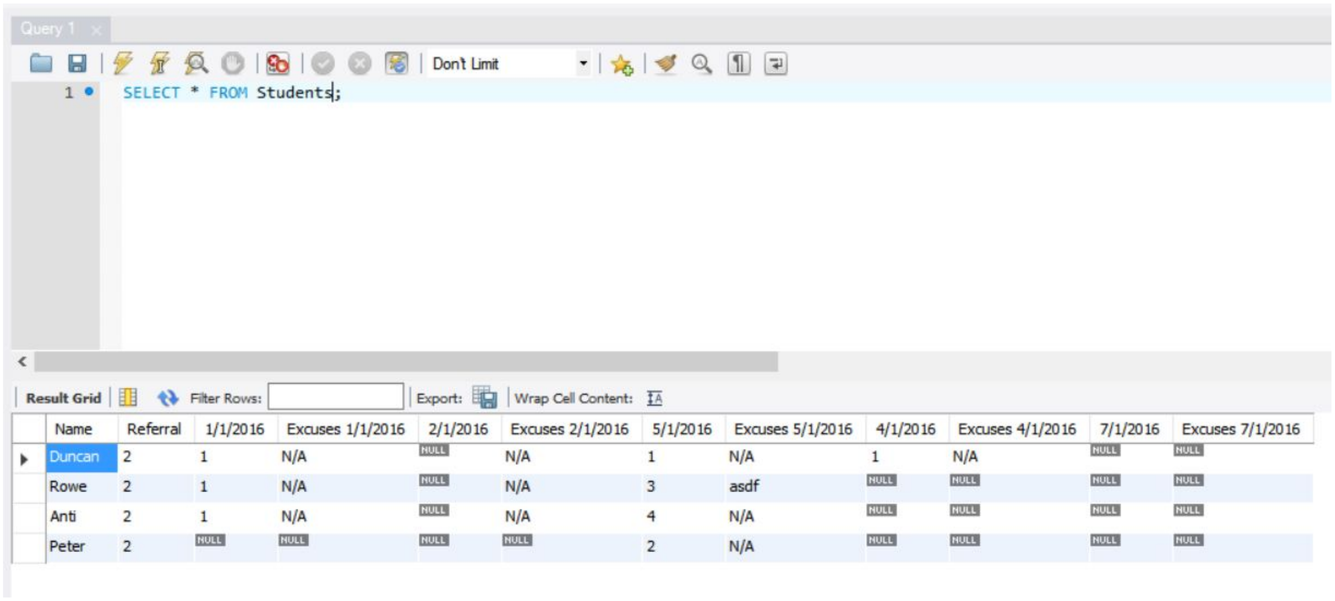
Criterion C: Development

Developing the product

I chose to use a MySQL database as I am familiar with the SQL language and the interface organizes the data well, ensuring both data integrity and data redundancy. To implement my MySQL database, I chose MySQL workbench as I have previous experience with it and it has worked well in the past. I started off my program by initializing a database in MySQL Workbench and creating two tables. One table, called Students, would contain all of the attendance information, while another table called Dates, would contain all the dates. The Dates table was used to easily traverse the Students table and retrieve specific attendance values of students in the Students table.

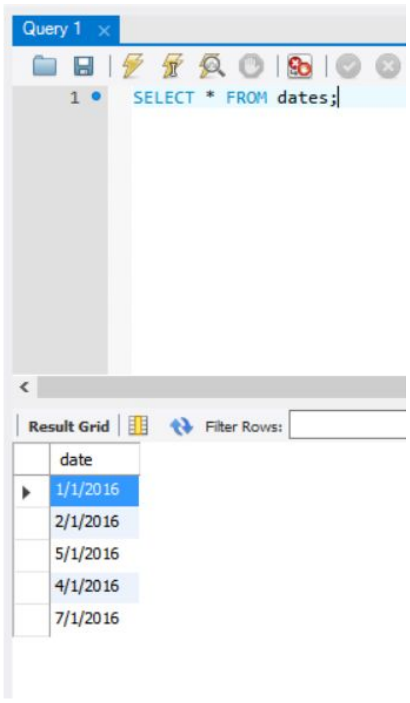


```
SELECT * FROM Students;
```

Name	Referral	1/1/2016	Excuses 1/1/2016	2/1/2016	Excuses 2/1/2016	5/1/2016	Excuses 5/1/2016	4/1/2016	Excuses 4/1/2016	7/1/2016	Excuses 7/1/2016
Duncan	2	1	N/A	NULL	N/A	1	N/A	1	N/A	NULL	NULL
Rowe	2	1	N/A	NULL	N/A	3	asdf	NULL	NULL	NULL	NULL
Anti	2	1	N/A	NULL	N/A	4	N/A	NULL	NULL	NULL	NULL
Peter	2	NULL	NULL	NULL	NULL	2	N/A	NULL	NULL	NULL	NULL

As you can see, when the Students table is initiated there are two default columns created, Name and Referral. The Name column is dynamically filled while the Referral column has a default value of 0 for each student when created. Two additional columns are added for each SG meeting, a column for the date and a column for excuses on that date as shown below.

```java
//Used by new attendance, adds columns to Student Table and adds dates to the Date Table
public static void newAttendanceSQL(String day, String month, String year) throws SQLException, ClassNotFoundException{

    String input = month + "/" + day + "/" + year;

    Connect();

    PreparedStatement ps = connect
            .prepareStatement("ALTER TABLE Students ADD `" + input  + "` int");

    ps.executeUpdate();

    ps = connect
            .prepareStatement("ALTER TABLE Students ADD `Excuses " + input  + "` varchar(255)");
    ps.executeUpdate();

    ps = connect
            .prepareStatement("INSERT INTO Dates (date) VALUES('" + input + "')");
    ps.executeUpdate();
}
```



