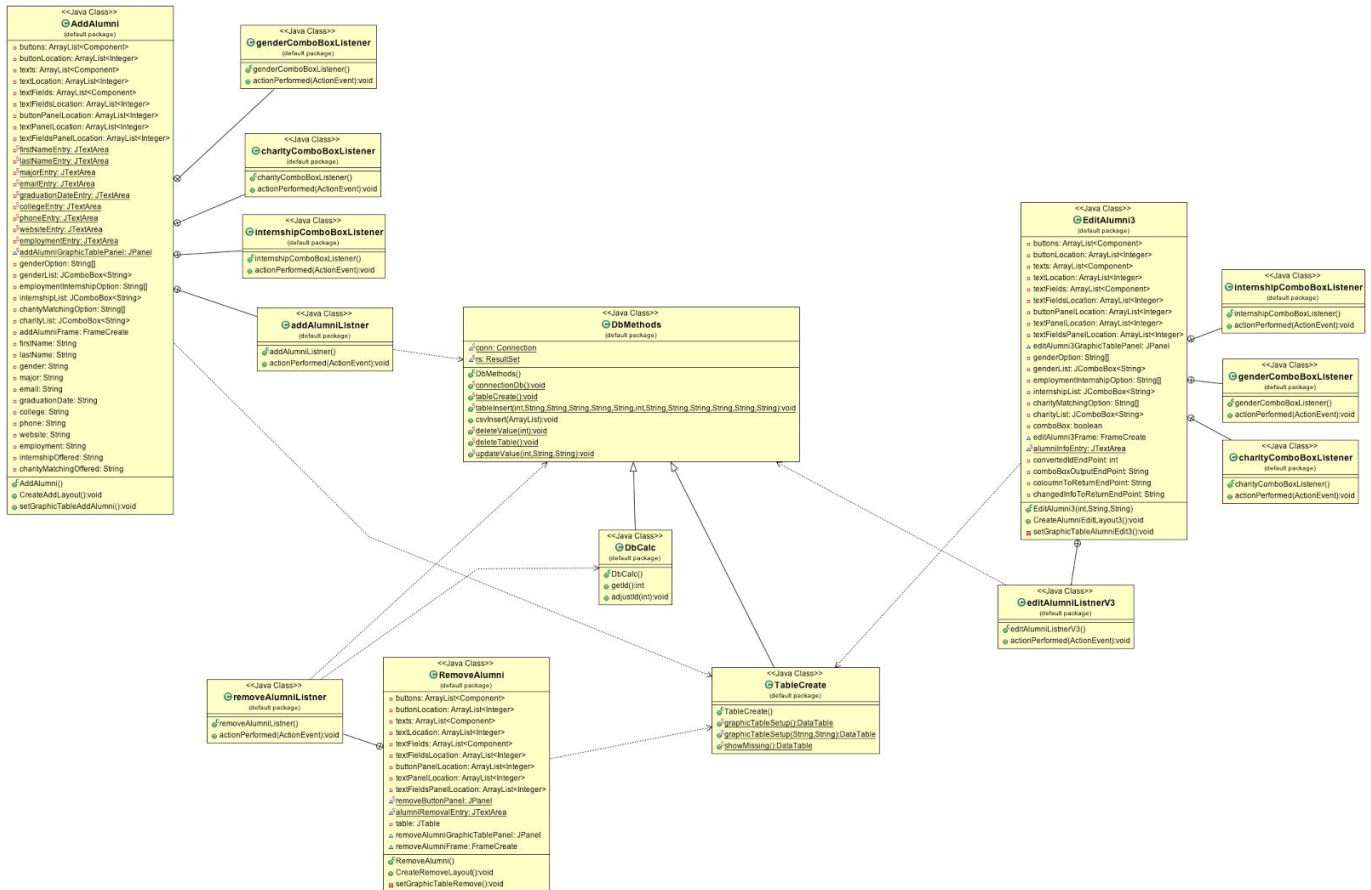


Criterion C: Development

Developing the product:

Database Relationships

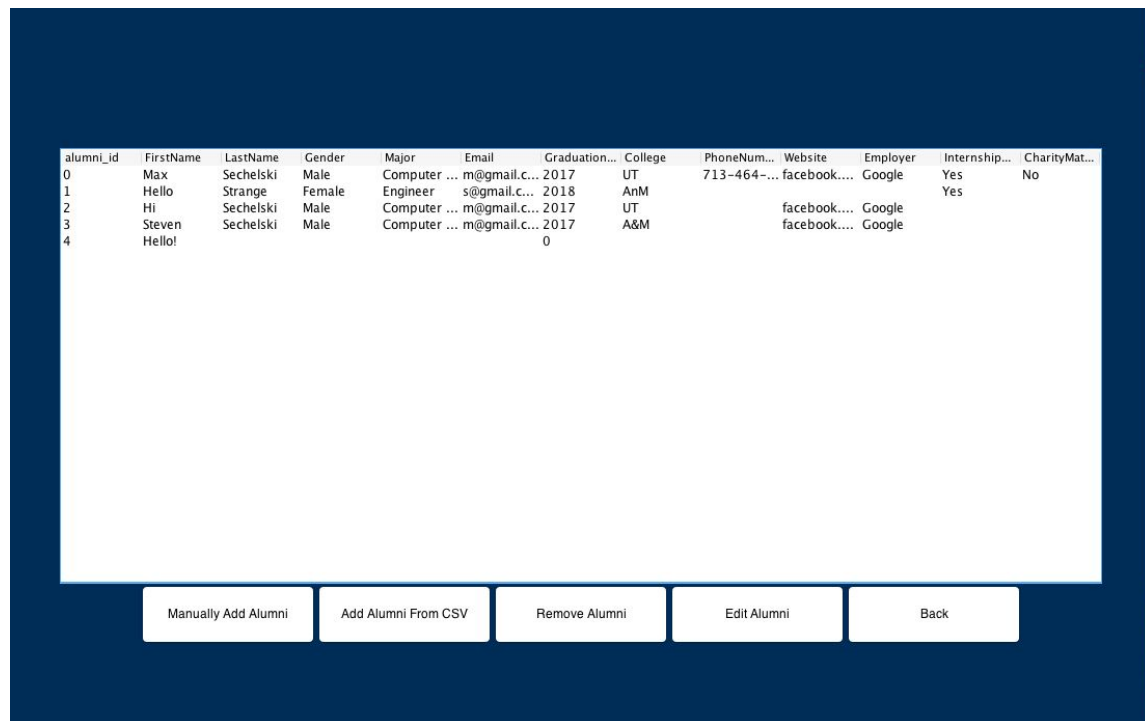
The entire application revolves around the access and manipulation of a database. The UML below shows the connections between classes that communicate with the database.



A majority of the classes above are related the the edit page of the database, as methods such as adding, removing, and editing entries need to communicate with the database to make the changes that they have been assigned.

TableCreate

However, the TableCreate class is used by any page that creates a graphical table to show the user what is stored in the database. An example of this is shown in the picture below.



alumni_id	FirstName	LastName	Gender	Major	Email	Graduation...	College	PhoneNum...	Website	Employer	Internship...	CharityMat...
0	Max	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT	713-464-...	facebook....	Google	Yes	No
1	Hello	Strange	Female	Engineer	s@gmail.c...	2018	AnM				Yes	
2	Hi	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT		facebook....	Google		
3	Steven	Sechelski	Male	Computer ...	m@gmail.c...	2017	A&M		facebook....	Google		
4	Hello!					0						

Manually Add Alumni Add Alumni From CSV Remove Alumni Edit Alumni Back

TableCreate packages data from the database into two ArrayLists, one for column names and one for data.

```
9 public class TableCreate extends DbMethods
10 {
11     //The following method packages data of all alumni into ArrayLists and returns an object of DataTable that stores those ArrayLists
12     public static DataTable graphicTableSetup() throws SQLException, ClassNotFoundException
13     {
14         //I Should be here
15         ArrayList<String> columnNames = new ArrayList<String>();
16         ArrayList<Object> data = new ArrayList<Object>();
17         Statement s2 = conn.createStatement();
18
19         String sql = "SELECT * FROM alumni";
20         ResultSet rs = s2.executeQuery(sql);
21
22         //Finding the amount of columns
23         ResultSetMetaData md = rs.getMetaData();
24         int columns = md.getColumnCount();
25
26         //Populating the ArrayList for column names
27         for (int i = 1; i <= columns; i++)
28         {
29             columnNames.add( md.getColumnName(i) );
30         }
31
32         //Populating the ArrayList for data
33         while (rs.next())
34         {
35             ArrayList<Object> rowCrawler = new ArrayList<Object>(columns);
36
37             for (int i = 1; i <= columns; i++)
38             {
39                 rowCrawler.add( rs.getObject(i) );
40             }
41
42             data.add(rowCrawler);
43         }
44
45         //returning an object of type DataTable that stores the ArrayLists created above
46         return new DataTable(columnNames, data);
47     }
48 }
```

