Vishal Yathish

28 November 2023

Programming Project #4 Report

**PROJECT IMPLEMENTATION**

The first thing I implemented was the Passenger class. This was relatively easy; it just required some basic getter and setter methods for the mClass, mEmbarked, mName, mSurvived, and mFare variables.

The next thing I implemented was the PassengerDatabase class. I worked on the getters first. getPassengers() returns the whole passengers vector. getPassenger\_byName() iterates through passengers vector, checks if each passenger has the same name as the input string, and if so, it returns that Passenger pointer object. getPassengers\_byClass()/byEmbarcation() iterate through the passengers vector, see if each element has the chosen attributes (Class, Embarcation, etc.), and if so, pushes it to an output vector. The resulting vector will be a filtered subset of the passengers vector with the chosen attributes. Next, I implemented PassengerDatabase’s version of the inherited csvData() method (from the DataCollectorCallback class). This creates a new Passenger pointer and uses the setter methods from Passenger class to assign attribute values from the map from the CSVFile class. The clearAllPassengers() method was interesting to work on. It clears the passenger vector; while the passenger is not empty, it pops the last pointer off the vector and deletes it (to prevent memory leaks). The first time I implemented this method, I tried using an iterator object, but this did not work properly, as it breaks when the pointer is deleted. The destructor method for this class, since they accomplish the same task, simply calls the clearAllPassengers().

The last component was the Output Tables: the PassengerOutputTable class and the OutputTableRow class. Aside from the basic getters and setters, the most significant method of PassengerOutputTable was the display() method, which produces the overall table. The first step was to use an iterator object and while loop to traverse the passengers vector get summary statistics/counts of survival and totals by class, embarkation, and fare. Once the summary statistics have been fully filled, I assigned row values by mField and within these, I assigned specific row descriptions for each value within the field. When displaying each row, I called each OutputTableRow object’s display() method. I used the “\t” character to pad each value, and align the columns in the final output table, instead of implementing the private padToThreeCharacters() method. Howard said that this was acceptable and would not be tested by the scorer.

**PROJECT TESTING**

My debugging process was quite straightforward since my logic and computations appeared to be quite accurate. I tested this by creating the basic tables: List of all passengers by class, List of all passengers by embarcation, and List of all passengers by fare. I then compared the values found here to Howard’s final numbers he added in the Discussion Forum, and they matched.

Formatting was an iterative process. Initially, with the row descriptions, I had the title and the dash next to them in the actual row description, but this posed an issue when calling the getRowDescription() functions, since they weren’t standardized. For example, assert (row.getDescription() == ">$50"). So, I removed the escape character and the dash from the actual description and moved it to the display() function. I also iterated the final output table to remove some unnecessary spacing, and added special cases to the row display() function for the fare descriptions to align their columns (since they had fewer characters than the other Field row descriptions).

I also tested my files in g32 to make sure that there were no memory leaks and it built and ran without errors.

A number of passengers

Description automatically generated with medium confidence

A number of passengers

Description automatically generated

A close-up of a ticket

Description automatically generated