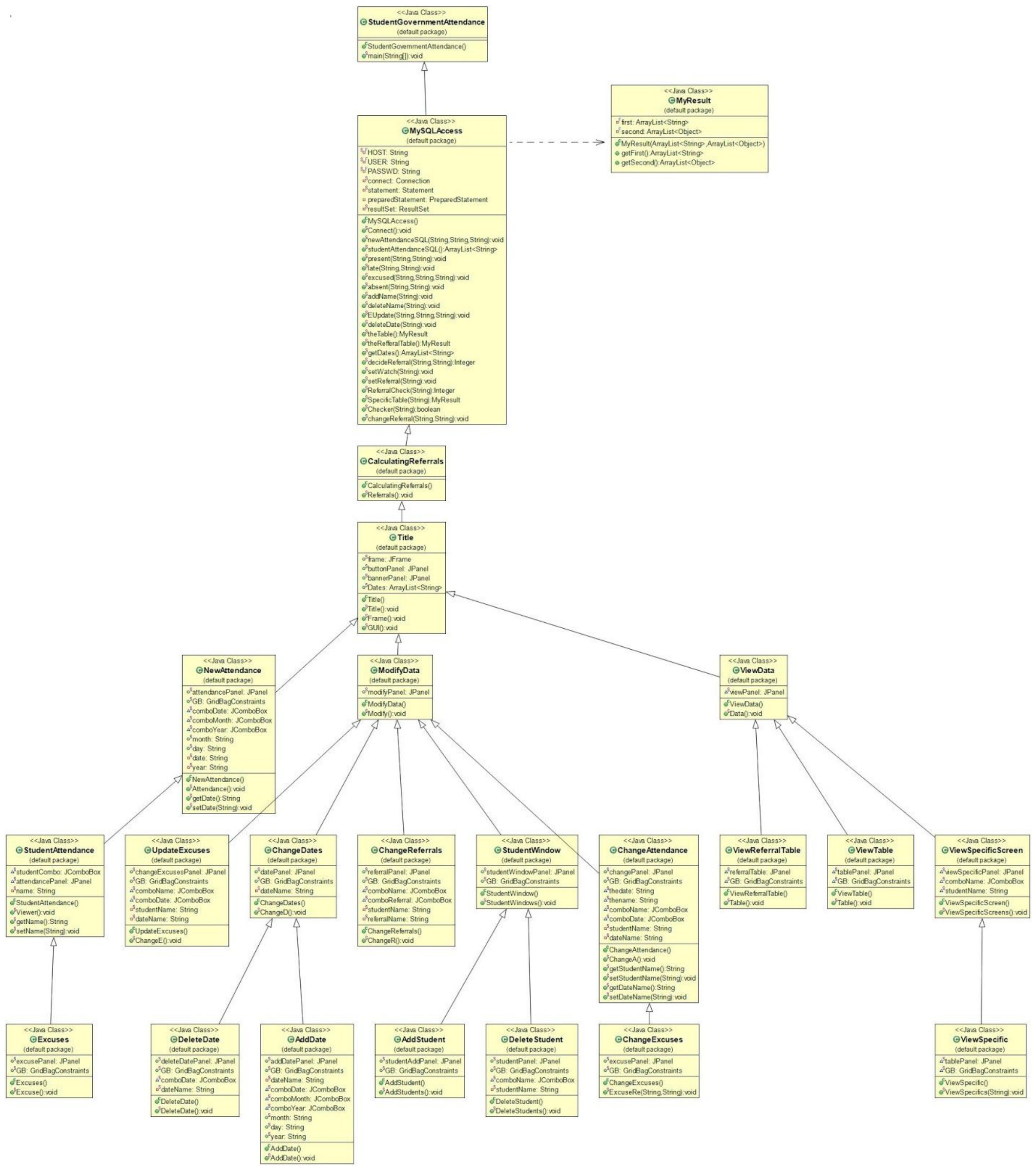
```sql
1 SELECT * FROM dates;
```

date
1/1/2016
2/1/2016
5/1/2016
4/1/2016
7/1/2016

The last prepared statement in the code adds the date value to the table Dates shown above. These two tables are heavily used throughout my program and are necessary to my project.

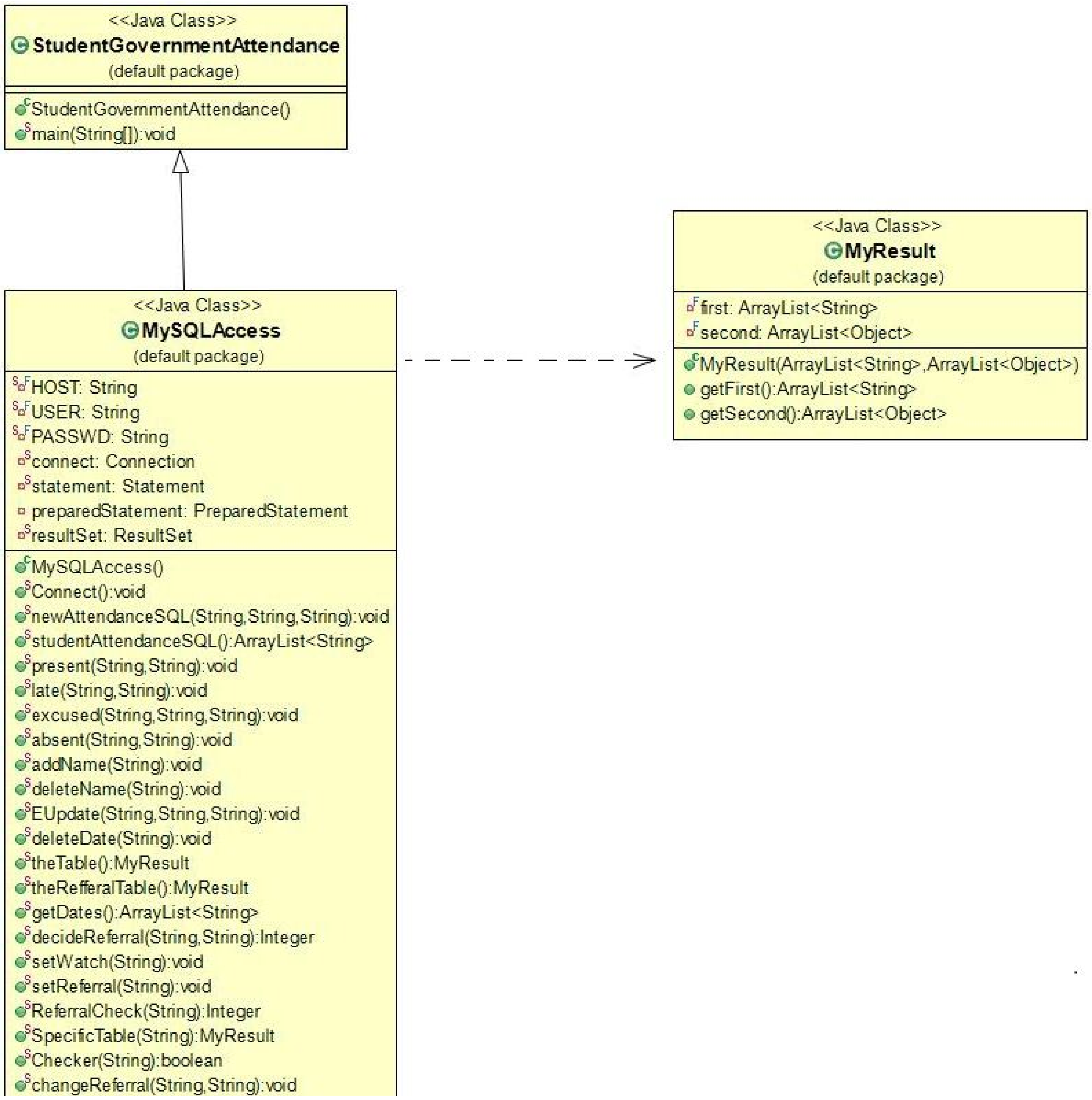
When I started coding my program, I had a clear view of the hierarchical nature of the classes. To implement this hierarchical structure I leveraged java's object oriented property of inheritance to link all of my classes together and avoid duplication of my code. This allowed me to use the same MySQL methods, panel names and variables from other classes in my program, making my coding more efficient and faster to create . The hierarchical nature of my program is shown in the following UML diagram.

Overall UML Diagram



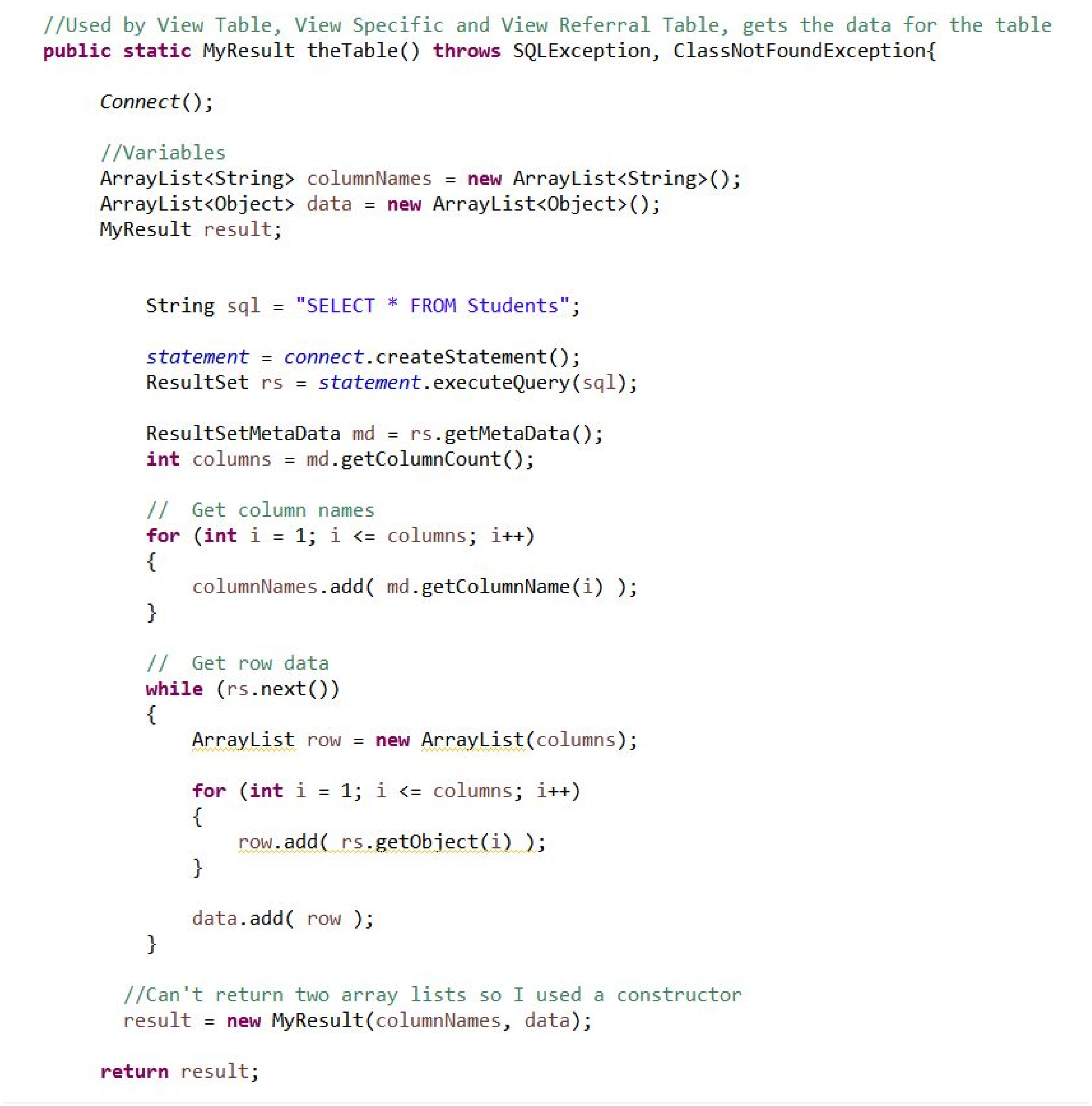
While this diagram is very small, it will be examined in further detail later as it illustrates the overall nature of my program.

