Project 6 - Bushman

I first started with the Town class. I made sure to implement Comparable<Town> and created my town name variable. I created the basic constructor to assign the given town name to my class variable. I returned the class name variable in the getName method. For the compare To method, I returned the name compared to the given object's name. I was little confused on the toString method since it basically follows the getName method. I just returned the class name variable. For the hashCode method, I returned the town name with the hashCode method. For the equals method, I first created a new Town object using their given object. I created a boolean variable to hold the true or false result. I used an if statement to compare if the town name was equal to the given objects town name. If it was, it returns true. If not, it returns false. For the road class, I first implemented the Comparable<Road>. I created a town source, town destination, town degrees and town name variables. For the basic constructor, I assigned each given variable to the correct class variable. For the next constructor, I did the same thing expect assigned the weight to 1. For the contains method, I created my boolean variable and checks if the town source was equal to the town or if the town destination was equal to the town. If it was, I returned true. If not, I returned false. I returned the town name for both the toString and getName method. I returned the town source and the town destination for each getSource and getDestination method. For the compareTo method, I returned the town name compared to the object's name. I returned the class weight for the getWeight method. For the equals method, I created a boolean variable along with the new Road object based on the given object. I checks if the town source was equal to the given Road's source, if the town source was equal to the given road's town destination, if the town destination was equal tot he road's town destination or if the town destination was equal to the road's town source. If so, I returned true. If not, I returned false. This made sure to compare both directions for the road.

Next, I worked on the Graph class and implemented the GraphInterface. I created a set of roads, a set of towns and a map for the previous vertex that will be used for the Dijstrka method. For the getEdge method, I first checked if the source vertex or designation vertex was equal to null. If so, I returned null. If not, I then used an enhanced for loop to run through each road of the roads set. Using two if statements, I checked if the current road"s designation was equal to the source vertex, if the current road's destination was equal to the destination vertex, if the current road's source was equal to the source vertex or if the current road's source was equal to the destination vertex. If so, I returned that current road. If not, I returned null. For the addEdge method, using two if statements, I checked if the source vertex was contained in the town set or if the designation vertex was contained. If not, I threw a new IllegalArgumentException. I then check if the source vertex or destination vertex was null. If so, I threw a new NullPointerException. I then created a new Road object with the given information. I then added that new road to the roads set. I used another if statement to check if the source vertex and destination vertex were not in the roads set. If not, I returned null. If so, I returned that newly added road. For the addVertex method, I created a boolean variable first and then checked if the given town was null. If so, I threw a new NullPointerException. I then checked if the town set contains the given town. If not, I added the town to the set and returned true. If not, I returned false. For the containsEdge method, I created a boolean variable and used a enhanced for loop to run through the road set. I checked if the current road in the set had an equal source vertex equal to the given source, if the current road's source is equal to the given destination vertex and vise versa. If so, I then returned true. If not, I returned false. For the containsVertex method, I created the boolean variables and used an enhanced for loop to run through the towns set. I checked if any of the towns were equal to the given town. If so, I returned true. If not, I returned false. For edgeSet and vertexSet method, I returned both the road set and towns set respectively. For the edgesOf method, I created my set of roads to hold the edges of the given town. I first checked to see if the vertex was not in the towns set. If not, I threw IllegalArgumentException. I checked if the given vertex was null

