Due 4/28

For programming assignment 7 we will be implementing a Car class. The Car class will implement the CarFunctions interface. Most of the functions in the CarFunctions interface have default implementations, so that you can download the program files and compile immediately. You will need to override all of the default functions in your implementation of the Car class. I have provided an implementation of Car.toString(). Additionally we will be implementing the ManageCarData class, which implements the ManageCarDataFunctions interface. I have included default implementations for all of the methods in ManageCarDataFunctions, so that you can compile. I've posted on Brightspace the two interfaces, mostly empty versions of the two classes, an implementation of the TotalRangeComparator comparator and an empty version of the RemainingRangeComparator comparator.

We will be making use of the fact that an interface can be used as a type. So, in the ManageCarDataFunctions and ManageCarData instead of returning lists of type Car we will be retuning lists of type CarFunctions. This is also true for the comparators. This should make implememtation of the code a little easier. We are also doing similarly for programs that are used for testing, using ManageCarDataFunctions as the type for ManageCarData objects.

Note that I could not include a default version of toString() in the CarFunctions interface, since that would override the toString() method in Object. So, I included an implementation in the Car class.

I've also included testCarData.java, which you can use to exercise your code. You will need to change "ManageCarDataFunctions manageCarData = **new** your_last_name_in_lower_case_ManageCarData();" to the obvious thing in testCarData.java.

Note in the below that I may reference the Car class as either Car or your_last_name_in_lower_case_Car. And likewise for ManageCarData class.

This programming assignment is giving us additional experience with:

1) Creating classes and implementing methods
2) Creating ArrayLists and PriorityQueues
3) Working with ArrayLists and PriorityQueues
4) Using an Iterator
5) Searching an ArrayList
6) Comparing fields of an object to values
7) Comparing fields in multiple objects
8) Reading text files
9) Parsing text
10) Making copies of ArrayLists and PriorityQueues
11) Further insight as to how objects interact
    a) Returning an original object in a method can have unintended effects
    b) Returning orignal objects in a copy of a list can have unintended effects

Below are the files and what they do and what you need to implement in each of them.

1) CarFunctions.java
   a) Interface that specifies the methods that need to be implemented in the Car class
   b) You don't need to make any changes to this file

2) ManageCarDataFunctions.java
   a) Interface that specifies the methods that need to be implemented in the ManageCarData class
   b) You shouldn't need to make any changes to the file

3) TotalRangeComparator.java
   a) Comparator for the the PriorityQueue of CarFunctions in total range and id order
   b) You don't need to make any changes to this file

4) RemainingRangeComparator.java
   a) Comparator for the the PriorityQueue of CarFunctions in remaining range and id order
   b) You need to implement this class. You should use TotalRangeComparator.java as a guide as to what to do for this file.

5) testCarData.java
   a) This program can be used for initial testing of your code.
   b) Change the one occurrence of “your_last_name_in_lower_case_ManageCarData” to the obvious
   c) You should make whatever changes you think are needed to sufficiently test your code in this file.

6) your_last_name_in_lower_case_Car.java
   a) This is the Car class
   b) Change the name in the obvious way
   c) You will need to add four fields, which should be private and three of them should be final
      1. A string that identifies the instantiation of the Car class. I called mine id and will use that name in this program description.

      2. An int that stores the fuel economy in miles per gallon for the instantiation of the Car class. I called mine fuelEconomyInMilesPerGallon and will use that name in this program description.

      3. An int that stores the fuel capacity in gallons for the instantiation of the Car class. I called mine fuelCapacityInGallons and will use that name in this program description.

      4. A double that stores the current amount of fuel in gallons for the instantiation of the Car class. I called mine currentFuelInGallons and will use that name in this program description. This field will be updatable, so it cannot be final.

- d) You will need a four parameter constructor that assigns the four field values

- e) You will need implementations of the following methods
  1. public int getFuelEconomyInMilesPerGallon()
     1. Return the fuel economy in miles per gallon

  2. public int getFuelCapacityInGallons()
     1. Return the fuel capacity in gallons

  3. public double getCurrentFuelInGallons()
     1. Return the current fuel in gallons

  4. public String getId()
     1. Return the id

  5. public String toString()
     1. I've provided an implementation that should work

  6. public void setCurrentFuelInGallons(double v)
     1. Update the current fuel in gallons

  7. public double getTotalRangeInMiles()
     1. The default implementation of this should be sufficient

  8. public double getRemainingRangeInMiles()
     1. The default implementation of this should be sufficient

7) your_last_name_in_lower_case_ManageCarData.java
   - a) This class is used to create instances of the Car class and manage getting the required data from the Car class
     1. Car objects will be assigned to variables of type CarFunctions
     2. For what this is worth, I only have one statement in my class that calls the Car constructor, and that is in the readData method
   - b) Change the name in the obvious way
   - c) You will need to add the following three fields, which should all be private and final
     1. An ArrayList of type CarFunctions. I called mine carList and will use that name in this program description.

     2. A PriorityQueue of type CarFunctions using the TotalRangeComparator comparator. I called mine carListByTotalRange and will use that name in this program description.