Main Class UML


<<Java Class>>
StudentGovernmentAttendance
(default package)
StudentGovernmentAttendance()
main(String[]):void
<<Java Class>>
MySQLAccess
(default package)
HOST: String
USER: String
PASSWD: String
connect: Connection
statement: Statement
preparedStatement: PreparedStatement
resultSet: ResultSet
MySQLAccess()
Connect():void
newAttendanceSQL(String,String,String):void
studentAttendanceSQL():ArrayList<String>
present(String,String):void
late(String,String):void
excused(String,String,String):void
absent(String,String):void
addName(String):void
deleteName(String):void
EUpdate(String,String,String):void
deleteDate(String):void
theTable():MyResult
theRefferalTable():MyResult
getDates():ArrayList<String>
decideReferral(String,String):Integer
setWatch(String):void
setReferral(String):void
ReferralCheck(String):Integer
SpecificTable(String):MyResult
Checker(String):boolean
changeReferral(String,String):void
<<Java Class>>
MyResult
(default package)
first: ArrayList<String>
second: ArrayList<Object>
MyResult(ArrayList<String>,ArrayList<Object>)
getFirst():ArrayList<String>
getSecond():ArrayList<Object>

By using inheritance, I was able to maximize the efficiency of my MySQL class, a class I created to hold all SQL commands. I only needed to create one connection method and could reuse the different database methods in my different children classes.

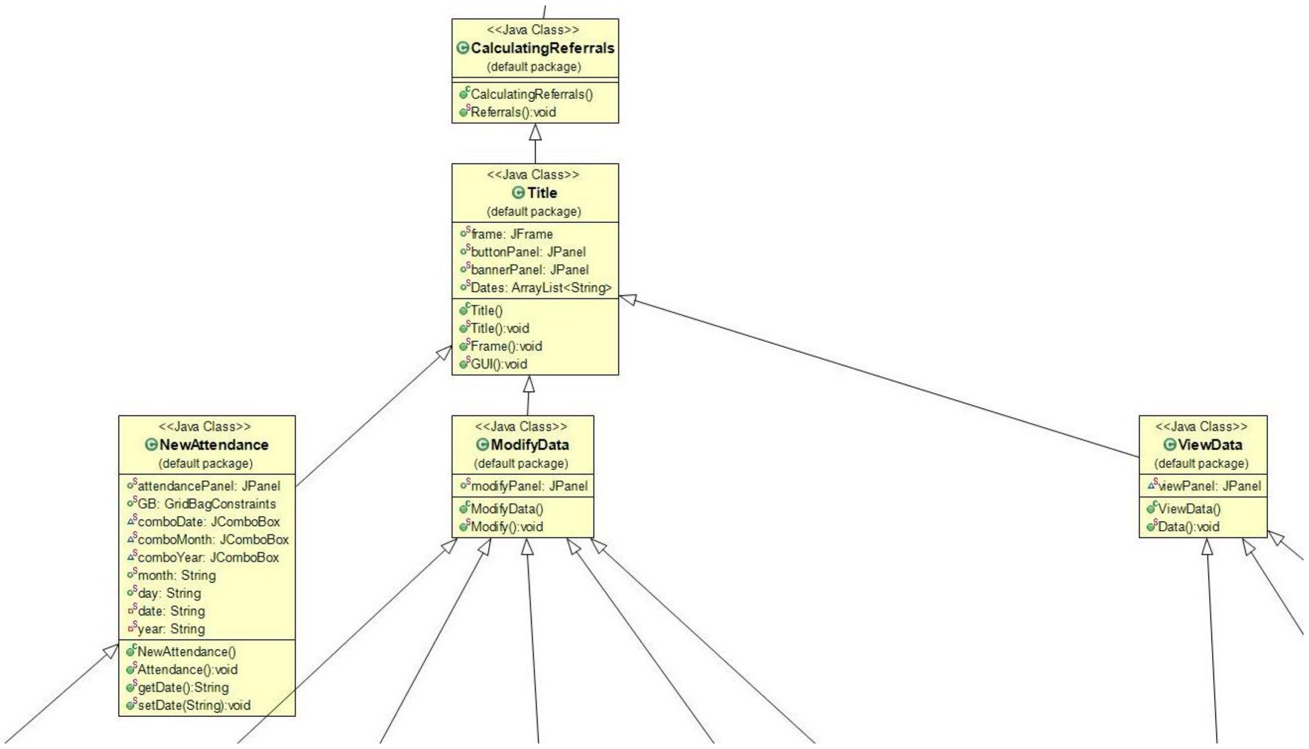
MyResult method in the MySQLAccess class



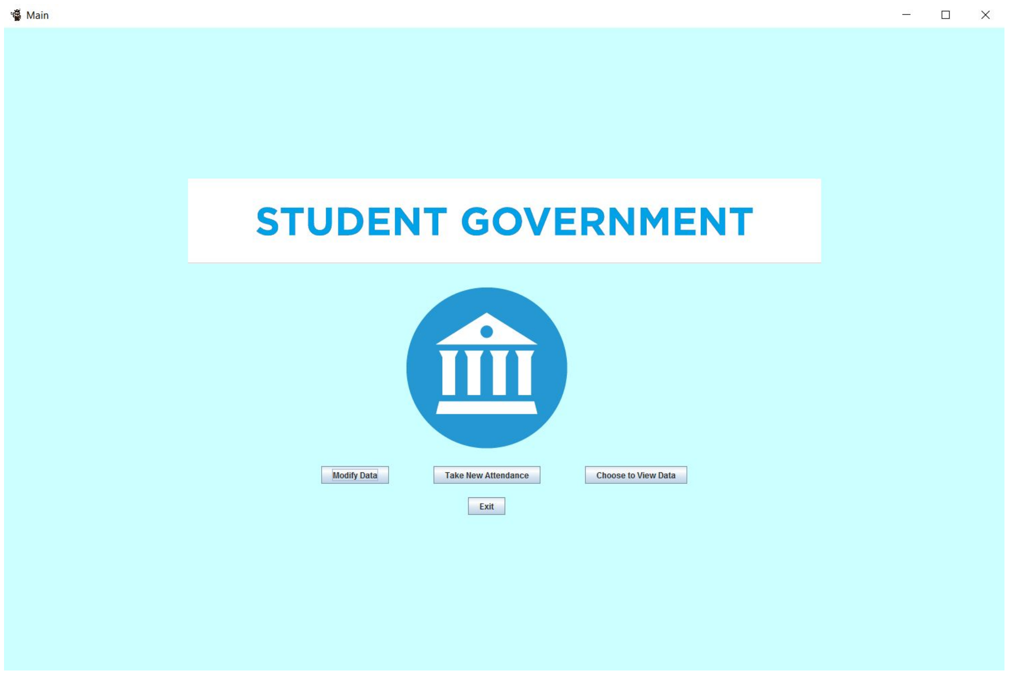
```
//Used by View Table, View Specific and View Referral Table, gets the data for the table
public static MyResult theTable() throws SQLException, ClassNotFoundException{

    Connect();

    //Variables
    ArrayList<String> columnNames = new ArrayList<String>();
    ArrayList<Object> data = new ArrayList<Object>();
    MyResult result;


        String sql = "SELECT * FROM Students";

        statement = connect.createStatement();
        ResultSet rs = statement.executeQuery(sql);

        ResultSetMetaData md = rs.getMetaData();
        int columns = md.getColumnCount();

        //  Get column names
        for (int i = 1; i <= columns; i++)
        {
            columnNames.add( md.getColumnName(i) );
        }

        //  Get row data
        while (rs.next())
        {
            ArrayList row = new ArrayList(columns);

            for (int i = 1; i <= columns; i++)
            {
                row.add( rs.getObject(i) );
            }

            data.add( row );
        }

      //Can't return two array lists so I used a constructor
      result = new MyResult(columnNames, data);

    return result;
```