and if so, threw NullPointerException. I then used a for loop to run through the roads sets. I checked if the current road's source was equal to the vertex or if the current road's destination was equal to the vertex. If either was true, I added that current road to the set of touching edges and returned it once the for loop was over with. For the removeVertex method, I created my boolean variable and checked if the towns set didn't contain the given town. If not, I returned false. If so, I used an enhanced for loop to run through the towns set. I checked if the current town was equal to the given town. If it was, I used another for loop to run through the edges of that current town. I then removed each specified road from the roads set. After that was done, I removed the town from the towns set and returned true. For the shortestPath method, I first called the dijkstraShortestPath to populate the class Map variable. I created my array list, town and road variables. Using a while loop to check if the destination vertex doesn't equal the given source vertex, I assigned the previous town variable to the previous town of the destination vertex. I made sure to check if the previous town variable is null and if so, to return the ArrayList. I then assigned the road variable using the getEdge method with the two given town variables. I then added the information to the array list with the first town name along with via and then the road name with to and then the next town name along with the miles. I made sure to assign the next town variable to the previous town to continue the while loop. For the dijkstraShortestPath method, I first created my array list variable to store unvisited towns, a hash map to store distances, town objects and int variables. I used a for loop to run through the towns set. I added each town to the array list and initialized each distance to max in he miles hash map. I then set the previous vertex from each town to null. I then set the distance for the source vertex to 0 in the miles hash map. Using a while statement to run through the array list, I created a shortest index and then used a for loop to run through the array list. I grabbed each town and checked if the distance of the first vertex was greater than the each element of the unvisited array list. If it was, I updated the shortest index to that index. I then updated the town object to remove the smallest vertex by the shortest index in the array list. Using a enhanced for loop, I assigned the adjacent town object to the current road in the for loop. I used an if statement I then assigned the adjacent miles to be the distance of the current vertex plus the road's weight. I used a if statement to check if the adjacent town's weight (just calculated) was less than the known distance. If it was, I set the new distance to the adjacent town along with inserted the adjacent town with the previous town in the previous hash map.

For the TownGraphManager class, I created a class graph variable which will be used throughout the methods. For the addRoad method, I created a boolean variable and two new town objects to hold the given two towns. I then created a new road object with the given towns, weight and road name. Using the getEdge method, I checked if the graph continue the two towns. If it didn't, I returned false. If it did, I returned true. For the getRoad method, I created two town objects with the given towns. I then created a new road object with the getEdge method with the two towns. I then returned the road object's name. For the addTown method, I created a boolean variable and a new town object with the given string. I then check to see if the town was added to the graph. If it was, I returned true, if not, I returned false. For the getTown method, I created my Set<Town> variable to hold the populate the towns of the graph. I used a for loop to run through the towns and checked if the current town's name was equal to the given name. If it was, I returned that town. If no town was found, I returned null. For the containsTown method, I created a boolean variable and new town object. I checked if the graph contained the given town and if it did, then I returned true and if not, I returned false. For the containsRoadConnection method, I created two town objects and a boolean variable. I checked if the graph contained an edge with the given two towns and if it did, I returned true. If it didn't, I returned false. For the allRoads method, I created an array list variable and then used a for loop to run through the graph's roads. I added each road's name of the set to the array list. Once all added, I used the sort method in collections to sort the array list alphabetically. For the deleteRoadConnection method, I created a boolean variable and two town objects. I created a new road object based on the given information as well. I checked if the new road equals to the given road name. If it did, I removed the from the graph using the removeEdge method. I then checked if the graph contained the given towns using the containsEdge

