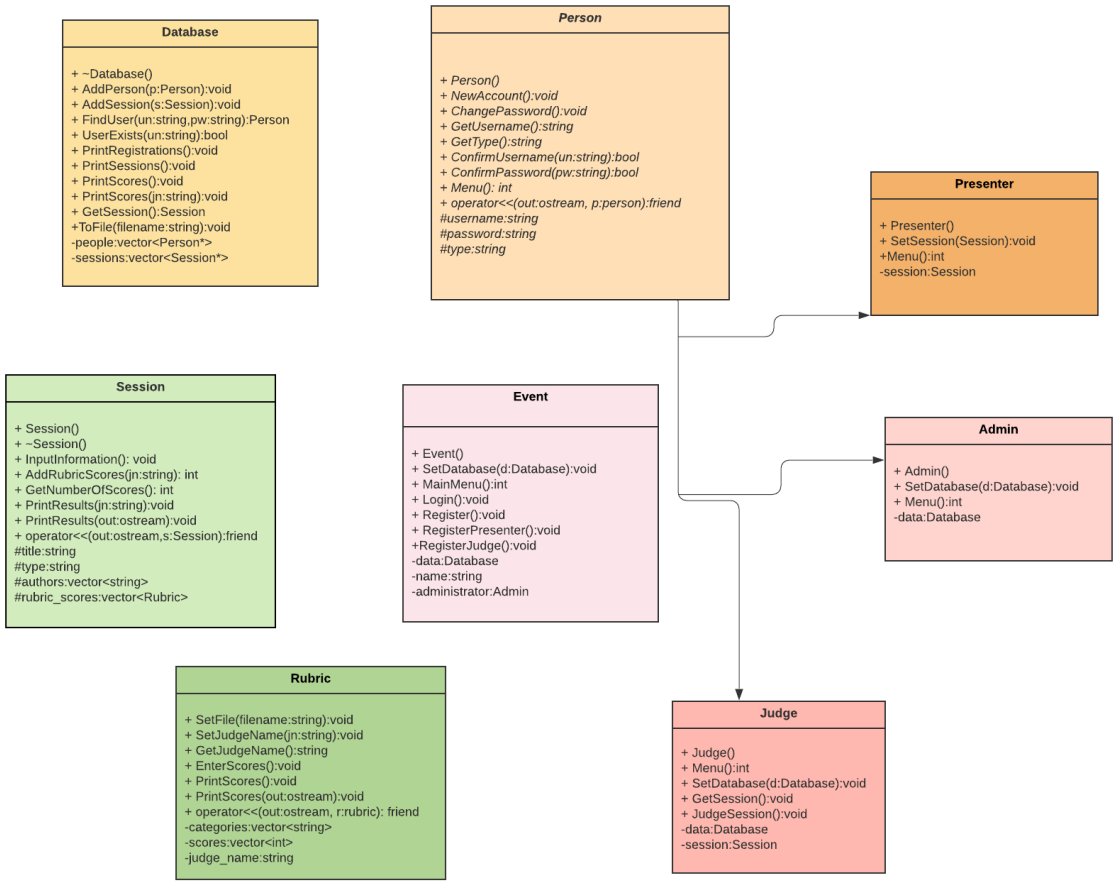
2022 Conference Class Design Documentation

Table of Contents:

UML DIAGRAM


Database
+ ~Database()
+ AddPerson(p:Person):void
+ AddSession(s:Session):void
+ FindUser(un:string,pw:string):Person
+ UserExists(un:string):bool
+ PrintRegistrations():void
+ PrintSessions():void
+ PrintScores():void
+ PrintScores(jn:string):void
+ GetSession():Session
+ToFile(filename:string):void
-people:vector<Person*>
-sessions:vector<Session*>
Person
+ Person()
+ NewAccount():void
+ ChangePassword():void
+ GetUsername():string
+ GetType():string
+ ConfirmUsername(un:string):bool
+ ConfirmPassword(pw:string):bool
+ Menu(): int
+ operator<<(out:ostream, p:person):friend
#username:string
#password:string
#type:string
Presenter
+ Presenter()
+ SetSession(Session):void
+Menu():int
-session:Session
Session
+ Session()
+ ~Session()
+ InputInformation(): void
+ AddRubricScores(jn:string): int
+ GetNumberOfScores(): int
+ PrintResults(jn:string):void
+ PrintResults(out:ostream):void
+ operator<<(out:ostream,s:Session):friend
#title:string
#type:string
#authors:vector<string>
#rubric_scores:vector<Rubric>
Event
+ Event()
+ SetDatabase(d:Database):void
+ MainMenu():int
+ Login():void
+ Register():void
+ RegisterPresenter():void
+RegisterJudge():void
-data:Database
-name:string
-administrator:Admin
Admin
+ Admin()
+ SetDatabase(d:Database):void
+ Menu():int
-data:Database
Rubric
+ SetFile(filename:string):void
+ SetJudgeName(jn:string):void
+ GetJudgeName():string
+ EnterScores():void
+ PrintScores():void
+ PrintScores(out:ostream):void
+ operator<<(out:ostream, r:rubric): friend
-categories:vector<string>
-scores:vector<int>
-judge_name:string
Judge
+ Judge()
+ Menu():int
+ SetDatabase(d:Database):void
+ GetSession():void
+ JudgeSession():void
-data:Database
-session:Session

