

Workshops Enrollment System

Project Report

ZHANG Wengyu 21098431d
CHEN Derun 21098424d
YE Haowen 21098829d

Computational Thinking and Problem Solving (COMP1002) - Project Group 2

1 Introduction

This document describes the process of solving the problem **Topic 3** by group 2. The project is part of the course *Computational Thinking and Problem Solving (COMP1002)* at PolyU.

2 Problem description

The aim of Topic 3 is providing a **Workshop Enrollment System** for students to enroll workshops as well as for administrators to manage. For students, they can enroll workshops, cancel enrollment, check enrollment, and search and sort workshops. For administrators, they have access to add a new workshop, update workshops, and search and sort workshops.

A workshop contains a) a unique **ID**, b) **Title**, c) **Location**, d) **Date**, e) **Time**, f) **Quota**, and g) **Remaining**.

The login function is also required to the system with the correct ID and corresponding password, no matter administrators or students.

3 Data abstraction

- We use multiple text files to store all workshops and each student's enrollments:

- `workshop.txt`: stores all workshops;
- `[sID].txt`: stores all workshop enrolled by a student with `[sID]`;

For each line in above 2 text files, the format is pre-designed as **ID:info1 info2,...**, like:

000001:workshop1 pq604a 2021/11/22 1200-1300 100 100

- Furthermore, we use text files for login:
 - `administrator.txt`: stores user name and password of all administrators for login verification;
 - `student.txt`: stores user name and password of all students for login verification;

Due to the ID of each workshop is unique and immutable, and each workshop has its information, we use the **Dictionary** data type to store the pair of these data. And **List** data type is used to store the information of a workshop which is mapped by the workshop ID. Moreover, **Dictionary** can be retrieved quickly by the key, and **List** is mutable which is handy to update. Therefore a nested **Dictionary** with a **List** as value is the main data structure used throughout the whole project.

4 Python implementation of the data types

- For nested **Dictionary** data model, representing workshop information, we convert data from text file to

```
{workshop ID:[workshop info1, info2,...]}
```

For example, from

```
"000001:workshop1 pq604a 2021/11/22 1200-1300 100 100" to
```

```
{"000001":["workshop1", "pq604a", "2021/11/22", "1200-1300", "100", "100"]}
```

- Firstly, using `split(":",)` to split the read string by ":", and the string will become a **list** with two elements, the first element is **workshop ID** and second is other information, i.e., `[ID, info1 info2 ...]`. After that, for the second element, using `split(" ")` to convert it to a new list, i.e., `[ID,[info1, info2, ...]]`. Finally, using `dict()` to convert it as `{ID:[[info1, info2, ...]]}` (See Figure 1).

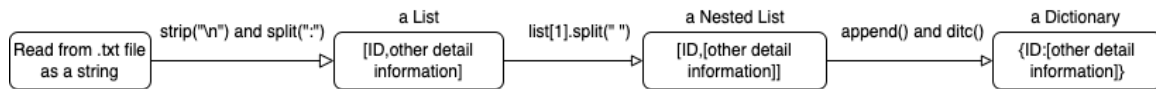


Figure 1: Data abstraction

5 Modular Design

5.1 Login model

We set a complete account login system for new users and exist users (See Figure 2). Entering the user system, it will ask you to choose whether administrator or student identity to login. After choosing the identity, the user is supposed to enter user name and password.

- For student, a successful login will happen with **exist user name and correct password**. During this section, the system will check whether the name exists in **student.txt** file or not initially. If not, the system will lead to **registering** section and ask user to register a new account. After successfully entering a **new user name** and password in register section, user will back to login section to login the registered account. If user does not choose to create a new account, the system will back to login section immediately.
- For administrator, the only difference is there is **no register** section.

At the same time, student or administrator has a chance of entering a wrong password **three times**.