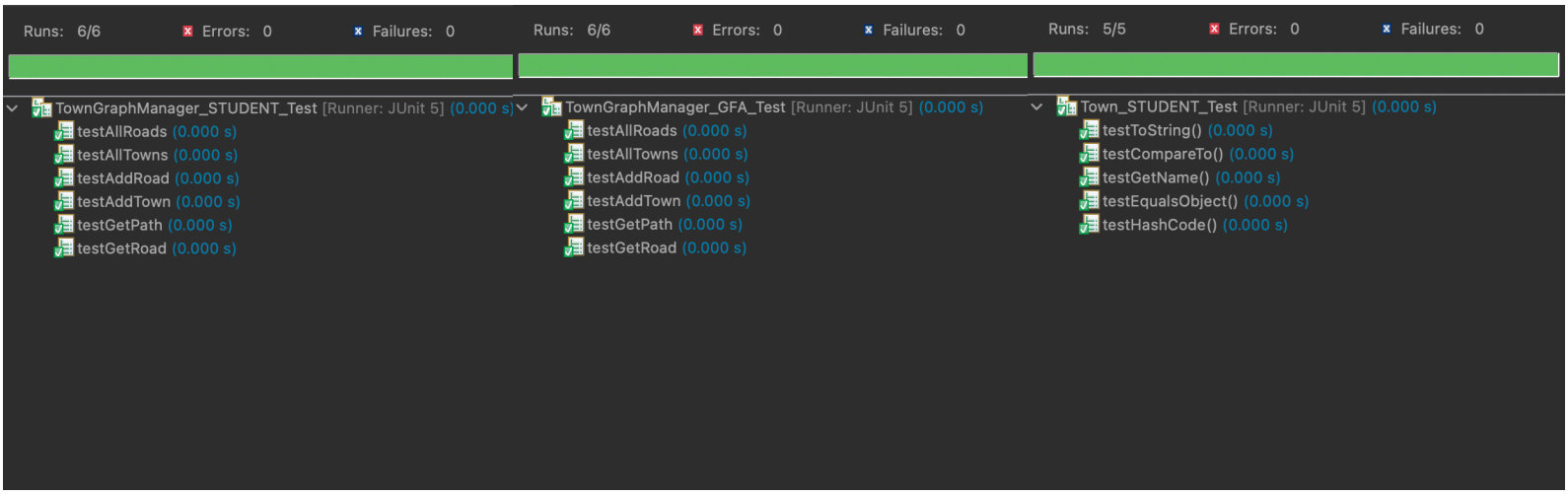
method. If it did, I retuned false and if it didn't I returned true since it was successfully deleted. For the deleteTown method, I created a Boolean variable and town object. I checked if the graph has removed the given town. If it did, I returned true and if it didn't, I retuned false. For the allTowns method, I created an array list variable and then ran through the graphs towns using a for loop. I then added each town to the array list and once done, I used the collection sort method to sort it alphabetically. For the getPath method, I created two town objects based on the given strings. I created a new array list to hold the path information based on the two towns and called the shortestPath method to grab that information. I then retuned that arraylist. For the populateTownGraph method, I created a scanner object, a string variables and string array variables. While the file had a next line, I divided the current line by split it into the array based on the semicolon. I then focused on the first element of that current line and split that into an array based off the comma. I assigned the current road as the first element of that array and the weight as the second element of that array. I assigned the town source as the second element of the current line and the town destination as the third element of the current line. I then used the addTown method to add both towns along with using the addRoad method to add the connecting towns, weight(miles) and the road name.

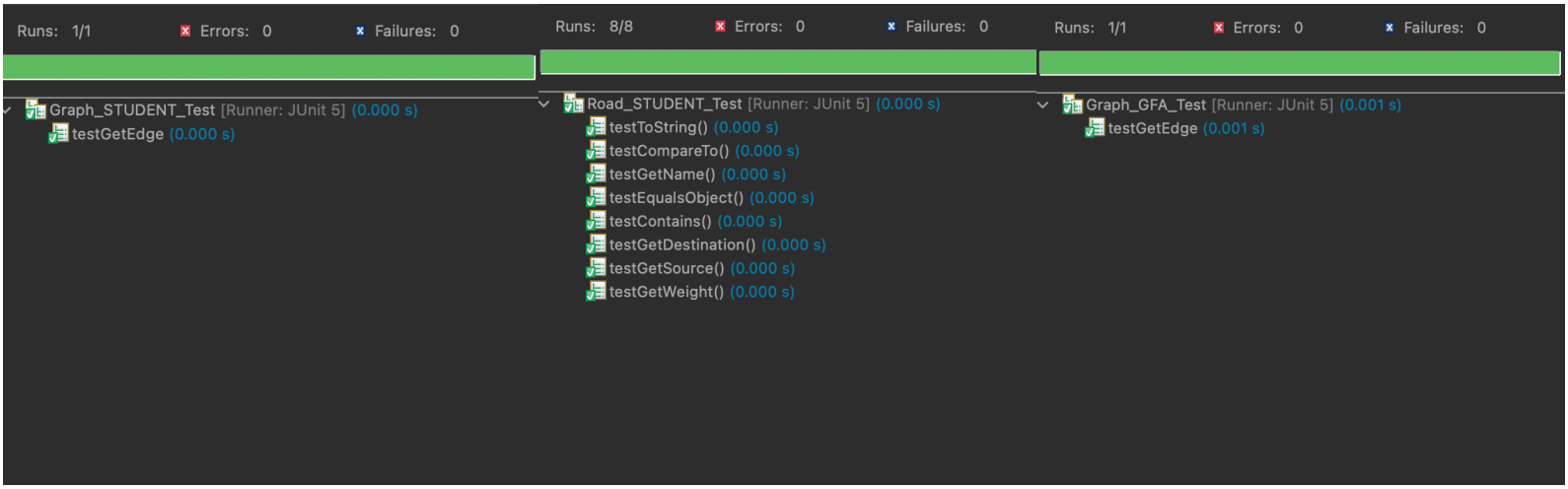
Learning Experience: I have learned a lot from this project. I reenforced my knowledge of creating proper equals and compareTo method for both the Town and Road classes. I learned how to properly create sets and hash maps along with inserting information into them as well as updating them. I got comfortable with enhanced for loops which were never my strong suit. It was either choosing using an iterator or learning how to properly use enhanced for loops which were much easier to deal with. Using an iterator seemed more work than it needed to be so I'm glad I finally started implementing enhanced for loops and probably should have been implementing them throughout all my projects. I learned how to properly added edges, vertices as well as getting those edges and vertices based on two given towns. I learned how to implement the Dijkstra shortest algorithm using a hash map, arraylist, town and road sets. I got more comfortable working with sets and hash maps, initializing information into them as well as updating those maps.

