Addison Goldwait & CJ Silverman 12.11.18
Professor Xia CS 150

CS Project III Report

I. Introduction

The country of Coffeeland contracted out the position of logistics manager to the consulting company, CS Consultants (CSC). The first assignment submitted to the CSC team was to create a program that optimized profit of their coffee shop. Most recently, Coffeeland requires the expertise of our team to schedule the truck routes that minimize the total distance traveled between the cities. CSC has created a program that outputs a list of truck routes that contains the list of cities in order of visitation and the cargos delivered to the warehouse in each city. There is an outline of the remainder of this document and its contents in as seen below.

II. Approach

Step 1: Setting Up the Problem- The first step in our approach is to define the parameters of the scenario and organize the given information.

Goal: Output an optimal truck schedule that minimized distance traveled

Given Information: A map of Coffeeland's cities and roads and a list of warehouses and their respective shipping requests. This information is provided to us by R&D in the form of three .txt files.

- Roads.txt → The first line is *n*, the number of roads, followed by *n* lines of the format *city1 city2 length*, where *city1* and *city2* are strings and *length* is an integer. Each city name is a string of English letters without spaces or special symbols. All roads are two way and every pair of cities are reachable from each other by a sequence of roads.

- Warehouses.txt → The first line is *m*, the number of warehouses, followed by *m* lines of the format *city* w_1 w_2 ... w_k, indicating the city of the warehouse, and the integer weights of the cargos the warehouse requests. All warehouses are located in distinct cities.

- Center.txt → The city of the logistics center

Parameters/ Restrictions: Each truck has the same capacity of 500 units that cannot be exceeded, and only one truck may be in operation at a time.

Step 2: Developing a Plan- Now, that we have all the initial information the programmers can start to see a plan come into place. We develop a general blueprint, that shows how we want the program to operate.

1. The program must translate the .txt files from R&D into more palatable form.
2. After the input data is loaded into the program, we can then run the simulation that schedules the truck routes. Starting from the current location of the truck, go to the closest warehouses *W* whose request has not been completely fulfilled and at least one of whose unfulfilled cargos can be added to the truck without exceeding its capacity. If no such warehouse exists, return to the logistics center. If more than one warehouses are equally close, choose the one whose city name is alphabetically first. Then we ship as many cargos as possible to *W* until either all cargos to *W* are shipped, or shipping anymore cargo to *W* will exceed the truck's capacity. The truck goes to the location W, and the process starts again. This simulation is run for each truck until all orders are filled.

3. The program outputs the truck schedules from the simulation that lists the cities in the order it visits and the cargos it delivers to the warehouse in each city, along with the total distance traveled by all the trucks.

<u>Step 3: Laying the Foundation</u>- Armed with our blueprint, we can now start setting up the class structure. When a project has as many moving parts as this one, it often helps to break down the classes.

> Graph→ We converted the map of Coffeeland's cities and roads into the Graph data structure because they decided it was the most efficient way to access and manipulate the data. The Cities are the vertices and the roads are the edges.
>
> Road→ This is a smaller class; each road object only needs to worry about the cities it connects and the distance between the two.
>
> City→This class is highly involved; City houses the warehouses that send the requests and the roads that connect it to adjacent cities.
>
> Warehouse→ This class contains information read in from the .txt files from R&D, such as shipping requests and cargo.
>
> Cargo→ Cargo is the object of a certain weight that is shipped via truck.
>
> Truck→ This class monitors the distance it travels and its weight capacity.
>
> Shipment→ The shipment class manages the requests and coordinates between Truck, Cargo, and Warehouse.
>
> ExperimentController→ This is the top-level class that controls all the moving parts.

III. Methods

Graph Class

a. addEdge: adds a two-way edge to the Graph
b. addDirectedEdge: adds a one-way edge to the Graph
c. getVertex: returns the vertex associated with the given name
d. hasWarehouse: Boolean return is a city has a warehouse
e. setWarehouse: assigns a warehouse to a city
f. bfs: runs a BFS from a given start vertex
g. dfs: runs a DFS from a given start vertex
h. recursiveDfs: runs a recursive DFS from a given vertex
i. shortestPath: returns the shortest path from the given start vertex when passed a String
j. shortestPath: finds the shortest path from the given start vertex when passed a Vertex
k. shortestPath: finds the shortest path when passed a city name and an integer
l. shortestPath: finds the shortest path when passed a vertex and an integer. This along with item k set a variable in each city called centerDist which is the distance back to the logistics center and does not change
m. returnDist: returns the distance variable of String
n. reset: resets the parameters of all vertices

Road

a. Road Constructor: assigns variables to City u, City v, and weight w.