Person class:

Overview:

Person class is the only abstract class. It is a general class which deals with people's information (accounts), such as password, username, and type. Username and password are used to log in to the system, and type is used as an arbitrary value to get the type of person that has the account. This is where it correlates directly to the presenter, admin and judge as the type will define how the inherited class behaves. This class correlates to the abstraction part of object-oriented programming. This object class cannot be instantiated and does not output anything.

Functions:

```
        Person();
//This function is an initializer function, setting the variable username and
password to "NONE"
        void NewAccount();
//This function makes it possible for the user to enter in their username and
password to make a new account, updating the current variables.
        void ChangePassword();
//This function makes changing the password possible, by first entering the old
password, checking the given old password, if true, insert the new password,
replacing the old one.
        string GetUsername();
//Getter function, gets the username of the user, set up by the NewAccount()
function.
        string GetType();
//Getter function, gets the type of the current user, does not get set up in this
class but in the inheriting classes.
        bool ConfirmUsername( string un );
//This function makes possible to confirm the username of the current user, by a
simple if comparison and return a Boolean identity.
        bool ConfirmPassword( string pw );
// This function makes possible to confirm the password of the current user, by a
simple if comparison and return a Boolean identity.

        virtual int Menu() = 0;
//Initializing function which will make this class abstract and will get
overridden by every class that inherits it.

        friend ostream & operator<<( ostream & out, const Person & p );
//Defines the cout of the main.cpp so that when a pointer of type person is
inputted the function prints the username and type of the given person.
```

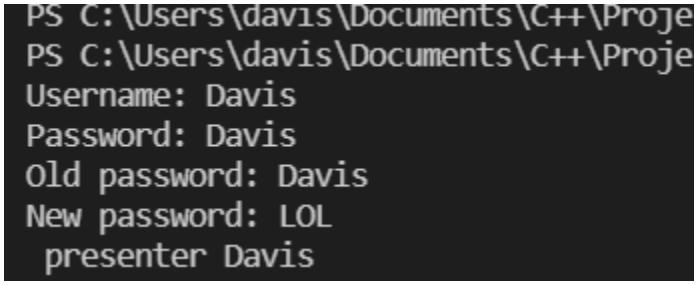
Using from the inheriting class Presenter:

```
    Presenter * p = new Presenter;
    p->NewAccount();
    p->ChangePassword();
    cout << *p;


Example output:

Username: Davis
Password: Davis
Old password: Davis
New password: LOL
 presenter Davis
```

TERMINAL OUTLOOK:



Admin Class:

Overview:

Admin class inherits the person class and uses the database class. This object class can be instantiated to type admin. This class has ability to check and view the database score, registrants, and presentations, based on the database that they want to set (where the database class comes to play). This class outputs the registrations, sessions, results, and database. It also accesses a file of the choosing and enters the data from the database. The change password function is inherited by the person class.

```
        Admin();
//Initializes the type of the class to “admin”
        int Menu() override;
//Overrides the menu to the desired menu output. Output example below.
        void SetDatabase( Database * d );
//This function makes it possible for the admin to set the session they want to
look at.

Example input:

    Admin *a = new Admin;
    a->Menu();

Example output:
Menu:
   1. View registrants
   2. View presentations
   3. View results
   4. Print database
   5. Change password
   0. Exit

Desired option: 2


****** Sessions ******
```

TERMINAL OUTLOOK:

```
Menu:
   1. View registrants
   2. View presentations
   3. View results
   4. Print database
   5. Change password
   0. Exit

Desired option: 2


****** Sessions ******

PS C:\Users\davis\Documents\C++\Project#2>
```

Presenter Class:

Overview:

Presenter class inherits the person class and uses the session class. This object class can be instantiated to type presenter. Resembling the admin class, it can set the session it wants to look at (where the session class comes to play) and all the information about it. The output will be the information of the session that it wants to look at. The change password function is inherited by the person class.

