DALHOUSIE
UNIVERSITY

CSCI5408 - Data Management, Warehousing, and Analytics

Name: Darshil Patel
Banner ID: B00946528

Problem- 2 Report

Task-1: created empty console java application

Initially, I started reading the Java Docs as a professor has specified in the pdf for rules and comments which should be implied in the code for better quality and understanding.

Then,

Created a new empty project using IntelliJ IDEA 2022.3.1 under the IntelliJ Build system and in Java as a programming language as per instruction given in the Assignment-1 pdf on Brightspace.

Before I started coding, I had to think of some architecture which would be useful at a later stage for navigation through code but unfortunately, I have less experience in Java and started drawing a rough workflow of the application inside the rough paper.

Below is the architectural drawing.

Architecture Description:

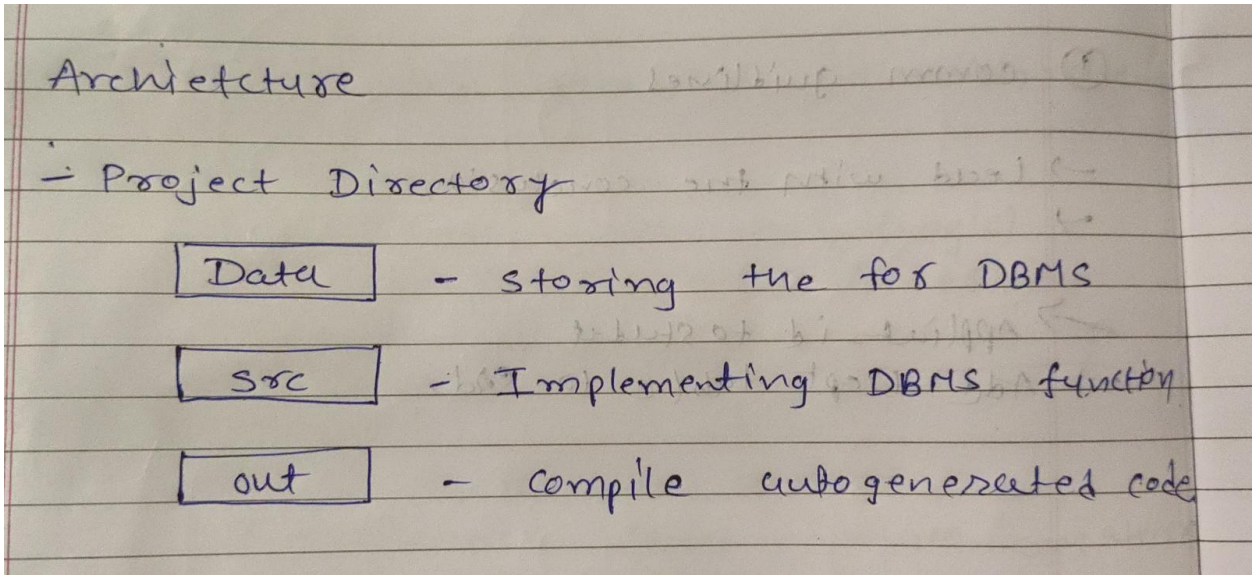


Fig 1.1: Java Application Architecture

This is the main directory structure and “Src” folder will have the source code and “out” is auto-generated when the program gets executed.

But “Data” folder is generated but the if does not exist because my application needs that folder to store various types of information.

(1) Src Directory: It will contain the all source code and implementation inside that file including class, method, and packages which will be helpful to navigate.