City

a. City Constructor: creates a city object
b. setWarehouse: assigns a warehouse to the variable cityWarehouse
c. setClosestCities: assigns an ArrayList of cityList to the variable closestCities
d. compareTo: an override method that compares the names of two cities

Warehouse

a. Warehouse Constructor: creates a Warehouse object and assigns a String to city
b. addCargo: adds a cargo to incomingCargo variable
c. cargoSort: sorts the incomingCargo ArrayList variable
d. compareTo: an overwrite method that compares two warehouses
e. getIncomingCargo: returns incomingCargo ArrayList

Cargo

a. Cargo Constructor: creates a cargo object

b. toString: override method that returns a String of Cargo’s variables
c. compareTo: returns the comparison of two Cargo objects’ weight

Truck

a. Truck Constructor: creates a truck object
b. printStrings: returns the loggedDistance
c. addShipment: adds shipment, s to shipList
d. returnTrip: adds the return trip distance to the variable distanceTraveled

Shipment

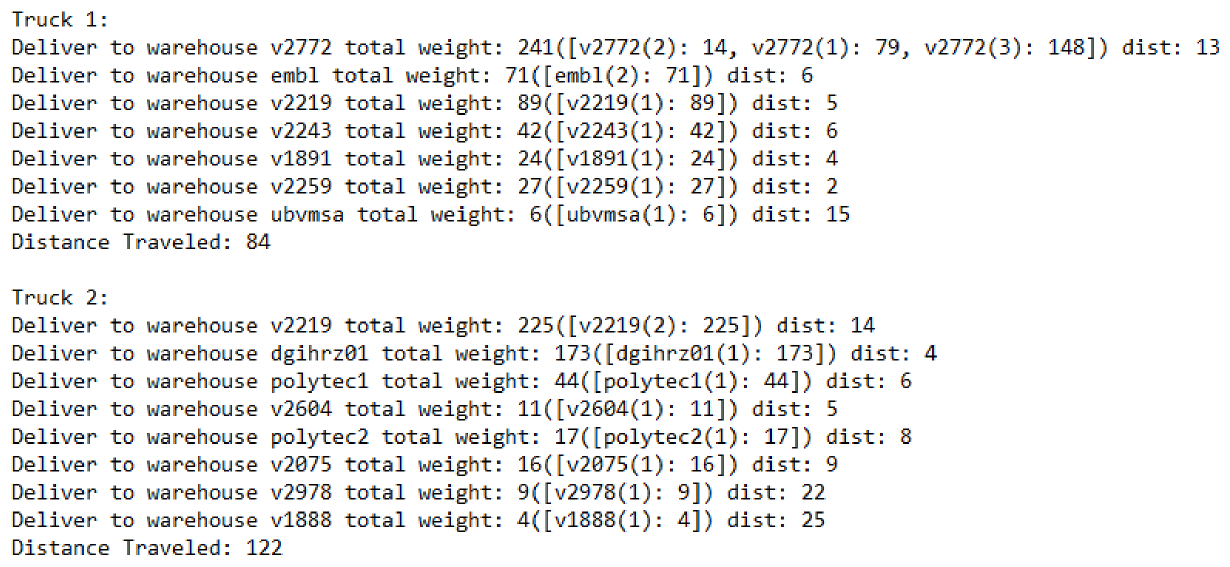
a. Shipment Constructor: creates a shipment object
b. toString: override method that returns the variables of Shipment
c. addCargo: adds cargo object, c to the shipmentCargo variable
d. setDestination: assigns destination variable to object City d
e. setDistance: assigns distance
f. getReturnDistance: returns distance from destination to the Logistics Center

ExperimentController

a. ExperimentController Constructor: Creates an ExperimentController object and assigns the R&D input .txt files
b. shortestPath: declares the shortest path of the city map
c. setCenterShortestPath: declasres the shortest path from the center
d. main method: controls the simulation process
e. mapCity: processes the input .txt files and translates them into a more palatable form

IV. Data and Analysis

The figure below displays the first two optimized truck schedules. The remainder of the truck schedules can be found in the file, output.txt.

