Criteria C: Development

TABLE OF CONTENT

List of Techniques:

1. **Database Connectivity**
2. **Use of SQL Commands**
3. **Simple Selection and Complex Selection**
4. **Loops and Nested Loops**
5. **Arrays**
6. **Sorting**

9. Constructors

10.Inheritance

11.Polymorphism

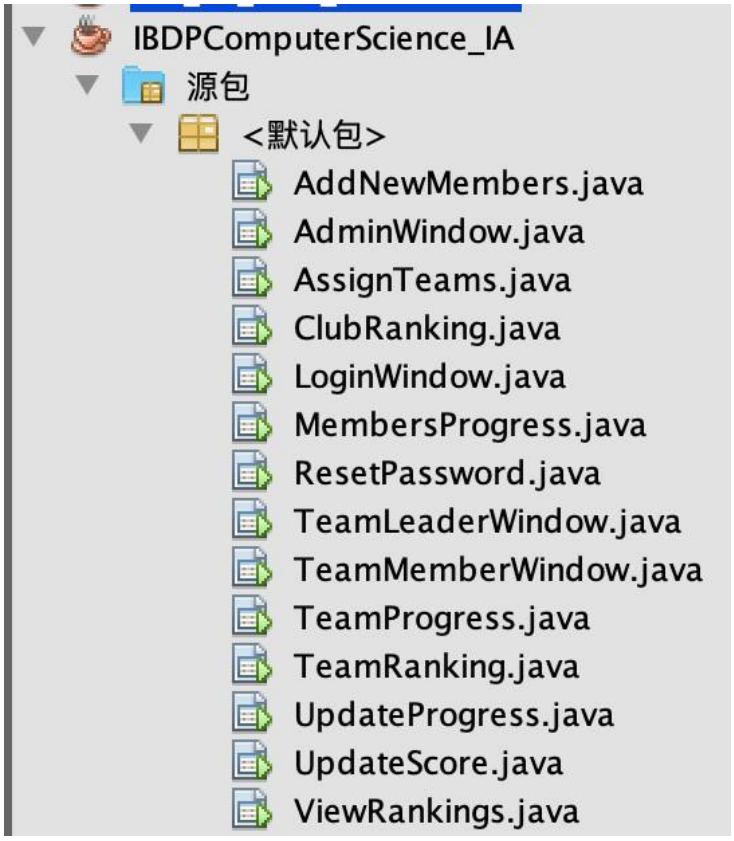
14.Using standard operations of collections

15.Use of sentinels or flags

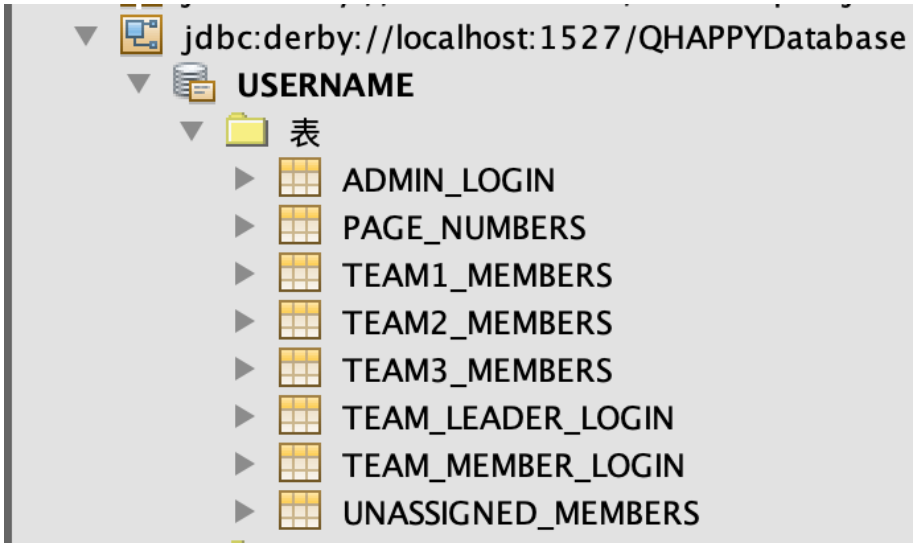
17.Use of Swing Controls, Swing Menus, and Swing Windows