Functions:

```
        Presenter();
/*This function initializes the type from the inherited person class to type:
presenter
        void SetSession( Session * s );
This function sets the session the presenter wants to look at.
        int Menu() override;
This function overrides the Menu, to whatever Menu the presenter has
accessibility to. */

Input case:

p->Menu();

Here is an example out of the unique Menu:
Menu:
   1. View session information
   2. Change password
   0. Exit

Desired option: 1
****** Session information ******
```

Judge Class:

Overview:

Judge class inherits the person class and uses the session and database class. This object class can be instantiated to type judge. Resembling the admin and presenter class, it has its own menu where it can look at the sessions and the function assigns the sessions, print the scores, and

change the password. It can also set its own database (where the database class comes to play) and unlike the other person related classes it also deals with the rubric scores, it can add the rubric scores for certain judges and help contain all the information about the sessions of the current judge. Thus, the output would mostly be about the sessions and the rubric scores, such as if there are any sessions to be assigned or printing the scores. The change password function is inherited by the person class.

Functions:

```
        Judge();
//This function initializes the type to “judge”.
        int Menu() override;
//This function overrides, the inherited menu function to the desired options for a judge to navigate. Uses the inherited class person.
        void SetDatabase( Database * d );
//This function gives the judge access to set any database they like so they can access the information they want.
        void GetSession();
//This is a getter function which returns the session with the lowest score. This function is derived from the session class.
        void JudgeSession();
//This function takes the judges name and makes it possible for the judge to enter all the scores, based on the rubric.txt file as a questionnaire. This function is derived and possible from the predefined functions of the sessions class and rubric.

Sample Input:

    Judge * j = new Judge;
    d->AddPerson( j ); //adds the judge in the database

    j->NewAccount();
    j->SetDatabase( d ); //previously initialized database
    j->JudgeSession();
    j->Menu();



Sample Output:

Username: Davis
Password: Davis
****** Rubric information ******
Overall appearance: A professional looking technical poster. Text and graphics are well-balanced. Author names and school/department are present. Appropriate poster size, font, images, and colors.
```

```
   1. Excellent
   2. Good
   3. Satisfactory
   4. Needs Improvement

Enter in a score:
3
Organization: The poster is well-organized and easy to read, with appropriate
headings, clearly defined objectives, methodologies, data, and conclusions.

   1. Excellent
   2. Good
   3. Satisfactory
   4. Needs Improvement

Enter in a score:
4
Engineering design: The poster demonstrates a clear understanding of engineering,
scientific,
and mathematical concepts embodied in their design/project and, if appropriate,
its relationship to social, economic, or environmental concerns.

   1. Excellent
   2. Good
   3. Satisfactory
   4. Needs Improvement

Enter in a score:
3
Professionalism: The authors display professionalism in all areas - initiative,
commitment, enthusiasm, and appearance.

   1. Excellent
   2. Good
   3. Satisfactory
   4. Needs Improvement
Enter in a score:
2
Question & Answer: The author(s) handle questions well, demonstrate knowledge of
concepts, and ask thoughtful questions of the judges.

   1. Excellent
   2. Good
   3. Satisfactory
```

```
   4. Needs Improvement

Enter in a score:
4









Menu:
   1. Judge session
   2. Print scores
   3. Change password
   0. Exit

Desired option: 1
```

Suggestions for improvement:

No score checker in the rubric class derived function JudgeSession(), does not check if the score is higher than 4 or less than 0.

Database class:

Overview:

Database class uses the person and session classes. It uses the person class to use its functions for the current users. We can add people to the people list, find users based on the peoples list, by confirming their usernames and password or we can confirm their existence. It also gives us access to print that list as just arbitrary values or store it in a file. Same as the person class being used, the sessions class is used to add sessions in the database sessions list. We can use that list then to either print the sessions, print the scores for the given sessions or get the lowest scoring session and return it. Same as what we can do with the registrations, we can also store this information, in an external file. This class is primarily important and deals with the people registered as well as classes and gives conventional functions for the user to be able to modify or edit the data. The outputs range from sessions to registrations and serves as a backbone for the person inherited classes.

```
        ~Database();
//This function makes sure the database is reset to having no information in any
of the registrars.
        void AddPerson( Person * p );
//Adds the parameter person to the registrar of people
        void AddSession( Session * s );
//Adds the parameter session to the registrar of sessions.
        Person * FindUser( string un, string pw );
//Finds user by first opening a vector and then going through the people list,
when found it adds the person to the empty vector all using person class
functions.
        bool UserExists( string un );
//Checks if the user inputted in the parameter is in the people registrar

        void PrintRegistrations();
//This function prints all the registrations in the people registrar in a list
manner.
        void PrintSessions();
//This function prints all the sessions in the session registrar in a list
manner.
        void PrintScores();
//This functions prints all the scores in the sessions registrar, by using the
printresult() function derived from the session class.

        void PrintScores( string jn );
//Print the results from the specific judge, using the PrintResults(string jn)
function derived from session class.

        Session * GetSession();
//Gets the lowest score session (session.h) and returns it.

        void ToFile( string filename );
//This function can output the current registrations and sessions on the
parameter file.

        vector<Person *> people; //people registrar
        vector<Session *> sessions; //sessions registrar


Example Input:

    Presenter * p = new Presenter;
    p->NewAccount();
    Session * s = new Session;
```

```
    Judge * j = new Judge;
    j->NewAccount();

    Database * d = new Database;

    d->AddPerson( p );
    d->AddSession( s );
    d->AddPerson( j );
    j->SetDatabase( d );
    d->PrintScores();

    d->PrintRegistrations();
    d->PrintSessions();



Example output:

Username: davis
Password: davis
Username: davis
Password: davis

****** Scores ******



****** Registrations ******
 presenter davis
      judge davis


****** Sessions ******


None
```

Improvements on the code suggested:

When trying to print the scores it will not do it and if it does print the scores it will break. The judge print result scores needs to have the session recognized otherwise it will not know what points to print.

The judge session ends after inputting everything thus making it impossible to test the scenario separately, could not find the reason as to why.

Session Class:

Overview:

Session class uses the rubric class, primarily to have a rubric object to store all the rubric scores and their authors and modify the information easily. The main purpose of the session class is to have a conventional way to store all the rubric information. We start classifying the rubrics by separating the type of information, if it's a poster or presentation, the title, number of authors and the names of the authors. We can then add to the rubric scores list based on an inputted judge, by setting up a file which then stores all the scores for the given judge. We can access the number of scores available and we can print the result scores list plainly, based on a judge's name or based on an output parameter. We can then print the classified information based on the type of rubric. Session class is responsible for keeping track of all the rubrics, event information (specifics of every event) and dealing with the modification and easy navigation of the rubrics. It outputs the event information and results based on the rubrics.

Functions:

```
        Session();
//Initializes the sessions type and title to None
        ~Session();
//Deletes and restores the session list so we can start fresh
        void InputInformation();
//This function makes inputting the information of the event possible. It asks
for the session information: (Poster/Presentation), Title and number of authors
along their names.
        void AddRubricScores( string jn );
//Makes the score input possible and in a form of a questionnaire based on the
Rubric.txt file. Sets the judge's name and enters the score meanwhile storing it
in the scores list.
        int GetNumberOfScores();
//Gets the rubric scores list size and outputs it. Getter function.
        void PrintResults();
//Prints all the score results of the rubric, by using the printscores() function
based on the database class.
        void PrintResults( string jn );
//Prints the score results for the specific judge inputted in the parameter.
        void PrintResults( ostream & out );
//Prints the score results for the specific output inputted in the parameter.

        friend ostream & operator<<( ostream & out, const Session & s );
//defines the << operator, cout to print the type, title and authors.
```

```
        string title; //title of the session
        string type; //type of the session
        vector< string > authors; //authors list
        vector<Rubric *> rubric_scores; //scores list


Example input:
    Presenter * p = new Presenter;
    p->NewAccount();
    Session * s = new Session;
    s->InputInformation();
    s->PrintResults();

    Judge * j = new Judge;
    j->NewAccount();


Example output:
Username: davis
Password: davis

Enter in session information:
   1. Poster
   2. Presentation

Type: 1

Title: Davis
Number of authors: 3
Author 1: Davis
Author 2: Davis
Author 3: Davis
****** RESULTS ******


Poster
      Title: Davis
    Authors: Davis, Davis, Davis

Username: davis
Password: davis
```

Suggested improvements:

No suggested improvements

Rubric Class:

Overview:

Rubric class does not use any other classes to reference itself off. In this class we define functions that we can use later on in session class such reading information from a file (add rubric scores) and inputting in the categories list, enter scores based on the rubric (also used in rubric scores), and help in printing the scores as a list method. This can also be useful to set the judges name and get the judges name which is then used to organize the information based on the inputted judge. We can also print out every category for the given rubric object. Rubric class is a backbone to the session class as it helps it in every step when classifying information.