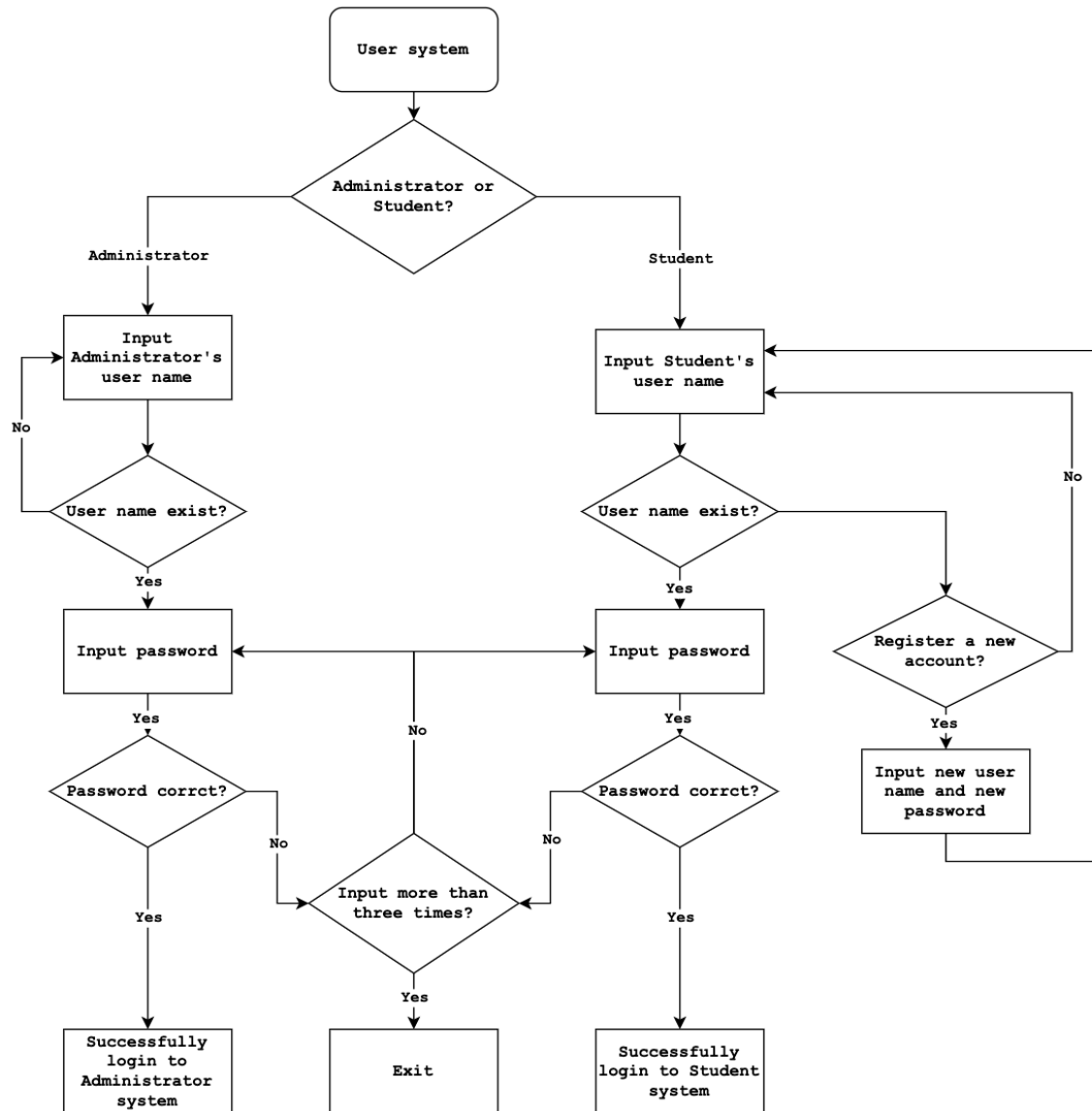


Figure 2: Login Procedure Flowchart

5.2 Key Functions

5.2.1 Function for Administrator

- **addWs(inData)**: Add a new workshop with **inData** into the workshop storage by administrator. The unique ID of the new workshop is generated by system automatically.
- **deleteWs(wID)**: Delete a workshop with a **wID** from the storage by the administrator.
- **updateWorkShop(wID,inData,Index)**: Update the information of a workshop by administrator.
 - **wID**: the ID of the workshop to be updated;
 - **inData**: the new information to be updated into the workshop;
 - **Index**: which information in the workshop need to be updated - 0: Name; 1: Location; 2: Date; 3: Time; 4: Quota; 5: Remaining. For example, **updateWorkShop(wID,inTitle,0)** can be used to update the title of a workshop.
- **Retrieval function**: Based on the **searchEngine**(see 5.3 Search Engine), this function returns corresponding workshop as a string based on the entering parameter(e.g., workshop ID, Title, Location,

Date, Time, Quota, and Remaining).

5.2.2 Function for Student

- **enroll(sID,wID)** : for a student-**sID** to enroll a workshop-**wID**. When remaining positions ≤ 0 , or the time of the workshop-**wID** is overlapped, system will not allow this enrollment.
- **cancel(sID,wID)** : for a student-**sID** to cancel a enrolled workshop-**wID**. When the workshop has not been enrolled by student, the system will not allow the cancellation.
- **listEnrolledWs(sID)**: list out all workshops enrolled by the student-**sID**.
- **Retrieval function**: Based on the class **searchEngine**(see 5.3 Search Engine), this function will return the corresponding workshop as a string based on the entering parameter(e.g., workshop ID, Title, Location, Date, Time, and Remaining, student have no access to the number of Quota).