***Full Source Code could be seen in Appendix. Here I would mainly present how these techniques are being implemented in the program.**

Current Classes



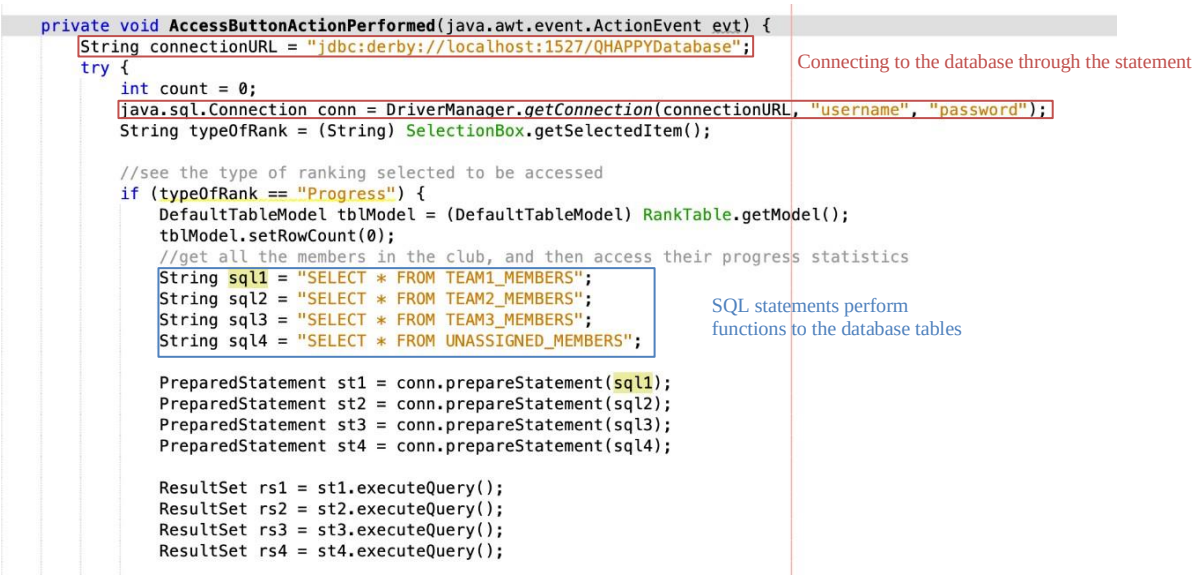
Database Tables



1. **Database Connectivity; Use of SQL Commands; Use of Swing Controls, Swing Menus, Swing Windows**

   These features are in fact shown virtually in all of my User Interfaces (UIs) designs through jFrames.

   For example, in the Club Ranking UI, I need to connect to the database and obtain all of the club members' current information of their overall progresses and points. In the image below, the **connectionURL** and **conn** is where I am attempting to connect to the database, and the different SQL Commands could be seen in **sql1**, **sql2**, **sql3**, **sql4**, etc.




```
private void AccessButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String connectionURL = "jdbc:derby://localhost:1527/QHAPPYDatabase";
    try {
        int count = 0;
        java.sql.Connection conn = DriverManager.getConnection(connectionURL, "username", "password");
        String typeOfRank = (String) SelectionBox.getSelectedItem();

        //see the type of ranking selected to be accessed
        if (typeOfRank == "Progress") {
            DefaultTableModel tblModel = (DefaultTableModel) RankTable.getModel();
            tblModel.setRowCount(0);
            //get all the members in the club, and then access their progress statistics
            String sql1 = "SELECT * FROM TEAM1_MEMBERS";
            String sql2 = "SELECT * FROM TEAM2_MEMBERS";
            String sql3 = "SELECT * FROM TEAM3_MEMBERS";
            String sql4 = "SELECT * FROM UNASSIGNED_MEMBERS";

            PreparedStatement st1 = conn.prepareStatement(sql1);
            PreparedStatement st2 = conn.prepareStatement(sql2);
            PreparedStatement st3 = conn.prepareStatement(sql3);
            PreparedStatement st4 = conn.prepareStatement(sql4);

            ResultSet rs1 = st1.executeQuery();
            ResultSet rs2 = st2.executeQuery();
            ResultSet rs3 = st3.executeQuery();
            ResultSet rs4 = st4.executeQuery();
```

In the UIs, I have mainly used jTable, jComboBox, jButton, and jRadioButton for rankings, selections, and performing functions. Some evidence could be seen in below in some of the designed UIs.

