Coding Assignment Instructions
Chapter 10
Interfaces and ArrayLists

This assignment is coordinated with the demonstrations that I have recorded in the Chapter 10 Tutorial Demos. The Transportation Class Model has been revised to Version 5 (see attached class diagram). The following describes the current state of the Version 5 Transportation Class Model and code **changes that have already been completed**:

1. The *VehicleDataSource* interface has been defined to allow for multiple plug-compatible implementations.
2. The *ArrayBasedVehicleTestDataSource* class has been modified so that it implements the VehicleDataSourceInterface.
3. The *ArrayBasedVehicleTestDataSourceTest* class has been modified to test the implementation of the new interface. The test has been run successfully.
4. An *ArrayListBasedVehicleTestDataSource* class has been created that uses an ArrayList to store the test vehicles. Test vehicles have been constructed for Cars, Motorcycles, and Snowmobiles. You will need to modify this class to create test data for Boats.
5. An *ArrayListBasedVehicleTestDataSourceTest* class has been created to test the new data source implementation class. A test has run successfully. You will need to modify the test case in this class to properly test for the presence of Boat test data. The test will need to be rerun.
6. The *ResourceChargeable* interface has been defined so that it can mark the *Snowmobile* class. This interface has only one method, *determineResourceChargeAmount().*
7. The *Snowmobile* class has been modified so that it implements the *ResourceChargeable* interface.
8. The *SnowmobileTest* class has been modified so that it tests the *determineResourceChargeAmount()* method. The test has been rerun and it has passed.
9. The *AnnualFeeClient* class has been redesigned and renamed *BillingClient*. This new client is responsible for billing all amounts due for vehicles. It makes use of the *VehicleDataSource* interface. By toggling comments in the code, this class can make use of either of the two defined *VehicleDataSource* implementation classes. The billing code has been expanded to provide better formatting of the lines for billing amount and to provide for a Total Amount Due line for each vehicle. Code has been added to check if a Vehicle is an instance of *ResourceChargeable*. When a such a vehicle is found, a charge line is generated for the Resource Charge amount.

Use my posted copy of the Chapter 10 Demo project as your starting point. Create a Chapter 10 Coding Assignment Solutions project. Then, complete the implementation of the Version 5 Transportation Class Model. To complete this work, **you are expected to do the following**:

1. Modify the *ArrayListBasedTestDataSource* class so that it includes 5 instances of Boat in the test data.
2. Modify the *ArrayListBasedTestDataSourceTest* class so that the test case expects 5 test Boats to be included in the data. Retest.

3. Run the *BillingClient* class to demonstrate that it properly displays all test data including the 5 newly added Boats.

