878     **/
879     {
880
881         int bAmount = 0;
882         LinkList nameSearch = new LinkList();
883         try
884         {
885             RandomAccessFile data = new RandomAccessFile("clubs.dat", "rw");
886             long records = (data.length()+299)/300;
887             if(amount == 0)
888             {
889                 for(long z = 0; z<records; z = z+1)
890                 {
891                     data.seek(z*300); //writes the name of the club
892                     String Lclub = data.readUTF();
893
894                     data.seek((z*300)+50); // writes the day
895                     String Lday = data.readUTF();
896
897                     data.seek((z*300)+60); // writed the name of the organizer
898                     String Lorganizer = data.readUTF();
899
900                     data.seek((z*300)+90); //writes the email address of the organizer
901                     String Lemail = data.readUTF();
902
903                     data.seek((z*300)+150); //writes the grade

```

```

904         String Lgrade = data.readUTF();
905
906         data.seek((z*300)+160); //writes the location
907         String Llocation = data.readUTF();
908
909         data.seek((z*300)+180); //writes type
910         String Ltype = data.readUTF();
911
912         data.seek((z*300)+200); //writes the description
913         String Ldescription = data.readUTF();
914
915         String compare =
Lclub+"/"+Lday+"/"+Lorganizer+"/"+Lemail+"/"+Lgrade+"/"+Llocation+"/"+Ltype+"/"+Ldescript
ion; //stick them all in one string
916         compare = compare.toUpperCase();
917         keyword = keyword.toUpperCase();
918         int position = compare.indexOf(keyword);
919         if(position <0)
920             {}
921         else
922
{nameSearch.add(Lclub,Lday,Lorganizer,Lemail,Lgrade,Llocation,Ltype,Ldescription);}
923     }
924 }
925
926 else
927 {
928     for(long z = 0; z<records; z = z+1)
929     {
930         data.seek(z*300); //writes the name of the club
931         String Lclub = data.readUTF();

```

```

932
933     data.seek((z*300)+50); // writes the day
934     String Lday = data.readUTF();
935
936     data.seek((z*300)+60); // writed the name of the organizer
937     String Lorganizer = data.readUTF();
938
939     data.seek((z*300)+90); //writes the email address of the organizer
940     String Lemail = data.readUTF();
941
942     data.seek((z*300)+150); //writes the grade
943     String Lgrade = data.readUTF();
944
945     data.seek((z*300)+160); //writes the location
946     String Llocation = data.readUTF();
947
948     data.seek((z*300)+180); //writes type
949     String Ltype = data.readUTF();
950
951     data.seek((z*300)+200); //writes the description
952     String Ldescription = data.readUTF();
953
954     String compare =
Lclub+"/"+Lday+"/"+Lorganizer+"/"+Lemail+"/"+Lgrade+"/"+Llocation+"/"+Ltype+"/"+Ldescript
ion; //stick them all in one string
955     compare = compare.toUpperCase();
956     keyword = keyword.toUpperCase();
957     int position = compare.indexOf(keyword);
958     if(position <0)
959     {}
960     else

```

```

961         {
962             if(amount == bAmount)
963                 {}
964             else
965                 { tMain.setText("");
966                   bAmount = bAmount+1;
967
nameSearch.add(Lclub,Lday,Lorganizer,Lemail,Lgrade,Llocation,Ltype,Ldescription);
968         }
969     }
970 }
971 }
972 data.close();
973 }
974     catch(IOException e)
975     { output("Problem searching for club"); }
976
977     display(nameSearch);
978 }
979
980     public void searchType(String type)
981     /**
982     This method will be used in order to search for a club through the use of the
type based search.
983     Before: There is a random access file with data in it
984     After: the user has searched and all clubs are displayed
985     Parameter: type
986
987     Pseudocode:
988     Opens the random access file
989     Searches through file and compares the type given in the parameter with the

```

```

type of each record
990     If they are equals
991     Displays that club in the area field
992
993     **/
994     {
995         LinkedList typeSearch = new LinkedList();
996         try
997         {
998
999             RandomAccessFile data = new RandomAccessFile("clubs.dat","rw");
1000             long records = (data.length()+299)/300;
1001             for(long z = 0; z<records; z = z+1)
1002             {
1003                 data.seek((z*300)+180); //goes to type
1004                 String ntype = data.readUTF();
1005
1006                 if(ntype.equals(type))
1007                 {
1008                     data.seek(z*300); //writes the name of the club
1009                     String Lclub = data.readUTF();
1010
1011                     data.seek((z*300)+50); // writes the day
1012                     String Lday = data.readUTF();
1013
1014                     data.seek((z*300)+60); // writes the name of the organizer
1015                     String Lorganizer = data.readUTF();
1016
1017                     data.seek((z*300)+90); //writes the email address of the organizer
1018                     String Lemail = data.readUTF();
1019

```

```

1020         data.seek((z*300)+150); //writes the grade
1021         String Lgrade = data.readUTF();
1022
1023         data.seek((z*300)+160); //writes the location
1024         String Llocation = data.readUTF();
1025
1026         data.seek((z*300)+180); //writes type
1027         String Ltype = data.readUTF();
1028
1029         data.seek((z*300)+200); //writes the description
1030         String Ldescription = data.readUTF();
1031
1032
1033
typeSearch.add(Lclub,Lday,Lorganizer,Lemail,Lgrade,Llocation,Ltype,Ldescription);
1034     }
1035 }
1036
1037     data.close();
1038 }
1039     catch(IOException e)
1040     { output("Problem Searching for day");}
1041 display(typeSearch);
1042
1043
1044
1045
1046
1047 }
1048
1049     public void search(String day)

```

```

1050     /****
1051     This method will be used in order to search for a club through the use of the
day based search.
1052     Before: The user hasn't searched for anything
1053     After: the user has searched and all clubs are displayed
1054     Parameter: day
1055
1056     Pseudocode:
1057     Opens the random access file
1058     Searches through file and compares the day with anything written in the file
1059     If it finds something that looks the same
1060     Displays that club in the area field
1061     ****/
1062     {
1063         LinkList daySearch = new LinkList();
1064         try
1065         {
1066
1067             RandomAccessFile data = new RandomAccessFile("clubs.dat","rw");
1068             long records = (data.length()+299)/300;
1069             for(long z = 0; z<records; z = z+1)
1070             {
1071                 data.seek((z*300)+50); //goes to day
1072                 String nday = data.readUTF();
1073
1074                 if(nday.equals(day))
1075                 {
1076                     data.seek(z*300); //writes the name of the club
1077                     String Lclub = data.readUTF();
1078
1079                     data.seek((z*300)+50); // writes the day

```

```

1080         String Lday = data.readUTF();
1081
1082         data.seek((z*300)+60); // writes the name of the organizer
1083         String Lorganizer = data.readUTF();
1084
1085         data.seek((z*300)+90); //writes the email address of the organizer
1086         String Lemail = data.readUTF();
1087
1088         data.seek((z*300)+150); //writes the grade
1089         String Lgrade = data.readUTF();
1090
1091         data.seek((z*300)+160); //writes the location
1092         String Llocation = data.readUTF();
1093
1094         data.seek((z*300)+180); //writes type
1095         String Ltype = data.readUTF();
1096
1097         data.seek((z*300)+200); //writes the description
1098         String Ldescription = data.readUTF();
1099
1100
1101
1102     daySearch.add(Lclub,Lday,Lorganizer,Lemail,Lgrade,Llocation,Ltype,Ldescription);
1103     }
1104 }
1105     data.close();
1106 }
1107     catch(IOException e)
1108     { output("Problem Searching for day");}
1109 display(daySearch);

```

```
1110      }  
1111  
1112      }  
1113
```

```

1114  /**
1115  =====
1116  =====
1117
1118  LINKLIST CLASS
1119
1120  =====
1121  =====
1122  ***/
1123
1124      class LinkList
1125      {
1126
1127          private Node head = null;
1128          private Node tail = null;
1129          private Node display = null;
1130
1131          public void add(String Lclub,String Lday,String Lorganizer,String
Lemail,String Lgrade,String Llocation,String Ltype,String Ldescription)
1132          /**
1133          The add method adds another node to the linklist
1134          Parameters: String Lclub
1135          String Lday
1136          String Lorganizer
1137          String Lemail
1138          String Lgrade
1139          String Llocation
1140          String Ltype
1141          String Ldescription
1142
1143          Pseudocode:

```

```

1144     if the head is equals to null, add at beginning
1145     else, add at end.
1146     **/
1147     {
1148
1149         if(head == null)
1150         {
1151             head = new Node();
1152             head.name = Lclub;
1153             head.day = Lday;
1154             head.organizer = Lorganizer;
1155             head.email = Lemail;
1156             head.grade = Lgrade;
1157             head.location = Llocation;
1158             head.type = Ltype;
1159             head.description = Ldescription;
1160             head.next = null;
1161             tail = head;
1162         }
1163         else
1164         {
1165             tail.next = new Node(); // adds node last
1166             tail.next.next = null; // needs to make new pointer for end
1167             tail = tail.next; //end needs to be changed because there is a new
pointer
1168
1169             tail.name = Lclub;
1170             tail.day = Lday;
1171             tail.organizer = Lorganizer;
1172             tail.email = Lemail;
1173             tail.grade = Lgrade;
1174             tail.location = Llocation;

```

```

1174         tail.type = Ltype;
1175         tail.description = Ldescription;
1176     }
1177 }
1178
1179     public void removeHead()
1180     /**
1181     The remove head method removes the head from the linklist
1182
1183     Pseudocode:
1184     if the head is equals to null, print error message
1185     else, head = head.next.
1186     **/
1187     {
1188         if(head == null)
1189             { System.out.println("The list is empty");}
1190         else
1191             {
1192                 head = head.next;
1193             }
1194     }
1195
1196     public void removeTail()
1197     /**
1198     This method reomves the tail from the linklist.
1199
1200     If the head is equal to null,
1201     give error message
1202     else if head.next is equal to null, both head and tail are null.
1203     else, if head.next.next = null, temp.next = null.
1204     **/

```

```

1205     {
1206         Node temp = head;
1207         if(head == null)
1208         {
1209             System.out.println("The list is empty");
1210         }
1211         else if(temp.next == null)
1212         {
1213             head=null;
1214             tail=null;
1215         }
1216         else
1217         {
1218             while(temp.next.next == null)
1219             {
1220                 temp.next = null;
1221             }
1222         }
1223     }
1224 }
1225
1226     public Node displayNextNode()
1227     /**
1228     This displayNextNode method displays the next node.
1229
1230     Pseudocode:
1231     if the node is equals to null, it is the head
1232     otherwise, node is equals to node.next.
1233     return the node.
1234     **/
1235     {

```

```
1236
1237     if(display == null)
1238     {
1239         display = head;
1240     }
1241     else
1242     {
1243         display = display.next;
1244     }
1245     return display;
1246 }
1247
1248
1249 }
1250
1251 class Node
1252 {
1253     String name=" ";
1254     String day=" ";
1255     String organizer=" ";
1256     String email = " ";
1257     String grade = " ";
1258     String location = " ";
1259     String type = " ";
1260     String description = " ";
1261     Node next;
1262 }
1263
```

```

1264  /**
1265  =====
1266  =====
1267
1268  ADMIN CLASS extends TEACHER
1269
1270  =====
1271  =====
1272  ***/
1273  class Admin extends Teacher
1274  {
1275      Button bNewnT = addButton("New Teacher",50,290,100,50,this);
1276      Button bDeleteAll = addButton("Delete All Clubs",50,340,100,50,this);
1277      Button bCountClubs = addButton("Stats",700,90,100,50,this);
1278      Button bDisplayTeachers = addButton("Display Teachers",700,140,100,50,this);
1279
1280      public Admin()
1281      {
1282          setSize(850,400);
1283      }
1284
1285      public void actions(Object source, String command)
1286      {
1287          if(source == bNewnT)
1288          {
1289              try{
1290
1291                  String tname = input("Name of teacher");
1292                  if(tname.length() < 1)
1293                  {
1294                      output("Please type in a name of a teacher");

```

```

1295     }
1296     else
1297     {String password = input("Password");
1298       String apassword = input("Re-type password");
1299       if(password.length() <1)
1300       {
1301         output("password too short");
1302       }
1303       else
1304       {
1305         if(password.equals(apassword))
1306         { checkDuplicated(tname,password); }
1307         else
1308         {
1309           output("passwords do not match");
1310         }
1311       }
1312     }}
1313     catch(Exception ex)
1314     {
1315       output("Could not add new teacher");
1316     }
1317   }
1318   if(source == bDeleteAll)
1319   { try
1320     {
1321       String answer = input("are you sure you want to delete ALL club list?");
1322       if(answer.equalsIgnoreCase("yes") || answer.equalsIgnoreCase("y"))
1323       { deleteAll(); }
1324       else
1325       { output("Clubs not deleted");}

```

```

1326         }
1327         catch(Exception ex)
1328         {
1329             output("Club aren't deleted");
1330         }
1331     }
1332     if(source == bDisplay)
1333     {
1334         display();
1335     }
1336     if(source == bSearch)
1337     {
1338         new Search();
1339     }
1340     if(command.equals("Help|Quit"))
1341     {
1342         System.exit(0);
1343     }
1344     if(command.equals("Help|Back to Main Menu"))
1345     {
1346         this.dispose();
1347         new Main();
1348     }
1349     if(command.equals("Help|Help"))
1350     {
1351         tDisplay.setText("Welcome, Admin. Please press any button in order to
perform its action.");
1352     }
1353     if(source == bNew)
1354     {
1355         output("Sign in as a teacher please");

```

```

1356     }
1357     if(source == bDelete)
1358     {
1359         output("Login as the teacher who created the club to delete it");
1360     }
1361     if(source == bCountClubs)
1362     { countAllClubs(); }
1363     if(source == bDisplayTeachers)
1364     {
1365         displayTeachers();
1366     }
1367
1368 }
1369 public void displayTeachers()
1370 /**
1371 The display Teachers method will be used by the admin to the find the
passwords of all the teachers.
1372 This could be used if the teacher has forgotten their password.
1373 Before: there must be a text file
1374 After: the passwords and the teachers are displayed.
1375
Pseudocode:
1376 read the file
1377 read name of teacher
1378 read password
1379 display them
1380 go on until file is done
1381
1382
1383 **/
1384 {
1385     tDisplay.setText("");

```

```

1386         try
1387         {
1388             BufferedReader file = new BufferedReader(new
1389             FileReader("f:\\teacherprofiles.txt"));
1390             while(file.ready())
1391             {
1392                 String nameTeacher = file.readLine();
1393                 String passwordTeacher = file.readLine();
1394                 tDisplay.setText(tDisplay.getText() + nameTeacher + "\n" +
1395                 passwordTeacher + "\n");
1396             }
1397         }
1398         catch(IOException e)
1399         {}
1400     }
1401     public void deleteAll()
1402     /**
1403     The delete all method will allow the administrator to delete all of the
1404     content of the random
1405     access file in one click. This is in case the year is over and there is
1406     definitely not going to be any use for the current list anymore.
1407     Before: the random access file has content in it
1408     After: the random access file has no content in it
1409     Pseudocode:
1410     Open up random access file
1411     Set length to zero
1412     */
1413     {

```

```

1413     try
1414     {
1415         RandomAccessFile data = new RandomAccessFile("clubs.dat", "rw");
1416         data.setLength(0);
1417         data.close();
1418         output("Clubs deleted");
1419     }
1420     catch(IOException e)
1421     { output("Could not delete file"); }
1422
1423 }
1424
1425     public void countAllClubs()
1426     /**
1427         This method is a very simple method to count all the clubs in the random
1428         access file.
1429         The administrator might use it in order to quickly check how many clubs were
1430         added since the last time
1431         he checked the program.
1432         Pseudocode:
1433         Open random access file
1434         Count all the clubs
1435         Display the count
1436     **/
1437     {
1438     try
1439     {
1440         RandomAccessFile data = new RandomAccessFile("clubs.dat", "rw");
1441         long records = (data.length()+299)/300;
1442         output(records + " club(s) in the file");

```

```

1442         data.close();
1443     }
1444         catch(IOException e)
1445     { output("Error occured while trying to count clubs");}
1446     }
1447
1448
1449     public void checkDuplicated(String name, String password)
1450     /**
1451     This method checks whether or not the administrator is trying to add in a
teacher profile that already exists.
1452     Before: teacher has not been checked for duplicated
1453     After: teacher has been checked for duplicated
1454     Parameters: name, password
1455
1456     Pseudocode:
1457     Open up file
1458     Compare all the teacher names to the name in the parameter
1459     If it is the same
1460     Do not let them add it
1461     If it is different
1462     Execute the add nprofile method
1463     ***/
1464     {
1465         try
1466         {
1467             BufferedReader file = new BufferedReader(new
FileReader("f:\\teacherprofiles.txt"));
1468             while(file.ready())
1469             {
1470                 String Cname = file.readLine();

```

```

1471         if(name.equals(Cname))
1472         {
1473             output("This teacher already exists");
1474             return;
1475         }
1476     }
1477 }
1478 output("adding new teacher");
1479 nprofile(name,password);
1480
1481 }
1482     catch(IOException e)
1483     {
1484         output("could not add new teacher, error occurred with file");
1485     }
1486
1487 }
1488
1489
1490     public void nprofile(String name,String password)
1491     /**
1492     This method writes in a new profile for the teachers.
1493     Before: there is no new teacher profile
1494     After: there is a new teacher profile
1495     Parameters: Name of profile and password
1496
1497     Pseudocode:
1498     Open file
1499     Write in the name, password
1500     Close file
1501     ****/

```