Challenges: Majority of the challenges came from shortestPath methods. The adjacency matrices didn't make too much sense to me in the book and I couldn't really understand it. Decided to use sets of towns and roads instead which seemed to work from my testing. I had an issue for the longest time of where I could only get the correct output when selecting Only other challenge I had not being able to clear the hash map when inserting two files. Once you insert one file, if you insert another, it combines the information without removing the previous files information.

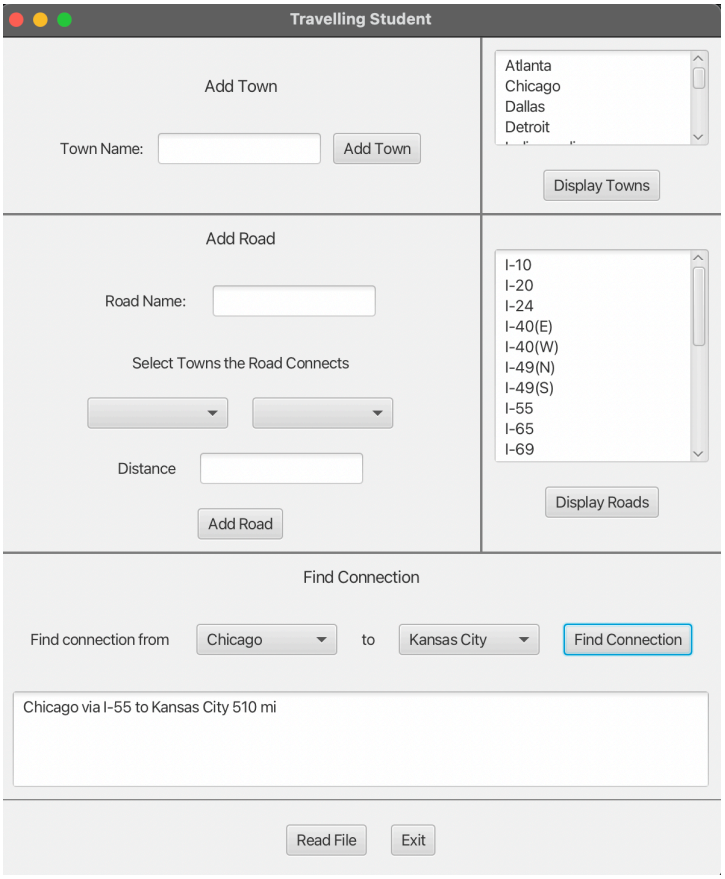
Assumption: No assumptions here. I deleted the package abc for the graph Junit since it wouldn't run otherwise.