2. **Inheritance: Constructors: Polymorphism**

In my source codes, all the UIs inherit from the javax.swing.JFrame class, with some evidence shown below in *Login Window UI, Team Ranking UI, Assign Teams UI*.

```
public class LoginWindow extends javax.swing.JFrame {

public class TeamRanking extends javax.swing.JFrame {

public class AssignTeams extends javax.swing.JFrame {
```

The "extends" shows the UIs are inherited

In addition to inheritance, the constructors are also visible in all of my UI designs. Some of the constructors might only contain the **initComponents()** method where lays out the JFrame, while some others may include my user-defined methods since I wish these methods to be automatically called once the UI is being shown (e.g., **updateBox()** in the AssignTeams Constructor).

```
public AssignTeams() {
    initComponents();
    Toolkit toolkit = getToolkit();
    Dimension size = toolkit.getScreenSize();
    setLocation(size.width / 2 - getWidth() / 2, size.height / 2 - getHeight() / 2); //center the GUI
    updateBox();
}

public MembersProgress() {
    initComponents();
    Toolkit toolkit = getToolkit();
    Dimension size = toolkit.getScreenSize();
    setLocation(size.width / 2 - getWidth() / 2, size.height / 2 - getHeight() / 2);
}
```

In the constructors, I would put the UIs onto the center of my screen

In some programs' structure, I also built polymorphism where some private variables are kept hidden as the access modifiers of the variables are set to **"private"** so that only the class itself could access and change the variable. This is evident in the Assign Team UI, where I needed to set specific counters, such as **HonorCounter**, **ScholasticCounter**, and **VarsityCounter** to keep track of the number of people included in an academic group, in case that after assigning the teams and academic groups, one academic group would have more than 3 members, which is against the rules, as mentioned in Criteria A. Combined these instance variables with the constructor and the methods, it forms a polymorphism.

```
public class AssignTeams extends javax.swing.JFrame {

    private int HonorCounter = 0;
    private int ScholasticCounter = 0;
    private int VarsityCounter = 0;
    private String teamSelected;

    /**
     * Creates new form AssignTeams
     */
    public AssignTeams() {
        initComponents();
        Toolkit toolkit = getToolkit();
        Dimension size = toolkit.getScreenSize();
        setLocation(size.width / 2 - getWidth() / 2, size.height / 2 - getHeight() / 2); //center the GUI
        updateBox();
    }

    private void updateBox() {
        String connectionURL = "jdbc:derby://localhost:1527/QHAPPYDatabase";
        try {
            java.sql.Connection conn = DriverManager.getConnection(connectionURL, "username", "password");
            String sql10 = "SELECT * FROM UNASSIGNED_MEMBERS";
            PreparedStatement st10 = conn.prepareStatement(sql10);
            //get all unassigned members and input them to each jComboBox
            ResultSet rs10 = st10.executeQuery();
            while (rs10.next()) {
                HonorGroupBox1.addItem(rs10.getString("NAME"));
                HonorGroupBox2.addItem(rs10.getString("NAME"));
                HonorGroupBox3.addItem(rs10.getString("NAME"));
                ScholasticGroupBox1.addItem(rs10.getString("NAME"));
                ScholasticGroupBox2.addItem(rs10.getString("NAME"));
                ScholasticGroupBox3.addItem(rs10.getString("NAME"));
                VarsityGroupBox1.addItem(rs10.getString("NAME"));
```

Setting the variables as "private" ensures the security of the data through polymorphism.