```
1502     {
1503
1504         try
1505         {
1506             PrintWriter file = new PrintWriter(new
FileWriter("f:\\teacherprofiles.txt",true));
1507             file.println(name);
1508             file.println(password);
1509             file.close();
1510         }
1511         catch (IOException e)
1512         { output("Could not write in name");}
1513         output("adding worked");
1514     }
1515
1516
1517
1518 }
1519
```

Usability

The program allows the user, specifically the teacher to add clubs in a very easy way. The use of buttons and advance GUI features give the program an advantage. The entire program has implemented buttons and menus. Notice how in the talks with my end users there was a demand for the users to be able to go back to the main menu at any point. This is implemented into my program and allows the user to switch to the splash screen at any point.

I thought what I did well was specifically in the adding a new club. While it is not perfect, I thought there were a few features in which I specifically paid attention to the issue of usability:



Notice how there is no place to enter the name of the organizer of the club. This is because when the teacher signs in, their name will automatically be considered as the organizer, and when they save a new club into the file, and their name will be saved as well. This prevents teachers from creating clubs that do not belong to them.

In addition to this, I made sure that for day and for type there is only an option bar, and they cannot make up something for this. This is so that when someone does a search there is only one way to search for day and type, and it creates much more reliable results.

Thus, I implemented this feature in the search class as well: right à This corresponds to my goal in section A, in that it is a useful search option which makes searching so easy. Also, it helps to create an effective database that will correspond with days and types (have the same format).

There is a help option in every class. This gives the user who is not sure what to do the option to see what to do in the particular window they are viewing. This help option can be accessed in the menu bar.



Notice how my program is green. With the aid of GUI I managed to create labels, buttons and menus and even color the background of my program. While this is a minor part, in the real life most software that is bought is usually bought because of how it looks and how it appeals to the customer, and not by the quality of the code, as the customer does not see that part of the software. Therefore, I felt that this was an important part of the program which made its usability high.

Another feature that would go in the usability section is the fact that the clubs database, also known as the random access file, is on the local server. Meaning anyone who has the program installed on their computer and is connected to the school server can access it. This makes it accessible to nearly everyone in the school community. It also means that adding clubs and deleting them will all be done in one file, which will not need to be updated as the program accesses it in its location.

Another feature is the outputs I have put in my program. As can be seen in part D of my program, there are a lot of outputs that give feedback to the users. For example when a teacher has added a club, the program tells him that the club has been added successfully. When an admin adds a teacher, the program tells him that the teacher has been added successfully. This is shown in a lot of parts of my program in part D.

To refer back to my A2 criteria:

A condense and effective database of all clubs and organizations in our school

Usability: the statement in itself is a usability issue. Therefore, I feel that I have actually completed this goal, my program should be a very “usable” program in itself.

Effective access and use of program in order to find out information about clubs and organizations

Usability: with the mentioned usability I have done in this section thus far (like the GUI for example and the fact that the random access file is stored on the servers), it allows the students to access the program easily to find out information

Useful search options, which will allow users to find clubs uncomplicatedly

Usability: the search options are a usability issue. I have made it so that there are three search options, which allows the user to find results that match what they need and what they are looking for. This is one of the most important features in my program.

Teachers will be able to add their own new clubs

Usability: teachers ARE able to add their own club. And when they do this on the shared random access files, the clubs are automatically accessible by all of the programs on the computers (that is if the program is installed on the computer).

Teachers are able to delete their club

Usability: Teachers are able to delete their own clubs that they have created (and thus no vandalism can be produced by teachers deleting other teachers clubs). The second they

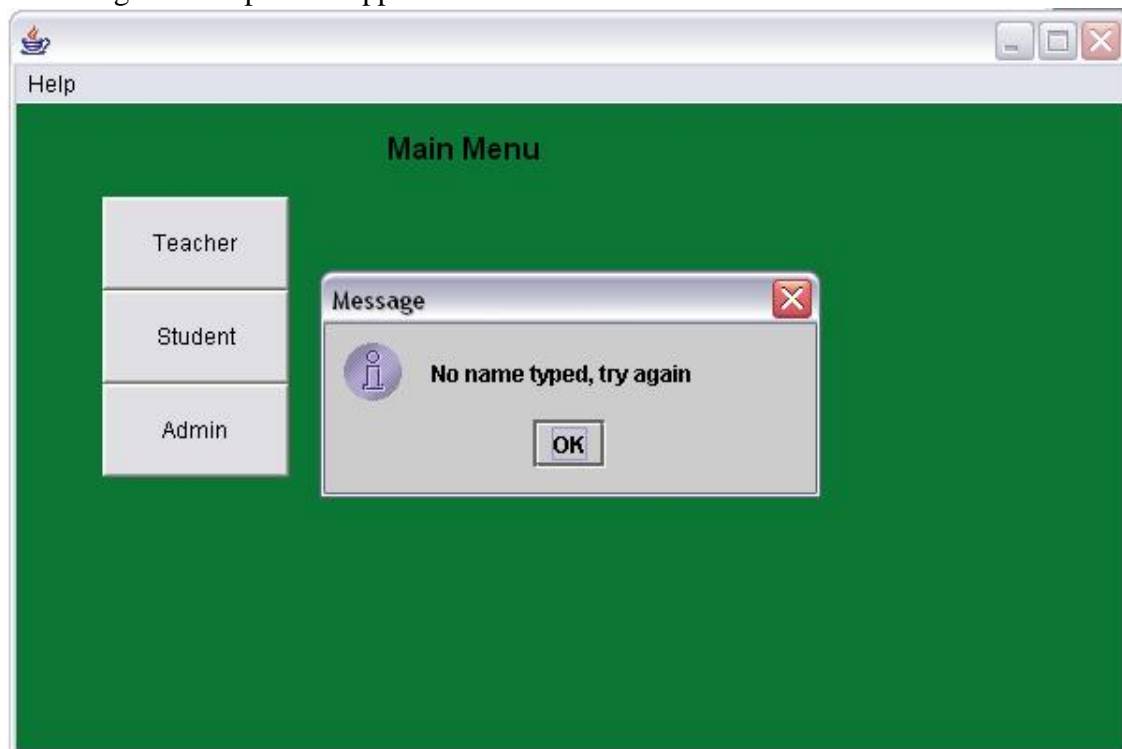
delete a club, since the random access file is on the server, all of the users will be affected by this.

Administration of the program

Usability: the administration of the program allows that there are teacher profiles which allow the creation and deletion process of clubs with an emphasis on security.

Handling Errors

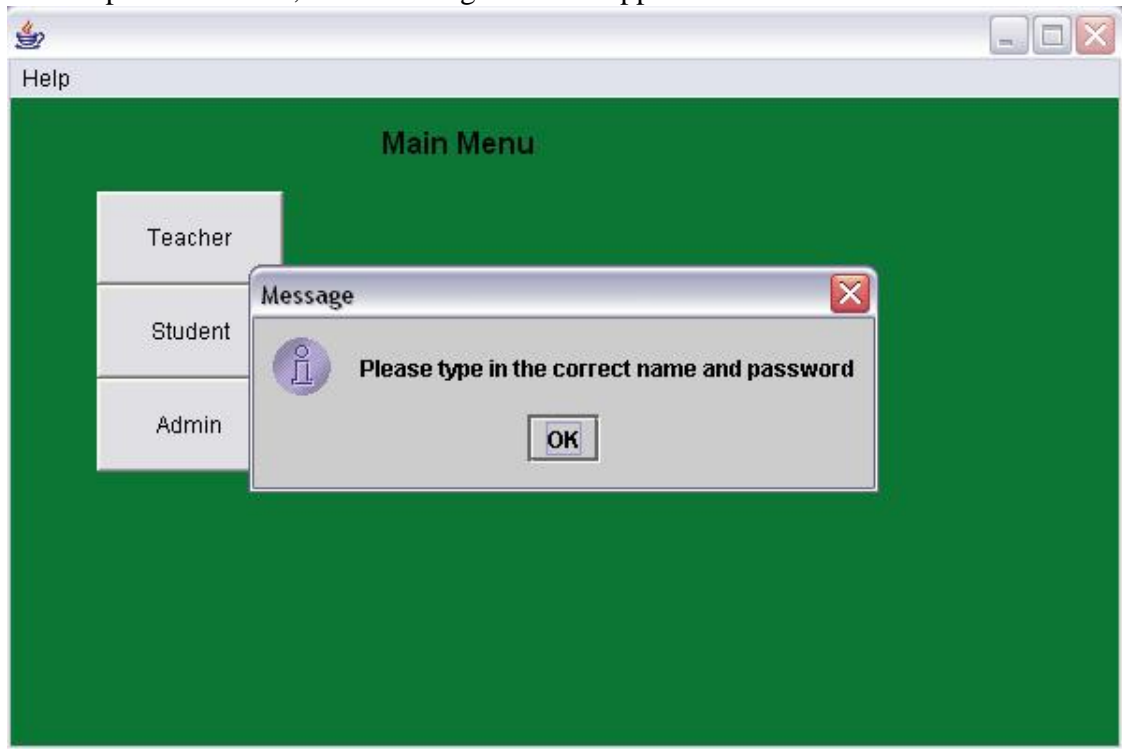
There are many error handling functions in my program. When trying to log in, and the teacher does not put in a name, the program sees that the string wasn't entered and the following error output will appear:



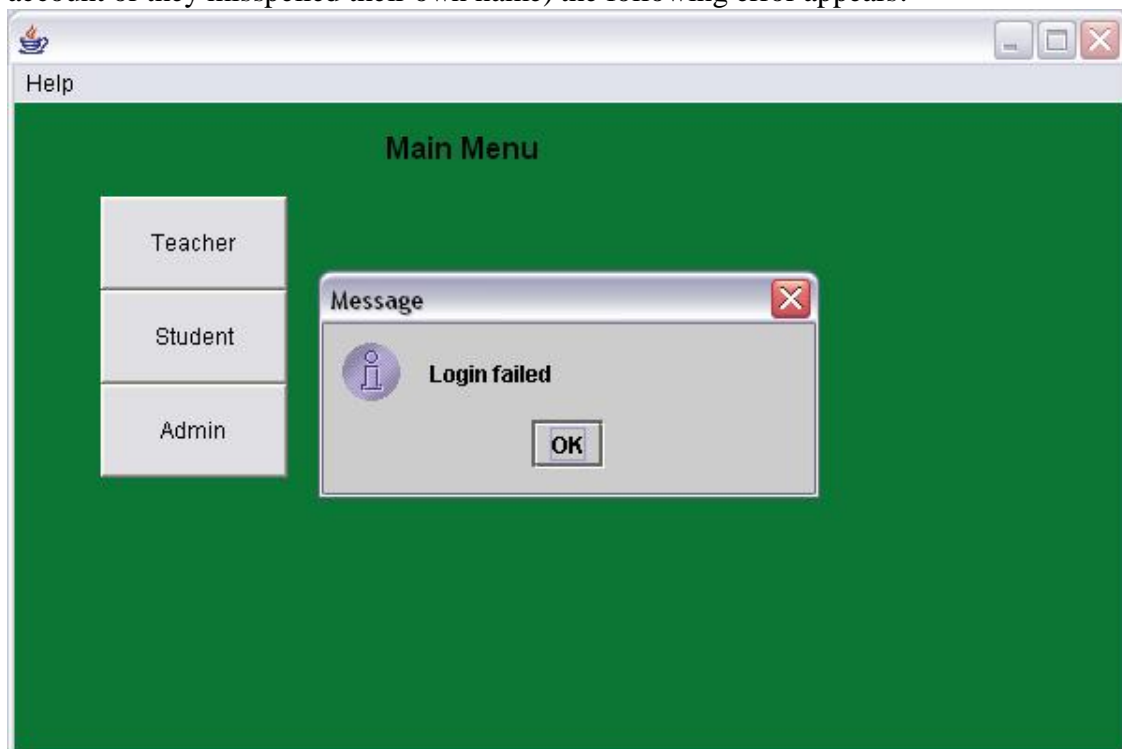
Here's the code for this error handling:

```
if(source == bTeacher)
{
    try{ //error handling
        String name = input("NAME: (Firstname Lastname)");
        String mpassword = input("PASSWORD");
        if(name.length() < 1)
        { output("No name typed, try again");}
        else if(mpassword.length() < 1)
        {}
        else
        {
            signT(name,mpassword);
        }
    }
    catch(Exception ex)
    { output("Login failed"); }
}
```

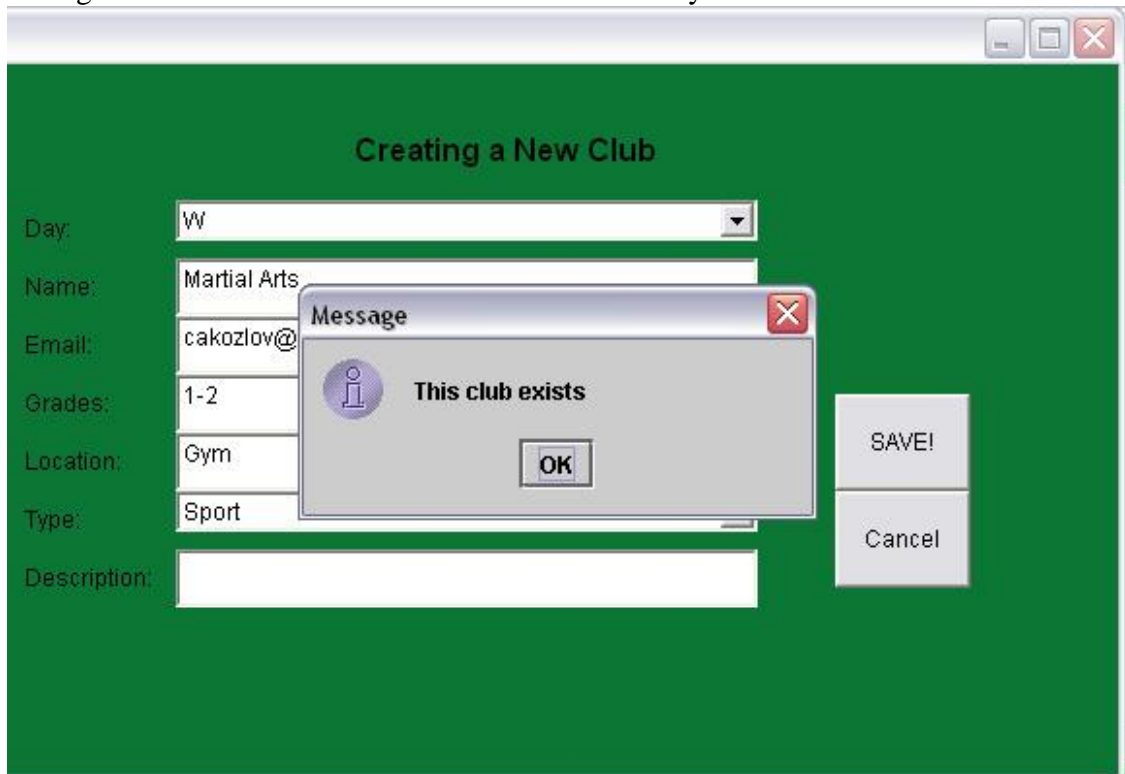
accordingly, if the password does not match the name while searching through the teacher profile text file, the following error will appear:



In addition, if the login does not work for a different reason (the teacher doesn't have an account or they misspelled their own name) the following error appears:

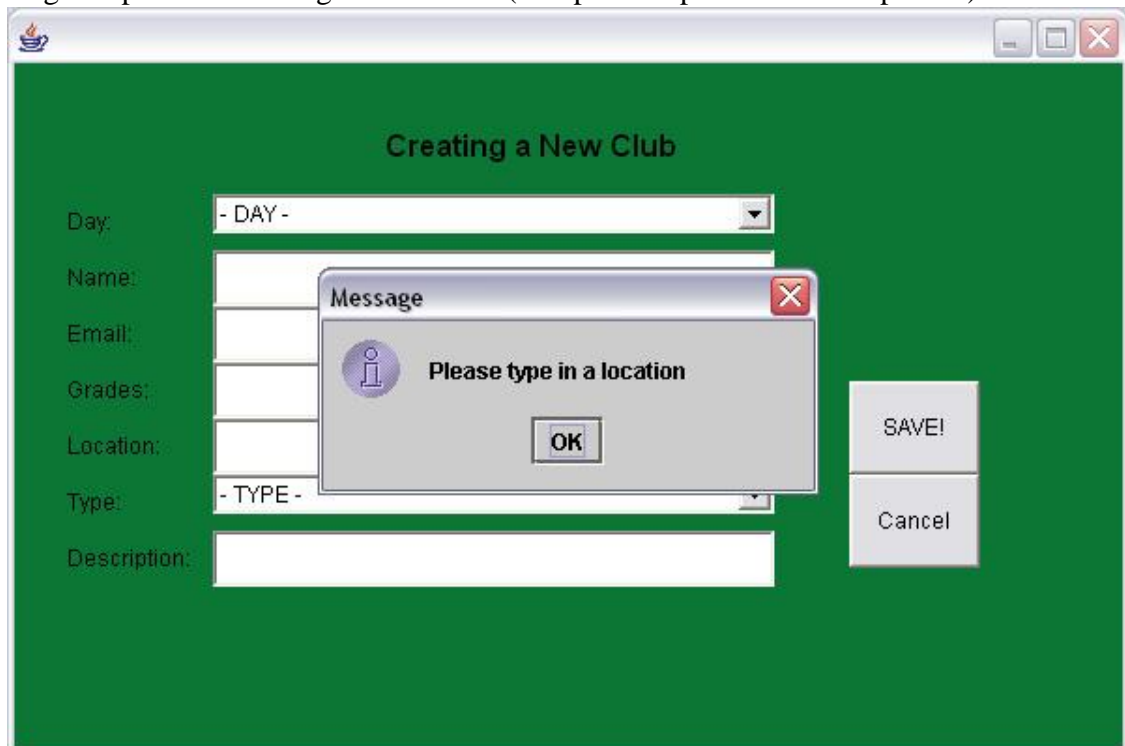


If the teacher does manage to log in, and they try to add a new club that already exists, the algorithm checks and if it finds that the club already exists:



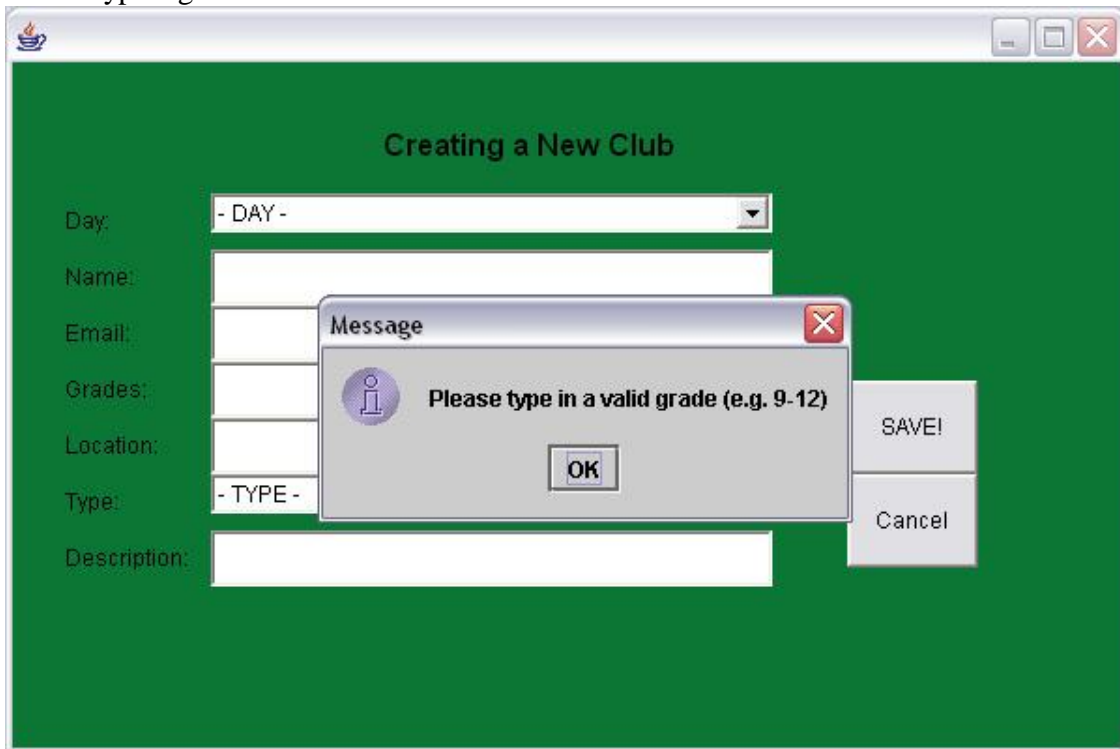
The screenshot shows a web form titled "Creating a New Club" with a green background. The form fields are: Day (W), Name (Martial Arts), Email (cakozi@...), Grades (1-2), Location (Gym), Type (Sport), and Description (empty). A modal message box is displayed over the form, containing an information icon and the text "This club exists" with an "OK" button. To the right of the form are "SAVE!" and "Cancel" buttons.

Nevertheless, if they try to add a club, one of the following errors might occur: they forget to put in something for the fields (except description which is optional):

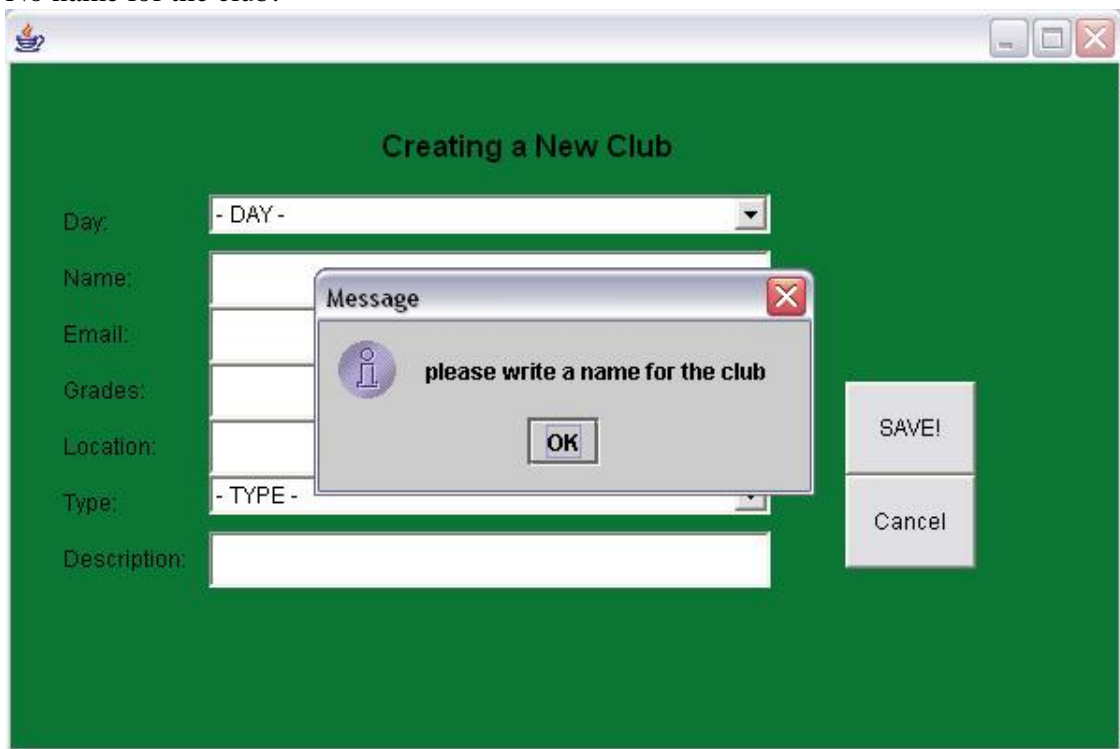


The screenshot shows the same "Creating a New Club" form, but with several fields empty: Day (- DAY -), Name, Email, Grades, Location, and Type (- TYPE -). A modal message box is displayed, containing an information icon and the text "Please type in a location" with an "OK" button. The "SAVE!" and "Cancel" buttons are visible on the right.

Forgot to put in location
Didn't type a grade:



No name for the club:



the following code demonstrated the error handling for my new club and for the past few screenshots:

```
if(name.length() < 1)
    {
        output("Please write a name for the club");
    }
if(day.equals("-DAY-"))
    {
        output("Please select a day");
    }
if(grade.length() < 1)
    {
        output("Please type in a valid grade (e.g. 9-12)");
    }
if(location.length() < 1)
    {
        output("Please type in a location");
    }
if(type.equals("- TYPE -"))
    {
        output("please select type");
    }
else
    {

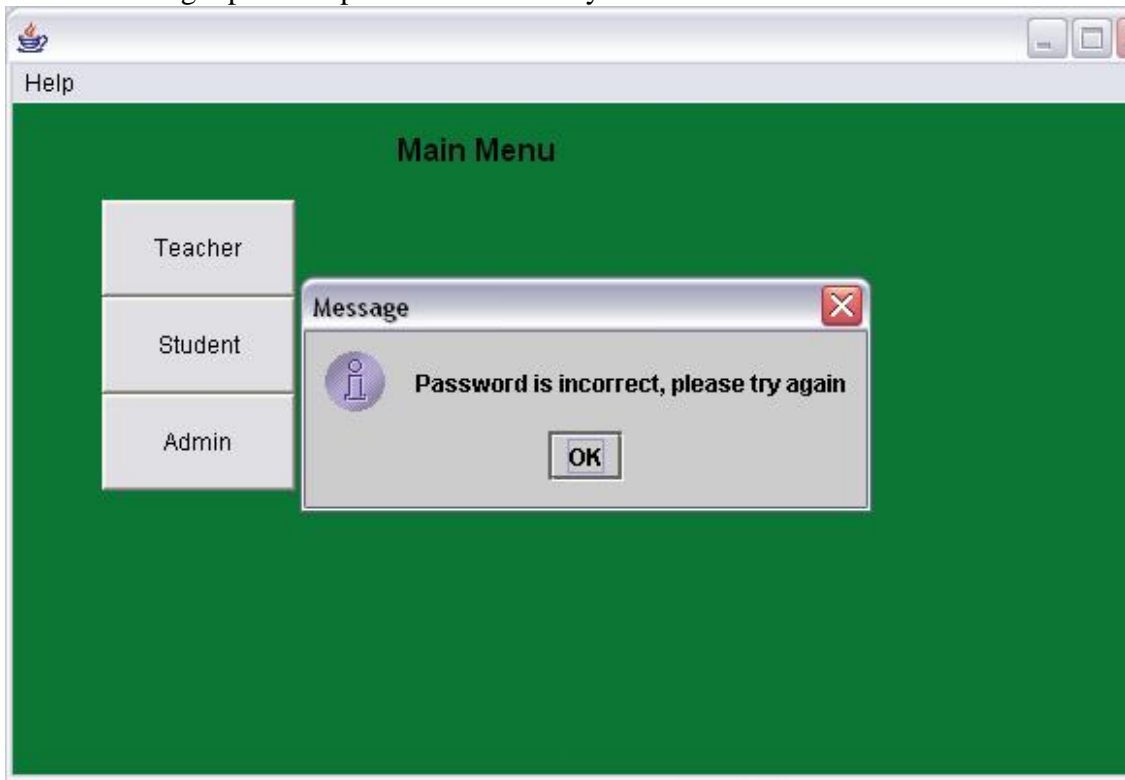
        validate(name,day,organizer, email, grade, location,type, description);
        this.dispose();
    }
}
```

and the validate method, which validates the email:

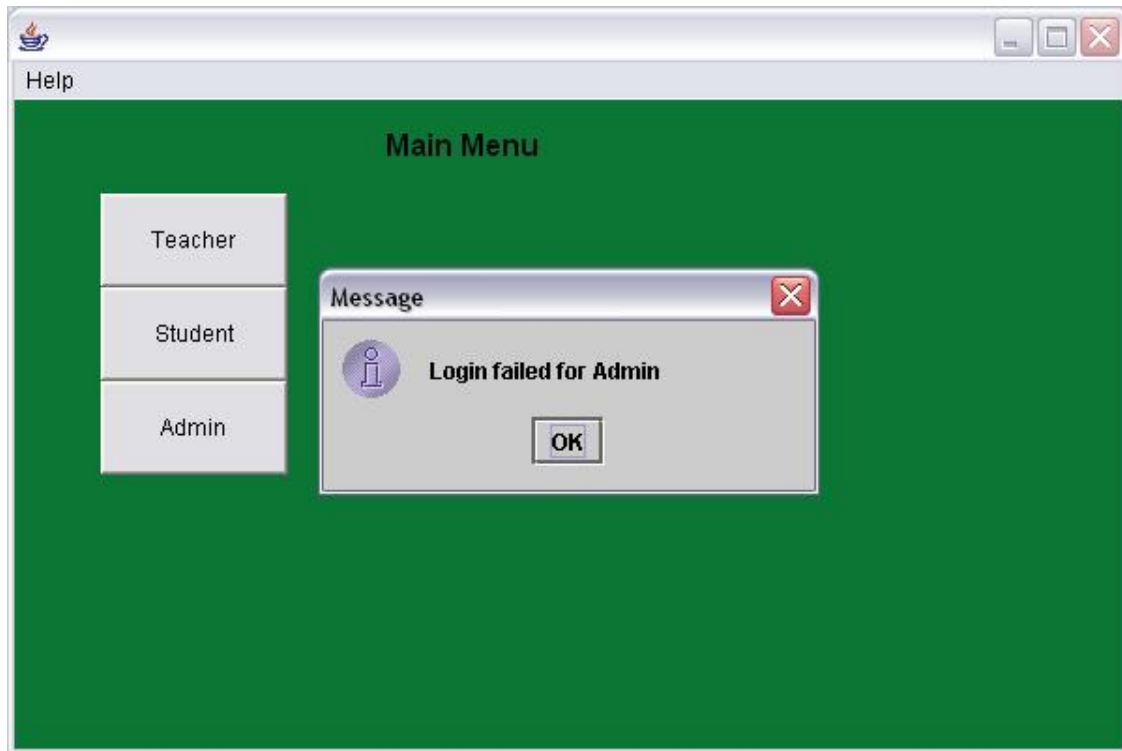
```
public void validate(String name, String day, String organizer, String email, String
grade, String location, String type, String description)
{