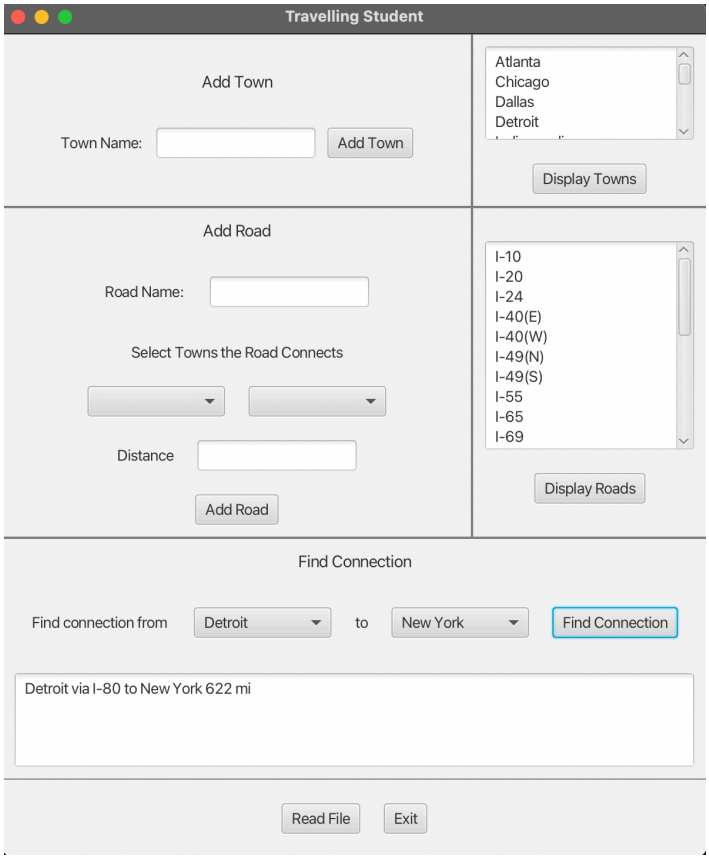
Junits

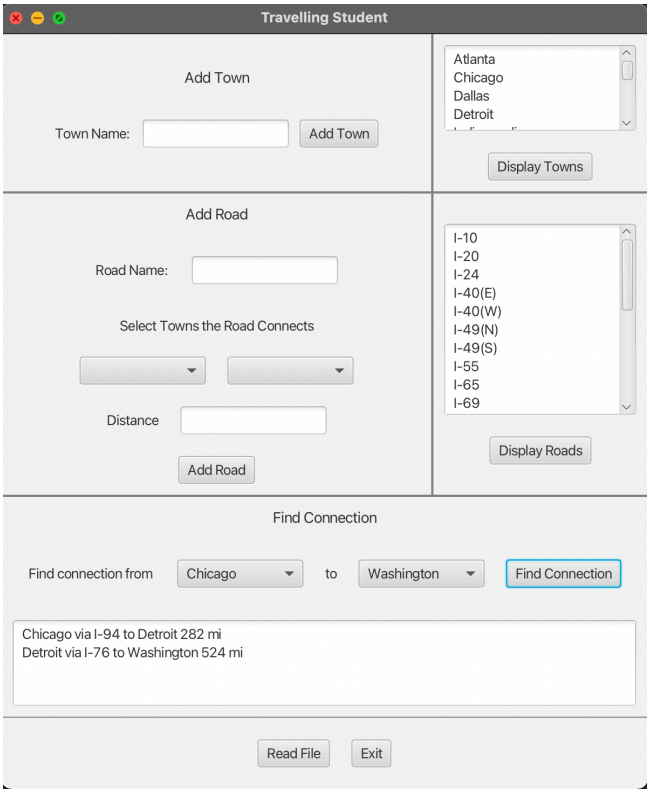

Runs: 6/6
Errors: 0
Failures: 0
TownGraphManager_STUDENT_Test [Runner: JUnit 5] (0.000 s)
testAllRoads (0.000 s)
testAllTowns (0.000 s)
testAddRoad (0.000 s)
testAddTown (0.000 s)
testGetPath (0.000 s)
testGetRoad (0.000 s)
Runs: 6/6
Errors: 0
Failures: 0
TownGraphManager_GFA_Test [Runner: JUnit 5] (0.000 s)
testAllRoads (0.000 s)
testAllTowns (0.000 s)
testAddRoad (0.000 s)
testAddTown (0.000 s)
testGetPath (0.000 s)
testGetRoad (0.000 s)
Runs: 5/5
Errors: 0
Failures: 0
Town_STUDENT_Test [Runner: JUnit 5] (0.000 s)
testToString() (0.000 s)
testCompareTo() (0.000 s)
testGetName() (0.000 s)
testEqualsObject() (0.000 s)
testHashCode() (0.000 s)