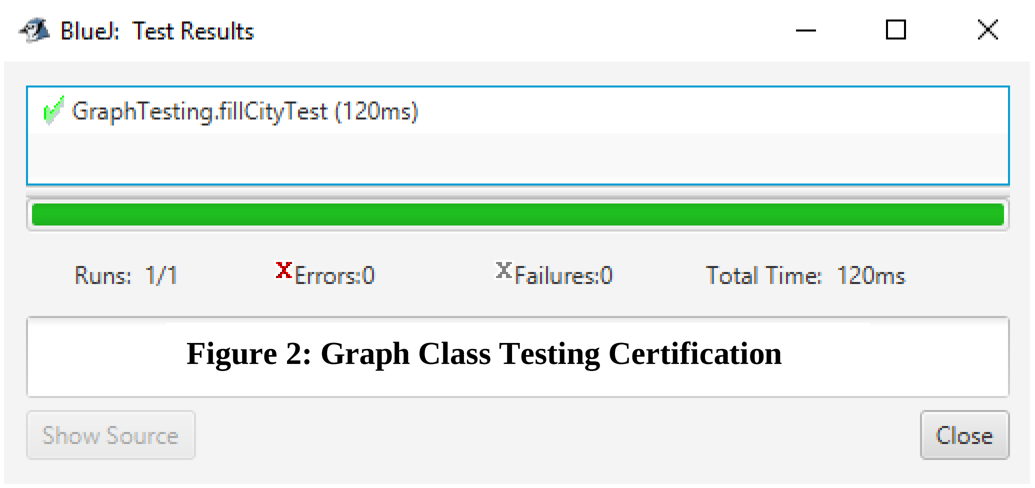


```
Truck 1:
Deliver to warehouse v2772 total weight: 241([v2772(2): 14, v2772(1): 79, v2772(3): 148]) dist: 13
Deliver to warehouse embl total weight: 71([embl(2): 71]) dist: 6
Deliver to warehouse v2219 total weight: 89([v2219(1): 89]) dist: 5
Deliver to warehouse v2243 total weight: 42([v2243(1): 42]) dist: 6
Deliver to warehouse v1891 total weight: 24([v1891(1): 24]) dist: 4
Deliver to warehouse v2259 total weight: 27([v2259(1): 27]) dist: 2
Deliver to warehouse ubvmsa total weight: 6([ubvmsa(1): 6]) dist: 15
Distance Traveled: 84

Truck 2:
Deliver to warehouse v2219 total weight: 225([v2219(2): 225]) dist: 14
Deliver to warehouse dgihrz01 total weight: 173([dgihrz01(1): 173]) dist: 4
Deliver to warehouse polytec1 total weight: 44([polytec1(1): 44]) dist: 6
Deliver to warehouse v2604 total weight: 11([v2604(1): 11]) dist: 5
Deliver to warehouse polytec2 total weight: 17([polytec2(1): 17]) dist: 8
Deliver to warehouse v2075 total weight: 16([v2075(1): 16]) dist: 9
Deliver to warehouse v2978 total weight: 9([v2978(1): 9]) dist: 22
Deliver to warehouse v1888 total weight: 4([v1888(1): 4]) dist: 25
Distance Traveled: 122
```

Figure 1: Optimal Truck Schedule

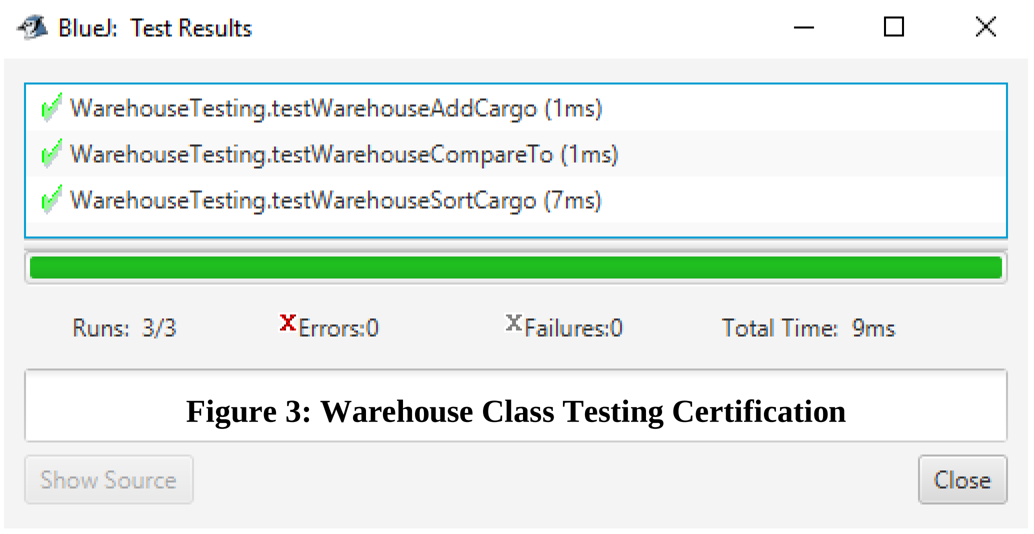
Here at CSC Consulting, you don't have to know everything about designing a program. Leave that to us! When a client entrusts CSC with their business, we guarantee a level of quality that can't be found anywhere else. To assure your program is firing on all cylinders, we put our code through rigorous testing, and give you easy-to-read results. The figures below show that this program passed all CSC standards of excellence and is ready to perform for you.