To build my GUI, I decided on using Swing as it has the same object oriented properties of Java, while also being included in the Java platform for creating GUI's. In order to view data in JFrame, I implemented Jtable, a component of Swing. To transfer the information from the MySQL server into JTable, I first sent a query retrieving all of the information in the table. Then, due to the dynamic nature of the problem, I inserted the information into two arraylists. However, due to Java's restrictions on being unable to return two results, I created a class called MyResult that combines both arrays into a MyResult object and returns that object instead. This allows for both arraylists to be returned and for the View classes to display the data, which will be shown later.

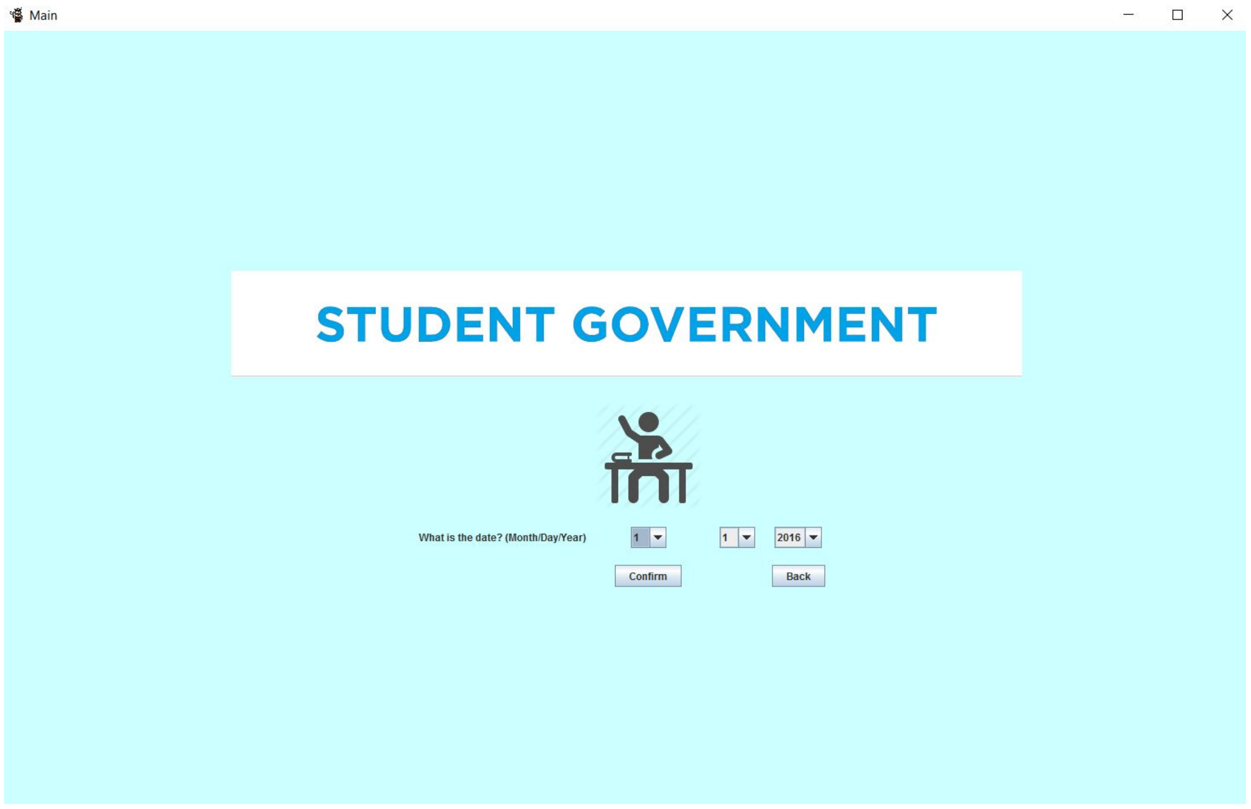
Title UML



The Title class is the main screen that the user sees when they open the program, it allows the user to choose between three different options and is very intuitive.



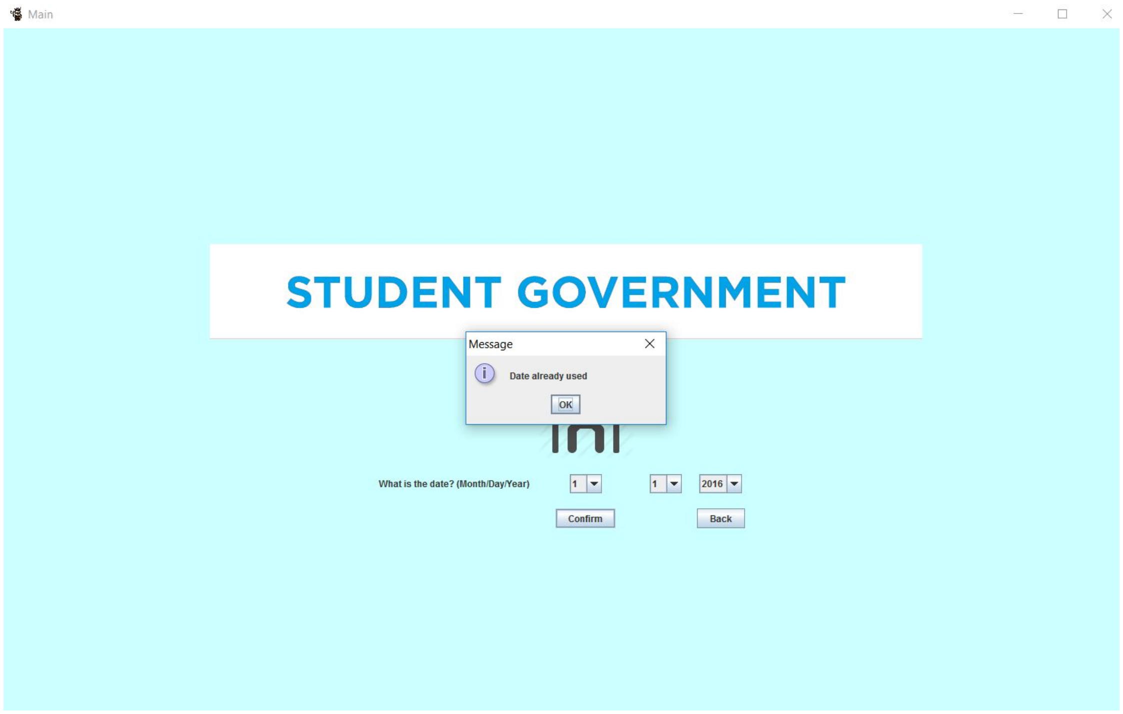
If they choose to take attendance, they will first be asked to select a date.