3. **Using Standard Operations of Collections: Arrays: Use of Sentinels or Flags**

These complexities are semantic to the *Club Ranking UI* alone. In Club Ranking, I need to get all members' information on their overall progresses and points. However, I only have fragmented databases recording these data entries (TEAM1_MEMBERS, TEAM2_MEMBERS, TEAM3_MEMBERS, UNASSIGNED_MEMBERS as shown above). Therefore, I concatenated the different information together and add these into an ArrayList object **allProgress** and **allPoint** so that I could then make these ArrayLists into arrays, **theArray**, to go through bubble sort to get sorted into descending order (oracle).

```
*/
ArrayList<Integer> allProgress = new ArrayList<>();
ArrayList<String> UsedNames = new ArrayList<>();

//insert all the points into the allProgress array list
while (rs1.next()) {
    allProgress.add(rs1.getInt("OVERALL_PROGRESS"));
}
while (rs2.next()) {
    allProgress.add(rs2.getInt("OVERALL_PROGRESS"));
}
while (rs3.next()) {
    allProgress.add(rs3.getInt("OVERALL_PROGRESS"));
}
while (rs4.next()) {
    allProgress.add(rs4.getInt("OVERALL_PROGRESS"));
}
//convert the array list into a 1-D array for bubble so
int arrLength = allProgress.size();
int[] theArray = new int[arrLength];
for (int i = 0; i < arrLength; i++) {
    theArray[i] = allProgress.get(i);
}
bubbleSort(theArray, arrLength);
```

The use of arrays concatenates all the elements together so that they could be sorted later

```
ArrayList<Integer> allPoint = new ArrayList<>();
ArrayList<String> UsedNames = new ArrayList<>();
while (rs1.next()) {
    allPoint.add(rs1.getInt("POINTS"));
}
while (rs2.next()) {
    allPoint.add(rs2.getInt("POINTS"));
}
while (rs3.next()) {
    allPoint.add(rs3.getInt("POINTS"));
}
while (rs4.next()) {
    allPoint.add(rs4.getInt("POINTS"));
}
int arrLength = allPoint.size();
int[] theArray = new int[arrLength];
for (int i = 0; i < arrLength; i++) {
    theArray[i] = allPoint.get(i);
}
bubbleSort(theArray, arrLength);
```

ArrayList is an ordered collection, and the specific functions of ArrayLists have been used in the programs here

Another ArrayList used here, as can be seen, is the **UsedNames** ArrayList. This is used to check if the found member has been displayed in the ranking already as there may be ties in their progress/point that leads to confusions in the system. To check the members, it goes through **.listIterator()** and **.hasNext()** method, all of which are standard operations of Collections ("Java ListIterator Interface").

```
int flag = 0;
//set a flag first
memberName = rs5.getString("NAME");
ListIterator<String> theCheck = UsedNames.listIterator();
//check if the member has been recorded in the UsedNames array list through the iterator
while (theCheck.hasNext()) {
    String comp = theCheck.next();
    if (comp.equals(memberName)) {
        flag += 1;
        //increment the flag by 1 if it has appeared in the UsedNames list
    }
}
if (flag == 0) {
    //only if the flag is 0 could the system add this searched member into the rank
    count += 1;
    String specificRank = String.valueOf(count);
    String tbData1[] = {specificRank, memberName, theStats};
    tblModel.addRow(tbData1);
    UsedNames.add(memberName);
}
```

The flag here is used to detect if there are multiple people with the same points/learning progresses and then delete them so that their names would not appear more than one time on the ranking

It could also be seen from the two images above that flags have been used (**swapped** in **bubbleSort()** and **flag** in checking the used names in the *Club Ranking UI*). **swapped** is used to check if further sorts need to be executed, while **flag** is to check if the member's name has already been put into the JTable in the *Club Ranking UI*.

4. **Simple Selection and Complex Selection**

Simple Selections could be seen in many places in my programs, where I want to determine if the user has made a selection or not, and then determine if the system needs to output an error message or make corresponding reactions in the system. For example, in the View Rankings UI, the main function of the code for **AccessButtonActionPerformed()** is simply to run through these simple selection programs.

```
private void AccessButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (TeamRankButton.isSelected() == true) {
        String currentSelect= (String) TeamBox.getSelectedItem();
        if(currentSelect.equals("Please Select")){
            JOptionPane.showMessageDialog(null, "Please Select a Team!");
            //let users select if they have yet to make a selection
        }else{
            //define the new teamUsed static variable
            LoginWindow.teamUsed = currentSelect;
        }
        //dispose to the team ranking UI
        new TeamRanking().setVisible(true);
        dispose();

    } else if (ClubRankButton.isSelected() == true) {
        //dispose to the club ranking UI
        new ClubRanking().setVisible(true);
        dispose();
    }else{
        //if have not select a kind of ranking, then output an error message
        JOptionPane.showMessageDialog(null, "Please Select a Kind of Ranking!");
    }
}
```

Here, the administrator needs to make two selections: type of ranking and if it is a team ranking, then select the team that he/she wants to see. Then, if there have been no selections, then the system would output error message indicating that the administrator should do so.