✓ GraphTesting.fillCityTest (120ms)

Runs: 1/1 Errors:0 Failures:0 Total Time: 120ms

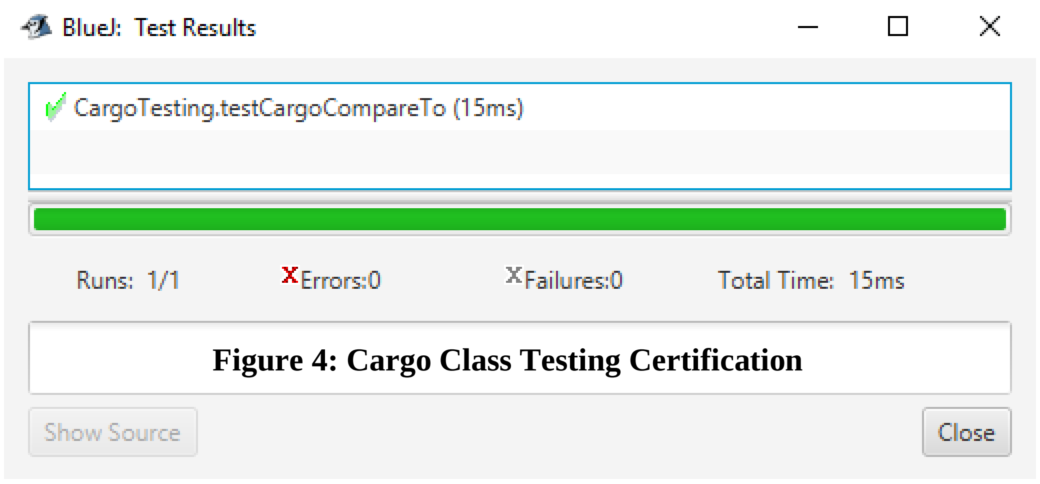
Figure 2: Graph Class Testing Certification



```
✓ WarehouseTesting.testWarehouseAddCargo (1ms)
✓ WarehouseTesting.testWarehouseCompareTo (1ms)
✓ WarehouseTesting.testWarehouseSortCargo (7ms)

Runs: 3/3    Errors:0    Failures:0    Total Time: 9ms
```

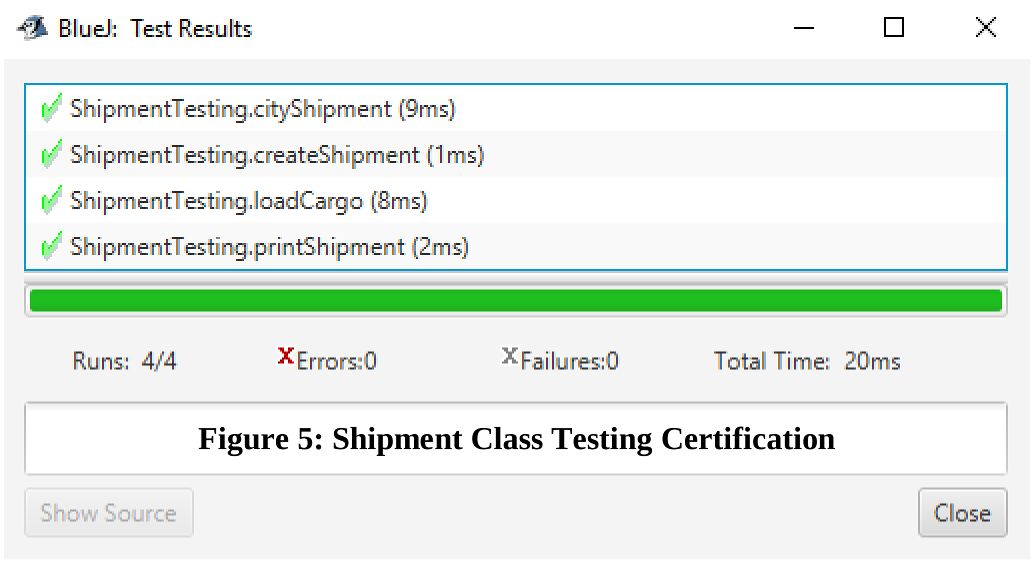
Figure 3: Warehouse Class Testing Certification



```
✓ CargoTesting.testCargoCompareTo (15ms)

Runs: 1/1    Errors:0    Failures:0    Total Time: 15ms
```