In order to ensure that a duplicate date is not selected, when the user hits confirm, a MySQLAccess database method is run called Checker.

```
 //Used by Add Date and New Attendance, gets date from Dates
 public static boolean Checker(String date) throws SQLException, ClassNotFoundException{

      Connect();
      resultSet = statement.executeQuery("SELECT date FROM Dates WHERE date = '" + date + "'");

      while(resultSet.next()){
          String x = resultSet.getString("date");

          if(x.equals(date)){
              return true;
          }
      }

      return false;
}
```

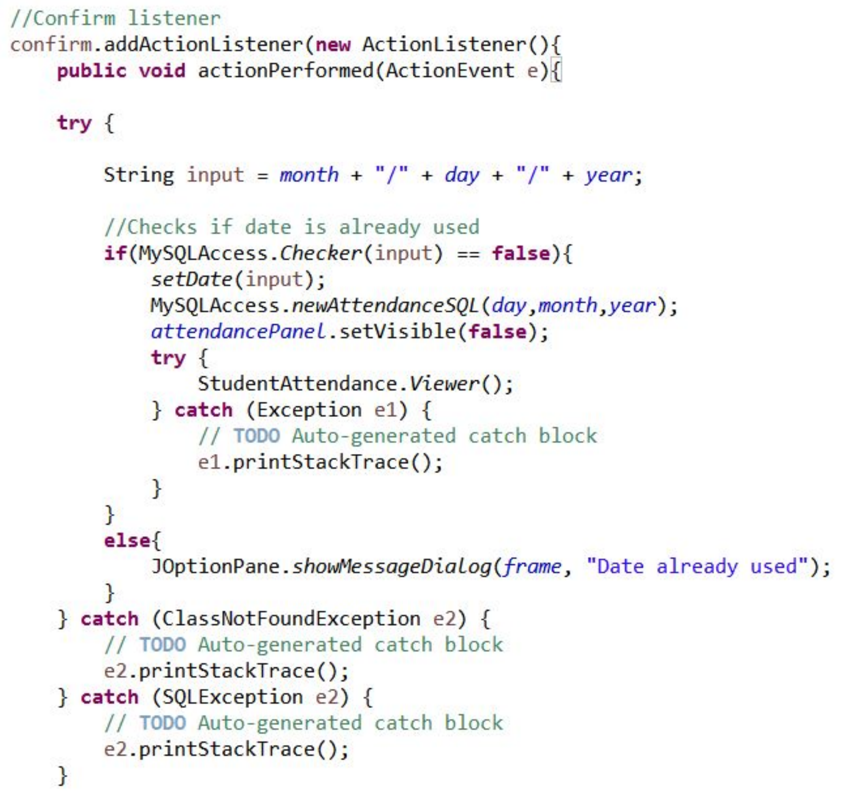
Checker runs through the date table and checks if the date entered is already in the table. If it is a duplicate date then an error is thrown, otherwise the user is allowed to continue.



Date already used

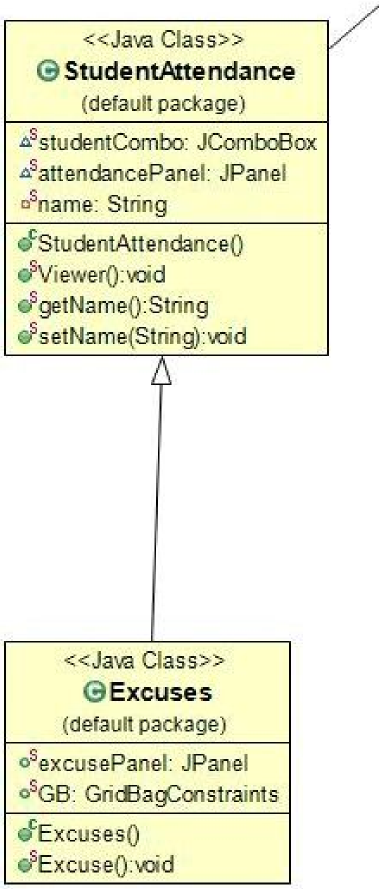
.

Furthermore, to ensure that my program does not crash due to errors, I have used multiple try-catch loops before I send data to the MySQL database as well as when I call other class's methods.

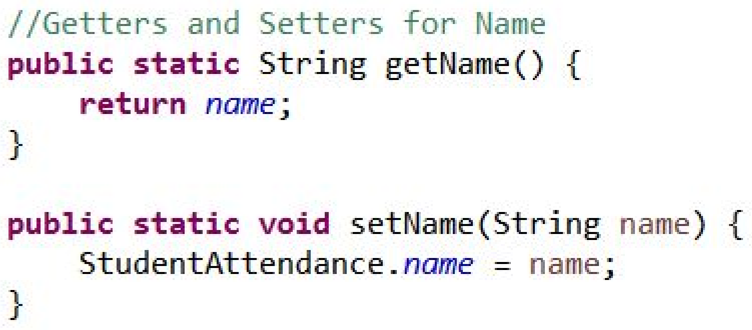


```java
//Confirm listener
confirm.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){

    try {

        String input = month + "/" + day + "/" + year;

        //Checks if date is already used
        if(MySQLAccess.Checker(input) == false){
            setDate(input);
            MySQLAccess.newAttendanceSQL(day,month,year);
            attendancePanel.setVisible(false);
            try {
                StudentAttendance.Viewer();
            } catch (Exception e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
        else{
            JOptionPane.showMessageDialog(frame, "Date already used");
        }
    } catch (ClassNotFoundException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    } catch (SQLException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    }
```

New Attendance UML

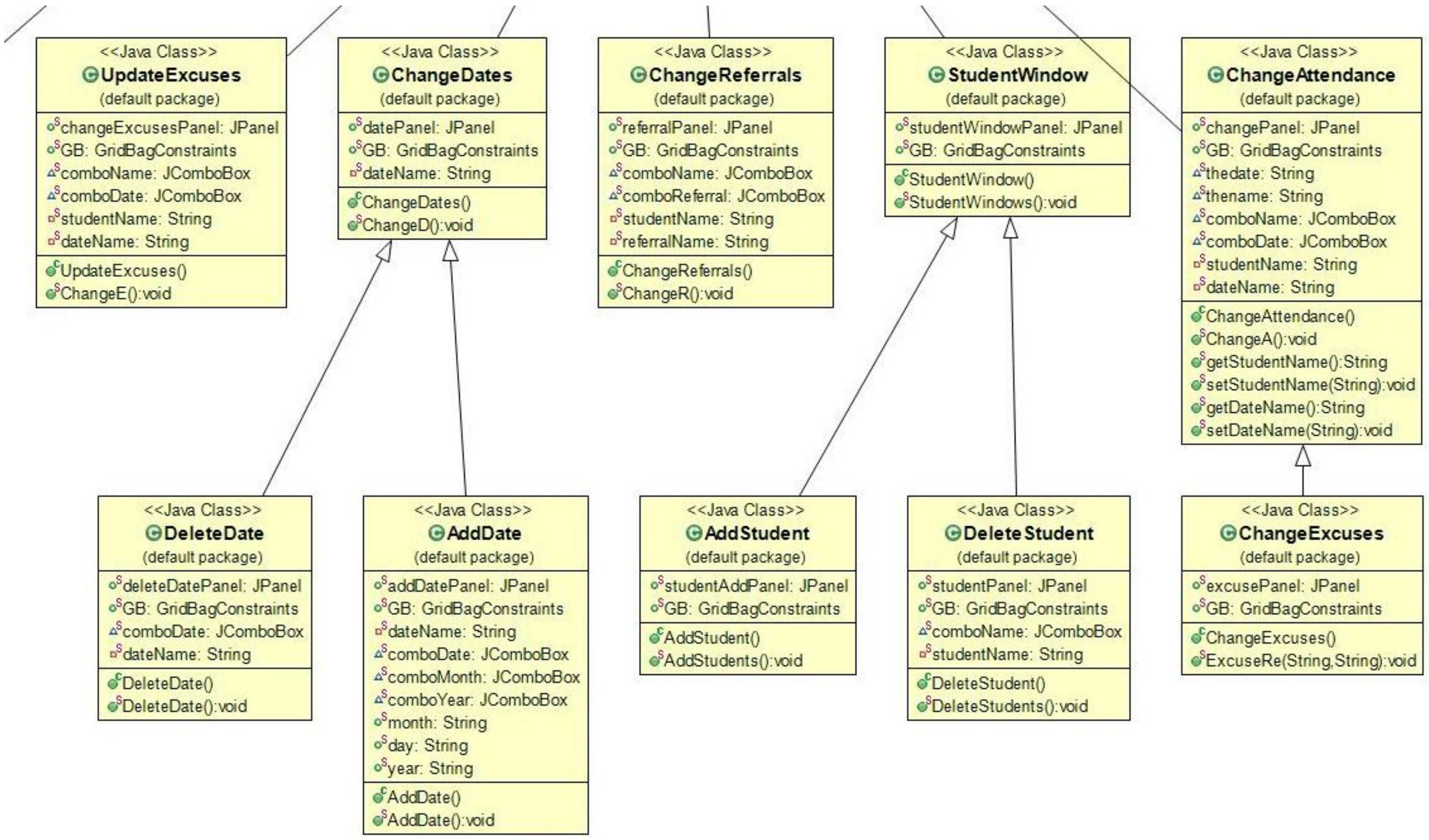


When the user selects the dates, they are lead to the NewAttendance screen. In this class, I have used the object oriented property of encapsulation on the name in order to maintain integrity and security. This helped with testing and debugging the two classes as well as allowed me to receive the NewAttendance date value from the StudentAttendance class, saving time and code.



```
//Getters and Setters for Name
public static String getName() {
    return name;
}

public static void setName(String name) {
    StudentAttendance.name = name;
}
```