5.3 Search Engine

- A class of **searchEngine** is designed for **Retrieval** workshops, which has 7 functions for searching corresponding workshop based on **workshop ID, Title, Location, Date, Time, Quota or Remaining number**. And each function will return a string **outStr**, for showing out the corresponding information if possible. Example of searching based on **workshop location**:
 - `input :`
 - `inDict`: a dictionary contains all workshop information
 - `inLocation`: the target location to search
 - `outList`: a list stores all found workshops for output
 - Repeat get element from `inDict` as item
 - if the location of item is equal to `inLocation`:
 - append the item workshop into the `outList`
 - Until got all elements
 - if the `outList` is empty, output "NO Found!"
 - else output the `outList`

6 Extra functions

We implement Search, Sort, Register function and Graphic User Interface as extra functions.

- **Search Function**: Our group provides a search function not only for administrator but also for student based on class **searchEngine**. This function provides a retrieve platform for student to search workshop based on **[i]ID; [tit]:Title; [l]Location; [d]Date; [t]Time; [r]Remaining** apart from **quota**.
- **Sort Function**: This function provides a better experience on searching and listing out workshops, which is based on **Bubble Sort**. The function `sort(inDict,inBase,inOrder)` will return a sorted **inDict** based on **inBase** section in the **inOrder** order. This function returns a **ascending order** search result based on **workshop ID** by default.
 - **inBase**: refers to the sort basis. It has value of **(-1,0,1,2,3,4,5)**, pointing to **Workshop ID, Title, Location, Date, Time, Quota and Remaining** of a workshop, respectively.

- **inOrder**: refers to the sort order. It contains boolean value **True** for **[i] increasing** order and **False** for **[d] decreasing** order.

For example, `findByName(inDict,0,True)` will search and return an **increasing order** dictionary based on workshop **name** from the `inDict` dictionary.

- **Register Function**: We provide register function for student who does not have an account for the system(See in 5.1 Login model).
- **Graphic User Interface**: Our group implement **Graphic User Interface(GUI)** successfully based on the `tkinter` package [1]. The GUI provides administrators and students a better operating experience, as well as an intuitive view on input and output when using the Workshop Enrollment System.

7 Object-oriented Design

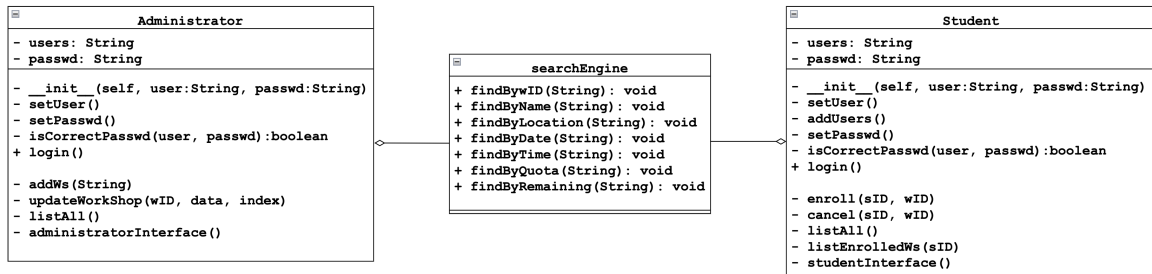


Figure 3: Object-oriented Design

According to the project description, the **Workshops Enrollment System** can be accessed by different administrators and students, so we abstract information from them and mainly use the **encapsulation** of Object-oriented(OO) Design in the program[2]. Overall, we need 3 objects - **administrator**, **student**, **search engine**, therefore, we create 3 corresponding classes to encapsulate their variables and functions(See Figure 3).

For **Administrator**, **Student** class, we encapsulate two variables - **users name** and **password** into them, constructed by the initializer when a **Administrator** or **Student** object is created in the project. These two variable not only can be used in the login verification, but also indicates the target .txt file modified by administrator or student. Furthermore, functions are also encapsulated in these classes, including login functions, file IO functions, and command line interface functions. Both **Administrator** and **Student** are accessed only through `login()` function in these two classes.

For **searchEngine** class, we encapsulate all searching functions into it, which can be easily used by administrators and students for retrieval.

References

- [1] Python Software Foundation, *Graphical User Interfaces with Tk*, 2021, last updated on Nov 30, 2021. [Online]. Available: <https://docs.python.org/3/library/tk.html>
- [2] Python Software Foundation, *Classes*, 2021, last updated on Nov 30, 2021. [Online]. Available: <https://docs.python.org/3/tutorial/classes.html>