Assign6Tests  
Jennifer Swofford

**Implementation:**

1. I started with a **searchRow** search that returns the index of the row located (or -1 if it’s not located). This is defined in the CityTable and in the StadiumTable. I made searchRow an abstract method in the AbstractTable class. I may refactor this to be completely to be a concrete method in the AbstractTable class depending on if time allows.
2. From there I decided that I needed to **change findRow** so that it leverages the searchRow and displays the row contents to the user. That made me realize what I would rather do is put a **concrete displayRow method in AbstratctTable** that takes an int (passed from SearchRow) and displays the contents of the row in a user-friendly format.
3. At this point, I removed the bulk of what I had in remove row and leveraged the newly created searchRow. Interestingly, I did find a bug in my program at this point. It turns out that when I removed a row and then saved it to a file, I wasn’t updating row count correctly. This is where I ended up removing the rowCount and started using fullTable.size().
4. The **bubblesort** was next – this took quite a bit of time in the CityTable. I encountered hiccups around needing to do some casting, and also had been running into issues where I wasn’t hitting the compareTo in my CityRow class. It kept using compareTo from the string class. It turned out I wasn’t passing the object but was passing compRow1.getCityId(). Once I corrected this, I didn’t have any issues.

I really wasn’t sure what to do with the sort specifically. I ended up making it so my program auto-sorts by ID whenever a row is added. This would mean any time a file is loaded or an individual row is added, the table will sort by CityID

1. From the sort, I jumped straight to the **join**. I found this part to be the most daunting, so I spent a lot of time here. I also wasn’t sure where to execute it in the main program. I ended up creating a new menu option for the merging of the tables that outputs a JoinedTables.txt.

I really had a tough time with this one around the types and casting I had to do. I ended up reaching out to tutor for assistance here because I couldn’t grasp how to make the for each loops work.

1. I then realized I missed the remove duplicate rows. I identified duplicates based solely on CityID. I am making the assumption that CityID is primary key on both tables, though one could argue it wouldn’t necessarily be the primary key on the stadium table. Regardless, for the sake of time I went this direction. The remove duplicates will advise the user which duplicate was found that’s being removed and will also confirm the # of rows removed after it completes. This confirmation message would also work well for if there are 0 duplicates found and there are no rows removed.

**Testing:**

Test cases that I performed were the same as the previous iterations of the program to make sure that it still functions with all my existing changes (regression testing).

For the new **sort** functionality, I tested loading a file and saving it to see the output in the new sort order. Then I tested adding induvial rows in various orders (ID 400, ID 7, ID 450, ID 35).

For testing the **search** I basically re-ran my previous test cases around findRow, but made sure that my new implementation had the same results (ex. Find a row loaded from a table. Find a row after adding a row, remove a row and attempt to find it

For testing the **join**, I tested scenarios where a stadium table had no data, a city table had no data, and where they both had data. Additionally, I handled a scenario if there were no matches found across the two tables.