Complex Selection is used the *Assign Teams UI*, where the system needs to check if the chosen new members for each academic group has repeated or not. Here, there is 36 *OR Statements* as the nine new members would all be compared to each other. If there is a found repetition of the member's name, then an error message would pop up, or else the program would move on to further assign these selected members to their newly allocated teams and academic groups.

```
//check if the user has selected a team to operate with
if (teamSelected.equals("Select a Team...")) {
    JOptionPane.showMessageDialog(null, "Please select a Team!");

    //make sure they haven't put in the same member into two different academic groups in a team
} else if (NewHonorMember1.equals(NewHonorMember2) || NewHonorMember1.equals(NewHonorMember3)
        || NewHonorMember1.equals(NewScholasticMember1) || NewHonorMember1.equals(NewScholasticMember2)
        || NewHonorMember1.equals(NewScholasticMember3) || NewHonorMember1.equals(NewVarsityMember1)
        || NewHonorMember1.equals(NewVarsityMember2) || NewHonorMember1.equals(NewVarsityMember3)
        || NewHonorMember2.equals(NewHonorMember3)
        || NewHonorMember2.equals(NewScholasticMember1) || NewHonorMember2.equals(NewScholasticMember2)
        || NewHonorMember2.equals(NewScholasticMember3) || NewHonorMember2.equals(NewVarsityMember1)
        || NewHonorMember2.equals(NewVarsityMember2) || NewHonorMember2.equals(NewVarsityMember3)
        || NewHonorMember3.equals(NewScholasticMember2) || NewHonorMember3.equals(NewScholasticMember1)
        || NewHonorMember3.equals(NewScholasticMember3) || NewHonorMember3.equals(NewVarsityMember1)
        || NewHonorMember3.equals(NewVarsityMember2) || NewHonorMember3.equals(NewVarsityMember3)
        || NewScholasticMember1.equals(NewScholasticMember2)
        || NewScholasticMember1.equals(NewScholasticMember3) || NewScholasticMember1.equals(NewVarsityMember1)
        || NewScholasticMember1.equals(NewVarsityMember2) || NewScholasticMember1.equals(NewVarsityMember3)
        || NewScholasticMember2.equals(NewScholasticMember3) || NewScholasticMember2.equals(NewVarsityMember1)
        || NewScholasticMember2.equals(NewVarsityMember2) || NewScholasticMember2.equals(NewVarsityMember3)
        || NewScholasticMember3.equals(NewVarsityMember1)
        || NewScholasticMember3.equals(NewVarsityMember2) || NewScholasticMember3.equals(NewVarsityMember3)
        || NewVarsityMember1.equals(NewVarsityMember2) || NewVarsityMember1.equals(NewVarsityMember3)
        || NewVarsityMember2.equals(NewVarsityMember3)) {
    JOptionPane.showMessageDialog(null, "Can't put one person in two groups!");
    successful = false;
} else {
```

A very complex selection is used here because a new member could only be assigned to one of the academic groups of one team