    if(email.indexOf(" ") > -1)
    {output("There can't be any blank spaces in the email address");}
    else if(email.indexOf("@") < 2)
    {output("Email is not valide, please try again");}
    else //if the email is validated, it will add it
    {addNewClub(name,day,organizer,email,grade,location,type,description);}
}
```

The admin might put their password incorrectly:



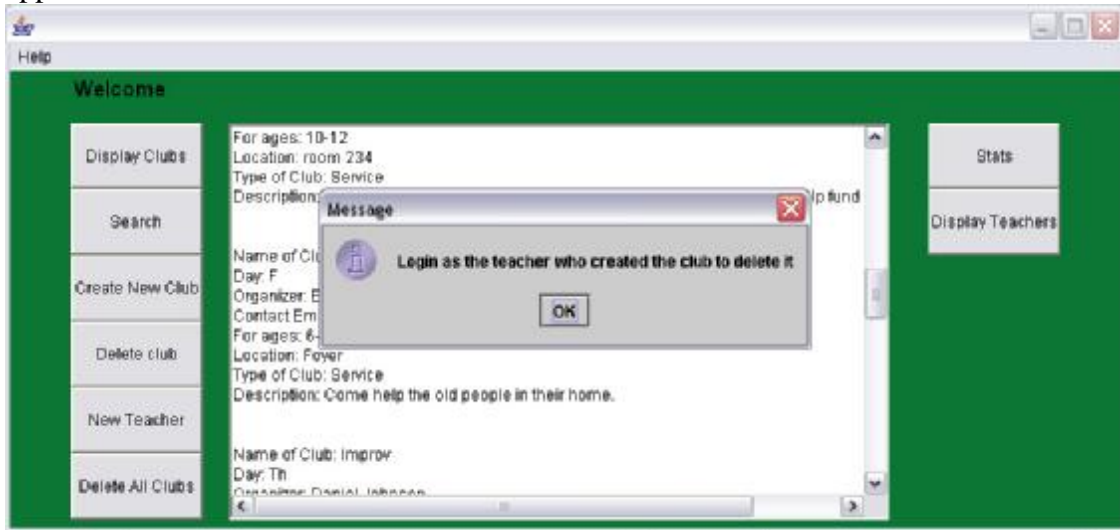
Or, they might press cancel instead of putting in a password, in this case using try and catch:



Here's the code I wrote for this part:

```
if(source == bTeacher)
{
    try{ //error handling
        String name = input("NAME: (Firstname Lastname)");
        String mpassword = input("PASSWORD");
        if(name.length() < 1)
        { output("No name typed, try again");}
        else if(mpassword.length() < 1)
        {}
        else
        {
            signT(name,mpassword);
        }
    }
    catch(Exception ex)
    { output("Login failed"); }
}
```

When the admin is logged in and tried to delete a specific club, the following error will appear:

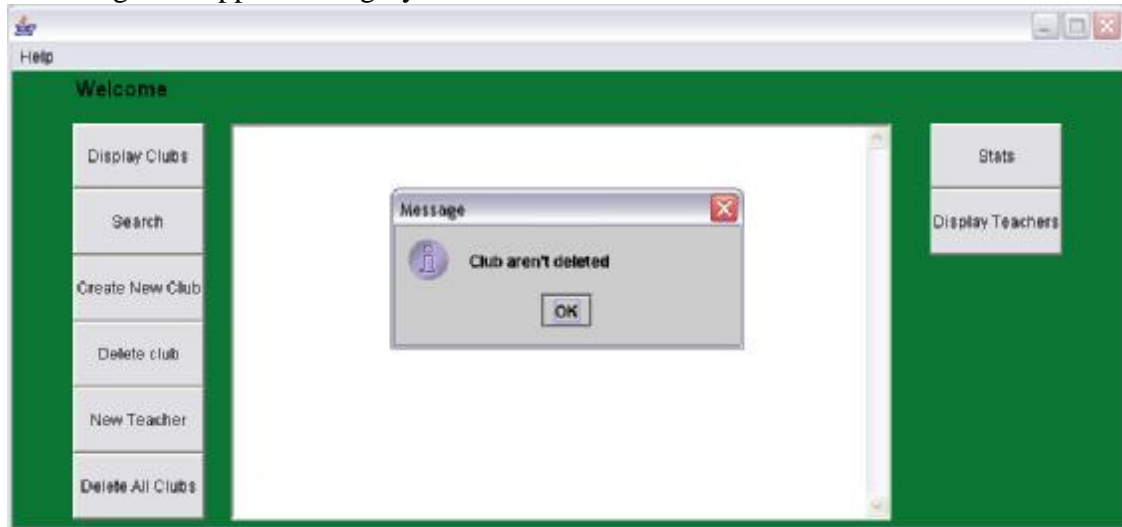


When trying to add a new club, the following error will appear for the admin:



this is to remind the admin that only teachers can add clubs.

When the admin presses the “delete all clubs” he might press cancel, in which case the following error appears using try and catch:

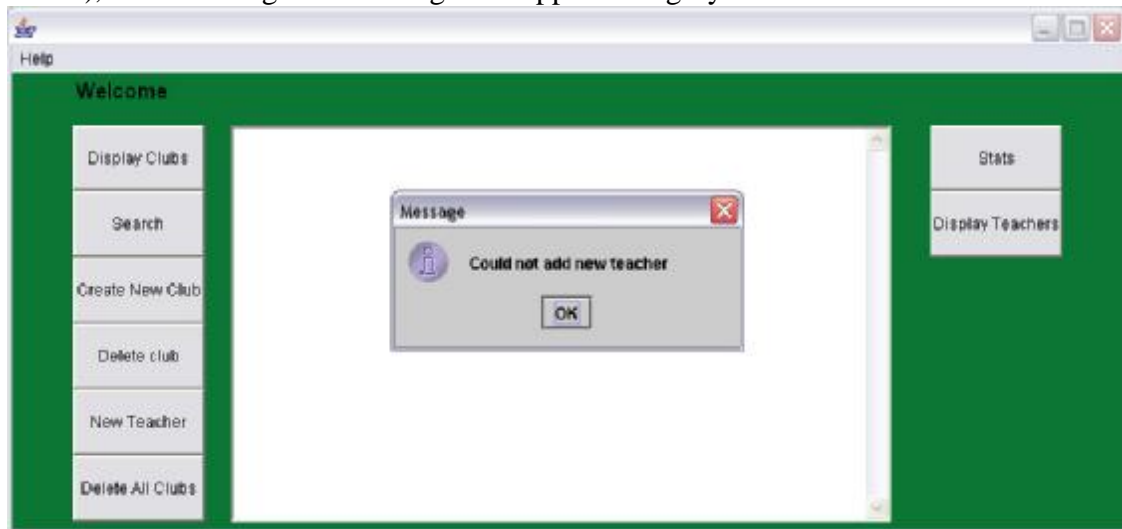


To demonstrate this, I have included the code:

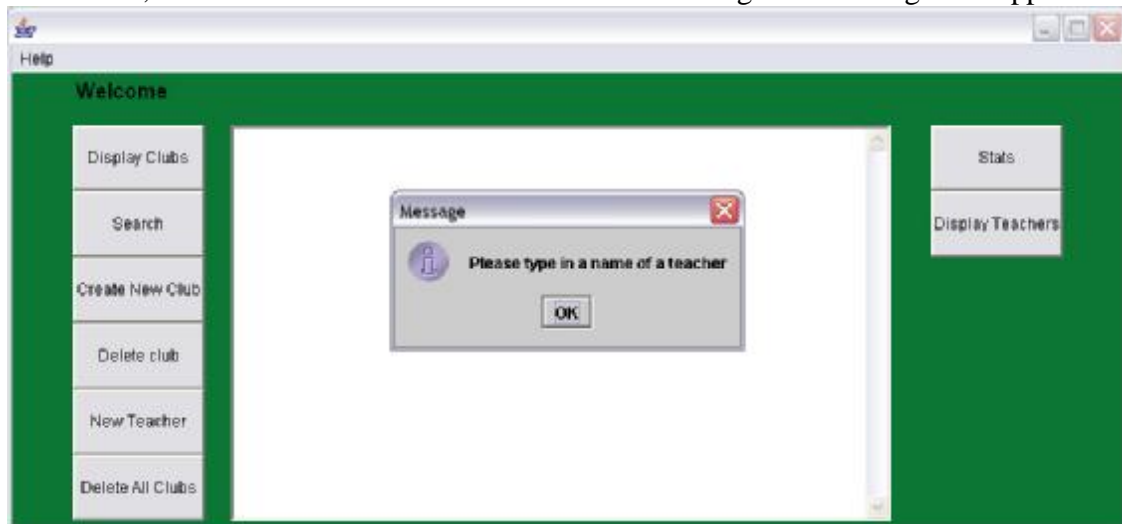
```
public void deleteAll()
{
    try
    {
        RandomAccessFile data = new RandomAccessFile("clubs.dat","rw");
        data.setLength(0);
        data.close();
        output("Clubs deleted");
    }
    catch(IOException e)
    { output("Could not delete file"); }
}
```

Note the use of the try and catch which is the reason for my error handling.

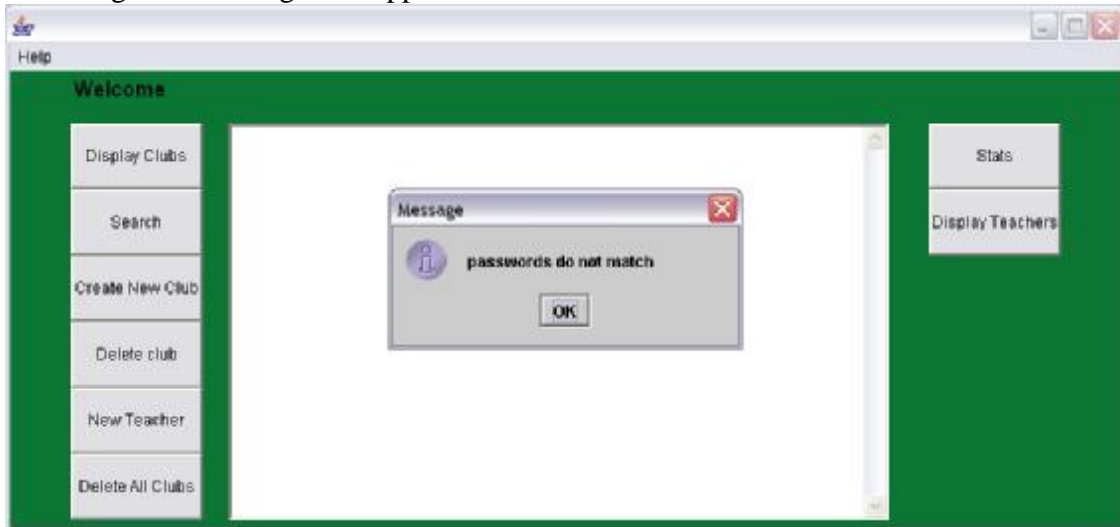
when the admin tries to add a new teacher, and fails to do so (that is, he cancels the action), the following error message will appear using try and catch:



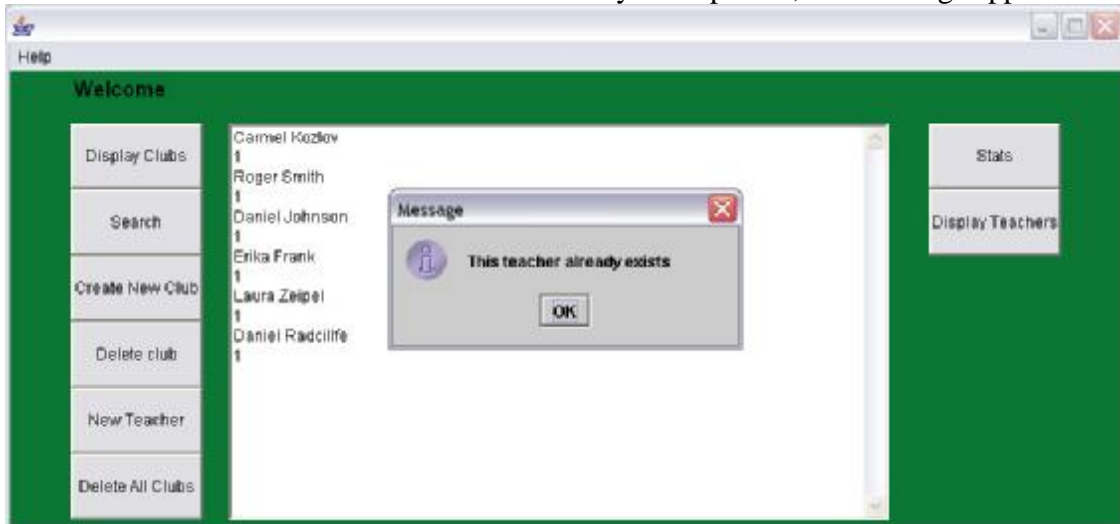
in addition, if there is no name for the teacher the following error message will appear:



When the password of the new teacher does not match the re-type of the password the following error message will appear:

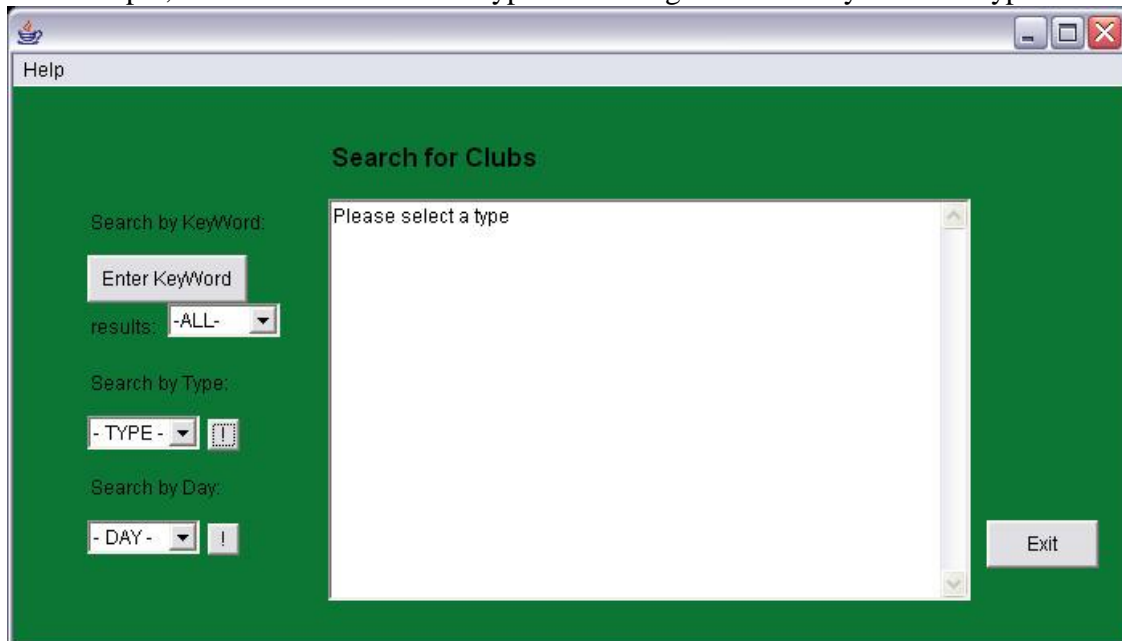


When the admin tries to add a teacher that already has a profile, this message appears:

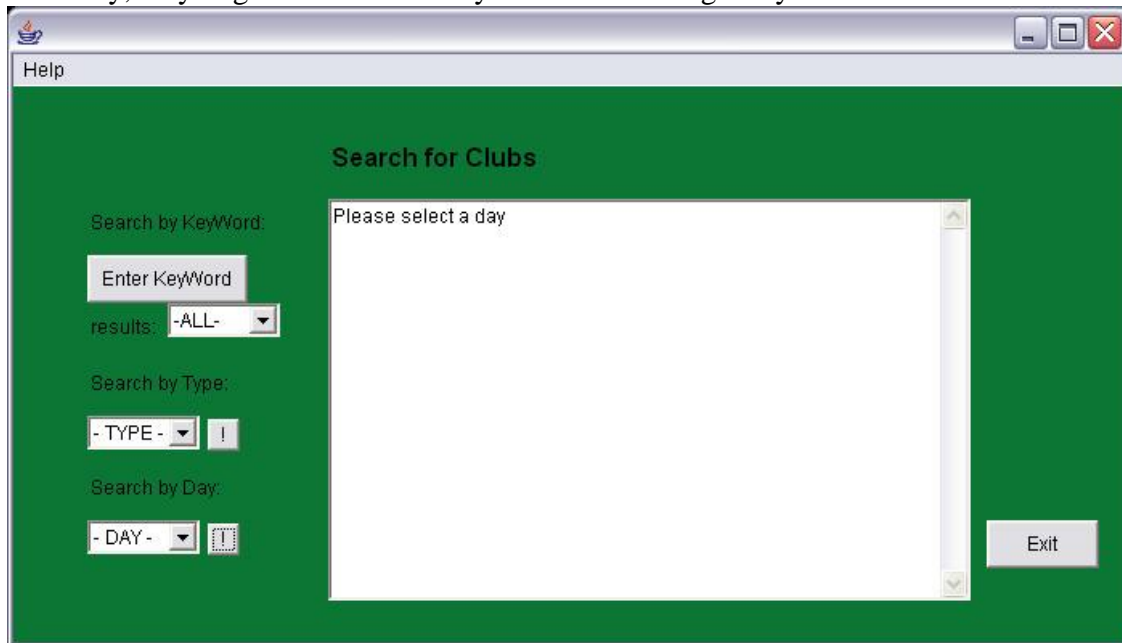


When a search class is open, there are a few errors that might occur.

For example, the user who commits a type search might not actually choose a type:



Similarly, they might not choose a day when committing a day based search:

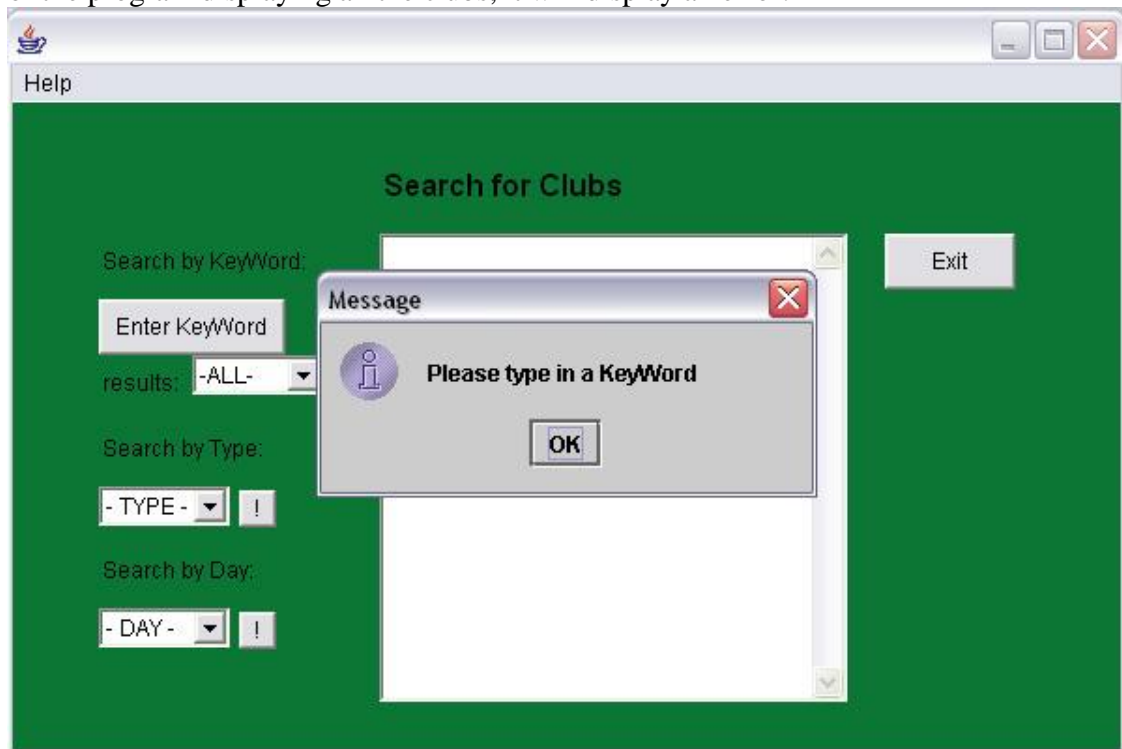


Here's the code for this error handling:

```
int sAmount;  
String amount = tChoiceb.getSelectedItemAt();  
if(amount.equals("-ALL-"))  
{sAmount = 0;}  
else
```

```
{sAmount = Integer.parseInt(amount);}
search(searchWord,sAmount);
```

When committing a keyword based search they might not put in a keyword, and instead of the program displaying all the clubs, it will display an error:



Many of the errors messages I have shown prevent the errors by try and catch. Especially in cases in which the “cancel” button was pressed.

An example taken out of the code, in would be the following:

```
public void countAllClubs()
/**
{
try
{
RandomAccessFile data = new RandomAccessFile("clubs.dat","rw");
long records = (data.length()+299)/300;
output(records + " club(s) in the file");
data.close();
}
catch(IOException e)
{ output("Error ocured while trying to count clubs");}
}
```

In this case, this didn't appear on the screenshots as the club data file does exist and is where it belongs. Therefore, this error wasn't something that I wanted to produce, as if I

could have not replaced the clubs file afterwards all the work I had put into creating new clubs (my example data) would have been lost and I wouldn't have been happy. This is an example, out of many, of how I used try and catch, but didn't produce screenshots for. I felt it is also unnecessary to put in all the code that has try and catch in it, as there is a lot of it. The try and catch error handling I have not put in the screenshots comes only from methods which use the data files (random access and text).

Goals listing with reference to section D

A condense and effective database of all clubs and organizations in our school

Everywhere (the whole program is this goal)

Effective access and use of program in order to find out information about clubs and organizations

Everywhere (the way the user interacts with the program, GUI, buttons, simple but effective search options, etc).

Useful search options, which will allow users to find clubs uncomplicatedly

D3-D8

Teachers will be able to add their own new clubs

D12-D13

Teachers are able to delete their club

D14-D15

Administration of the program

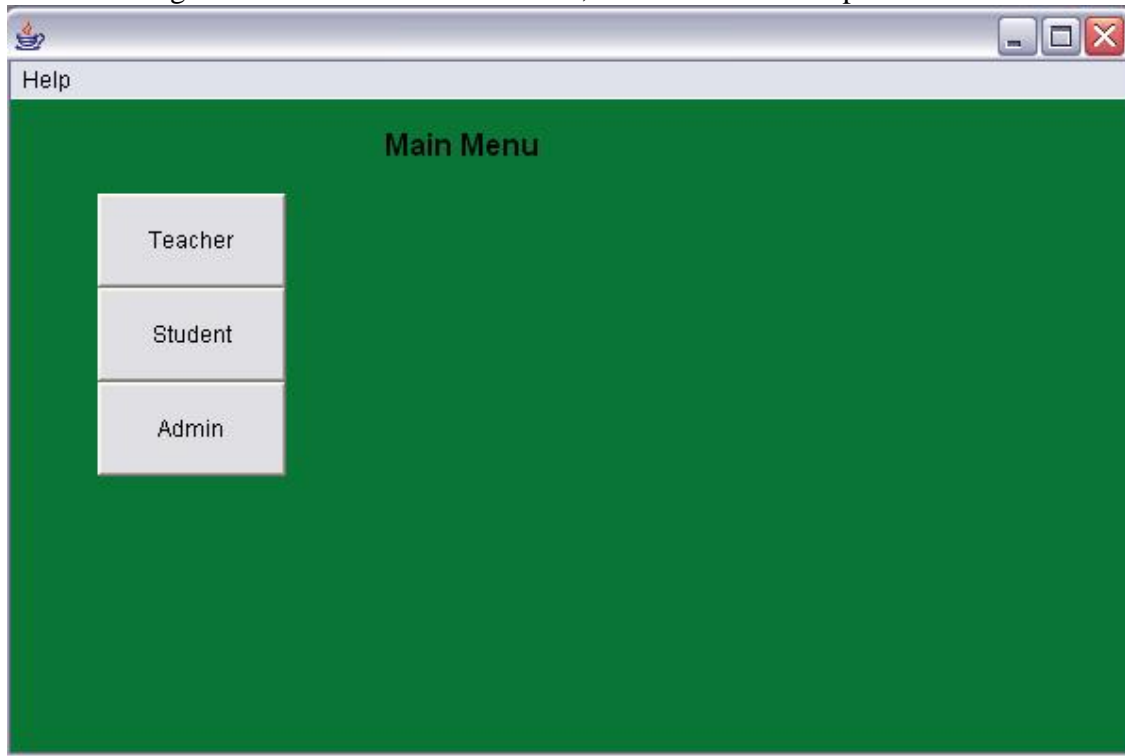
D16

Part D – Student Club and Organization

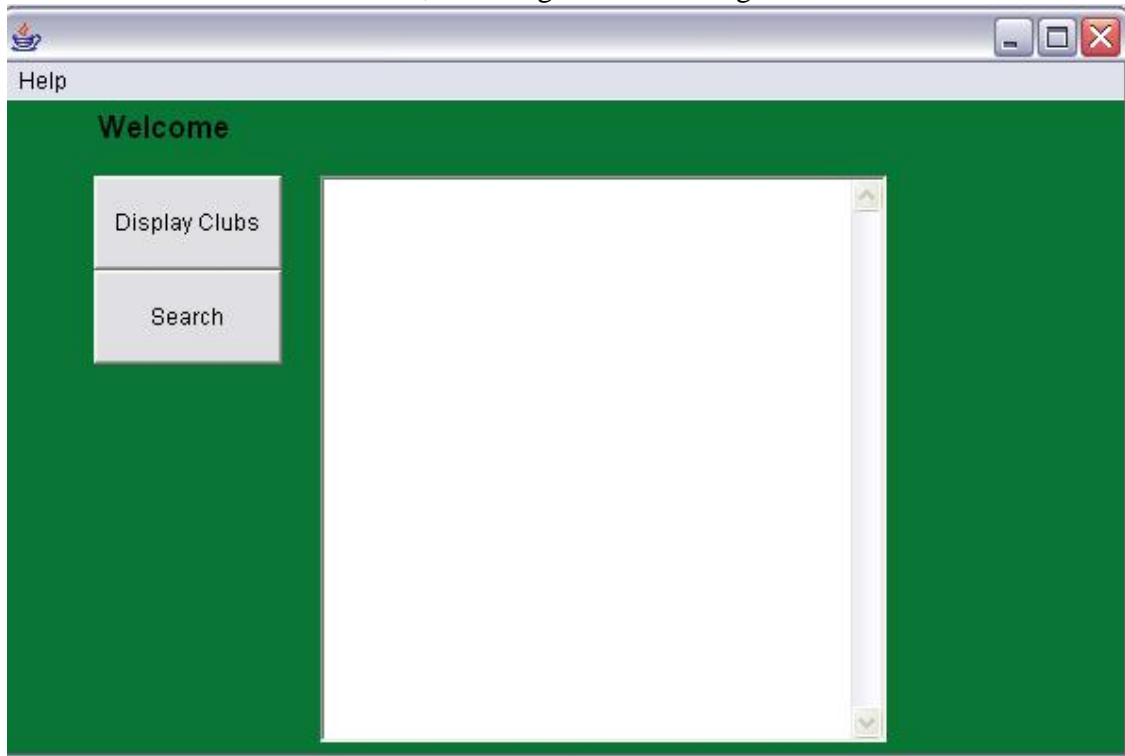
Documentation

The following screenshots are a regular run of the program that 3 different kind of users might have done (teacher, student and admin). Read the comments I have made to follow it through.

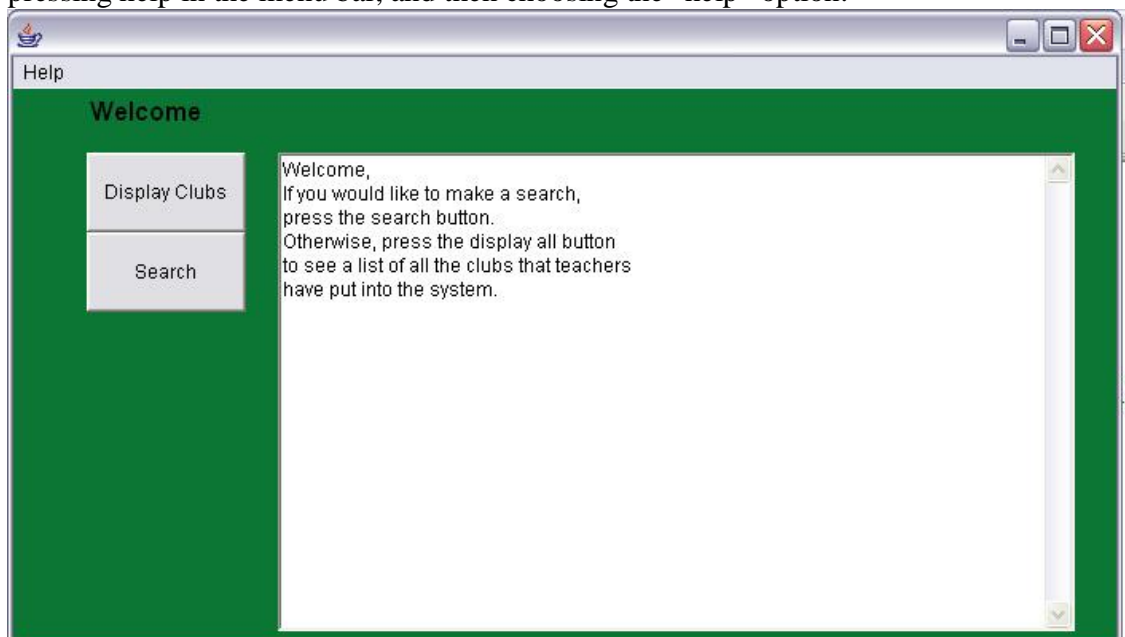
The following screenshot is of the main menu, also known as the Splash Screen.



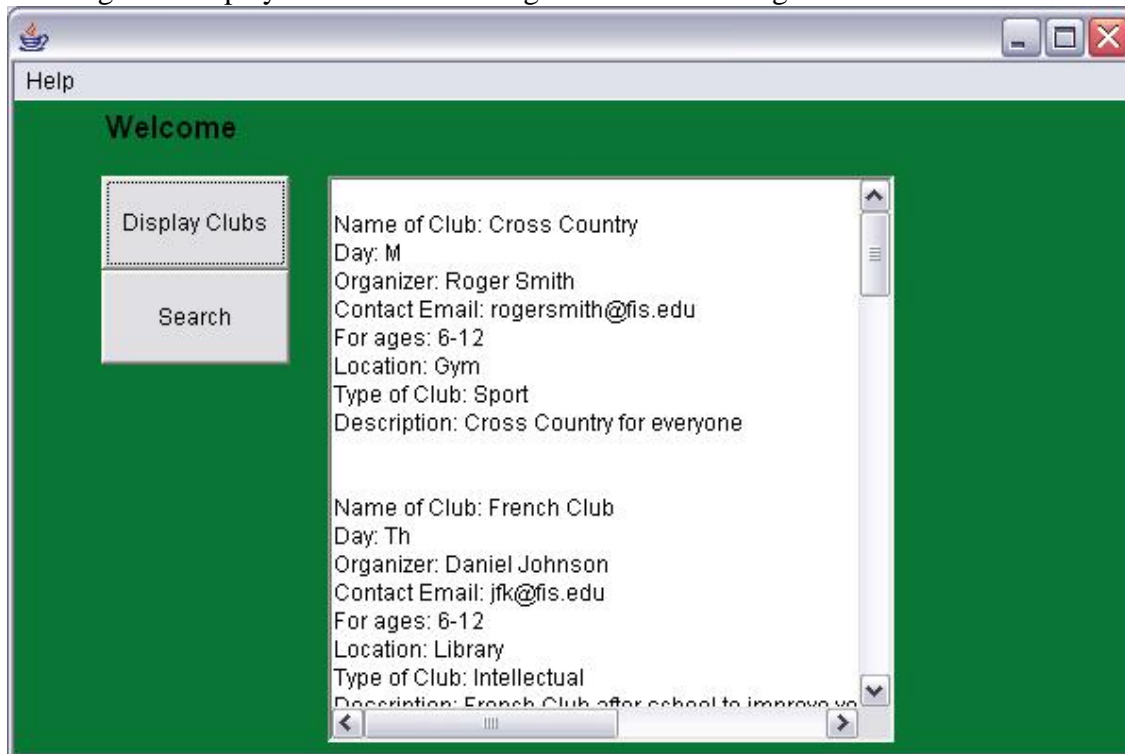
If we choose to enter as a student, we will get the following screen:



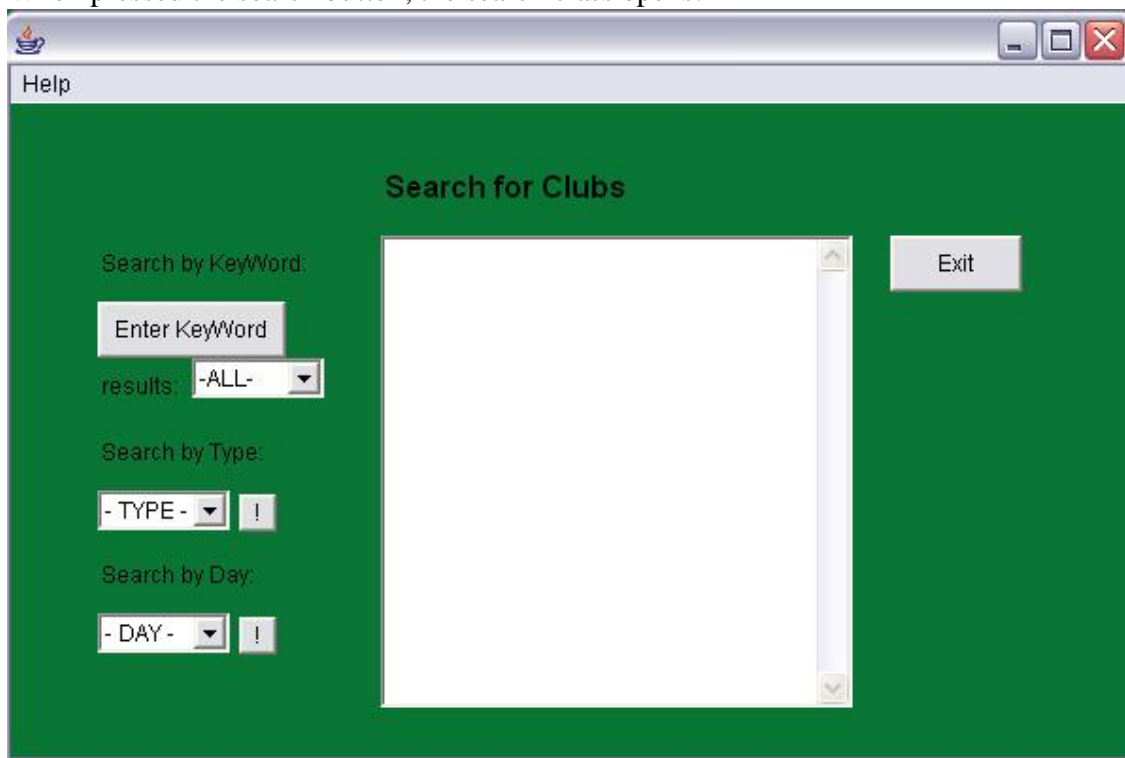
pressing help in the menu bar, and then choosing the “help” option:



Pressing the “Display Clubs” button will give us the following:

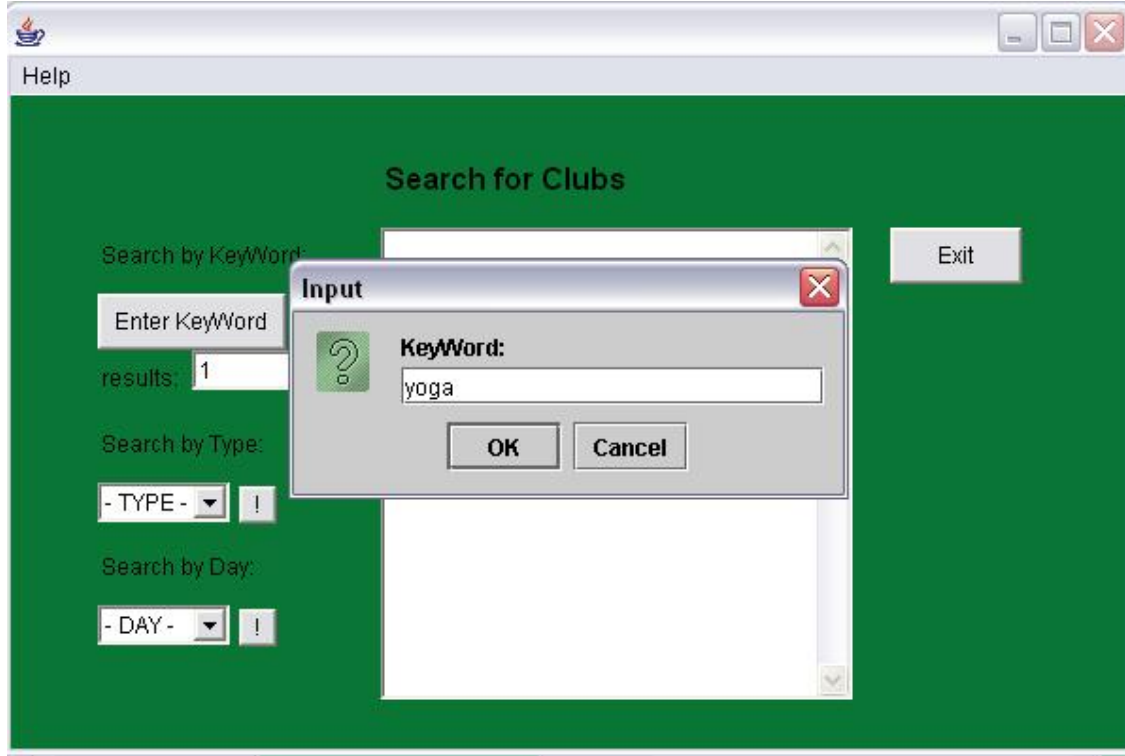


The clubs that are currently on the screen are in the random access file. When pressed the search button, the search class opens:

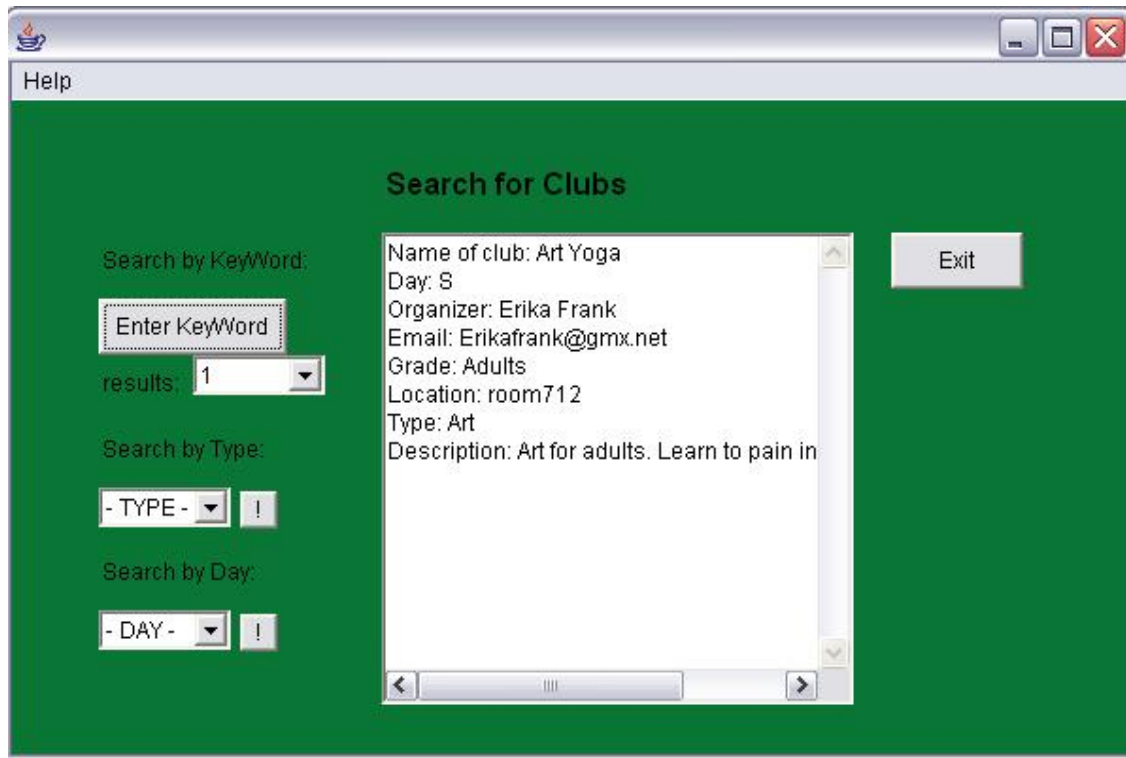


Here there are different ways of searching for a club. This class and different ways of searching fulfill my goal for creating a successful database in which students can search for club in A2 criteria.

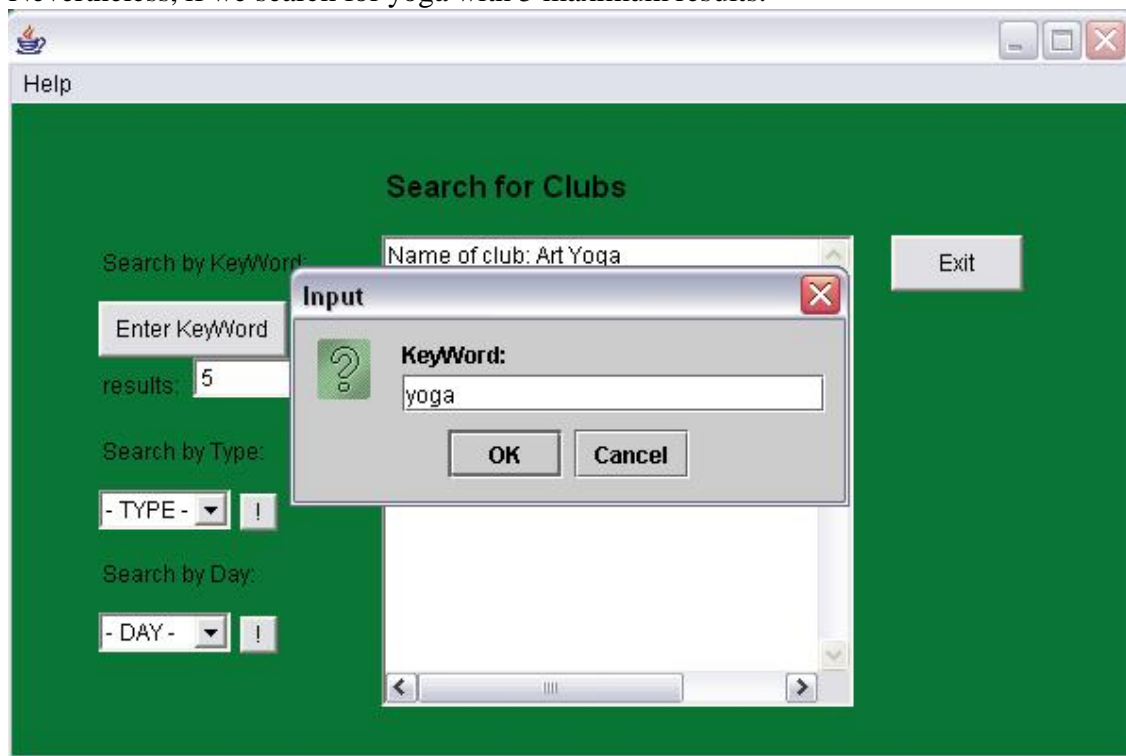
Searching by keyword, for the word “yoga”, with 1 maximum results gives the following:



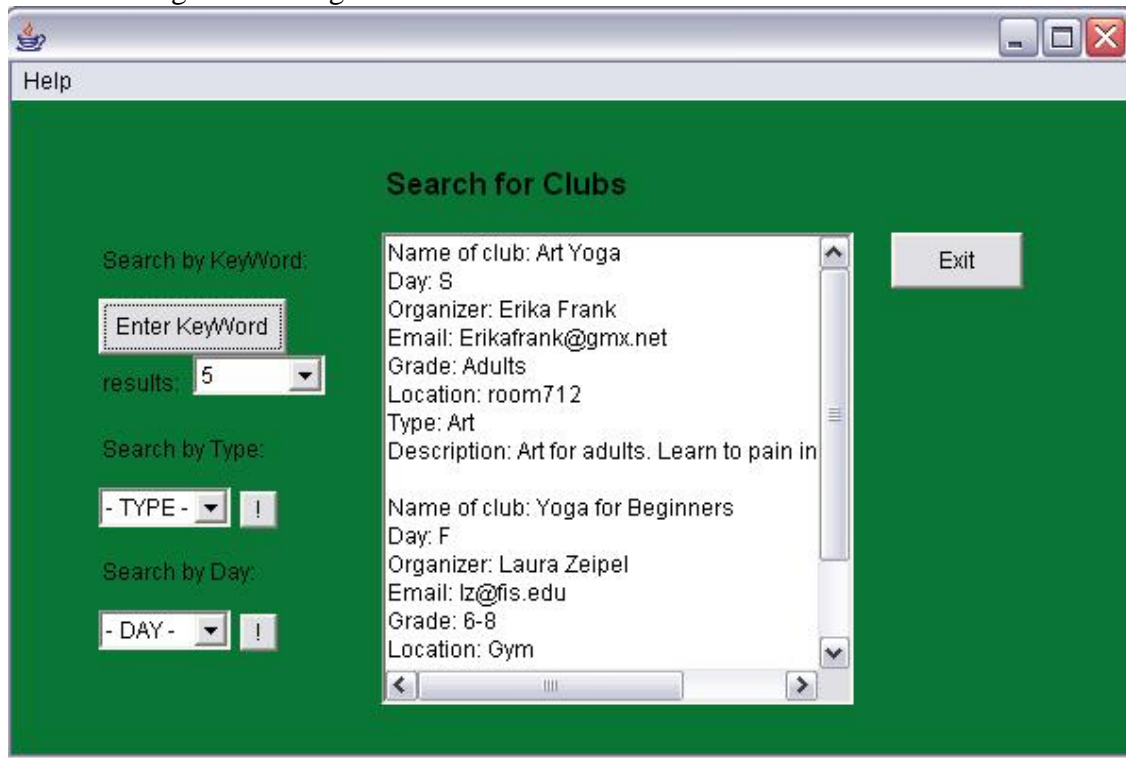
And the results:



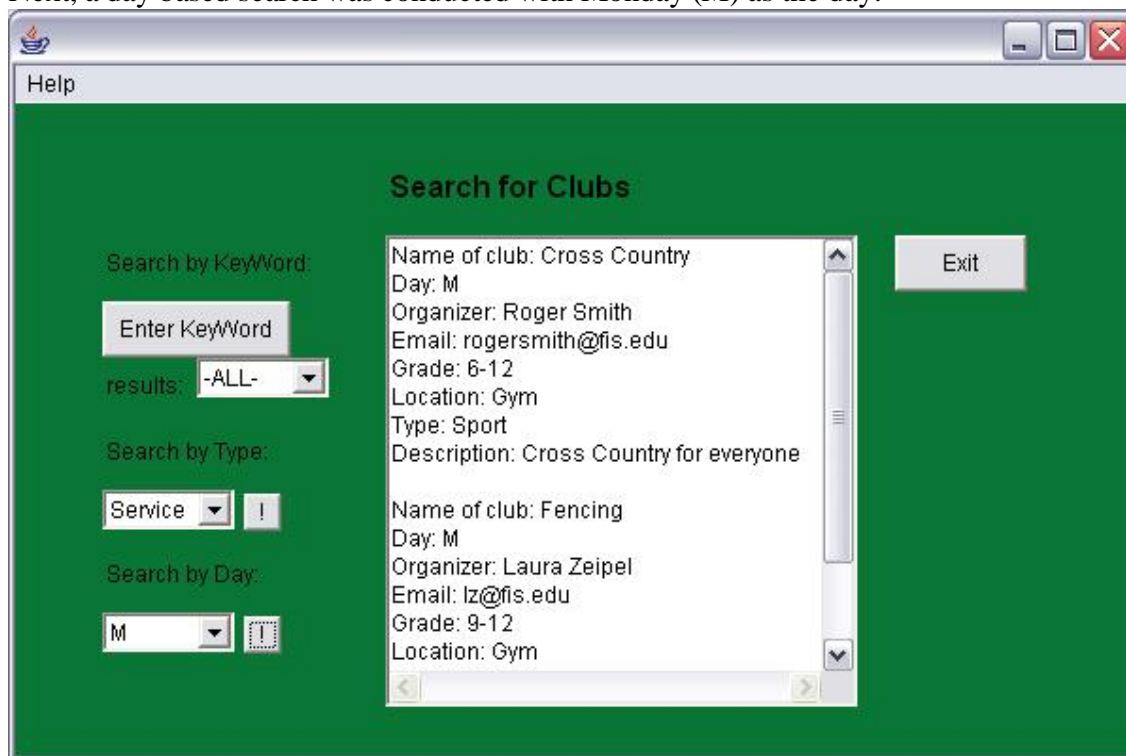
Nevertheless, if we search for yoga with 5 maximum results:



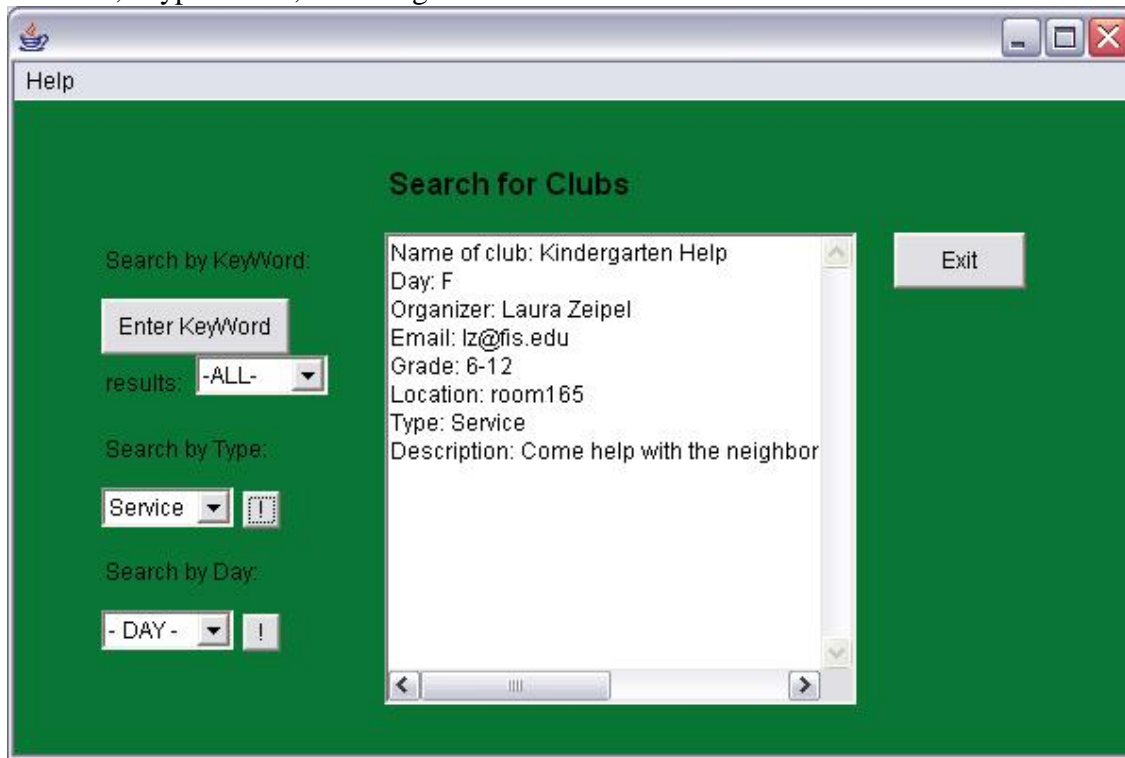
The following results are given:



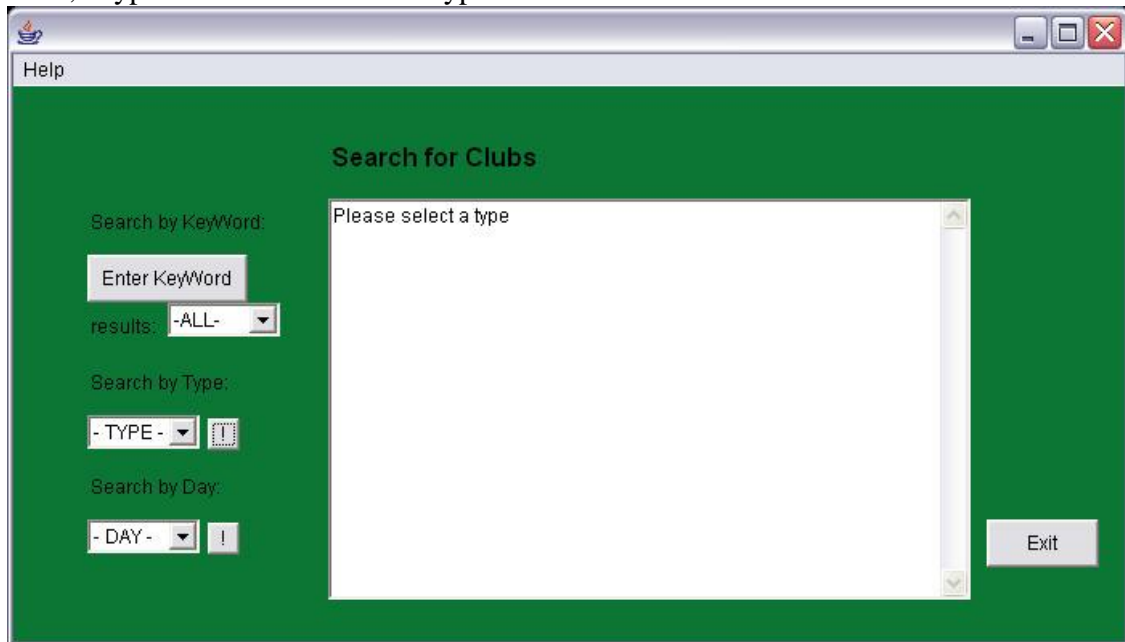
Notice how here there are more results now. Not necessarily 5 (as 5 is the MAXIMUM amounts of results possible), but as many with the corresponding keyword in their record. Next, a day based search was conducted with Monday (M) as the day:



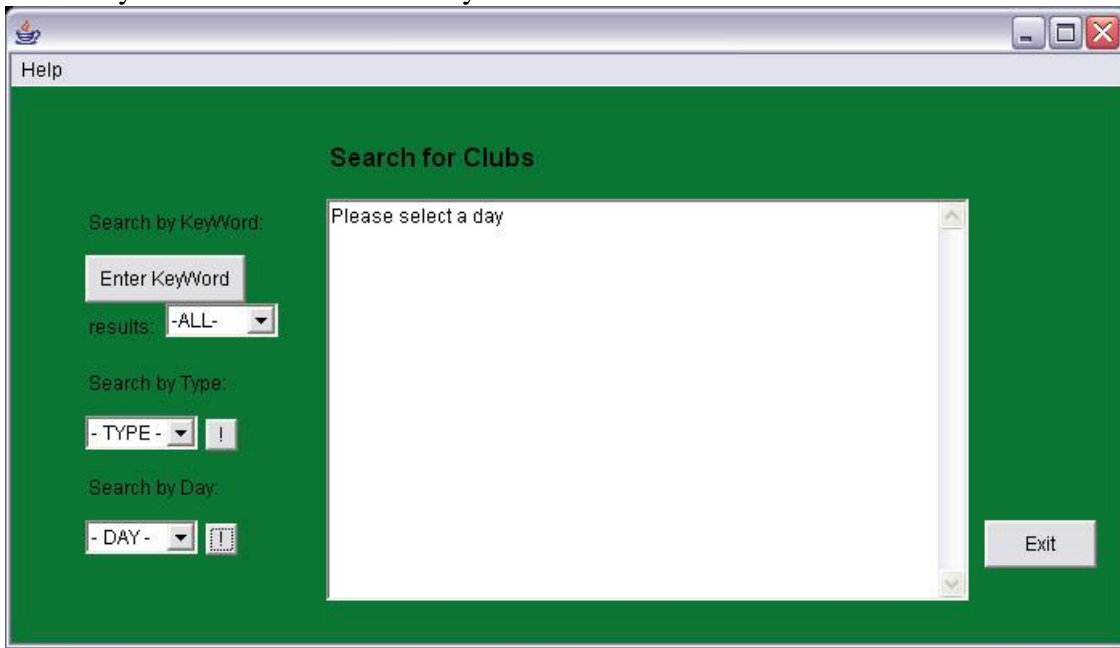
and next, a type search, searching for clubs based on service:



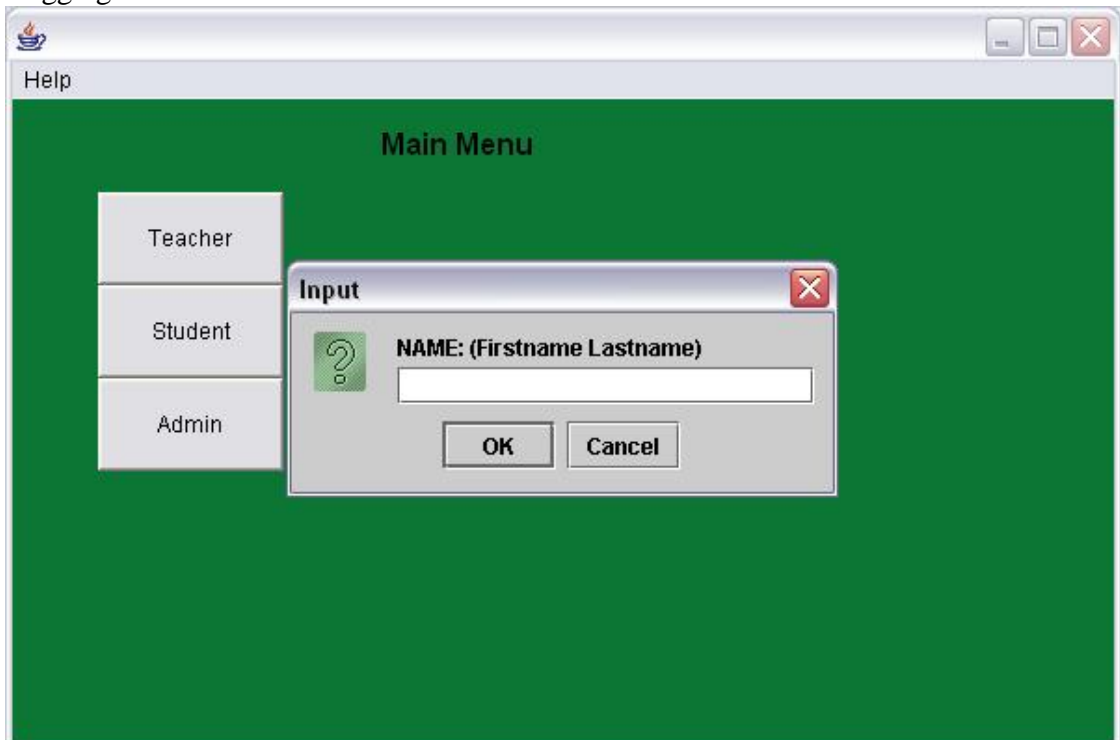
Next, a type based search with no type chosen:



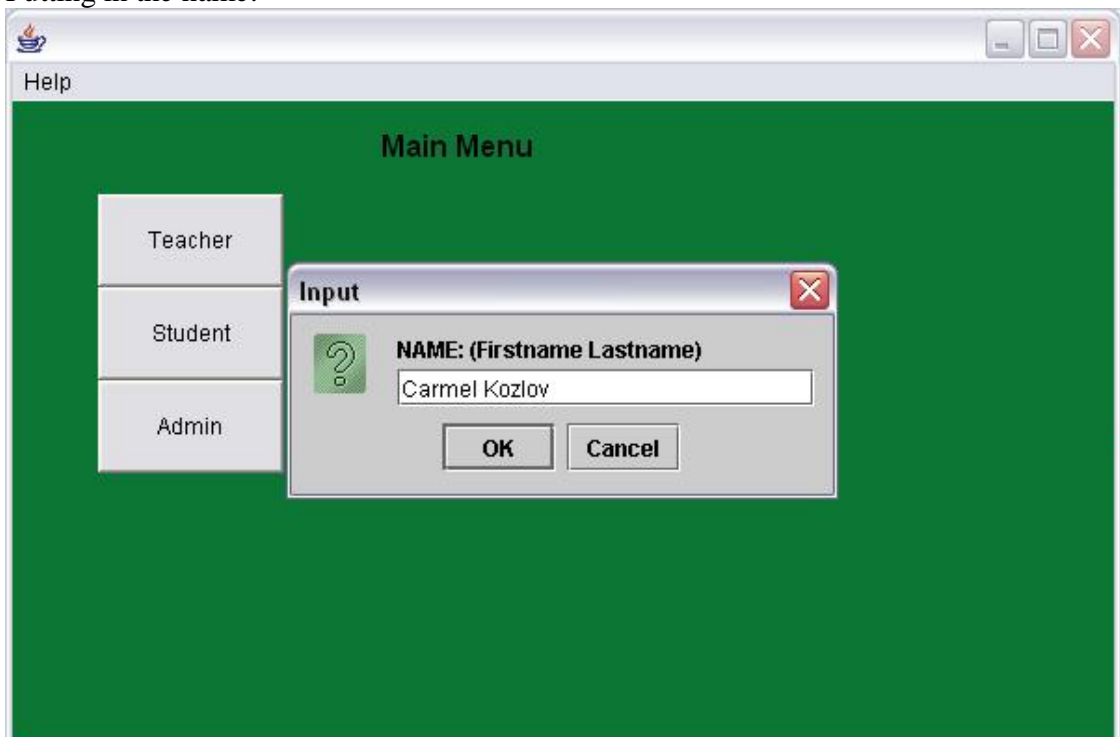
And a day based search without a day chosen:



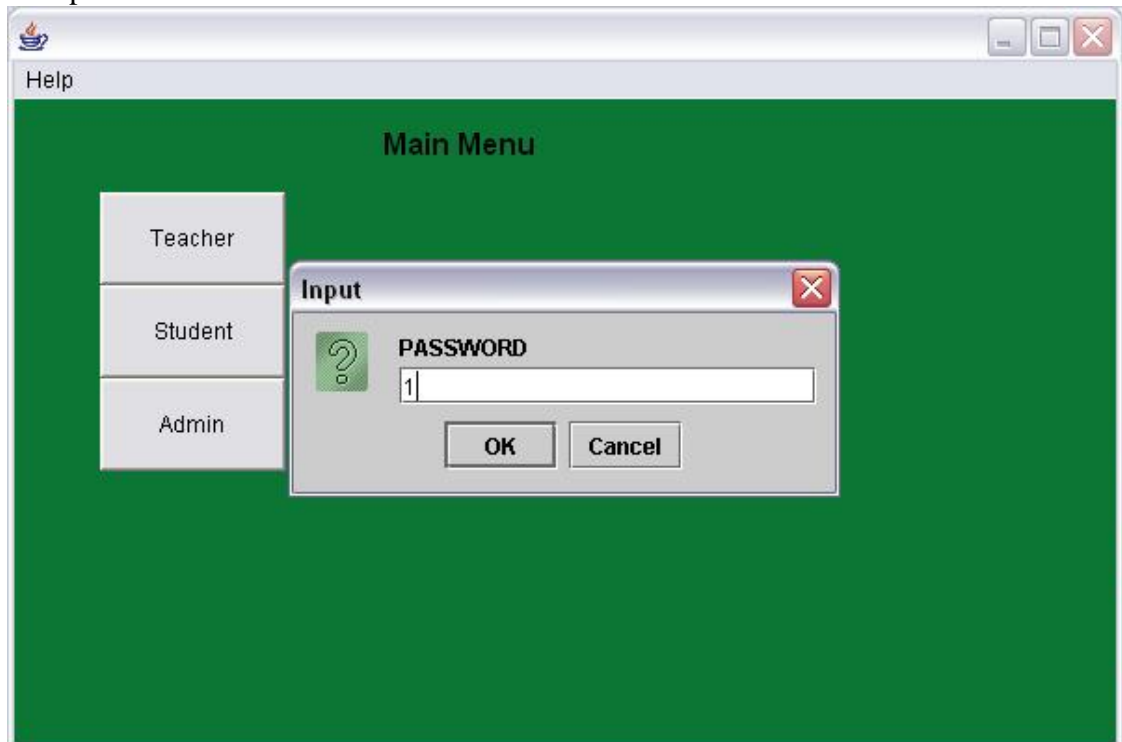
Logging in as a teacher:



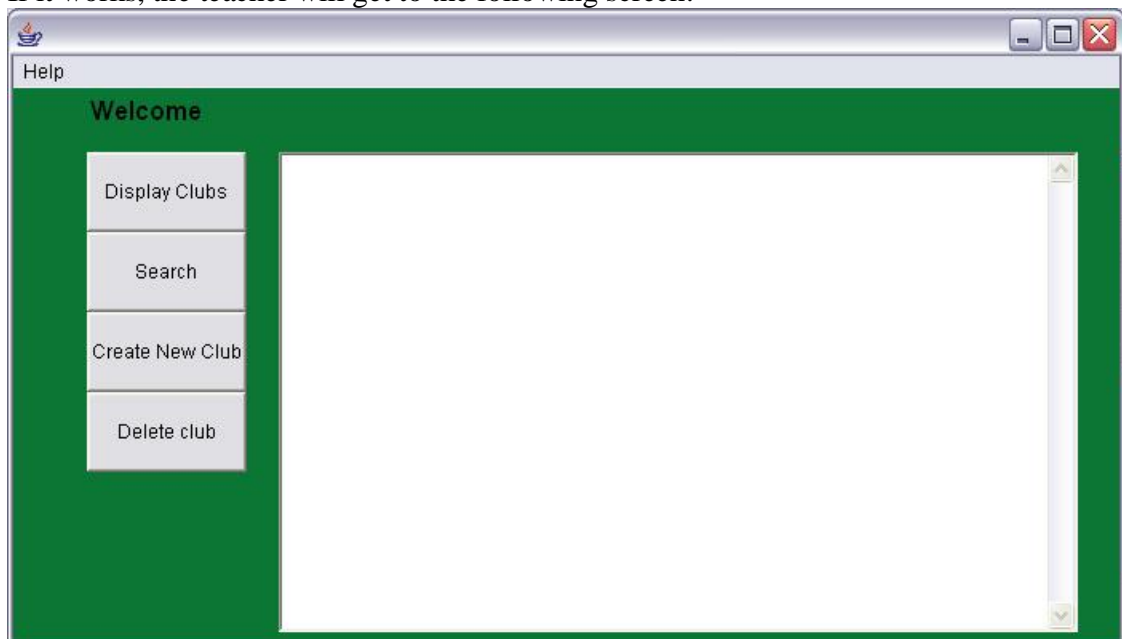
Putting in the name:



And password:



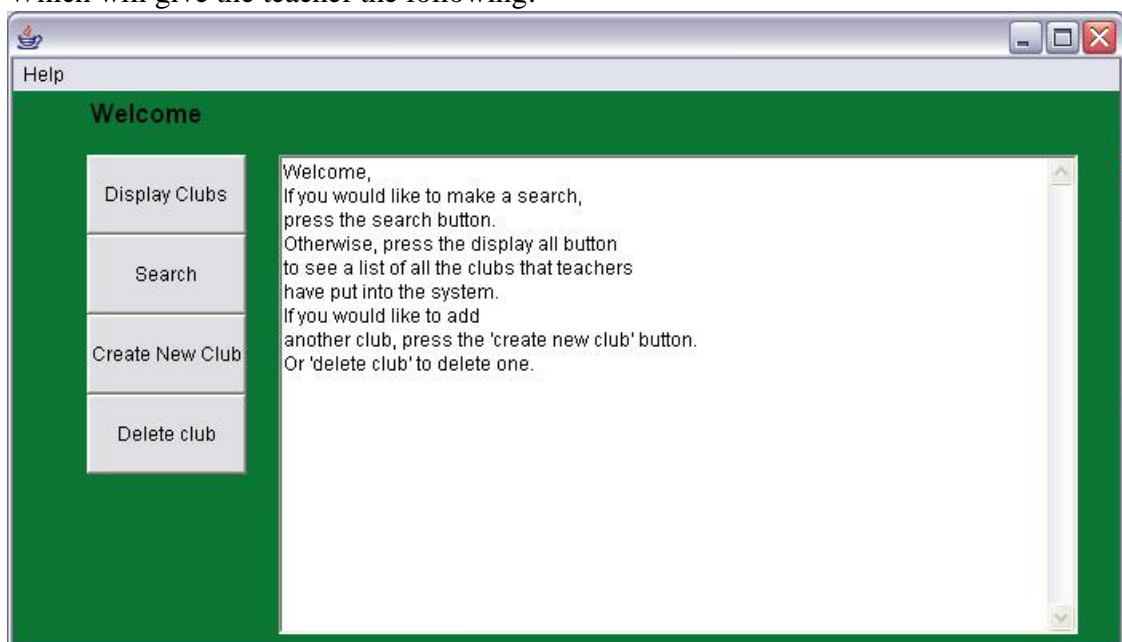
If it works, the teacher will get to the following screen:



Next, the teacher can use the help file:



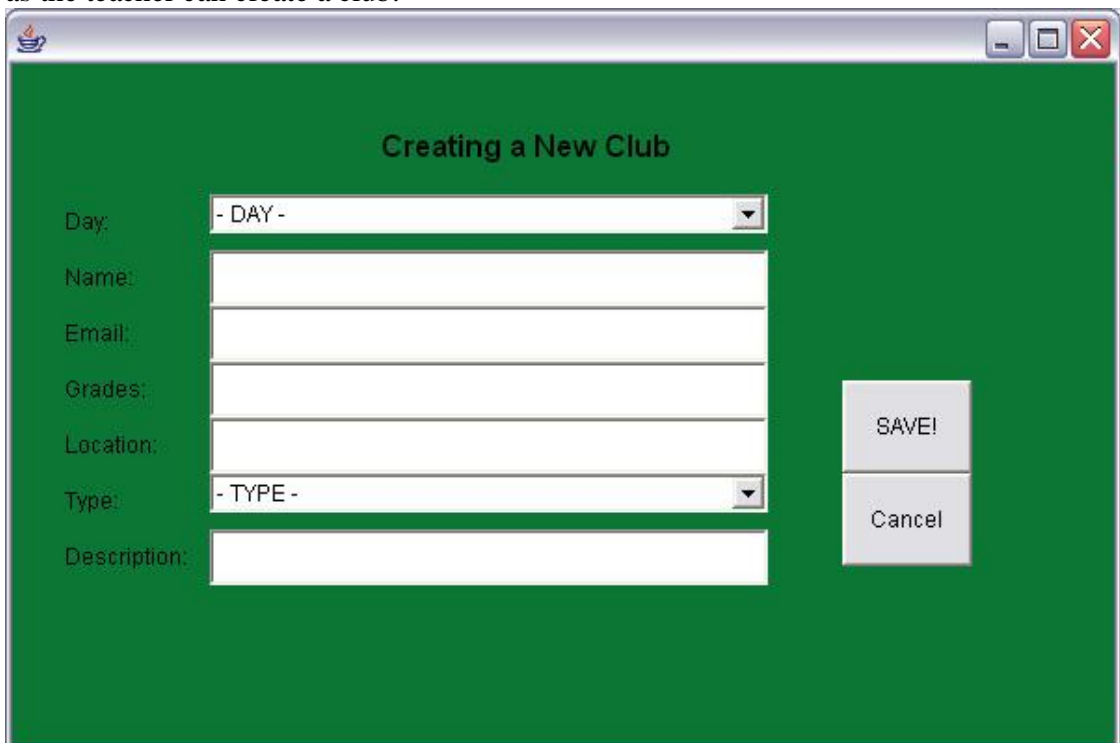
Which will give the teacher the following:



Displaying the clubs for the teacher will look like this:



Next, the teacher might want to add a new club, this again reflects my goal for section A2, as the teacher can create a club:



And fill in the club:

The screenshot shows a window titled "Creating a New Club" with a green background. The form contains the following fields and values:

Day:	T
Name:	Chess
Email:	cakozlov@gmail.com
Grades:	11-12
Location:	Library
Type:	Intellectual
Description:	Chess for all people in grades 11-12

Buttons for "SAVE!" and "Cancel" are located on the right side of the form.

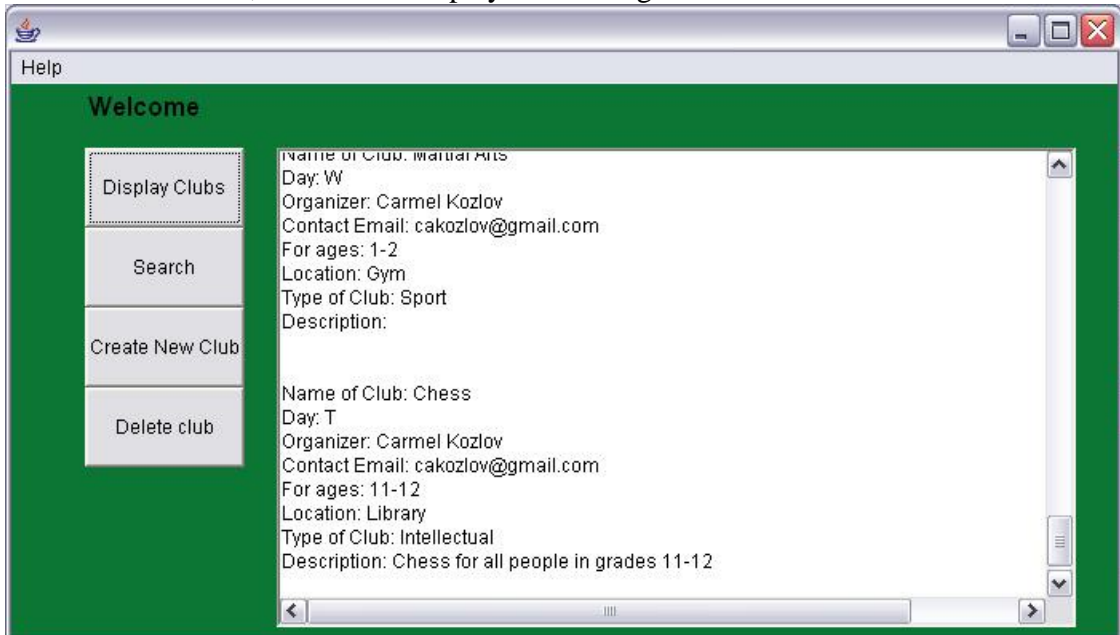
When the club has been created, a message appears:

The screenshot shows the same "Creating a New Club" window, but with a "Message" dialog box overlaid in the center. The dialog box contains the text "Your club has been added" and an "OK" button. The form fields in the background are now filled with different values:

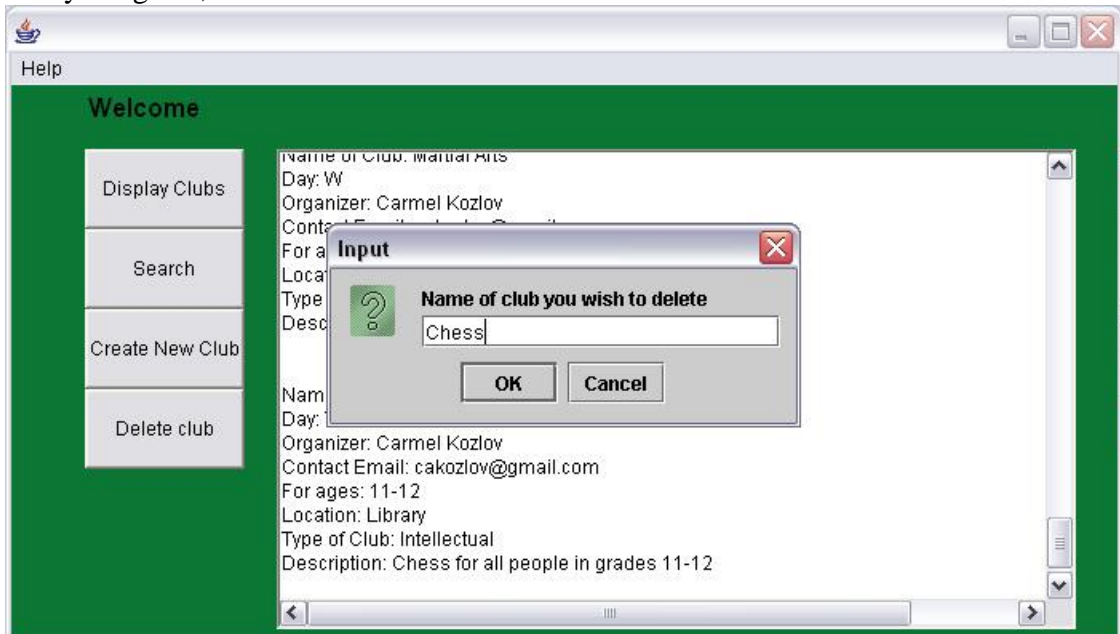
Day:	Th
Name:	RPG
Email:	georgetay
Grades:	10-12
Location:	room254
Type:	Other
Description:	Come and play D&D

The "SAVE!" and "Cancel" buttons are still visible on the right side of the form.

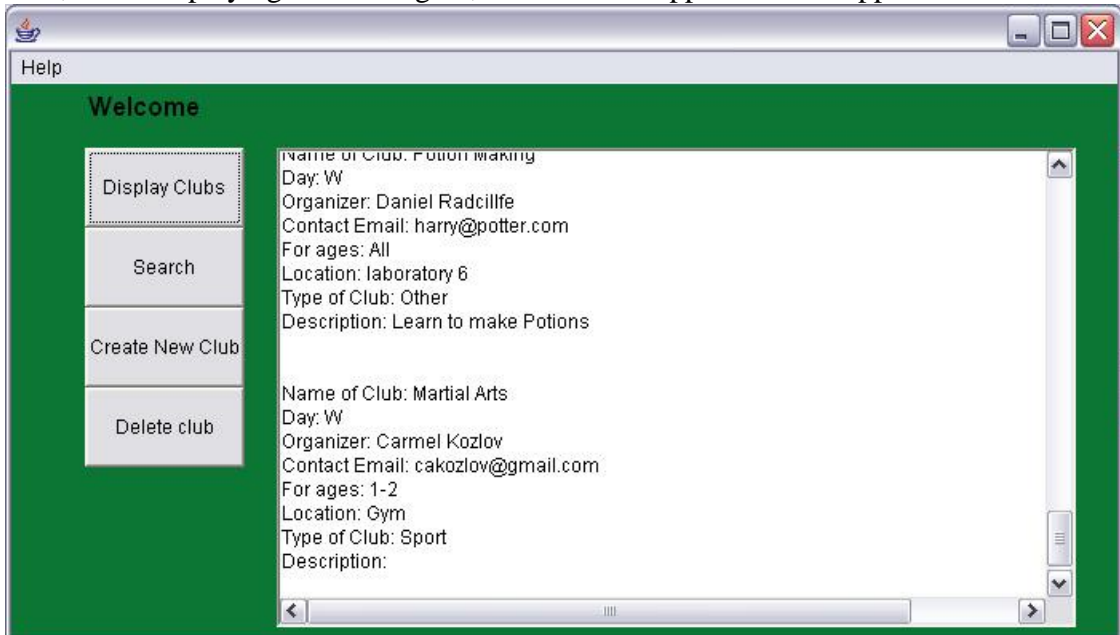
When club is saved, teacher can display all clubs against and look in the end of the list:



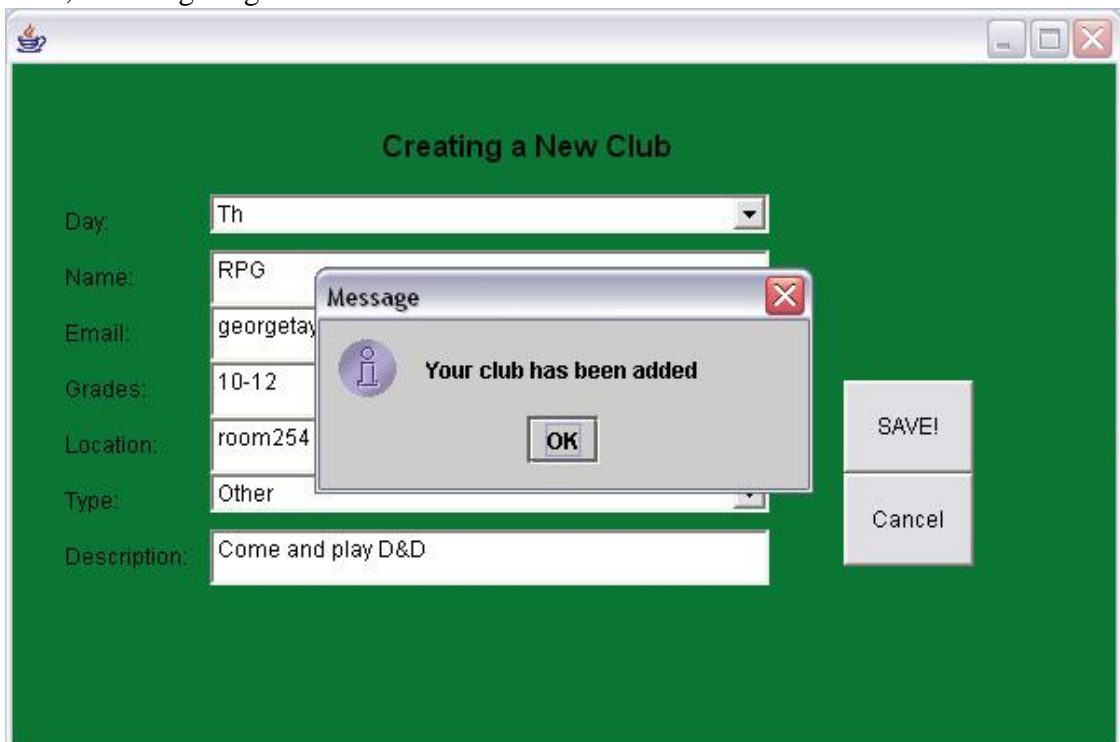
Now, the teacher will delete the club they have just created, which again reflect another of my A2 goals, as the teacher can delete a club:



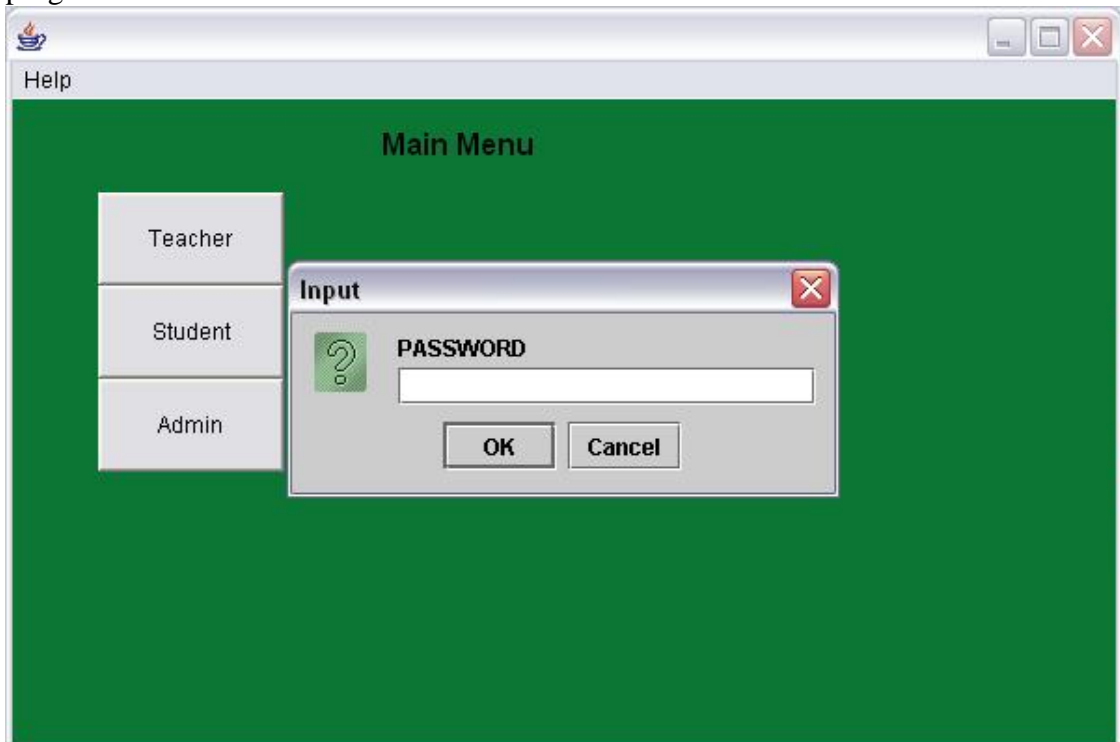
And, while displaying all clubs again, the chess has appeared to disappear:



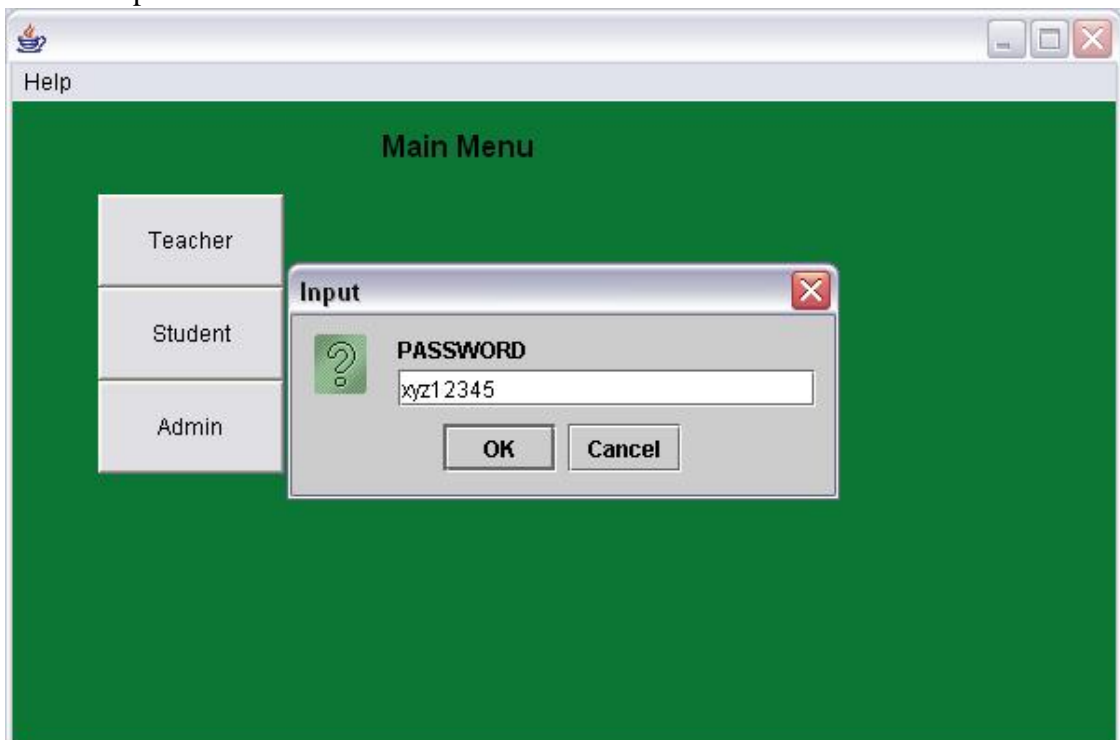
Also, a message is given out when a club is deleted:



Admin has to log-in, the admin class fills my goal in A2 for the administration of the program:



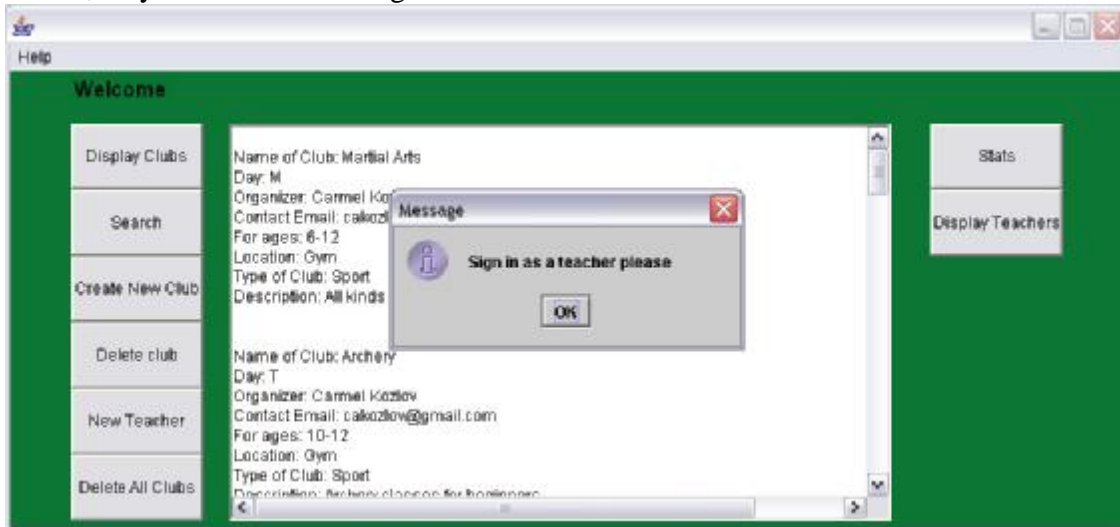
Put in the password:



This is how the admin menu looks like:



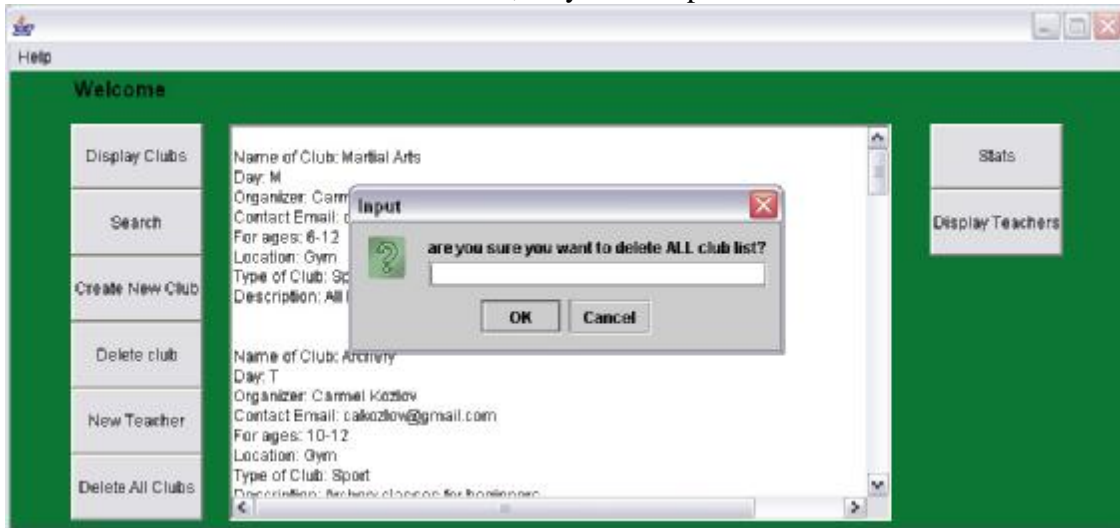
Admin cannot create a new club, only teachers can, so when they press the new club button, they are reminded to log in as a teacher:



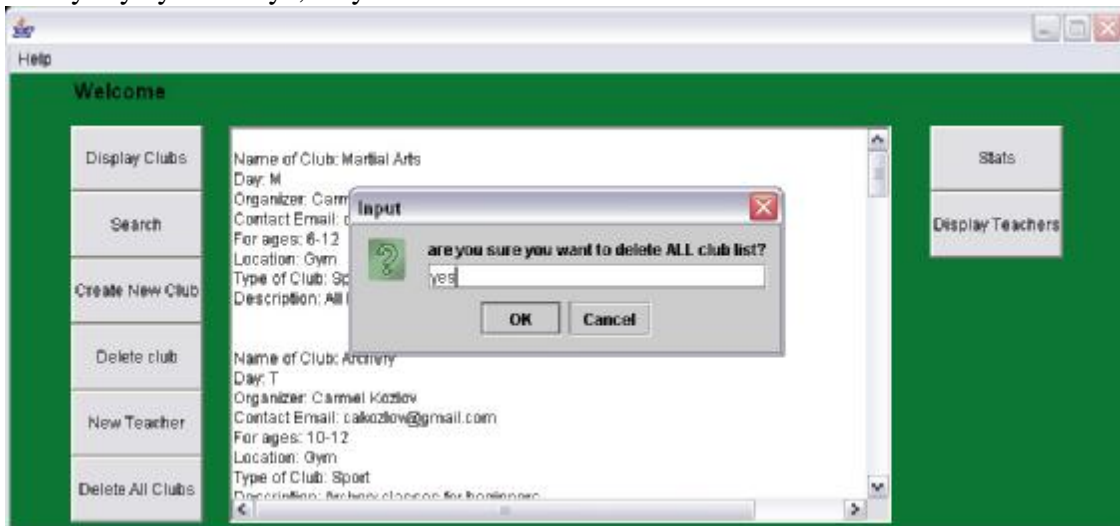
therefore, they cannot delete clubs either:



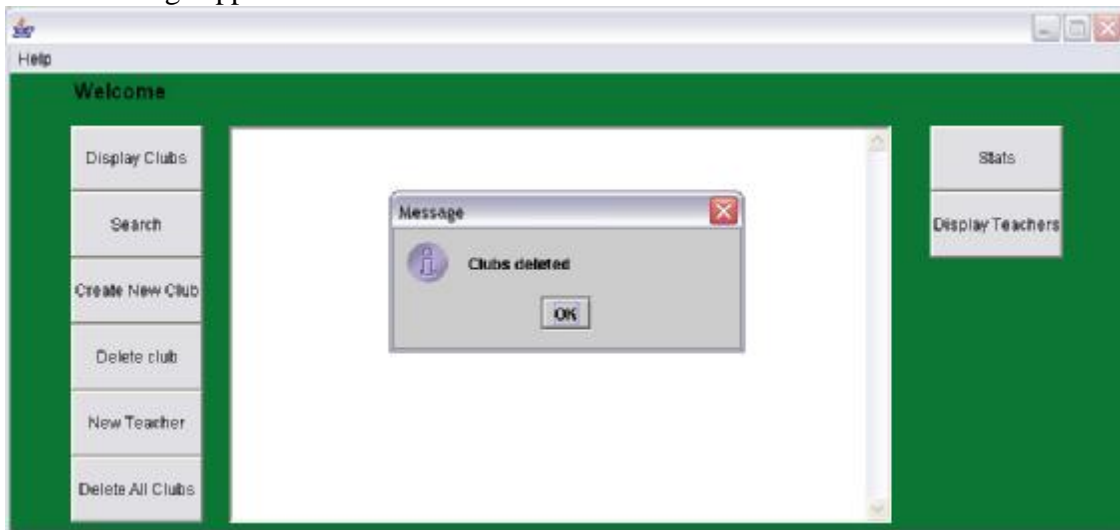
If the admin wants to delete all the clubs, they need to press the delete all clubs button:



if they say “yes” or “y”, they will delete all the clubs:



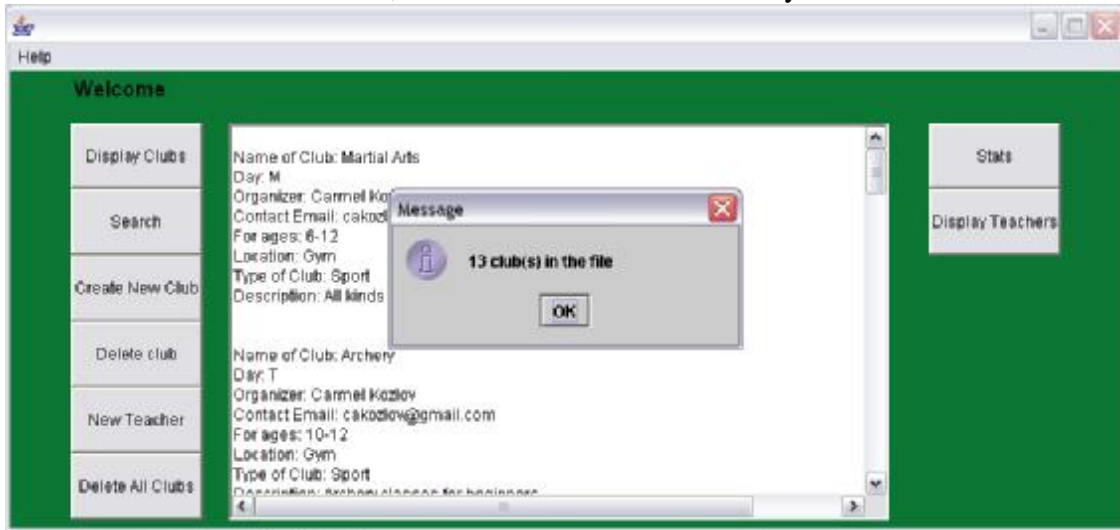
And a message appears:



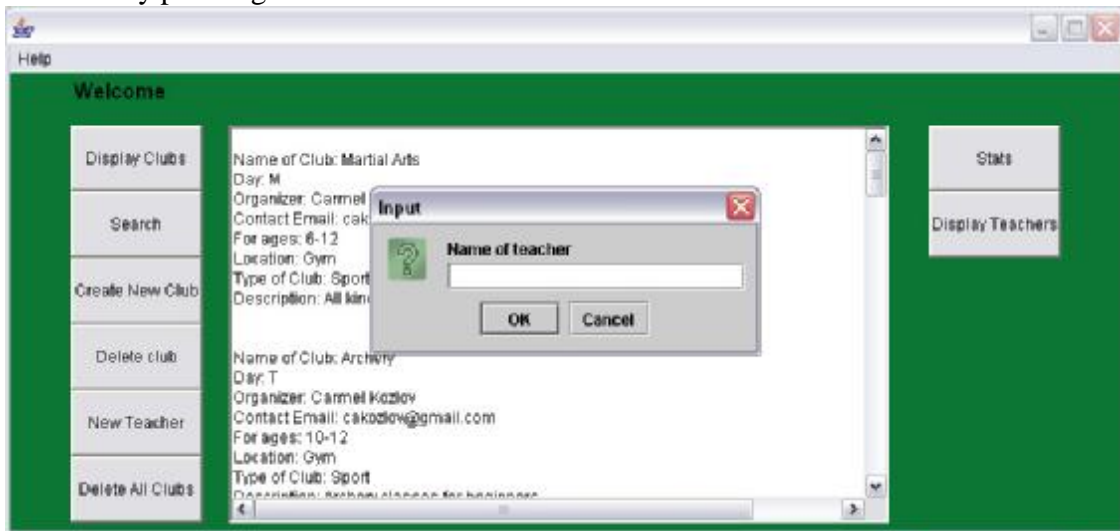
Thereafter, pressing the display clubs button will give the admin this:



when there are clubs in the file, the admin can check how many clubs there are:



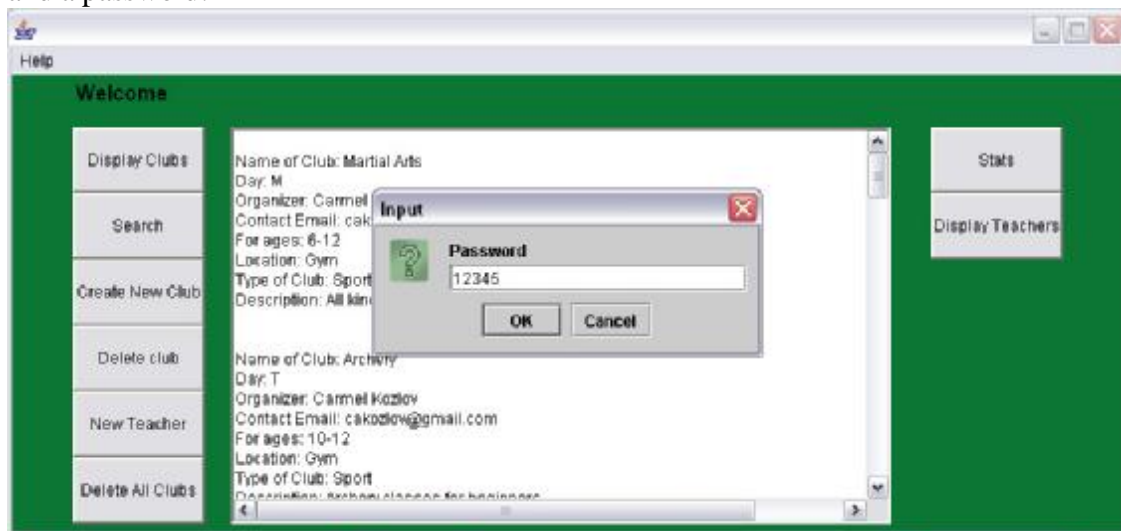
in order for more teachers to add clubs, they need to have an account. The admin can add accounts by pressing the new teacher button:



then, by entering a name:



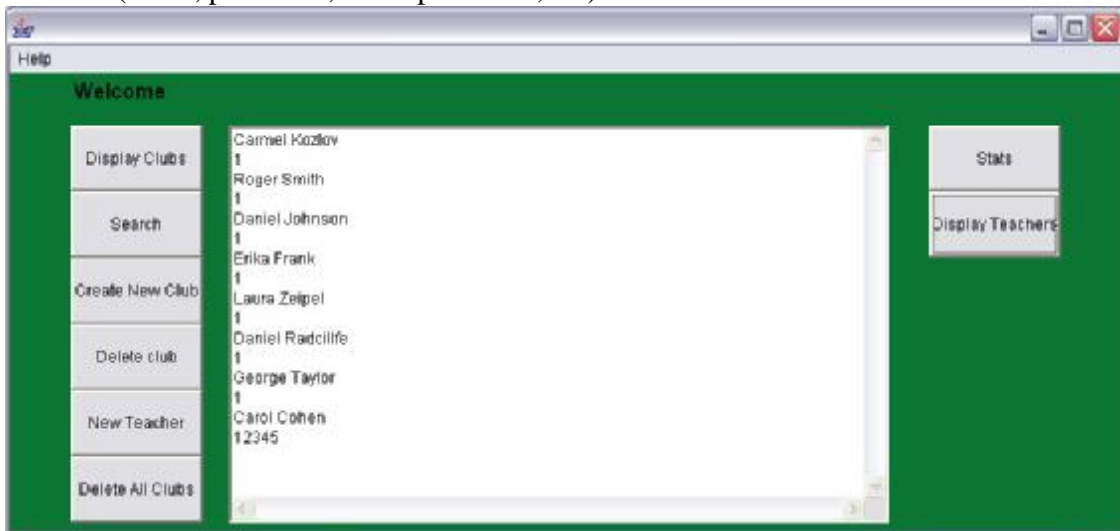
and a password:



They then need to retype the password:



Finally, they can view the entire teacher profiles and their passwords by displaying the teachers (name, password, name password, etc):



Evaluating Solutions

Did my program work?

My program worked. Surprisingly enough I found that it worked and that I actually did a good job with it. The random access file, while at first was a little difficult to deal with, worked. Clubs can be saved, deleted and loaded. The admin can create new teacher profiles and all of the clubs data can be put on a network and accessed by different computers.

Did my program address the criteria for success?

My program addressed the criteria for success. I have felt that I have completed 11 mastery aspects like I addressed it in part B. after analyzing the program I had managed to find a solution which worked, which in my opinion was a big part of the criteria for success. Not only that, I managed to do so with a detailed design stage which I followed through and actually helped me a lot. The end user was also satisfied with the end result and it was helpful to have him during the various stages of my program.

For my goals, here are the recap and my explanation:

A condense and effective database of all clubs and organizations in our school

Once the program will be used effectively, there will be a condense and effective database of all clubs and organizations in our school. Since I have to input example data when running the program, I am already starting to do this.

Effective access and use of program in order to find out information about clubs and organizations

I've used GUI and a reasonable user interface in order to make the program effective to use. To search and find clubs only takes a few steps, each consisting of a click or two. Therefore, I feel that this criteria has been achieved

Useful search options, which will allow users to find clubs uncomplicatedly

My goal was to have 3 kinds of searches – day based search, keyword based search and a type based search. I have successfully achieved these. In addition to achieving these, they are also very efficient as they are very quick algorithms and there is no waiting time for the user.

Teachers will be able to add their own new clubs

Teachers are in fact able to add their own clubs. Not only that, teachers have their own profile with their own personalized password.

Teachers are able to delete their club

The program can detect which teacher added what club, therefore, if a teacher who didn't create a specific club wants to delete it, they will not be able to do so. Nevertheless, if they want to delete the club they have created, they can do so.