graphicTableSetup() retrieves all the entries from the database and inserts them into a ResultSet. A ResultSet is a table that represents the database data that the SQL command returns. The code then gets the metadata of the ResultSet which stores things such as the number of columns and column names. The code uses this information to create a loop that runs for as many columns as the ResultSet contains, and for each loop the name of a column is added to one of the two arraylists that will be returned by TableCreate.

This SQL command can be modified to return specific data, which the filter page of the application uses to query specific results from the database and update the table to show the filtered information. An example of a modified SQL command for this purpose is seen in the graphicTableSetup(String column, String filterChoice) method.

```

49 //The following method packages data of alumni with a certain column
50 public static DataTable graphicTableSetup(String column, String filterChoice) throws SQLException, ClassNotFoundException
51 {
52     ArrayList<String> columnNames = new ArrayList<String>();
53     ArrayList<Object> data = new ArrayList<Object>();
54     Statement s2 = conn.createStatement();
55     String sql;
56
57     if(column == "GraduationDate")
58     {
59         int filterChoiceFinal = Integer.parseInt(filterChoice);
60         sql = "SELECT * FROM alumni WHERE " + column + " = '" + filterChoiceFinal + "'";
61     }
62     else
63     {
64         sql = "SELECT * FROM alumni WHERE " + column + " = '" + filterChoice + "'";
65     }
66
67     ResultSet rs = s2.executeQuery(sql);
68
69     ResultSetMetaData md = rs.getMetaData();
70     int columns = md.getColumnCount();
71
72
73     for (int i = 1; i <= columns; i++)
74     {
75         columnNames.add( md.getColumnName(i) );
76     }
77
78     while (rs.next())
79     {
80         ArrayList<Object> rowCrawler = new ArrayList<Object>(columns);
81
82         for (int i = 1; i <= columns; i++)
83         {
84             rowCrawler.add( rs.getObject(i) );
85         }
86
87         data.add( rowCrawler );
88     }
89
90     //System.out.println(columnNames);
91     //System.out.println(data);
92     return new DataTable(columnNames, data);
93 }

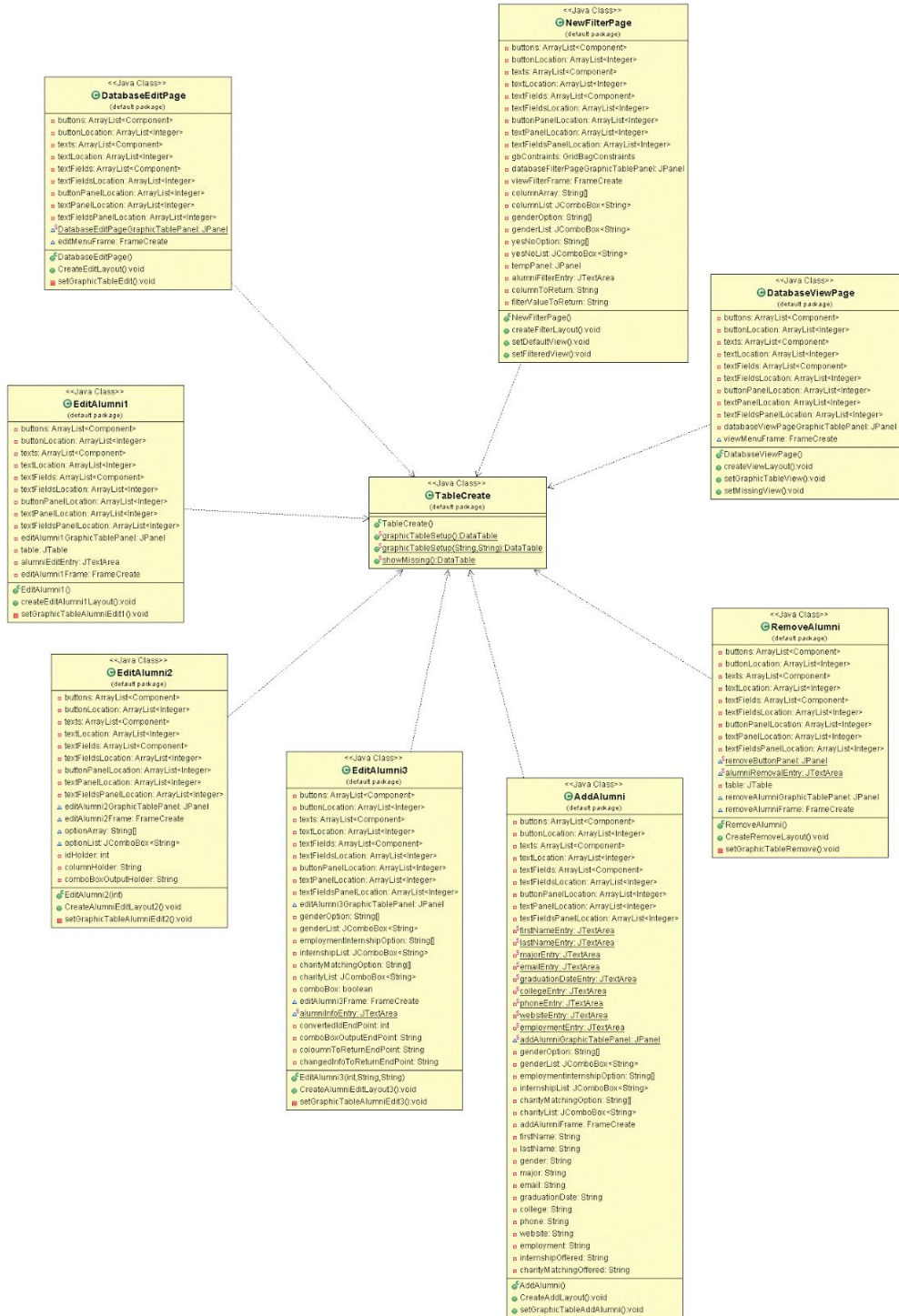
```