Functions:

```
        void SetFile( string filename );
//opens a file and takes the files information (categories) and inputs it in the
categories list(the green and white comments are done on purpose here, making it
more… realistic).


        void SetJudgeName( string jn );
//Sets the judge's name to the parameter, setter function for the string
judge_name
        string GetJudgeName();
//Getter function to output the judge's name
        void EnterScores();
//This function serves to prompt the user to enter the scores based on the rubric
(tied in with the judge class).
        void PrintScores();
//This function serves as a score printer in a list manner, based on the
categories list and scores list.
        void PrintScores( ostream & out );
//This function servers also as a score output printer in a list manner, based on
the output as input.

        friend ostream & operator<<( ostream & out, const Rubric & r );
//This function defines the <<, cout operator to print "Rubric Scores:" and then
the categories of the rubric.

        vector<string> categories;//categories list
        vector<int> scores; //scores list
        string judge_name; //judge name
```

```
Input Example:

    Rubric *r = new Rubric;
    r->EnterScores();
    cout<<*r;



Output example:

****** Rubric information ******
   1. Excellent
   2. Good
   3. Satisfactory
   4. Needs Improvement
Enter in a score:
2

Rubric scores:
```

Suggested Improvements:

In the rubric class we should be able to set the session as this class is the backbone of the session class.

Event Class:

Overview:

The event class serves as the last product of the classes, where they all work together and create a system that will help any type of person, let that be administrator, presenter, or judge. It gives the ability to create a new event, enter the credentials, set up a common menu for every person, create a final register with a presenter and judge and make new accounts for them.

```
        Event();
//This function initializes the event function, it prints and asks for an event
administrator account, requests name of event.

        void SetDatabase( Database * d );
//Set's the database to the parameter, checks to see if the administrator is in
the register, if he not then he's added from the previous event() function call.
This uses the database class and session class.
        int MainMenu();
//This function makes the viewing of the menu possible. The menu has
accessibility to logging in, registering or exiting.
        void Login();
```

```
//Asks for password and username, checks to see if it exists, if it doesn't it prints "User not found." And if it does it continues the code. Uses the person class.
        void Register();
//Asks for type of person logging in, and whatever type it is it calls the corresponding function that registers them
        void RegisterPresenter();
//Registers the new person by logging them in with a new account, checks to see if the username exists and if it doesn't, it goes through with creating a new session, asking to input the information of the event, sets the session for the presenter and then adds the person and session to the database.
        void RegisterJudge();
//Register the new person by logging them in with a new account, checks to see if the username exists, and if it doesn't Set's the judge database to the current database and then adds the judge to current person list.

Example input:
    Judge * j = new Judge;

    j->NewAccount();

    Event * e = new Event;

    e->MainMenu();




Example output:
Username: davis
Password: davis









****** New Event ******
Enter in the credentials for the Event administrator.
Username: davis
Password: davis
```

```
Name of event: davis









****************************
Welcome to davis
 1. Login
 2. Register
 0. Exit

Desired option: 1
Logging in:
Username: davis
Password: davis
```

Examples of object-oriented programming in this collection of classes

All the classes in some manner comply to object-oriented programming. The key characteristics of object-oriented programming, are:

1. Encapsulation

All the classes above have private and public data members. Private data members can't be accessed by other classes as those are what defines the class characteristics. Just as the same way we can't control someone's emotions and we have no right to, so does the class have it's own data characteristics which should not be touched.

2. Abstraction

We can take any high-level class such as the event class, where we use a bunch of already predetermined functions to make possible an easy user interface. Abstractions encapsulates the user interface, using the idea that to use an object we don't need to know how the object was made.

3. Inheritance

Inheritance is product of abstractions and encapsulation. It is the equivalent of a parent and child situation. In this case the person class served as a parent class for the judge, presenter, and admin class. While inheriting all the functions it also can modify a function by overriding it. Just like a child inheriting genes, while like their parents they always their little hint of modification.

4. Polymorphism

This word just means a lot of shapes. (object-oriented) (pun Intended)

Not in this case, in class terms this means we can take a child class and use it just as a parent class, so we don't confuse the types. Inherits the function, implements it own version of these functions and thus can be use in a mixed collection of whatever objects the classes represent. Thus, helping you in not redefining the inherited function a lot of times. We see this with the judge, admin and presenter class.

General Code Improvements:

The sessions should be checked automatically when a presenter is logging in their account or opening a new account. There should be a navigatefunction for every session and have the sessions be personalized based on the person with the file name being the person and every session. Rather than being able to set the session by the presenter as a step up for abstracted function.

I generally liked going through this project and to see how useful classes really are especially for big projects. Learning about the perks of inheritance, abstraction of classes and how easy to navigate a seemingly "complex" project can be made with proper use of classes. Generally a well-produced project and a very meaningful one.