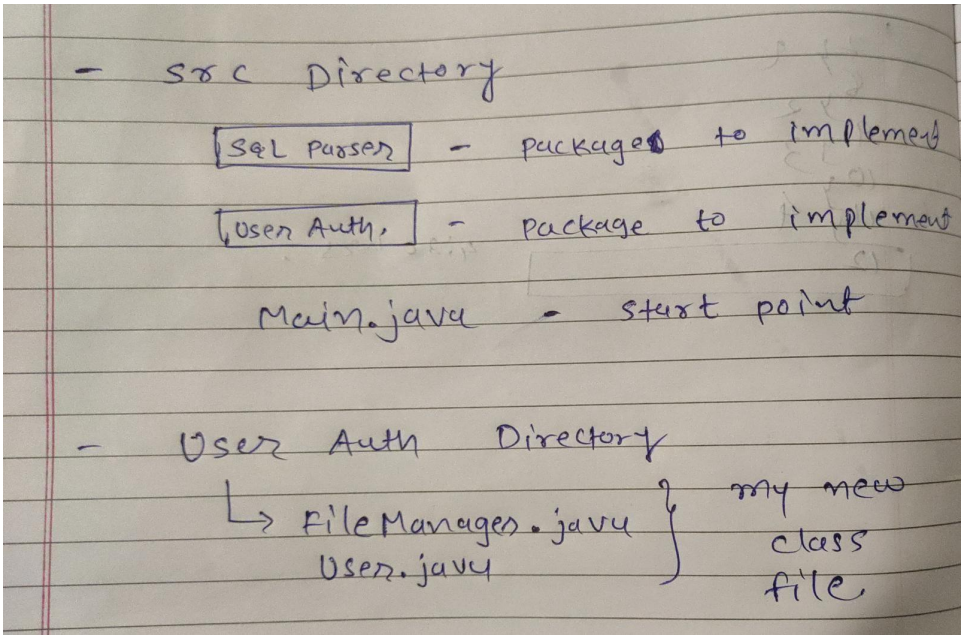


Fig 1.2: Architecture

Taske - 2 Stated implementing the project

1. **User Authentication**
   Here we are dealing with the user, I created a **User class** for **storing user information** in the memory like username, password, security- questions and answer and the name of the database.

```
Main.java  User.java  userinfo.txt
7   public class User {
8
9
        6 usages
10      static FileManager manager = new FileManager();
        1 usage
11      static String question = "Set your security question for future verification. \nWhat is your favourite color?";
        1 usage
12      static String questionLogin= "What is your favourite color?";
13
        2 usages
14      int userRowIndex  = -1;
        6 usages
15      String userId, password, answer, database = "";
        6 usages
16      public String userDBpath = FileManager.path + this.database + "/";
17
        1 usage  Darshil Patel +1
18      public User(){
19
20      }
        1 usage  Darshil +1
21      public User(String userId,String password, String answer){
22          this.userId = userId;
23          this.password = password;
24          this.answer = answer;
25      }
```

Fig: 2 User Class

FileManager class has all the methods which are related to **file handling** and **keep a track of the user information**

Fig 3 FileManager Class

After watching these two images it is clear that the User and file manager classes are connected because you can see in the image **Fig2 User** class that it has a static instance of **FileManager class**

That instance is created for handling the operation of the users with files on the hard disk.

FileManager instance automatically creates Data Folder which I mentioned in the beginning.

Based on this architecture I have started and implemented the methods and classes which are now in functional condition. Now let's see method of FIileManager class.

FileManager class important methods:

1. **Private void createUserInfofile()**
   Creates a Data folder if it does not exist and if it does then loads all the information about the user from the static file path.

```
2 usages  Darshil
private void createUserInfofile(){

    File file = new File( pathname: "data/userinfo.txt");

    try {
        if (file.createNewFile()) {
            System.out.println("File Created.");
        }
        else
        {
              System.out.println("Loading the file....");
            loadfiles(file);
        }
    }
    catch (Exception e) {
        System.out.println("userinfo.txt file could not be loaded.");
    }
}
```

Fig 4 createUserInfofile()

2. **Private void loadfile(File file)**
   This function is called by the above function to **load the data** from **data/userinfo.txt file** and add to the **ArrayList of type class user**

```
1 usage  Darshil
private void loadfiles(File file){

    try{
        Scanner fileReader = new Scanner(file);
        String line;


        while (fileReader.hasNextLine()){
            line = fileReader.nextLine();
            addUserToClass(line);
        }

        fileReader.close();
        System.out.println("User info is Loaded.");
    }
    catch (Exception e){
        System.out.println("Failed");
        e.printStackTrace();
    }

}
```