     3. A PriorityQueue of type CarFunctions using the RemainingRangeComparator comparator. I called mine carListByRemainingRange and will use that name in

this program description.

d) A zero parameter constructor that instantiates the three fields

e) You will need implementations of the following methods
   1. public void readData(String filename)
      1. You need to read the text file
         1. For each line
            1. Parse the id, fuel economy, fuel capacity, and current fuel
            2. Instantiate a new Car object with the above four values
            3. Add the Car object to the ArrayList and two PriorityQueues

   2. public ArrayList<CarFunctions> getCarList()
      1. Return a copy of carList
      2. We return a copy, since the code that made a call to the method could modify the ArrayList
      3. I looped over the carList elements, and added each to the a new ArrayList of type CarFunctions, and returned that
      4. This won't stop the calling method from changing the current fuel in the objects in the returned ArrayList, which would actually change the current fuel in carList, carListByTotalRange, and carListByRemainingRange
         1. We could have made copies of each Car object as part of the copy to ensure that what we returned was completely independent of the fields of our Car class.

   3. public PriorityQueue<CarFunctions> getCarListByTotalRange()
      1. Return a copy of carListByTotalRange
      2. We return a copy, since if the code that made the call to the method ever does a poll() of the returned object, we don't want that to effect our original copy of the data.
      3. I looped over the carList elements, and added each to the a new PriorityQueue of type CarFunctions instantiated with the appropriate comparator, and returned that

   4. public ArrayList<CarFunctions> getCarListByTotalRangeUsingIterator()
      1. Create a new ArrayList of type CarFunctions
      2. Create an Iterator of type CarFunctions for carListByTotalRange
      3. Loop over the elements of carListByTotalRange using the iterator
      4. Add each element of carListByTotalRange returned by the iterator to the ArrayList
      5. Return the ArrayList

   5. public PriorityQueue<CarFunctions> getCarListByRemainingRange()
      1. Return a copy of carListByRemainingRange
      2. We return a copy, since if the code that made the call to the method ever

    does a poll() of the returned object, we don't want that to effect our original copy of the data.
    3. I looped over the carList elements, and added each to the a new PriorityQueue of type CarFunctions instantiated with the appropriate comparator, and returned that

6. public ArrayList<CarFunctions> getCarListByRemainingRangeUsingIterator()
    1. Create a new ArrayList of type CarFunctions
    2. Create an Iterator of type CarFunctions for carListByRemainingRange
    3. Loop over the elements of carListByRemainingRange using the iterator
    4. Add each element of carListByRemainingRange returned by the iterator to the ArrayList
    5. Return the ArrayList

7. public ArrayList<String> getCarListByTotalRangeViaPoll(double minTotalRange, double maxTotalRange)
    1. Create a new ArrayList of type String
    2. Loop over the elements of carListByTotalRange using the poll() method, this will return the elements of carList in total range in miles and then id order
        1. For each element that has total range in [minTotalRange,maxTotalRange]
            1. Call the current Car element currentCar
            2. Get the string representation of the Car, call it currentCarString (currentCar.toString() should be what you want)
            3. Loop over the ArrayList carList elements
                1. If the current carList element is equal to currentCar, there will be exactly one, append a tab and the current index to currentCarString
                    1. You can use the Object.equals() to determine which element of carList is equal to currentCar
                    2. You can probably also use the ArrayList.indexOf(Object o) to accomplish this without using a loop
            4. Loop over the ArrayList carList elements
                1. If the current carList element has the same fuel economy as currentCar, there will be at least one, append a tab and the current index to currentCarString
                    1. Use the Car.getFuelEconomyInGallons() method for the comparison of the two Car's fuel economy
            5. Note: if you create two Strings, one for the index of the Car that equals the currentCar and one for the Cars that have the same fuel economy, you only need to loop over carList once, and then append the two to currentCarString after exiting the loop, this is what I did
            6. Since poll() removes the elements from the PriorityQueue, at the end of the method you will need to add all of the elements back into the PriorityQueue, which can be done by looping over the elements of

        carList, and then using the PriorityQueue.add() method to add the current element of carList to carListByTotalRange

8. public ArrayList<String> getCarListByRemainingRangeViaPoll(double minRemainingRange, double maxRemainingRange)
    1. Create a new ArrayList of type String
    2. Loop over the elements of carListByRemainingRange using the poll() method, this will return the elements of carList in remaining range in miles and then id order
        1. For each element that has remaining range in [minRemainingRange,maxRemainingRange]
            1. Call the current Car element currentCar
            2. Get the string representation of the Car, call it currentCarString (currentCar.toString() should be what you want)
            3. Loop over the ArrayList carList elements
                1. If the current carList element is equal to currentCar, there will be exactly one, append a tab and the current index to currentCarString
                    1. You can use the Object.equals() to determine which element of carList is equal to currentCar
                    2. You can probably also use the ArrayList.indexOf(Object o) to accomplish this without using a loop
            4. Loop over the ArrayList carList elements
                1. If the current carList element has the same fuel economy as currentCar, there will be at least one, append a tab and the current index to currentCarString
                    1. Use the Car.getFuelEconomyInGallons() method for the comparison of the two Car's fuel economy
            5. Note: if you create two Strings, one for the index of the Car that equals the currentCar and one for the Cars that have the same fuel economy, you only need to loop over carList once, and then append the two to currentCarString after exiting the loop, this is what I did
            6. Since poll() removes the elements from the PriorityQueue, at the end of the method you will need to add all of the elements back into the PriorityQueue, which can be done by looping over the elements of carList, and then using the PriorityQueue.add() method to add the current element of carList to carListByRemainingRange

