Contents

[Assignment Description 2](#_Toc116817168)

[AbstractTable Class 2](#_Toc116817169)

[AbstractRow Class 2](#_Toc116817170)

[Test Cases 2](#_Toc116817171)

[TestCase1: City Table Walkthrough 2](#_Toc116817172)

[Description: 2](#_Toc116817173)

[Steps: 2](#_Toc116817174)

[Expected Outcomes: 3](#_Toc116817175)

[Actual Outcome: 3](#_Toc116817176)

[TestCase2: Stadium Table Walkthrough 3](#_Toc116817177)

[Description: 3](#_Toc116817178)

[Steps: 3](#_Toc116817179)

[Expected Outcomes: 3](#_Toc116817180)

[Actual Outcomes: 3](#_Toc116817181)

[TestCase3: City/Stadium Tables in the same run 4](#_Toc116817182)

[Description: 4](#_Toc116817183)

[Steps: 4](#_Toc116817184)

[Expected Outcome: 4](#_Toc116817185)

[Actual Outcome: 4](#_Toc116817186)

[TestCase4: Load Empty Files 4](#_Toc116817187)

[Description: 4](#_Toc116817188)

[Steps: 5](#_Toc116817189)

[Expected Outcomes: 5](#_Toc116817190)

[Actual Outcomes: 5](#_Toc116817191)

[TestCase5: Add and locate cities composed of multiple strings, regardless of case 5](#_Toc116817192)

[Description: 5](#_Toc116817193)

[Steps: 5](#_Toc116817194)

[Expected outcomes: 5](#_Toc116817195)

[Actual outcomes: 5](#_Toc116817196)

[TestCase6: Load a city table of 100 rows 5](#_Toc116817197)

[Description: 5](#_Toc116817198)

[Steps: 6](#_Toc116817199)

[Expected Outcome: 6](#_Toc116817200)

[Actual Outcome: 6](#_Toc116817201)

[TestCase7: Load a stadium table of 100 rows 6](#_Toc116817202)

[Description: 6](#_Toc116817203)

[Steps: 6](#_Toc116817204)

[Expected Outcome: 6](#_Toc116817205)

[Actual Outcome: 6](#_Toc116817206)

# Assignment Description

## AbstractTable Class

In the AbstractTable class, once I got my CityTable class & Stadium Table class, I looked at the methods that existed in both City & Stadium *and* took the same parameters. I then went back to the AbstractTable class and added them as abstract methods.

Therefore, in addition to the requirement of loadTableFromFile & saveTable, I also added findRow and removeRow as abstract classes. addRow I left out of the AbstractTable class as the two tables have differing row lengths.

I also identified that my String splitter was being used by both Stadium & City classes, so I added that to the AbstractTable class as a concrete method.

## AbstractRow Class

In the AbstractRow class, in addition to the equals method, I also added a toString method as an abstract method to be implemented in the subclasses.

# Test Cases

## TestCase1: City Table Walkthrough

### Description:

This test case will validate that a City Table can be loaded, rows can be added, the new row can be found, old rows can be found, a row can be deleted, and the table can be saved to an output file.

### Steps:

1. Load cities.txt - Option 1
2. Add a row of city data (Sparta, 700, 1.2) – Option 3
3. Locate the new row – Option 5
4. Locate a pre-existing row – Option 5
5. Delete the first row – Option 4
6. Save the table as citiesTest1.txt – Option 2
7. Quit (Option 11)

### Expected Outcomes:

1. File will load without errors
2. Row will be added without errors
3. Will locate the new row
4. Will locate the existing row
5. Row will get deleted and the remaining rows will get re-indexed
6. File will save to the project folder with the new row at the end of the list and the first row removed.
7. Program will close.

### Actual Outcome:

1-7: As expected

## TestCase2: Stadium Table Walkthrough

### Description:

This test case will validate that a Stadium Table can be loaded, rows can be added, the new row can be found, old rows can be found, a row can be deleted, and the table can be saved to an output file.

### Steps:

1. Load stadia.txt - Option 6
2. Add a row of city data (Jen's Stadium, 72, Jen’s Team, 7000) – Option 8
3. Locate the new row – Option 10
4. Locate a pre-existing row – Option 10
5. Delete the first row (Mercedes-Benz Stadium)– Option 9
6. Save the table as stadiaTest2.txt – Option 7
7. Quit (Option 11)

### Expected Outcomes:

1. File will load without errors
2. Row will be added without errors
3. Will locate the new row
4. Will locate the existing row
5. Row will get deleted and the remaining rows will get re-indexed
6. File will save to the project folder with the new row at the end of the list and the first row removed.
7. Program will close.