Fig 5 loadfile()

3. **Public boolean CreateDatabase(String db, int userIndex)**
   Creates a database for a particular user, I mean folder when I say the word database as a custom application.

```
1 usage  Darshil
public boolean createDatabase(String db,int userIndex){
    File fileDir = new File( pathname: path+db);
    if(!fileDir.exists()){
        fileDir.mkdir();
        userData.get(userIndex).database = db;
        userData.get(userIndex).userDBpath = path + db + "/";
        userDataWriter();
        return true;
    }
    else{
        return false;
    }
}
```

Fig 6 create a database for the user

4. **Private void addUserToClass(String line)**
   It parses the information of userinfo.txt file and create user class to add to the ArrayList<User>

```
2 usages  Darshil Patel + 1
private void addUserToClass(String line){

    String [] s = line.split( regex: ";");

    User user = new User();

    user.userId = s[0];
    user.password = s[1];
    user.answer = s[2];
    try {
        user.database = s[3];
        user.userDBpath = FileManager.path + user.database;
    }
    catch (Exception e){
        user.database = null;
    }

    userData.add(user);
}
```

Fig 7 add UserToClass(String line)

5. **Public void saveUser(String userString) & Public void userData writer()**
   This is basically for writing back all the user-updated values and new user data to userinfo.txt file

```
public void saveUser(String userString){
    addUserToClass(userString);
    userDataWriter();
}

2 usages  Darshil +1
private void userDataWriter(){
    try {
        FileWriter fw = new FileWriter( fileName: "data/userinfo.txt");
        for (User userDatum : userData) {
            fw.write(userDatum.__str__());
        }
        fw.close();
          try{
              File f = new File("data/userinfo.txt");
              loadfiles(f);
```

Fig 8 Two method work together

User class Main methods:

```
1. public static String stringProtection(String password)
2. public static User Login
3. public static void SignUp()
```

The first method is for making password protected it converts text into md5 string
The second method is for the login functionality of the user
The third method is for creating new users in the system.

Now let's see the Main class and how UserAuthentication module is connected in main.java file with out put and testing Screenshots.

Main.java

1. **Few static methods to print in the output for user convenience**
   These methods are just for user convenience to print the data on the output so the user can navigate easily but once the user starts running the queries user will not be able to see this information. They need to rerun the program to start again.

```
1 usage  Darshil +1
public static void init()
{
    System.out.println("================================================================================");
    System.out.println("                    Welcome to Darshil's DBMS System                            ");
    System.out.println("================================================================================");

}

3 usages  Darshil +1
public static void options(){
    System.out.println("--------------------------------------------------------------------------------");
    System.out.println(" To access the DBMS System, Either use credential or Setup new credential");
    System.out.println("--------------------------------------------------------------------------------");
    System.out.println("1. Sign In\n2. Register/Create Account\n3. Exit");
    System.out.println("--------------------------------------------------------------------------------");
}
```

Fig 9.1 init() & option()

```
1 usage  Darshil
public static void UserOperations(User user){

    System.out.println("================================================================================");
    System.out.println("                                User Operations                                 ");
    System.out.println("--------------------------------------------------------------------------------");
    System.out.println("User: "+user.getUserId());
    System.out.println("Database: "+user.getDatabase());
    System.out.println("Note: One Database is allowed per User. Delete the old DB to create New one ");
    System.out.println("--------------------------------------------------------------------------------");
    System.out.println("          Functions          |          Syntax               ");
    System.out.println("1. Create Database - "+ (user.getDatabase()==null?"Enabled ":"Disabled")+" | CREATE DATABASE DATA
    System.out.println("2. Delete Database - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | DROP DATABASE DATAB
    System.out.println("3. CREATE TABLES   - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | CREATE TABLE TABLE_N
    System.out.println("4. SELECT QUERY    - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | SELECT FIELD_NAMES,
    System.out.println("5. INSERT QUERY    - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | INSERT INTO TABLE_NA
    System.out.println("6. UPDATE QUERY    - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | UPDATE TABLE_NAME SE
    System.out.println("7. DELETE QUERY    - "+ (user.getDatabase()!=null?"Enabled ":"Disabled")+" | DELETE FROM TABLE_NA
    System.out.println("--------------------------------------------------------------------------------");
    while (true){
        Query.SQLEditor(user);
    }
```