4. Create the .java file for the *LuxuryTaxable* interface. This interface has one method:

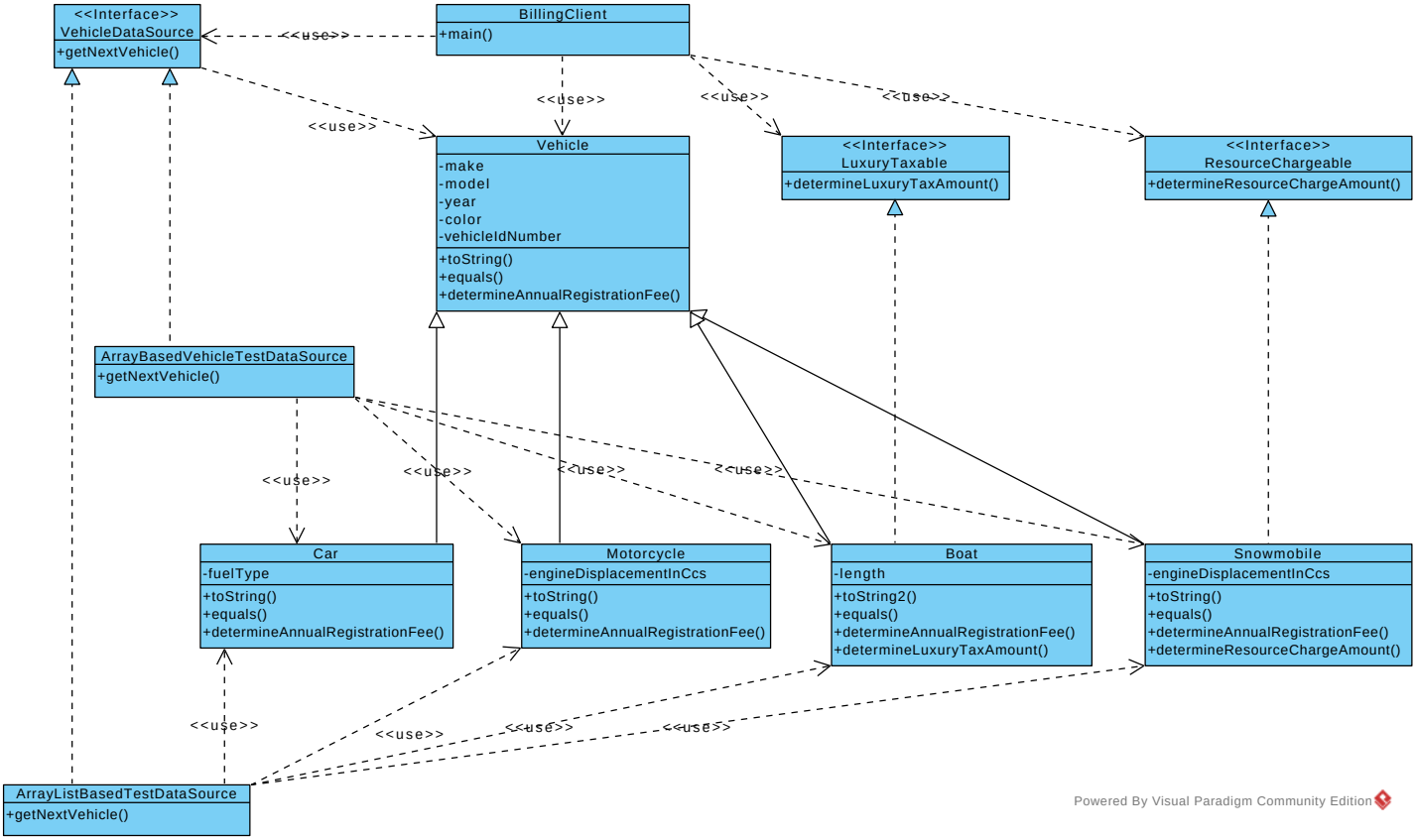
   public double determineLuxuryTaxAmount();

5. Modify the *Boat* class. Have the class implement the *LuxuryTaxable* interface. Provide an implementation of *determineLuxuryTaxAount()* that conforms to the following requirements:

Length in Feet	**Luxury Tax Amount**
< 30	0.00
30 to 39	100.00
40 to 49	200.00
50 to 69	500.00
> 69	750.00

6. Modify the *BoatTest* class to provide appropriate test cases for *determineLuxuryTaxAmount().* Make sure that a test case is provided on each side of tax amount boundary (30, 31, 39, 40, etc.). Retest.

7. Modify the *BillingClient* class so that it checks if each vehicle is an instance of *LuxuryTaxable*. For vehicles that qualify, call the determineLuxuryTaxAmount() method to retrieve the amount, add this amount to the Total Amount Due, and print a charge line. Do not print a luxury tax charge line if the luxury tax due is zero. When finished, run this class and inspect the output to confirm that the test Boats have been charged appropriately.


<<Interface>>
VehicleDataSource
+getNextVehicle()
BillingClient
+main()
<<use>>
Vehicle
-make
-model
-year
-color
-vehicleIdNumber
+toString()
+equals()
+determineAnnualRegistrationFee()
<<Interface>>
LuxuryTaxable
+determineLuxuryTaxAmount()
<<Interface>>
ResourceChargeable
+determineResourceChargeAmount()
ArrayBasedVehicleTestDataSource
+getNextVehicle()
Car
-fuelType
+toString()
+equals()
+determineAnnualRegistrationFee()
Motorcycle
-engineDisplacementInCcs
+toString()
+equals()
+determineAnnualRegistrationFee()
Boat
-length
+toString2()
+equals()
+determineAnnualRegistrationFee()
+determineLuxuryTaxAmount()
Snowmobile
-engineDisplacementInCcs
+toString()
+equals()
+determineAnnualRegistrationFee()
+determineResourceChargeAmount()
ArrayListBasedTestDataSource
+getNextVehicle()
Powered By Visual Paradigm Community Edition