When looking at lines 60-64, the modified SQL code can be seen. The method takes the arguments from the parameters and then inserts it into the SQL command, allowing for customized access to the database that other methods can use to serve filtered information to the user through the JTable. This use of SQL shows the advantage of using a database, as it allows for specialized retrieval of data that is seen in this method. The application relies on this ability of SQL to filter data so that the user can sort data to their specification, as required by the success criteria.

This example also exemplifies the use of polymorphism in my code, as I use an overloaded method. This allows me to use a common name for two different functions, and determine the action by what arguments are added when calling the method. Doing this allows for greater flexibility in my program, as a method can handle varied input arguments without resulting in an error.

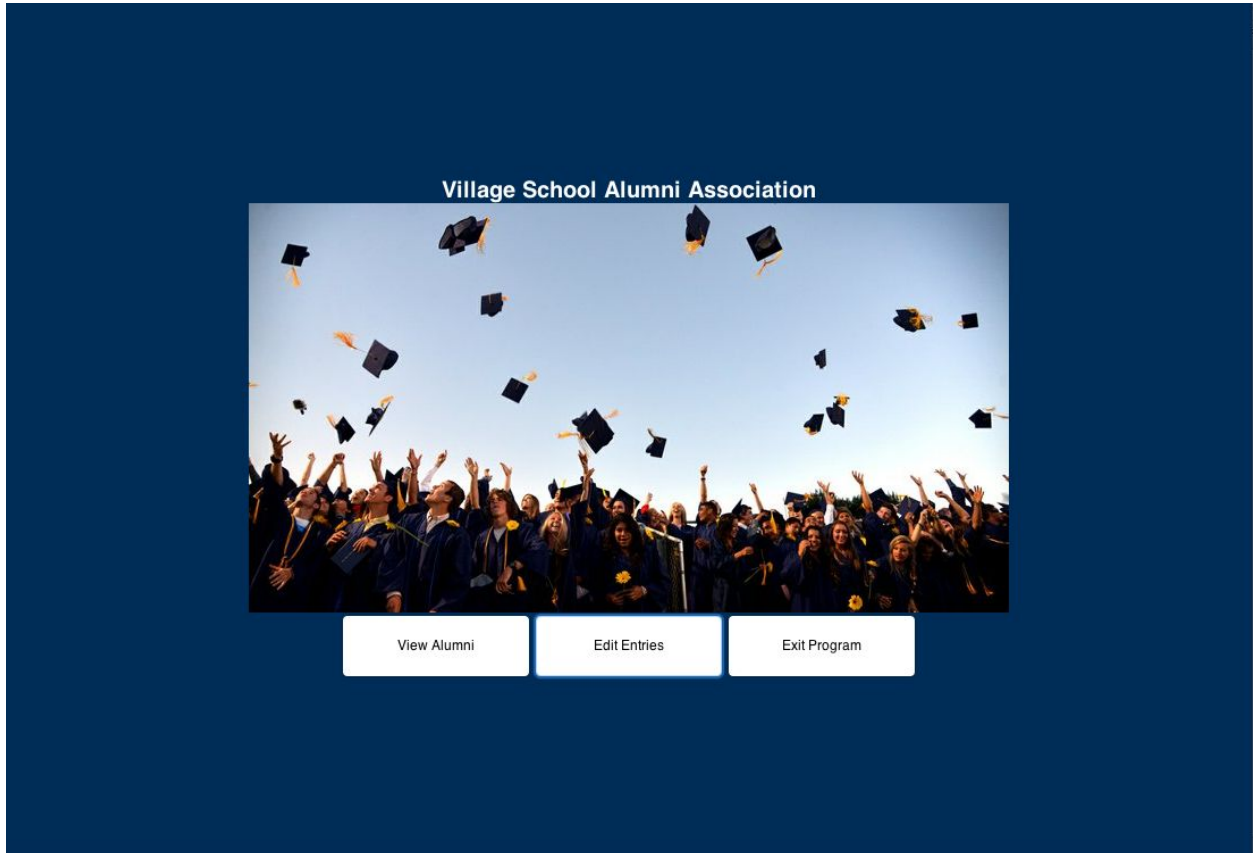
A ResultSet has a cursor, and for both methods, the while loop runs for as long as that cursor has a next row. For each row, a for loop is run for as many columns as there are in the ResultSet, adding each value until it gets to the end of the row. The for loop then terminates and the while loop continues on. This populates the second of the two arraylists that will be returned by TableCreate.

TableCreate is called often by pages that either allow the user to edit or view the database, as it is required to get the information into a format that is insertable into a JTable. The following UML shows the classes that call on TableCreate.