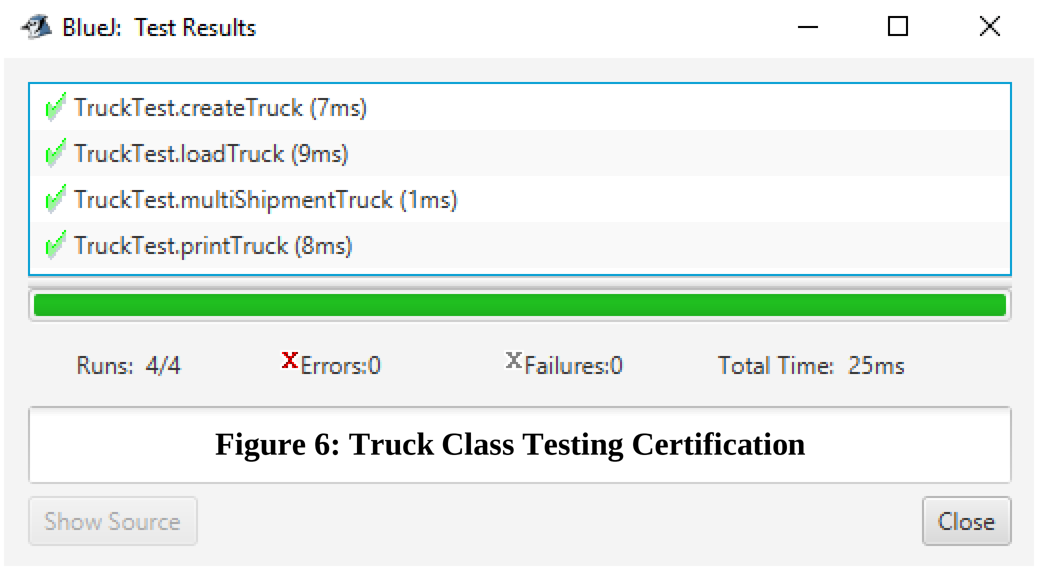
Figure 4: Cargo Class Testing Certification



```
✓ ShipmentTesting.cityShipment (9ms)
✓ ShipmentTesting.createShipment (1ms)
✓ ShipmentTesting.loadCargo (8ms)
✓ ShipmentTesting.printShipment (2ms)

Runs: 4/4    Errors:0    Failures:0    Total Time: 20ms
```

Figure 5: Shipment Class Testing Certification


BlueJ: Test Results

```
✓ TruckTest.createTruck (7ms)
✓ TruckTest.loadTruck (9ms)
✓ TruckTest.multiShipmentTruck (1ms)
✓ TruckTest.printTruck (8ms)

Runs: 4/4    Errors:0    Failures:0    Total Time: 25ms
```

Figure 6: Truck Class Testing Certification

V. Conclusion

Coffeeland, contracted out the responsibilities of Logistics Manager to CS Consultants to optimize the delivery schedule. CSC has created a program that outputs a list of truck routes that contains the list of cities in order of visitation and the cargos delivered to the warehouse in each city. CS Consulting wants to thank you for your loyal patronage, and for all your future consulting needs, choose the company that aims to please, CS Consulting.

VI. References

Java API's:

PrintWriter- https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html

HashMap<K,V>- https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

String- https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Integer- https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

Interface Comparable<T>- https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Textbook- Weiss, Mark Allen. *Data Structures & Problem Solving Using Java*. Pearson Education, 2010.

FIGURE 4