In my version of testCarData, I included the code below to verify that my code was returning copies of the PriorityQueues instead of the original objects. I will eventually also add something to verify that I returned a copy of the ArrayList of CarFunctions instead of the original object.

```
// poll out the elements of the PriorityQueue ordered by remaining range
PriorityQueue carsByRemaingRange = manageCarData.getCarListByRemainingRange();
while( carsByRemaingRange.size() > 0 )
```

```
{
            carsByRemaingRange.poll();
}

// poll out the elements of the PriorityQueue ordered by total range
PriorityQueue carsByTotalRange = manageCarData.getCarListByTotalRange();
while( carsByTotalRange.size() > 0 )
{
            carsByTotalRange.poll();
}
```

The above will cause any calls that happen after the above to output the PriorityQueue data to be empty if the original object was returned instead of a copy. I didn't include the code in the version of testCarData that I posted on Brightspace, since I didn't want that to slow your initial progress with the program.

Returning a copy should not be a major endevour, below is how I do it for the PriorityQueue that orders the Cars by total range.

```
public PriorityQueue<CarFunctions> getCarListByTotalRange()
{
            PriorityQueue<CarFunctions> tempCarListByTotalRange = new PriorityQueue<>(new TotalRangeComparator());
            for( CarFunctions c : carList )
            {
                        tempCarListByTotalRange.add(c);
            }
            return tempCarListByTotalRange;
}
```

To submit your program, e-mail it me at bi92798@binghamton.edu by midnight on the due date, with a subject of “cs 140 program 7”. Attach the files “your_last_name_in_lower_case_Car.java” and “your_last_name_in_lower_case_ManageCarData.java” to the e-mail. Do not put you java file in a zip file, tar file, rar file, or any other archive file.

Below is the content of sample_data2.txt:

```
1       20      40      25
2       24      40      20
3       20      30      28
4       24      30      12
5       20      30      6
```

Below is the output for “java testCarData sample_data2.txt 700 1000 400 600”

```
carList
1       20      40      25.0    800.0 500.0
2       24      40      20.0    960.0 480.0
3       20      30      28.0    600.0 560.0
4       24      30      12.0    720.0 288.0
5       20      30      6.0     600.0 120.0
```

```
carListByTotalRange iterator
3	20	30	28.0	600.0 560.0
5	20	30	6.0	600.0 120.0
1	20	40	25.0	800.0 500.0
2	24	40	20.0	960.0 480.0
4	24	30	12.0	720.0 288.0

carListByTotalRange iterator local
3	20	30	28.0	600.0 560.0
5	20	30	6.0	600.0 120.0
1	20	40	25.0	800.0 500.0
2	24	40	20.0	960.0 480.0
4	24	30	12.0	720.0 288.0

carListByRemainingRange iterator
5	20	30	6.0	600.0 120.0
4	24	30	12.0	720.0 288.0
3	20	30	28.0	600.0 560.0
1	20	40	25.0	800.0 500.0
2	24	40	20.0	960.0 480.0

carListByRemainingRange iterator local
5	20	30	6.0	600.0 120.0
4	24	30	12.0	720.0 288.0
3	20	30	28.0	600.0 560.0
1	20	40	25.0	800.0 500.0
2	24	40	20.0	960.0 480.0

carListByTotalRange.poll().getTotalRangeInMiles() in [700.0,1000.0]
4	24	30	12.0	720.0 288.0 3	1	3
1	20	40	25.0	800.0 500.0 0	0	2	4
2	24	40	20.0	960.0 480.0 1	1	3

carListByRemainingRange.poll().getRemainingRangeInMiles() in [400.0,600.0]
2	24	40	20.0	960.0 480.0 1	1	3
1	20	40	25.0	800.0 500.0 0	0	2	4
3	20	30	28.0	600.0 560.0 2	0	2	4

carListByTotalRange iterator (if empty, you didn't refill carListByTotalRange after polling all of the elements)
3	20	30	28.0	600.0 560.0
5	20	30	6.0	600.0 120.0
1	20	40	25.0	800.0 500.0
2	24	40	20.0	960.0 480.0
4	24	30	12.0	720.0 288.0

carListByRemainingRange iterator (if empty, you didn't refill carListByRemainingRange after polling all of the
elements)
5       20      30      6.0     600.0 120.0
4       24      30      12.0    720.0 288.0
3       20      30      28.0    600.0 560.0
1       20      40      25.0    800.0 500.0
2         24        40        20.0      960.0    480.0
```