I feel that this is one of the most important features of the program since it not only took me a while to figure out and when it worked I was very happy, but also because it prevents vandalism and creates order in my program. It is a security issue which nowadays in real life software programming is a very important issue.

Administration of the program

There is a whole admin class. I felt that this was a good achievement. It's also a security issue as they are the only ones able to create new teacher profiles.

Did it work for the same data sets but not for others?

The program has two data sets. The club file data set and the data set for the teacher profiles. If the user (Teacher or Admin in this case) decide to put something into the input values that does not work, the program will usually reject it. That is, if for example a teacher decides to create a new club but puts in random names and emails that make no sense, the program will not be able to detect it. Nevertheless if the teacher puts too many words into the description and then doesn't fill out some other information the program will not accept it.

The admin can only put in names of teachers and their passwords. Therefore, the admin might also put in names of teacher who do not exist. The program cannot check if the names are in fact correct.

It would be too complex and beyond my time and resources to create a program that checks for the validity of the above mentioned things. The program was made with the intention of it benefiting the school community and therefore, also expecting the users who use it to have some common sense and not vandalize it.

Does the program in its current form have any limitations?

Yes. Like many software these days, my program is not 100% fully completed and there are limitations.

First of all, and the most important in my opinion, is the fact that you cannot sort the clubs. This feature has been missing but bothering me the whole time. Frankly, it was not 100% necessary in order to complete the program, but it was on my list of things to do in case there is extra time and resources to allow it. I didn't get the chance to program this part, and while the program does work without it, it would have benefited the user a whole lot.

In addition, it will accept a lot of values for many of the inputs. Something, that again, would have taken me a lot of time to program since there are so many input options (especially when creating new clubs). Nevertheless, it is up to the users not to vandalize and use their common sense when working with the program.

What additional features could the program have?

As I mentioned, better input rejection and a sorting algorithm (or perhaps a few sorting algorithms). If this could be done in the revision of the program in the future, I would consider the program to be pretty much in perfect condition for the use in the school.

A log file of when teachers have added clubs or deleted so that the admin can look over it. Therefore, a function which will let the admin to see over time how many clubs have been added, and will have more statistical information rather than just the amount of clubs in the file at the moment.

Also, allowing the admin to delete teacher profiles or change their passwords and an algorithm that will sort them in an alphabetical order in the text file.

Was the initial design appropriate?

It appears that the design was in fact appropriate. Perhaps the only thing that could have been done better was the way the clubs were presented. By that I mean that perhaps not everyone will have the time and will put any effort into scrolling up and down, but there

was no real way to avoid that using java. That is, I couldn't think of a better method to present the clubs other than the text areas. In addition, I didn't want the program to be too big when it runs. I didn't want it to cover the whole screen as I felt that was unnecessary and a lot of people would not like it. Therefore, I kept it as small as possible, considering the fact that the clubs need to appear. It is possible, in the future, if requested by many users to make the screen and the text areas bigger if they feel it is necessary. Since the computer science internal assessment doesn't have a part in which we let many people use the program over a longer period of time and assess it, I could not get response from many people over the design and could only refer back to my opinion and the end users opinion. In a program like mine that will be used by the whole school community. this is not good as so many people use the program. In the future therefore, I would let people give feedback on the design, even though I tried to make it as appropriate as possible.