Fig 10 UserOperation(User user)

2. One Main method

```
public class Main {
    no usages  Darshil +1 *
    public static void main(String[] args) {

        init();
        options();

        Scanner s = new Scanner(System.in);

        String choice;

        boolean flag = true;

        do {
            System.out.print("Enter your choice: ");
            choice = s.nextLine();

            switch (choice){
                case "1":

                    User user = User.Login();
                    if(user == null){
                        options();
                        continue;
                    }
                    else {
                        flag = false;
                    }
                    UserOperations(user);
                    break;
                case "2":
                    User.SignUp();
                    break;
                case "3":
                    flag = false;
                    System.out.println("Application Terminated.");
                    break;

                default:System.out.println("Wrong choice, enter either 1 or 2.");
                    break;
            }
            if(flag == false){
                break;
            }
            options();

        }while(flag);

        System.out.println();
    }
```

Fig 11 Main method()

Here is the conclusion of how these work together.

- **User Authentication Module -** Handles user management system for this DBMS system.
- **Main Java fie -** Start point of program execution for a user to login signup and exit. Once user login() then he will continuously see the sql editor.

Outputs:

Test Case 1: Successful Signup, Successful Login, Friendly Navigation

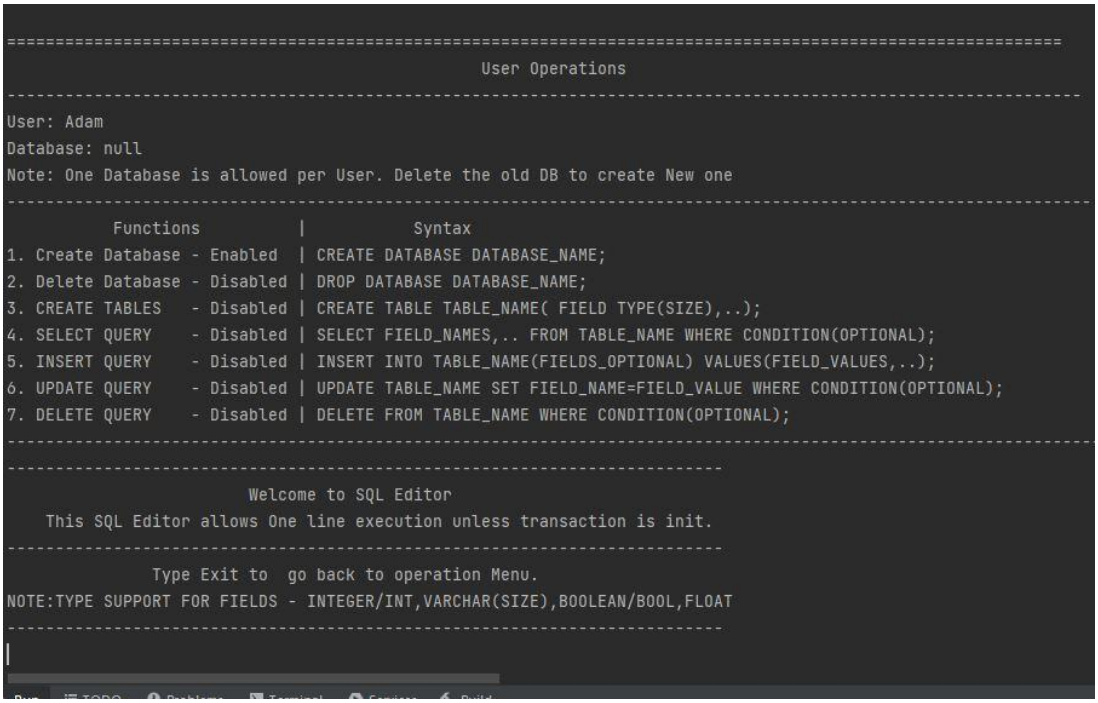
```
================================================================================
                    Welcome to Darshil's DBMS System
================================================================================
--------------------------------------------------------------------------------
 To access the DBMS System, Either use credential or Setup new credential
--------------------------------------------------------------------------------
1. Sign In
2. Register/Create Account
3. Exit
--------------------------------------------------------------------------------
Enter your choice: 2
User info is Loaded.
Enter User id: Adam
Enter User password: 123
Set your security question for future verification.
What is your favourite color?: idont
Successfully Signed Up, Please go to Login to use your account.
```

