Car Rental Billing Problem

Background

A car rental company provides the following three types of vehicles to its corporate customers:

- Sedan
- SUV
- 4x4

At the end of each month, a rental fee statement is sent to clients in plain text format. The company already has a program that computes and sends rental fees statements. However, many of the customers asked to receive the statement in JSON format instead of plain text because they want a format they can easily integrate with their accounting software. The detailed formulas of rental fee calculation are not well-documented, and the staff relies on the software to compute it!

Your manager, the engineering director of the company, called you into a meeting and gave you an overview of existing code that generates the monthly statement of each customer. She told you that she took a look at the "Customer" class and found it really ugly with many problems. She specifically told you that she noticed dead code, violations to the standard language naming conventions, lack of using modern language features, methods that are too long and not easy to understand, lack of proper usage of OOP, and lack of design patterns that could make this code easier to read and modify!
She also explained that this code is in ECMAScript 6, and designed to run both in Node.js, and through Browserify in a React based front-end, but she isolated the part you need to modify and created a simple build/test workflow for it in Node.js.

Your Assignment

The development director gave you the following two-part assignment:
1- Refactor/redesign this code so that:

- The code is easier to read and understand
- It becomes easy to introduce new output formats whenever needed
- It becomes easy to introduce new types of vehicles in the future

2- Develop the JSON format output, and make it possible to specify the output format per individual customer (JSON or plain text).

The code as you received it includes a characterization test written in Mocha/Chai. The director explained that this was written by a smart guy in QA, and MUST continue to pass as it guarantees that the design changes you're going to make do not break existing customer expectations. If you don't know what a characterization test is, read this:
https://en.wikipedia.org/wiki/Characterization_test

Note that you need to study this test to see how the system is instantiated and invoked, as there is no "main" entry point in the code and you need to rely on tests to exercise your changes. The test is also a great way to ensure you did not introduce regressions as you modify the code.

Artifacts Given to You:

The following is a list of the artifacts given to you as they appear in the archive "Car_Rental_Billing_JavaScript.zip":

File	Can I modify it?	Description and remarks
Customer.js	Yes	Customer class, this is where most of the complexity is. *Tip: start at "Statement" method.*
Vehicle.js	Yes	Vehicle Class
Rental.js	Yes	Rental Class, it represents a single rental of a specific vehicle.
Test/BillingTests.js	Yes	Put your unit tests here, this is handed over as an empty boilerplate.
Test/CharacterizationTest.js	Do not modify the asserts	While you might need to modify the test to change the way objects are created or setup, you should NEVER modify the assertions inside the test, otherwise there will be no guarantee that old behavior is intact!
test.sh	No	Compiles and tests your code on Mac/Linux
test.bat	No	Compiles and tests your code on Windows
Package.json	No	npm package.json file. You may decide to include other npm packages if needed through "npm install"

Our Expectations

We are expecting you to carry out both parts of your assignment as explained previously by the development director, and to upload "Car_Rental_Billing_JavaScript.zip" after completing the assignment to https:// tp.virgingates.com using the same user name and password you already have there. Note that you will need to change the archive name slightly as will be explained later below.

The workflow we think you would prefer is as follows:

1- Install or update Node.js on your computer as it is needed to run this assignment. You need a Node.js version that supports ECMAScript 6, so version 8 or higher is recommended.
2- Unpack the archive "Car_Rental_Billing_JavaScript.zip" in a folder on your hard disk.
3- Open a terminal/command line window in the folder where you unpacked "Car_Rental_Billing_JavaScript.zip", then run the following command while connected to the internet to install all needed npm packages for the assignment.

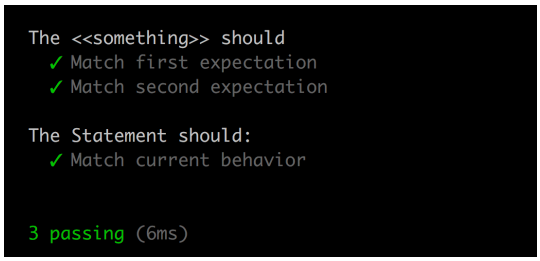
```
npm install
```

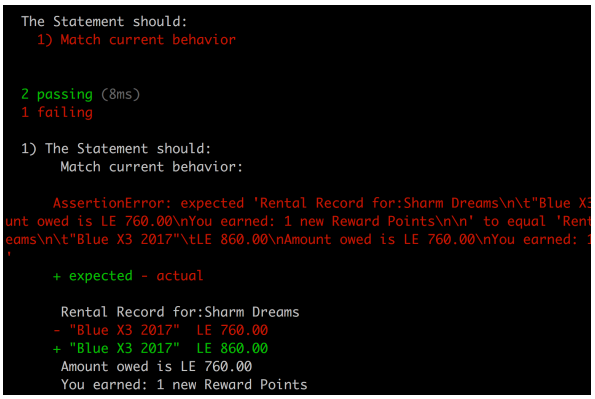
4- Run "test.sh" (Mac/Linux) or "test.bat" (Windows) and make sure it runs without failure.
5- Now you are ready to carry out you're assignment. Edit the files listed above, add new files if needed, while running "test.bat" or "test.sh" frequently to make sure you did not break anything.
6- When you're ready to upload your work, run "test.sh" or "test.bat" again to make sure all is well, then:.
7- Delete "node_modules" folder to reduce upload size

8- Pack all files in folder in a zip file and give it the name we mentioned in the email you received for this assignment. Make sure you do not pack any unnecessary files, and make sure the zip file is smaller that 1 MB.
9- Login to https://tp.virgingates.com, go to "Car Rental Billing Assignment - Javascript" and upload it, then click "End Test" to submit.

Tools

You are free to use your preferred Editor / IDE for development, or change the suggested workflow as you see fit. However, we will run your code only through running "test.sh" or "test.bat". An example output of "test.sh" when all is well, and when some tests fail are below side by side for your reference. Both "test.sh" and "test.bat" are simple scripts that invoke the Mocha test runner. We encourage you to take a look at them.
To know more about Mocha test runner or Chai assertion library, please visit their websites.





Note that by convention, you need to keep all test classes (if any) you develop in Test folder.

Evaluation

Your code will be assessed as follows:

- We make sure it runs through "test.sh" or "test.bat" without interpretation errors and without test failures.
- We make sure you did not change existing plain text output in any way.
- We evaluate the cleanliness of your code, in terms the size and tidiness of classes and methods, the quality of the names of your variables, classes, methods, the accuracy and usefulness of the comments if they are found, etc…
- We evaluate your design approach, how you used OOP to your advantage, how do you use patterns and abstraction and whether they add value or are just there to show off your knowledge.
- We evaluate your JSON representation for proper understanding of JSON nature and how it represents data elements.
- We evaluate the quality of the unit tests you introduced and their coverage.

Good luck!