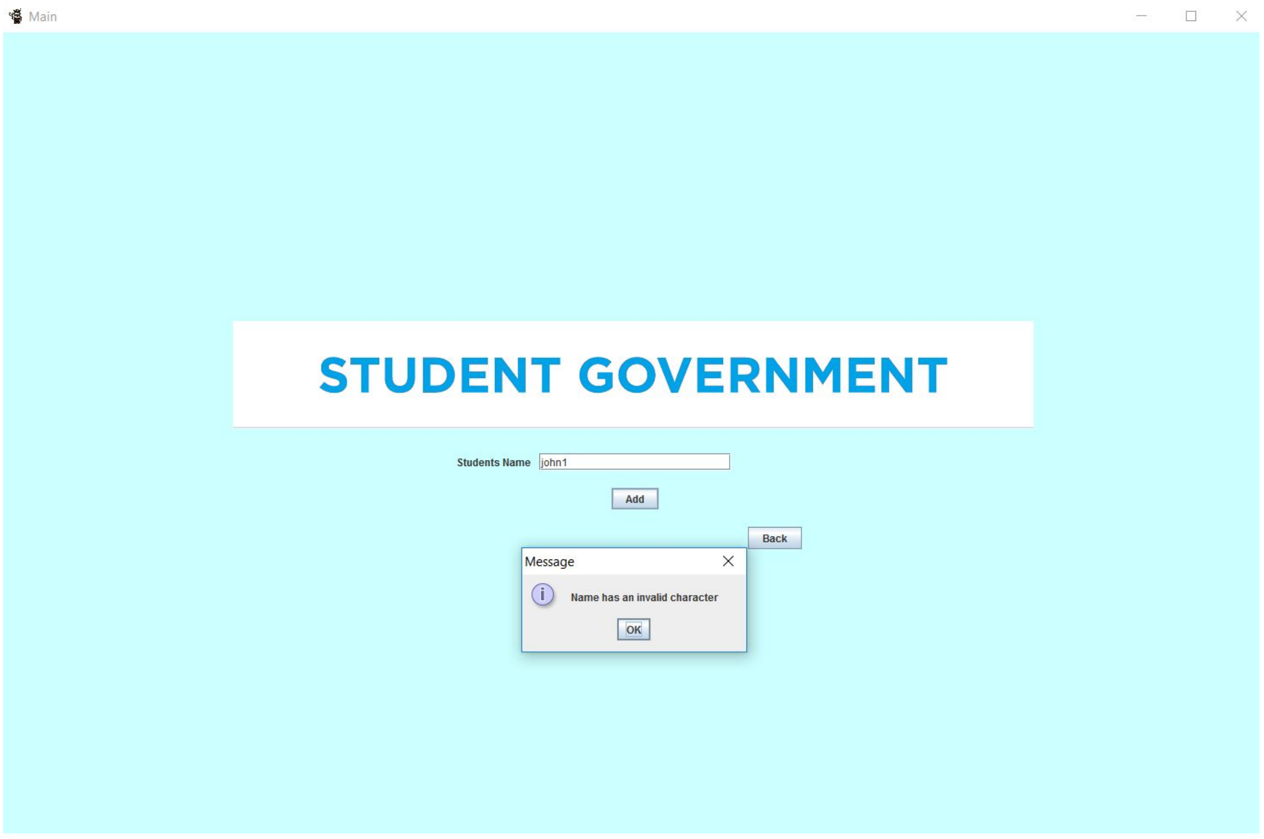
Modify Data UML



Another possible button the user could have clicked was Modify Data. There are five main areas a user could change: the names, excuses, referrals, dates or the overall attendance. Many of these areas require heavy manipulation of the SQL language and have been implemented in MySQLAccess's methods. Throughout my program, in order to increase its robustness, I took every opportunity to decrease the amount of user inputs available by using drop-down lists. However I was unable to do this with names as they are dynamic. To make my program more secure and robust when handling names, I created a whitelist of only alphabetical characters shown below.

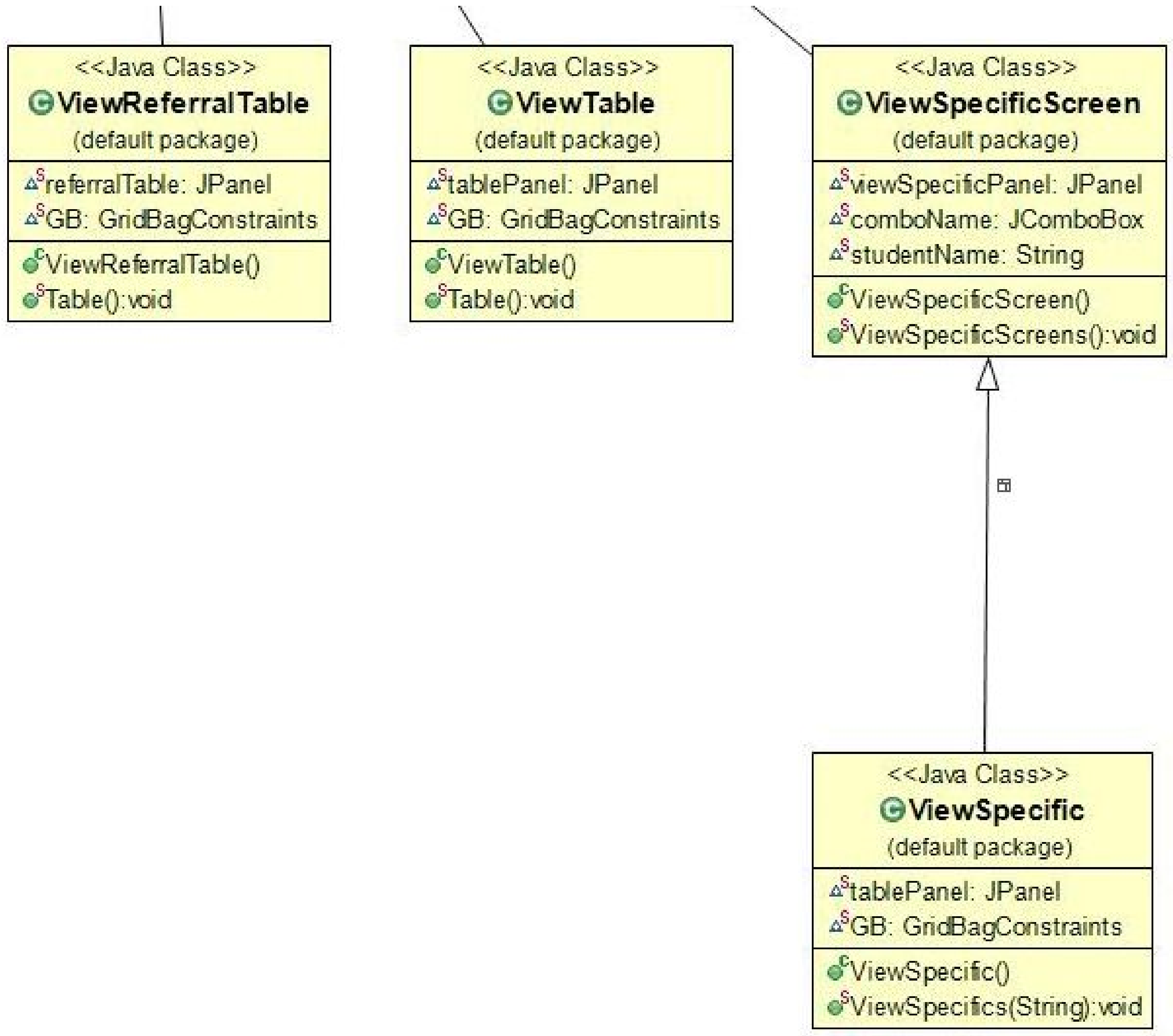
```
//Add button listener
Add.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){

    try {
        //Filter for adding names
        if(nameField.getText().matches("[A-Z a-z]+")){
            MySQLAccess.addName(nameField.getText());
            studentAddPanel.setVisible(false);
            studentWindowPanel.setVisible(true);
        }
        else{
            JOptionPane.showMessageDialog(frame, "Name has an invalid character");
        }
    } catch (ClassNotFoundException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    } catch (SQLException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    }

    }
});
```