Application Structure

The application opens on the main page:



The main page allows for the user to accomplish the two main functions of the application: Query the database for information from the View menu, and edit the database from the Edit menu.

Looking at the Edit menu:

alumni_id	FirstName	LastName	Gender	Major	Email	Graduation...	College	PhoneNum...	Website	Employer	Internship...	CharityMat...
0	Max	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT	713-464-...	facebook....	Google	Yes	No
1	Hello	Strange	Female	Engineer	s@gmail.c...	2018	AnM				Yes	
2	Hi	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT		facebook....	Google		
3	Steven	Sechelski	Male	Computer ...	m@gmail.c...	2017	A&M		facebook....	Google		
4	Hello!					0						

Manually Add AlumniAdd Alumni From CSVRemove AlumniEdit AlumniBack

The user is again presented with options.

Say the user wants to manually add an alumni entry, when they click on the corresponding button the following menu opens:

alumni_id	FirstName	LastName	Gender	Major	Email	Graduation...	College	PhoneNum...	Website	Employer	Internship...	CharityMat...
0	Max	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT	713-464-...	facebook....	Google	Yes	No
1	Hello	Strange	Female	Engineer	s@gmail.c...	2018	AnM				Yes	
2	Hi	Sechelski	Male	Computer ...	m@gmail.c...	2017	UT		facebook....	Google		
3	Steven	Sechelski	Male	Computer ...	m@gmail.c...	2017	A&M		facebook....	Google		
4	Hello!					0						

BackAdd Entered Alumni

First Name:

Last Name:

Gender:

▼

Major:

Email:

Graduation Date:

College:

Phone Number:

Website:

Employment:

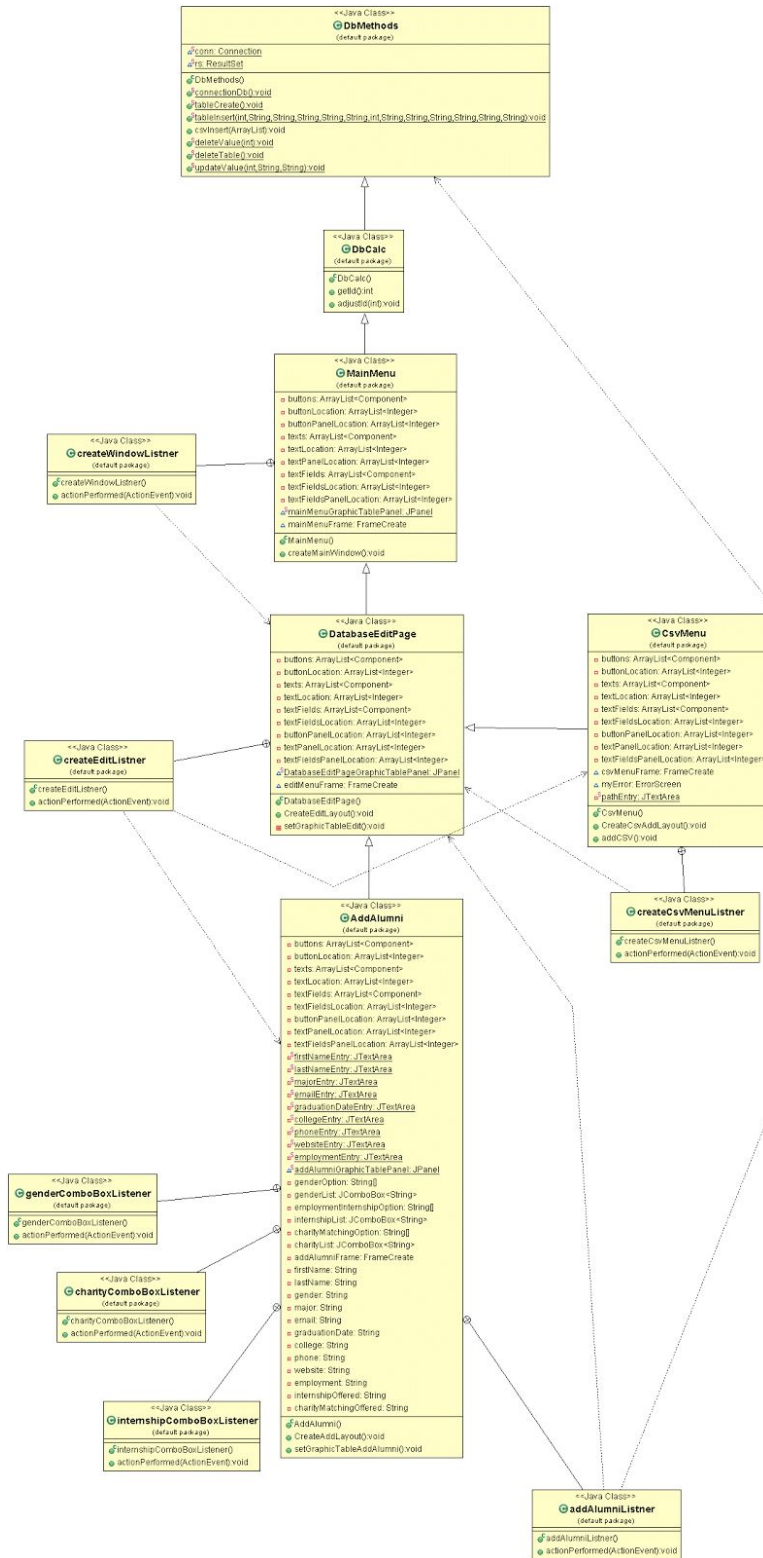
Does their employer offer student internships?:

▼

Does their employer offer charity matching?:

▼

The user would then enter their entries and add the alumni to the database. This process to get from the main menu to eventual end action is uniform across the application. This process also provides an example of how the code is structured, shown below:



This exemplifies the inheritance structure of the application. Starting with the DbMethod Class and cascading downwards until the end function is reached. This allows for each class to be linked, and reduces the amount of code that needs to be written. This structure is mimic throughout the application for all end functions. The use of inheritance allows for more code reuse and as a result, faster development as I no longer need to implement new methods to accomplish the same thing as methods in other classes. Inheritance also reduces errors inside the application, for as long as the original method works correctly and is checked for errors, any subsequent use in classes that inherit the method will be free of error.

Word Count: 793