5. Loops and Nested Loops: Sorting

The most obvious loops I have used if **while loops**, since I need to go through all the searched entries after a **"SELECT"** SQL Command. For example, in the *Login Window UI*, the while loops are used to check whether the input username exists in **rs1**, **rs2**, or **rs3**, which correspondingly mark the status as team member, team leader, and administrator.

```
while (rs1.next()) {
    String getUsername = rs1.getString("QDNumber");
    String getPassword = rs1.getString("Password");
    String getName = rs1.getString("Name"); //re-check if usernames and passwords match
    if (userName.equals(getUsername) && passWord.equals(getPassword)) {
        /**define the static variables, set the login status
         * (either team member, team leader, or administrator)
         * same for the remaining while loops
         * */
        nameUsing = getName;
        status = "member";
        found = true;
        JOptionPane.showMessageDialog(null, "Hi " + nameUsing + ", Welcome to the System");
        teamMemberLogin = true;
    }
}

while (rs2.next()) {
    String getUsername = rs2.getString("QDNumber");
    String getPassword = rs2.getString("Password");
    String getName = rs2.getString("Name");
    if (passWord.equals(getPassword) && userName.equals(getUsername)) {

        nameUsing = getName;
        status = "leader";
        found = true;
        teamLeaderLogin = true;
        JOptionPane.showMessageDialog(null, "Hi " + nameUsing + ", Welcome to the System");
    }
}

while (rs3.next()) {
    String getUsername = rs3.getString("QDNumber");
    String getPassword = rs3.getString("Password");
    String getName = rs3.getString("Name");
    if (userName.equals(getUsername) && passWord.equals(getPassword)) {
        nameUsing = getName;
        status = "admin";
        found = true;
        adminLogin = true;
        JOptionPane.showMessageDialog(null, "Hi " + nameUsing + ", Welcome to the System");
    }
```

Another kind of loop used is **for loops**. For example, in the *Members Progress UI*, I need to display a specific member's progress in each subject into a JTable. To do this, I form rows of arrays of objects, **allProgresses** and **subjects**, and insert them into the JTable, using for loops (Knowledge to Share).

```
//extract the information on their specific progresses
String Lit_Progress = String.valueOf(rs4.getInt("LIT_PROGRESS"));
String Art_Progress = String.valueOf(rs4.getInt("ART_PROGRESS"));
String Sci_Progress = String.valueOf(rs4.getInt("SCI_PROGRESS"));
String SocSci_Progress = String.valueOf(rs4.getInt("SOCSCI_PROGRESS"));
String Mus_Progress = String.valueOf(rs4.getInt("MUS_PROGRESS"));
String Mat_Progress = String.valueOf(rs4.getInt("MAT_PROGRESS"));
String Econ_Progress = String.valueOf(rs4.getInt("ECON_PROGRESS"));

//compile a new row to show all the progresses on the seven subjects
String[] allProgresses = {Lit_Progress, Art_Progress, Sci_Progress, SocSci_Progress,
    Mus_Progress, Mat_Progress, Econ_Progress};
String[] subjects = {"Literature", "Art", "Science", "Social Sciece", "Music", "Mathematics", "Economics"};

for(int i = 0; i < 7; i ++){
    //add the new information into the jTable
    String tbData[] = {subjects[i],allProgresses[i]};
    tblModel.addRow(tbData);
}
```

A for loop is used here so that the data entries could be inserted into the jTable

Nested loop could be observed from the **.bubbleSort()** method which is used to sort out the order of points and learning progresses to make rankings ("Bubble Sort – GeekforGeeks.").

```
private void bubbleSort(int arr[], int n) {
    /* a bubble sort method is needed since
    the whole club ranking wouuld need to concatenate the
    databases together, rather than just using "ORDER BY" in
    sql commands for a single database
    */
    int i, j, temp;
    boolean swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        if (swapped == false) {
            break;
        }

    }
}
```

Works Cited

"Bubble Sort - GeeksforGeeks." *GeeksforGeeks*, 2 Feb. 2014, www.geeksforgeeks.org/bubble-sort/. Accessed 3 June 2022.

"Java ListIterator Interface." *Programiz*, www.programiz.com/java-programming/listiterator. Accessed 2 June 2022.

Knowledge to Share. "JTable in JAVA Swing | ADD Data into JTable." *YouTube*, 19 Jan. 2019, www.youtube.com/watch?v=CQMpXGwHeYQ&t=785s. Accessed 2 June 2022.

oracle. "ArrayList (Java Platform SE 8)." *Oracle.com*, 11 Sept. 2019, docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html. Accessed 3 June 2022.

Word Count: 1027

*The word count excludes the title page, in-text citations, and works cited page.

Appendix
APPENDIX_2_Source_Code.pdf