If any character entered is not a letter, an error will show up and the user will not be able to continue as shown below. I chose a whitelist instead of a blacklist, where only certain characters aren't allowed as a blacklist is less secure as you could miss characters. Furthermore I know names can only contain alphanumerical characters.

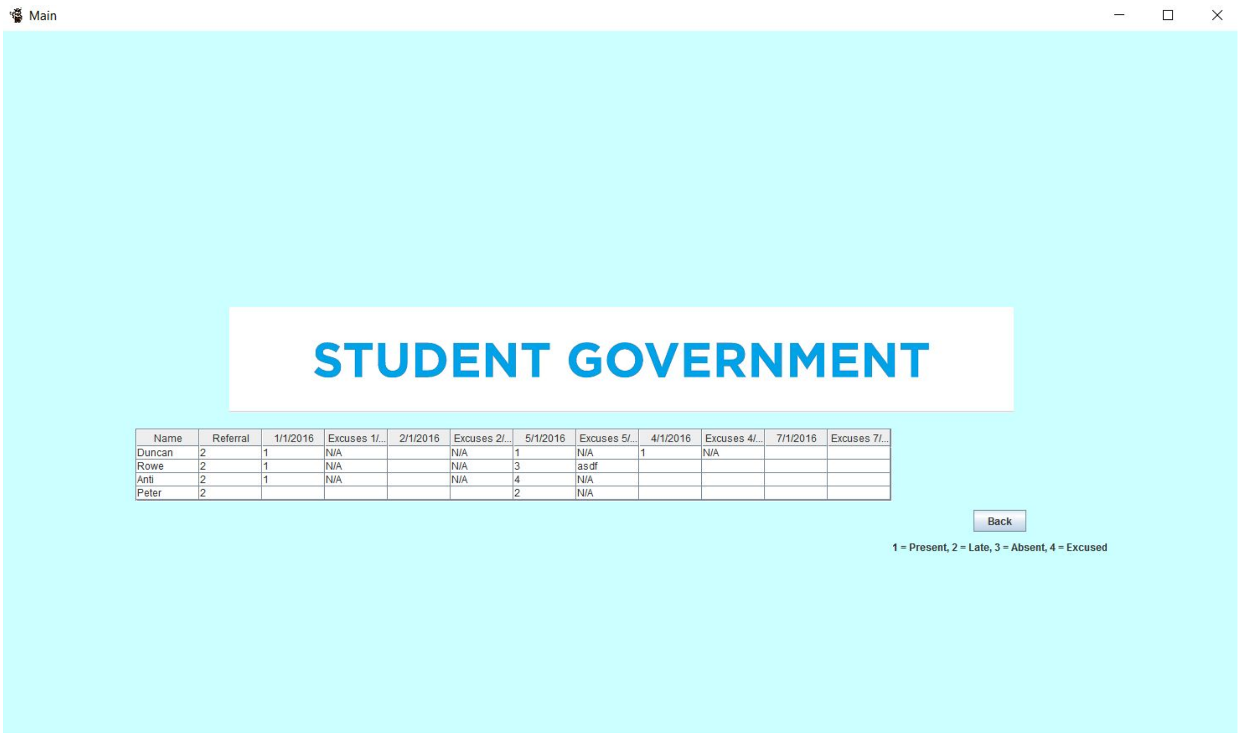


Name has an invalid character

Viewer UML



Lastly we have the third option the user could choose, viewing the data. As mentioned before, in order to view the MySQL table, the Swing component Jtable is used as shown below.



STUDENT GOVERNMENT

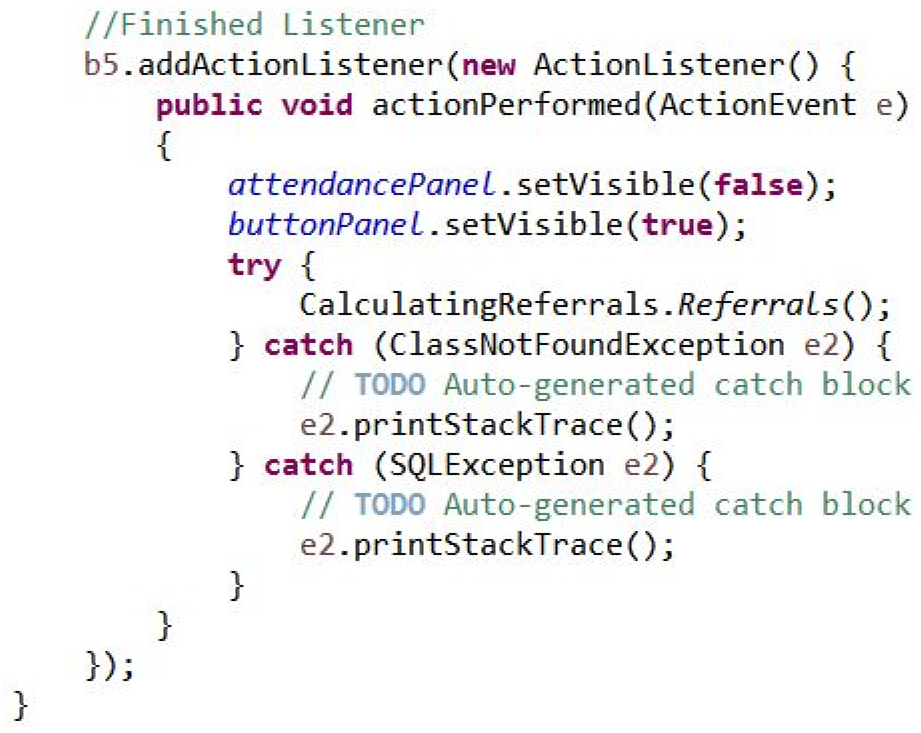
Name	Referral	1/1/2016	Excuses 1/...	2/1/2016	Excuses 2/...	5/1/2016	Excuses 5/...	4/1/2016	Excuses 4/...	7/1/2016	Excuses 7/...
Duncan	2	1	N/A		N/A	1	N/A	1	N/A		
Rowe	2	1	N/A		N/A	3	asdf				
Anti	2	1	N/A		N/A	4	N/A				
Peter	2					2	N/A				

1 = Present, 2 = Late, 3 = Absent, 4 = Excused

```
MyResult lists = MySQLAccess.theTable();

//Adds information to ArrayLists
ArrayList<String> columnNames = lists.getFirst();
ArrayList<Object> data = lists.getSecond();

JLabel legend = new JLabel("1 = Present, 2 = Late, 3 = Absent, 4 = Excused");
JButton b5 = new JButton("Back");

Vector columnNamesVector = new Vector();
Vector dataVector = new Vector();



//Adds data from ArrayLists into Vectors
for (int i = 0; i < data.size(); i++)
{
    ArrayList subArray = (ArrayList)data.get(i);
    Vector subVector = new Vector();
    for (int j = 0; j < subArray.size(); j++)
    {
        subVector.add(subArray.get(j));
    }
    dataVector.add(subVector);
}

for (int k = 0; k < columnNames.size(); k++ ){
    columnNamesVector.add(columnNames.get(k));
}

//Create table with database data
JTable table = new JTable(dataVector,columnNamesVector);

//Adds table to a scrollPane for additional features
JScrollPane scrollPane = new JScrollPane(tablePanel);
table.setFillsViewportHeight(true);
table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
table.setPreferredScrollableViewportSize(table.getPreferredSize());
```

As seen by the code above, the two arraylists are converted into vectors as Jtable requires vectors as its inputs.To do this, I simply ran through the array lists with a for loop, added the data to the vectors keeping the same order and then created the Jtable.