(1)

```
 To access the DBMS System, Either use credential or Setup new credential
--------------------------------------------------------------------------------
1. Sign In
2. Register/Create Account
3. Exit
--------------------------------------------------------------------------------
Enter your choice: 1
Enter User id: Adam
Enter User password: 123
What is your favourite color?: idont

User Authenticated.......
```

(2)

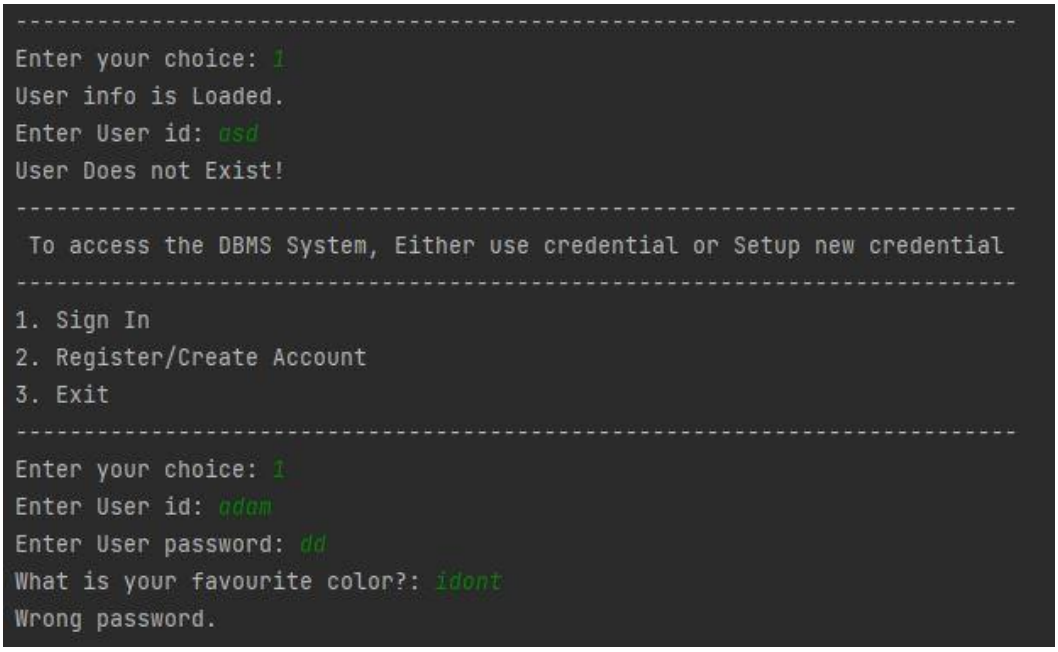


(3)

Fig 12 Test Case -1

Here we can see successful execution as shown in the output.

Test Case - 2: successful Signup(), Failed Login() user-id failed, password failed



(1)

Test Case -3: successful Signup(), Failed Login() user-id userid -pass, password-pass, security-failed.

```
Enter your choice: 1
Enter User id: adam
Enter User password: 123
What is your favourite color?: asd
Wrong Security answer.
--------------------------------------------------------------------------
```

(2)

Fig 13 Test Case-2

Let's Check where we are for this project

- **User Authentication - Done**
  - **Storing User - Fetching the User Data/Stroing the User - Done**
  - **Password protection - Done**
  - **2-F Authentication - Done**

Now,

SQL Parser is the package which I have implemented and has functionality like classes and methods to **parse the SQL queries** and **file operation** based on the queries will also be handled by that package.