Searching

The searching algorithms in my main program differ a little bit. Take for example my day based search. It goes through the whole file, but it accesses for every record, straight to the place where the day is stored and compares it with that the user has chosen. Therefore, it is a very quick search. Even if there were over 200 clubs in the file, it would not take long to do the search. This works the same way for the type based search. For the keyword search the algorithm creates a long string out of every record and see if the string entered by the user matches anything from that long string the algorithm produced. This doesn't take long either. I felt therefore that my searching algorithm was efficient as searching algorithms usually are with random access files.

Alternative Approaches

I felt that my design was most appropriate for the program. The design was based on an object oriented approach.

There are in fact differences (and more features) between the initial prototype in part A and the end result in part D. obviously, since ideas develop further once I have actually started working on the program and realizing what I can and cannot do.

I feel that for my search class it wasn't as effective as I wanted it to be.

Perhaps a filter based search would have been more successful, in which all clubs are displayed and the user chooses filters which make the clubs that do not fit in those categories disappear of the screen and the user ends up with a few clubs that fit the filters they have chosen. Some of these filters could have been: location, day, time, grades, etc. but instead I chose a search option of 3 different searches.

Nevertheless, this could have been a different approach that might have even been more successful than the one I have chosen. Other than that, as I have already mentioned, a lot of parts of my program would have been more successful with the addition of sorting algorithms.

User Documentation

For the program to run on a computer:

Windows XP, 512MB RAM, at least 1GB free of disk storage

Connection to local Frankfurt International School Intranet and servers

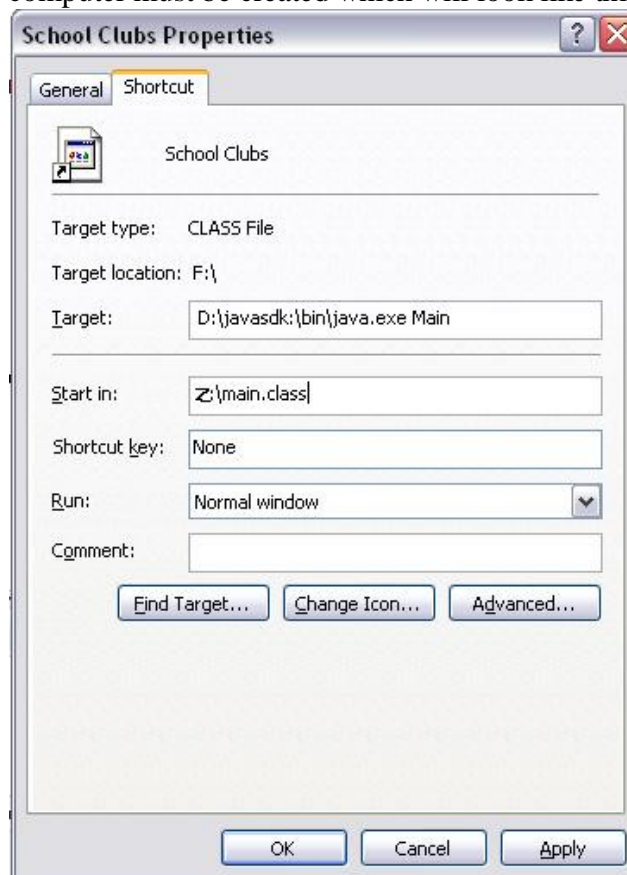
Java version 1.4 must be installed on the computer.

Admin must:

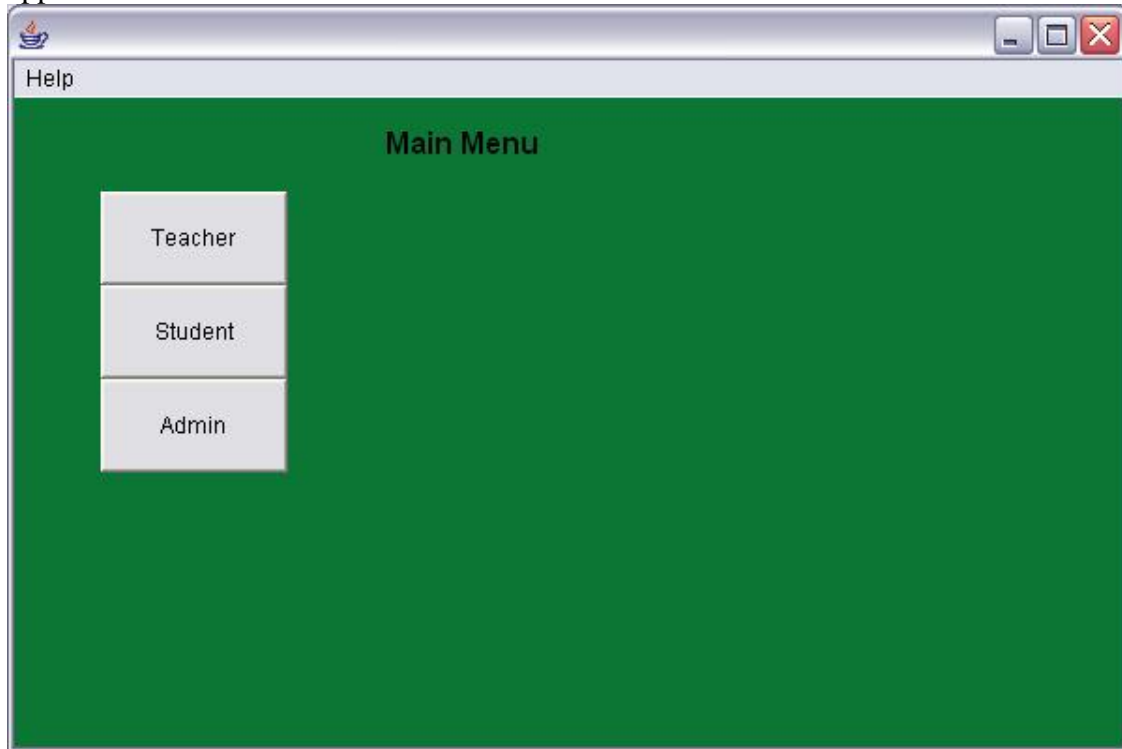
Create/copy the club data file and the program code into the Z: drive in the local school server, in the following location: Z:\CarmelKozlov\

The clubs file data must be called clubs.dat and the teacher profile text file must be called teacherprofiles.txt.

A shortcut on each computer must be created which will look like this:



The admin should check that each shortcut works on the computer, and that when he runs the shortcut the following screen appears:



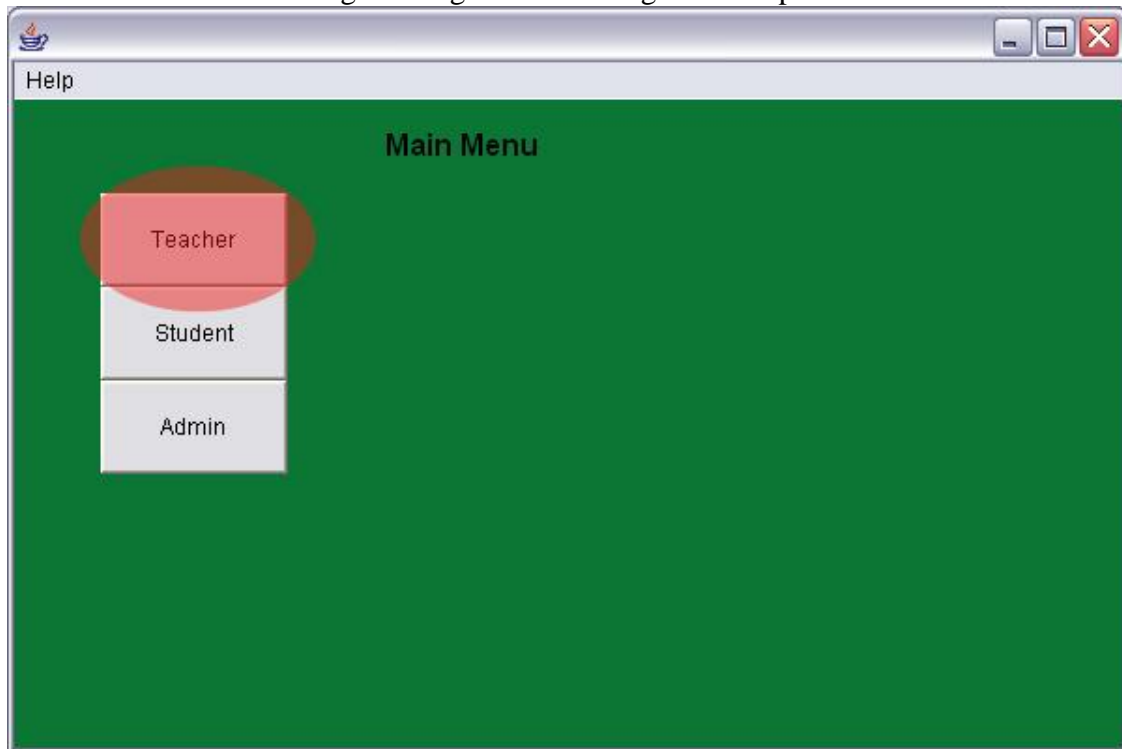
Then, the admin must create teacher profiles so that they can add clubs. After logging in with the given password for the admin (which is set to xyz12345), the admin must create new teacher profiles:



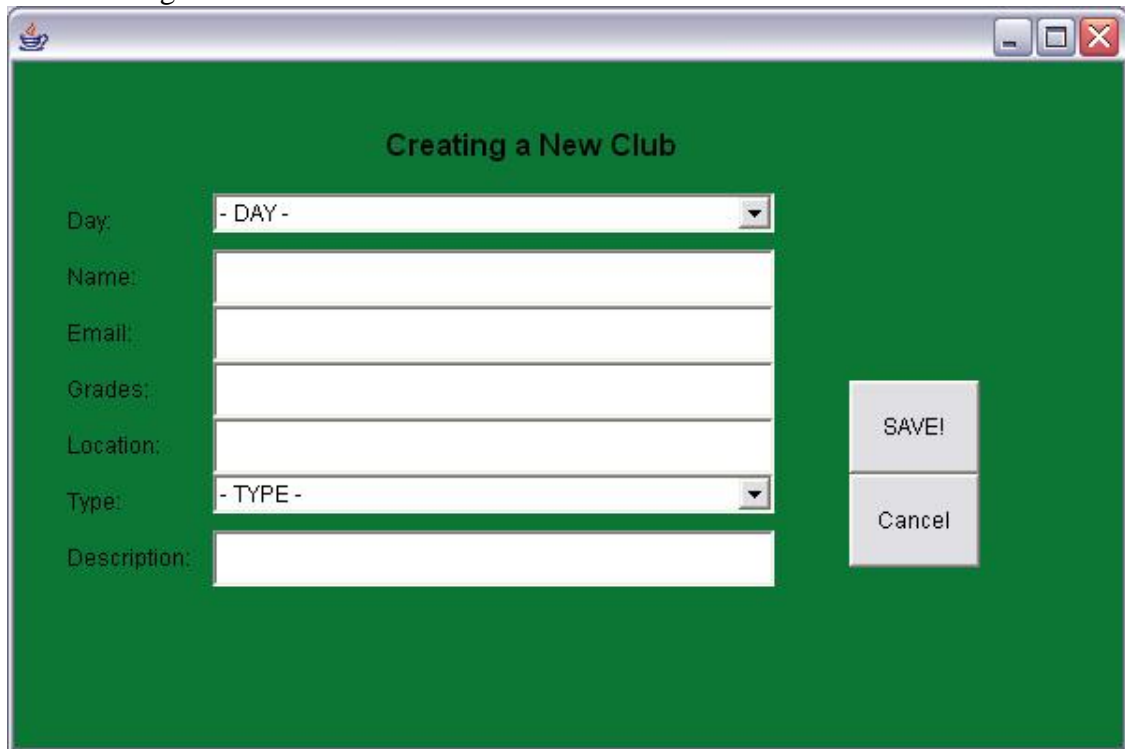
When there are a few teacher profiles, the admin can look at all of them and see their passwords:



And the teachers should log-in using the teacher login in the splash screen:



Teachers can then create new clubs by pressing the create new club button and filling in the following form:

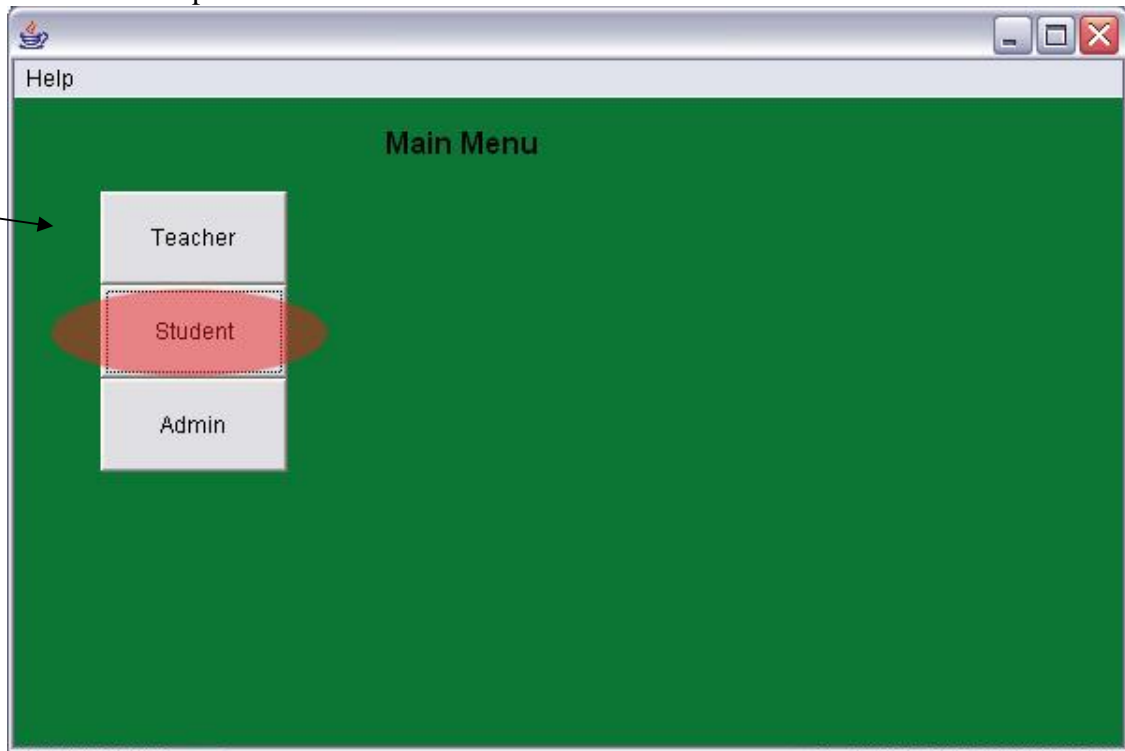


The image shows a software window titled "Creating a New Club" with a green background. The window contains a form with the following fields and controls:

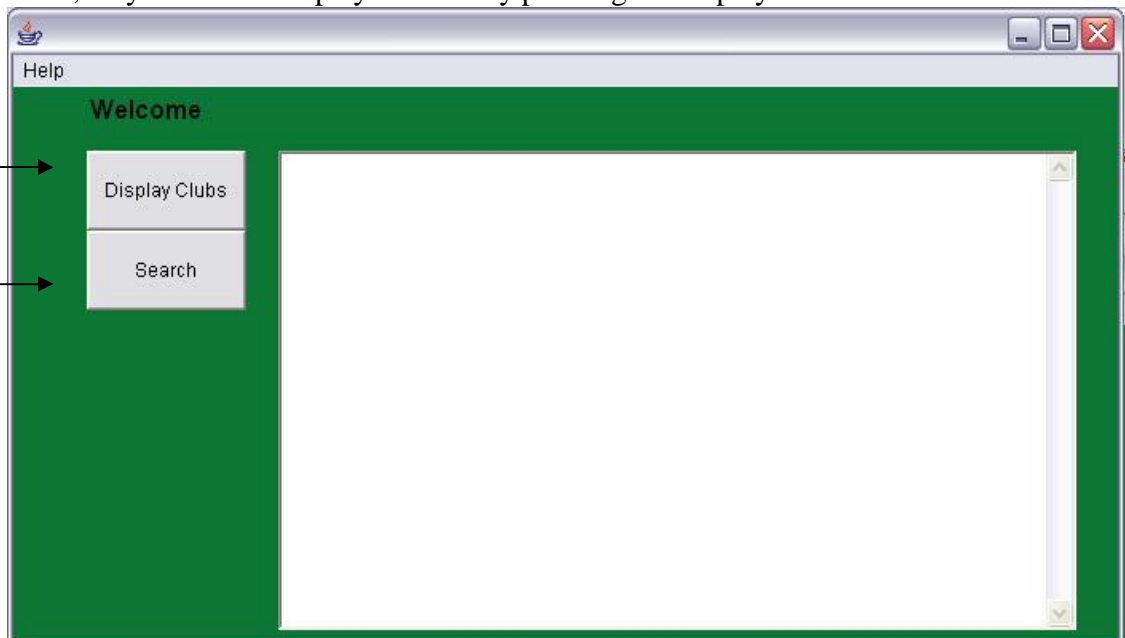
- Day:** A dropdown menu with the text "- DAY -".
- Name:** A text input field.
- Email:** A text input field.
- Grades:** A text input field.
- Location:** A text input field.
- Type:** A dropdown menu with the text "- TYPE -".
- Description:** A text input field.
- SAVE!** and **Cancel** buttons are located to the right of the form fields.

When day stands for day of club that takes place, name is the name of the club, email is the email of the teacher organizing it, grades that can participate in the activity, typical meeting location for the club, the type of club or activity and an optional part of description.

When a student would like to search for a club all they need to do is press the student button on the splash screen:

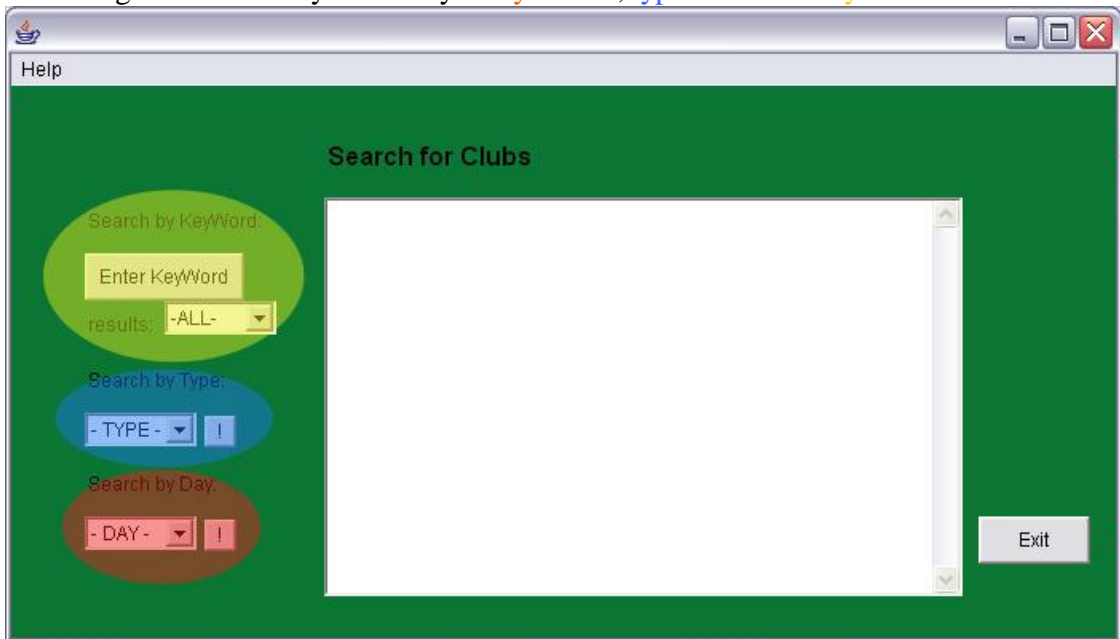


Then, they can either display the club by pressing the display button:

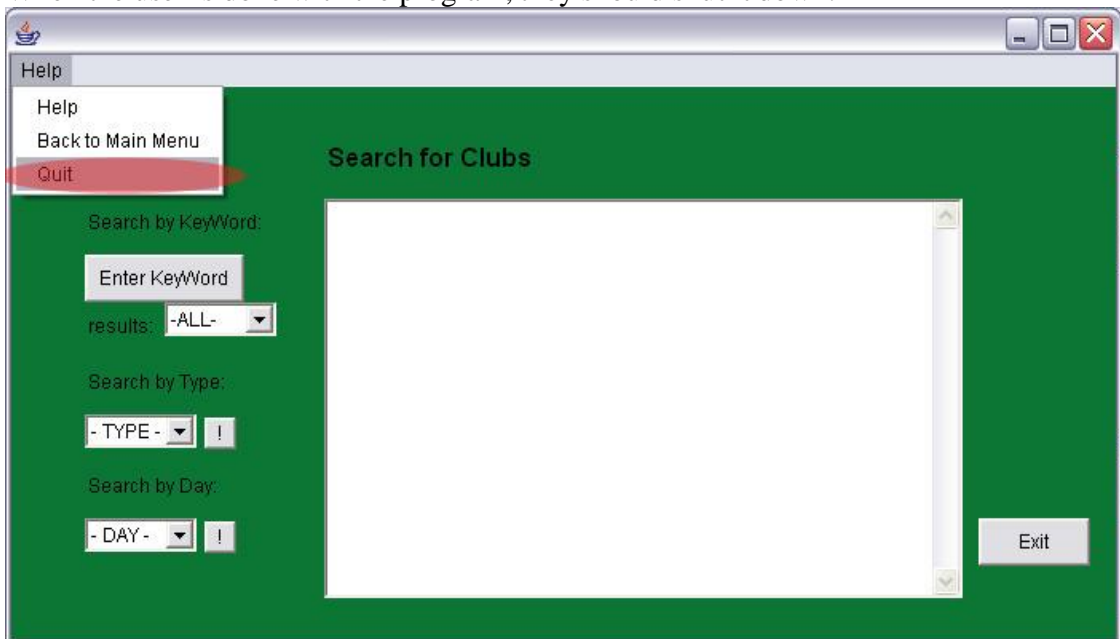


or they can search.

Searching can be done by three ways: **day search**, **type search** or **keyword search**:



When the user is done with the program, they should shut it down:



Or press help at any given point during the program.

Admin