### Actual Outcomes:

**First Run**: Wasn’t able to find the stadium; Turned out the split in findRow was looking at the wrong element. Changed from split[1] to split[0].

**Second run**: As expected.

## TestCase3: City/Stadium Tables in the same run

### Description:

This test case will hit all the commands in the program in the same run.

### Steps:

1. Load cities.txt - Option 1
2. Load stadia.txt - Option 6
3. Add a row of city data (Holmen, 50, .05) – Option 3
4. Add a row of stadium data (Another Stadium, 50, Another Team, 50) – Option 8
5. Locate the new city row – Option 5
6. Locate the new stadium row – Option 10
7. Locate a pre-existing city row – Option 5
8. Locate a pre-existing stadium row – Option 10
9. Delete the first city row – Option 4
10. Save the table as citiesTest3.txt – Option 2
11. Delete the first row (Mercedes-Benz Stadium)– Option 9
12. Save the table as stadiaTest3.txt – Option 7
13. Quit (option 11)

### Expected Outcome:

1. City File will load without errors
2. Stadium File will load without
3. City Row will be added without errors
4. Stadium Row will be added without errors
5. Will locate the new city row
6. Will locate the new stadium row
7. Will locate the existing city row
8. Will locate the existing stadium row
9. City Row will get deleted and the remaining rows will get re-indexed
10. File will save to the project folder with the new row at the end of the list and the first row removed.
11. Stadium row will be removed and the remaining rows will get re-indexed
12. File will save to the project folder with the new row at the end of the list and the first row removed.
13. Program will close.

### Actual Outcome:

1-13: As expected

## TestCase4: Load Empty Files

### Description:

This test case will validate that users will get a warning message if they try to load an empty city table or empty stadium table.

### Steps:

1. Load empty.txt as a city table – Option 1
2. Load empty.txt as a stadium table – Option 6
3. Quit (Option 11)

### Expected Outcomes:

1. User will get an error message
2. User will get an error message
3. Program will close

### Actual Outcomes:

1-3: As expected

## TestCase5: Add and locate cities composed of multiple strings, regardless of case

### Description:

This test will validate that city names with multiple worders (ex. La Crosse, West Salem, etc.) are added to the table correctly, can be found, and saved to a file without losing formatting.

### Steps:

1. Add a row of city data (La Crosse, 700, .05) – Option 3
2. Add a row of city data (West Salem, 800, .005) – Option 3
3. Add a row of city data (New York City, 1, 1.2) – Option 3
4. Locate La Crosse – Option 5
5. Locate west salem – Option 5 (all lower case)
6. Locate new York CITY– Option 5 (mixed case)
7. Save the table as citiesTest5.txt – Option 2

### Expected outcomes:

1. City will get added without error
2. City will get added without error
3. City will get added without error
4. City will be found
5. City will be found
6. City will be found
7. Table will get saved and the 3 cities will be in the file in the correct format.

### Actual outcomes:

**First Run:** Header was persisted as null. Updated Save table to check for header value

**Second Run**: As expected & header present

## TestCase6: Load a city table of 100 rows

### Description:

This test case will validate that we can load a full city table of 100 rows and manipulate it and save it. A test file call 100City.txt has been staged to work with

### Steps:

1. Load 100City.txt – Option 1
2. Add a row of city data (Jen, Jen, Jen) – Option 3
3. Remove a row of city data (Seattle92, 92, 91.77) – Option 4
4. Add a row of city data (Jen, Jen, Jen) – Option 3
5. Save the table as citiesTest6.txt – Option 2
6. Quit (Option 11)

### Expected Outcome:

1. File will load without issues/errors
2. An error message will display that the table is full
3. The row will get removed and the remaining rows re-indexed
4. The new row will get added
5. The final file will get saved to the project folder with the deleted row missing and the row added.

### Actual Outcome:

1-5: As expected

## TestCase7: Load a stadium table of 100 rows

### Description:

This test case will validate that we can load a full stadium table of 100 rows and manipulate it and save it. A test file call 100Stadia.txt has been staged to work with

### Steps:

1. Load 100Stadia.txt – Option 6
2. Add a row of stadium data (Jen, Jen, Jen, Jen) – Option 8
3. Remove a row of stadium data (BBVA Stadium98)– Option 9
4. Add a row of stadium data (Jen, Jen, Jen, Jen) – Option 8
5. Save the table stadiaTest7.txt – Option 7
6. Quit (Option 11)

### Expected Outcome:

1. File will load without issues/errors
2. An error message will display that the table is full
3. The row will get removed and the remaining rows re-indexed
4. The new row will get added
5. The final file will get saved to the project folder with the deleted row missing and the row added.

### Actual Outcome:

1-5: As expected