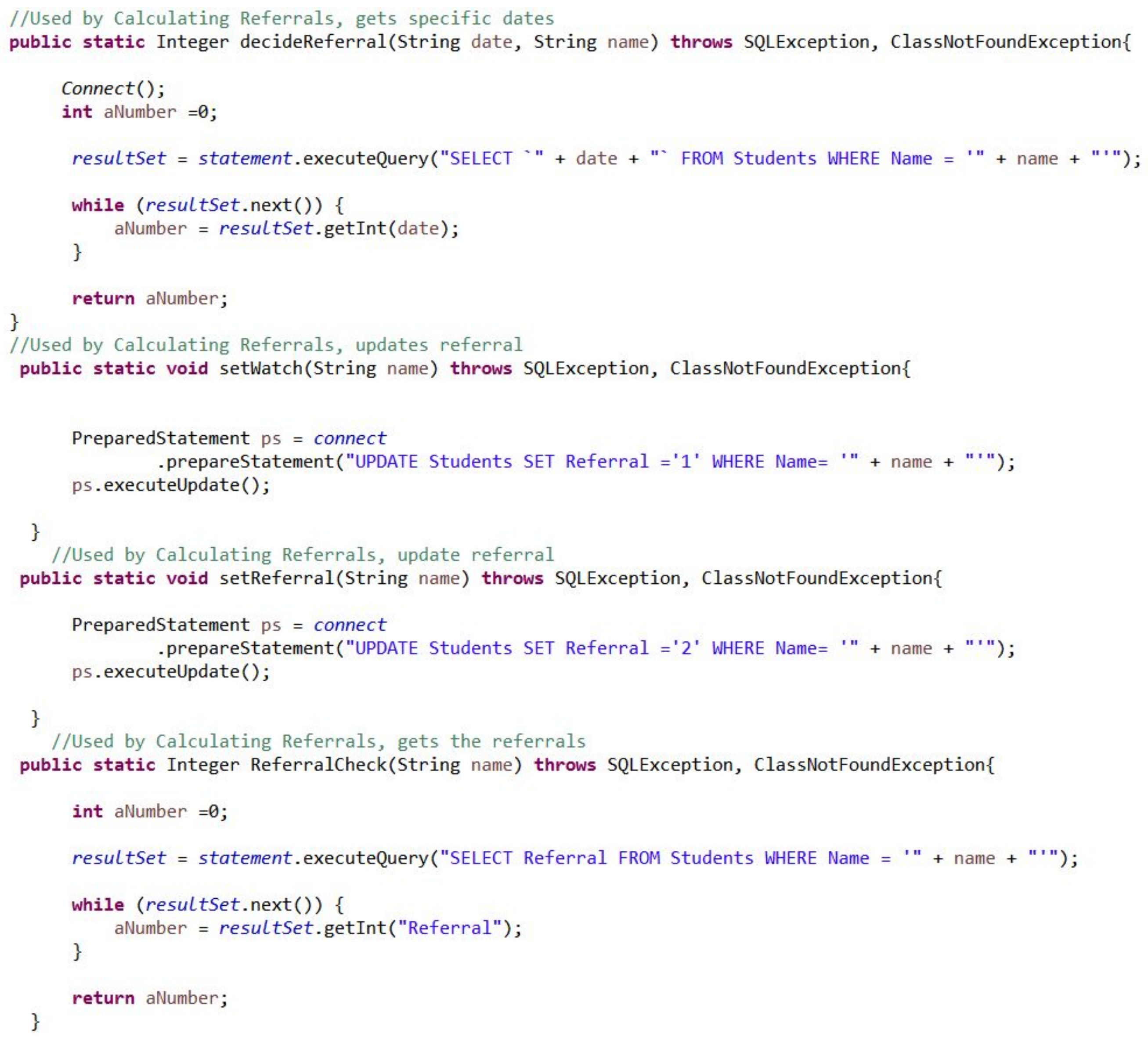
One of my most challenging aspects of the table as ensuring that my referrals were correct and that people would not be penalized twice. In order to combat this, I decided to run CalculatingReferrals only when the user finishes taking new attendance.



```
        //Finished Listener
        b5.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                attendancePanel.setVisible(false);
                buttonPanel.setVisible(true);
                try {
                    CalculatingReferrals.Referrals();
                } catch (ClassNotFoundException e2) {
                    // TODO Auto-generated catch block
                    e2.printStackTrace();
                } catch (SQLException e2) {
                    // TODO Auto-generated catch block
                    e2.printStackTrace();
                }
            }
        });
    }
```

This makes the most logical sense as Mrs.Conroy would only be updating referrals when she enters in new data. As for my algorithm calculating referrals, I ran across my next challenge. If, at the beginning, there are only three student government meetings and someone misses two, they will be on a watch list as they missed one and then be elevated to a referral list as they did not attend twice in a row. To ensure that this does not occur, I used an if statement to first checked if they had missed more than 70% of the meetings, if they did not then I checked if they had missed the last two meetings as to not double penalize them. The code below uses methods from the MySQLAccess to do this which will also be showed below.

```
//Runs through each name and then each date
for(String n:name){
    int count = 0;

    for(String d:date){
        attendance.add(MySQLAccess.decideReferral(d,n));
    }
        for(int x = 0; x < attendance.size();x++){
            if(attendance.get(x) == 1){
                count++;
            }
        }
        //Checks if attendance is 70% otherwise it checks the last two meetings
        if(((double) (count / (attendance.size()))) <= .7){

            int check = MySQLAccess.ReferralCheck(n);

            if(check == 1){
                MySQLAccess.setReferral(n);
                break;
            }
            if(check == 0){
                MySQLAccess.setWatch(n);
                break;
            }
        }
        else{
            if(attendance.size() >= 2){

                int first = attendance.get(attendance.size()-1);
                int second = attendance.get(attendance.size()-2);

                if((first != 1) && (second != 1)){

                    int check = MySQLAccess.ReferralCheck(n);

                    if(check == 1){
                        MySQLAccess.setReferral(n);
                        break;
                    }
                    if(check == 0){
                        MySQLAccess.setWatch(n);
                        break;
                    }
                }
```



```java
//Used by Calculating Referrals, gets specific dates
public static Integer decideReferral(String date, String name) throws SQLException, ClassNotFoundException{

     Connect();
     int aNumber =0;

      resultSet = statement.executeQuery("SELECT `" + date + "` FROM Students WHERE Name = '" + name + "'");

      while (resultSet.next()) {
          aNumber = resultSet.getInt(date);
      }

      return aNumber;
}
//Used by Calculating Referrals, updates referral
 public static void setWatch(String name) throws SQLException, ClassNotFoundException{


      PreparedStatement ps = connect
              .prepareStatement("UPDATE Students SET Referral ='1' WHERE Name= '" + name + "'");
      ps.executeUpdate();

  }
    //Used by Calculating Referrals, update referral
 public static void setReferral(String name) throws SQLException, ClassNotFoundException{

      PreparedStatement ps = connect
              .prepareStatement("UPDATE Students SET Referral ='2' WHERE Name= '" + name + "'");
      ps.executeUpdate();

  }
    //Used by Calculating Referrals, gets the referrals
 public static Integer ReferralCheck(String name) throws SQLException, ClassNotFoundException{

      int aNumber =0;

      resultSet = statement.executeQuery("SELECT Referral FROM Students WHERE Name = '" + name + "'");

      while (resultSet.next()) {
          aNumber = resultSet.getInt("Referral");
      }

      return aNumber;
  }
```

Throughout my program, I took advantage of the OOP properties of inheritance and encapsulation, however I found no need to use polymorphism as I did not need to use duplicate method names. In order to make my program useable for my client, I created a batch script that ran the jar file so that she could run the application from her desktop easily.

Word Count : 1187