The password for admin is “xyz12345” and it is not possible to change this. Make sure you do not forget it or lose it.

To create a new teacher profile, you need to press the teacher profile button. Enter the name of the teacher in the form of Firstname Lastname. You can only change these if you access the profiles through the text file, but the intention of this program is not to change it and rather for you to put in correctly in the first place. Middle names are also acceptable, but realize that the program had a certain name capacity and cannot exceed 50 bytes.

If a teacher forgets their password you can click the display all button and it will show you all the names of the teachers with their passwords beneath their names in the order you have created the profiles.

You cannot create new clubs or delete any clubs, the buttons are there to remind you that if you want to do so, you need to sign in as a teacher (in case the admin has also an account for themselves and would like to add a new club).

The Stats button will tell you how many clubs there are in the file. This is for you to see how many teachers have added clubs and to see if there are improvements from the last time you checked how many clubs there are in the file.

The delete all button will actually empty the random access file, so make sure that you only do something like this in the end of the year. It will accept the input of “y” or “yes”, otherwise any other input will be ignored.

Teacher

Log in with your EXACT name that you have chosen with the admin or that he has given you, and type in your password.

You can create clubs and delete the ones you have created. You cannot delete clubs that you have not created. You cannot create clubs with the same name.

When you are creating a new club, use your common sense. that is, for location put the most common location your club meets in. for the name of the club, put the exact name. for the day, choose the correct day, and for weekends, choose S. You do not have to type a description but it is better if you do, just do not make it too long, as the program will have to cut your description short. For grades, something like this: 9-12 would be a good input. You may also write “adults” if it is not intended for the students of the community but for their parents. It is up to you to decide who can come to your club.

You may also use any other part of the program besides the admin one.

Student

You may search for clubs using three kinds of search: day based search, type based search and keyword based search. You may put anything in the keyword based search, even one letter will produce answers for you.

Do not vandalize the system in any way.

When you do a search based search, choose the write letter for the names of the day. S will stand for weekend in general; it doesn't necessarily mean Saturday or Sunday. If you need more information other than the one given to you in the program, you may email the teacher who has set up the club. That is what their email information is for.

Mastery Factors

Mastery Factor	Explanation	Code line, if applicable
Adding to Random Access File	My random access file, "clubs.dat", has a method to add clubs to it in the New Club class, called addNewClub().	640-680
Deleting from Random Access File	Teachers can delete clubs from the file using the delete() method in the teacher class.	262-350
Searching in Random Access File	Searching in the random access file is done in the Search file, and can be done with three methods, search(String day), search(String word, int amount) and searchType(String type).	700-1100
Recursion	None	
Merging two sorted data-structures	None	
Polymorphism	In my search class, I have two search methods, search(String day) and search(String word, int amount).	1049, 857
Inheritance	My admin class extends my teacher class which extends my student class.	1273, 174, 372
Encapsulation	In my linklist class, there are private nodes that can only be accessed using get and set.	1127-1129
Parsing a text file	None	
Hierarchical composite data structure	I've got linked list, in which each node has more than one data record in it. To be more precise, each node in my linked list has 6 strings in it.	1251-1261
Five SL mastery factors	Searching in both random access file and normal text file, use of additional libraries, user-defined objects, simple selection, loops, nested loops, user defined methods with parameters.	Everywhere
ADT # 1 – Add and Retrieve data	This will be in my Link List class, displayNextNode() and add().	1115-1263

ADT # 2 – Handeling Ends	Is also in my LinkList class, can be seen in removeTail()	1115-1263
ADT # 3 – Many error-handeling	Spread throughout my LinkList class, in nearly all of the methods.	1115-1263
ADT # 4 – All methods, Robust	None	