SQL Parser has two classes **Table and Query Class**, Query class parses a particular sql query and sends it to the Table class internal table to perform and store the data in the table file.

- **Query Class & important methods**
  These are some important classes and methods which is used for parsing the data.
  Not going into the details because they are long to explain.

  For parsing the information Regex is used, which makes data to be fetched from string easier but unfortunately not all the methods are able to successfully implement where conditions.

```
public class Query {

    17 usages
    public static User currentUser;

    1 usage   Darshil Patel +1
    public static boolean CREATE(String query){...}

    1 usage   Darshil Patel +1
    public static void SELECT(String query){...}

    1 usage   Darshil Patel +1
    public static boolean INSERT(String query){...}

    1 usage   Darshil +1
    public static boolean UPDATE(String query){...}

    1 usage   Darshil
    public static boolean DELETE(String query){...}

    1 usage   Darshil +1
    public static void QueryHandler(String query,boolean transaction){...}

    1 usage   Darshil
    public static void initSQL(){...}
```

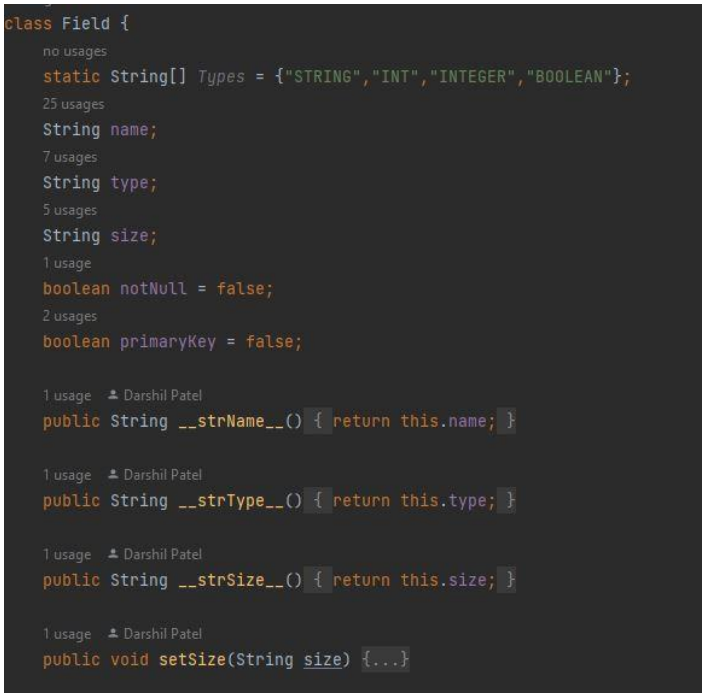
Fig 14 Query Class

- **Table Class & important methods.**
  I loved how all the work together once they are implemented, which makes it look cooler, especially this class.

  Each object of the table class can be considered as a table and you can perform the operation by running its implemented methods.

  The Query class uses the Table class to perform and invoke methods.

(1)



(2)

Field class is for handling a particular field in the table its object is created in the Table class as a data member. Field class has few methods to validate information when storing the data in the file for example type checking etc.

Now let's get back to the flow, last time we ran the program we were successfully able to go the editor.

At the end of the main class, it invokes the method of the Query class called **Query.SQLEditor(user);** to start execution of taking SQL commands and running queries.

Now let's see each individual query with the input and see how it looks in the output.

- **Database and create table successfully.**

```
--------------------------------------------------------------------------
                        Welcome to SQL Editor
    This SQL Editor allows One line execution unless transaction is init.
--------------------------------------------------------------------------
              Type Exit to  go back to operation Menu.
NOTE:TYPE SUPPORT FOR FIELDS - INTEGER/INT,VARCHAR(SIZE),BOOLEAN/BOOL,FLOAT
--------------------------------------------------------------------------
create database  mydb;
DB created.
create table student (name varchar (100) primary key, id int);
------------------------------------------------------------
create table student (name varchar (100) primary key, id int)
name varchar (100) primary key, id int
Table created Successfully
```