Runs: 1/1
Errors: 0
Failures: 0
Graph_STUDENT_Test [Runner: JUnit 5] (0.000 s)
testGetEdge (0.000 s)
Runs: 8/8
Errors: 0
Failures: 0
Road_STUDENT_Test [Runner: JUnit 5] (0.000 s)
testToString() (0.000 s)
testCompareTo() (0.000 s)
testGetName() (0.000 s)
testEqualsObject() (0.000 s)
testContains() (0.000 s)
testGetDestination() (0.000 s)
testGetSource() (0.000 s)
testGetWeight() (0.000 s)
Runs: 1/1
Errors: 0
Failures: 0
Graph_GFA_Test [Runner: JUnit 5] (0.001 s)
testGetEdge (0.001 s)

Running application

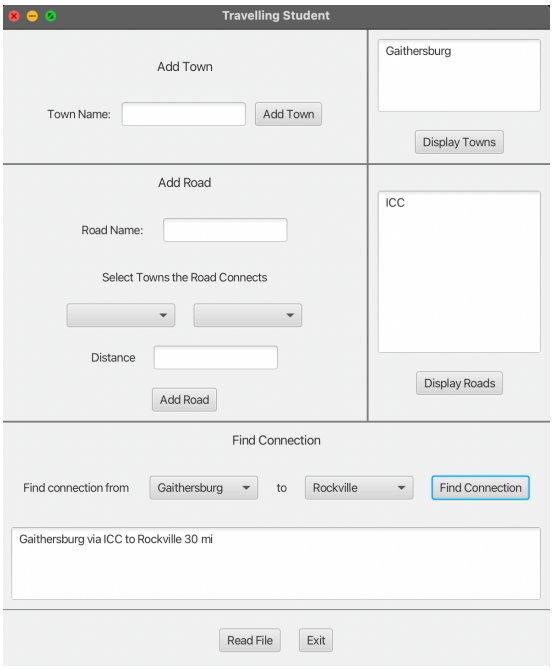






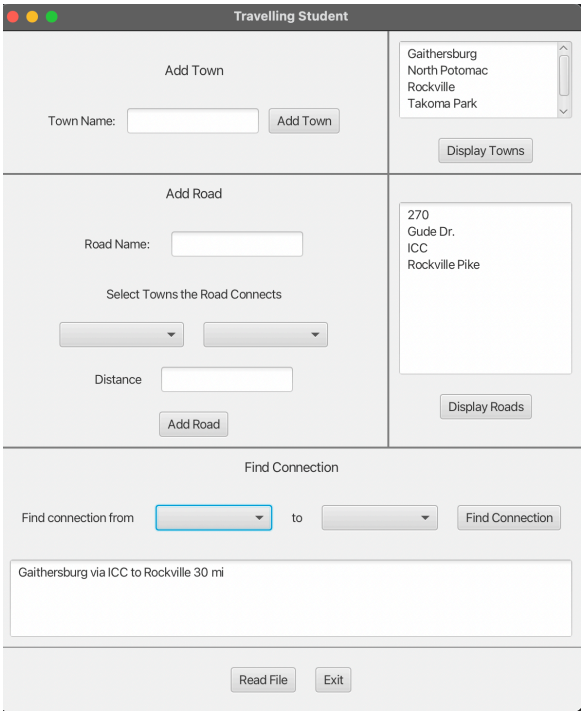
Testing

Expected: Add the town gaithersburg and rockville and a connection of the road ICC with 30 miles
Reality: Successfully adds the town gaithersburg and rockville and a connection of the road ICC with 30 miles



Expected: add Gaithersburg, North Potomac, Rockville and Takoma Park with roads of 270, Gude Dr., ICC and Rockville Pike. Takoma park via Rockville pike to rockville should be 15 miles, rockville via 270 to gaithersburg should be 5 miles

Reality: Successfully adds Gaithersburg, North Potomac, Rockville and Takoma Park with roads of 270, Gude Dr., ICC and Rockville Pike.



Expected: Find a connection between Takoma park and Gaitherburg from above. Should result in Tacoma park via Rockville Pike to Rockville 15 mi. Rockville via 270 to Gaitherburg 5 miles.
Reality: Successfully find a connection between Takoma Park and Gaitherburg.