Requirements

In this assignment you will continue to build a Car Configuration Application

I would like you to expand your proof of concept – so we will build API's for car configuration classes using interfaces and abstract classes and also add a custom exception handler to enhance your design.

For expanding proof of concept please consider the following requirements:

We will expand your existing design with these options:

- Define a set of methods in an interface (as API) that can be used to exercise the functionality of the existing class set.
- Create an exception handler so it handles at least 5 exceptions
- Enhance your design and code to create any abstract classes so your code is extensible and reusable.

Your Deliverable:

Design and code classes for these requirements and write a driver program to exercise your API and test the exception handler. Test your code adequately.

Concepts you will need to know.

- Object Theory
- Exception Handling
- Abstract Classes
- Interfaces

Plan of Attack

Refactoring

All class names should be declared as nouns. In the last unit we created Automotive that needs to be renamed to Automobile. Please complete this step before approaching this unit.

Step 1 – Writing API – Design and Construction

1. Analyze what you are being asked to do.
    a. You are being asked to build API for car configuration classes.
    b. Develop a custom exception handling mechanism to make your module more reliable.
2. From the information provided in the lab description following are the considerations for developing API.
    a. You will definitely need to use Interfaces.
    b. You have Model package (A component) that have Automobile class. Each Automobile class contains an instance of OptionSet and respective Options.
    c. In first unit you were asked to make the methods in OptionSet and Option class protected. Only methods of Automobile class are set as public.
    d. We are now trying to provide a set of methods that act as a Programming Interfaces for accessing functionality in Model package.
    e. Before we decide on how to create and implement the methods, let's read through and understand the meaning of a Java Interface. Please review the article at - http://docs.oracle.com/javase/tutorial/java/concepts/interface.html to learn the importance of Interfaces.
    f. Now let's decide on methods that should be exposed in the Interface.

```
1. public void BuildAuto(String filename);

//Given a text file name a method called BuildAuto can be
written to build an instance of Automobile. This method
does not have to return the Auto instance.

2. public void printAuto(String  Modelname);

//This function searches and prints the properties of a
given Automodel.

3. public void updateOptionSetName(String Modelname,
String OptionSetname, String newName);

//This function searches the Model for a given OptionSet
and sets the name of OptionSet to newName.

4. public void updateOptionPrice(String Modelname, String
```

```
Optionname, String Option, float newprice);

//This function searches the Model for a given OptionSet
and Option name, and sets the price to newPrice.

For our implementation we will implement these four
methods.
```

g. Few considerations for structuring the code:
    i. It is not important for us to expose the Automobile instance. We can create an Automobile instance and provide all the required functionality in the chosen API in last step.
    ii. So how do we organize the code so we can encapsulate Automobile.
        1. Create a new package called Adapter.
        2. Add two interfaces in package called Adapter:
            a. One called CreateAuto and add BuildAuto and printAuto methods in it.
            b. Another one called UpdateAuto and add updateOptionSetName and updateOptionPrice in it.
        3. Add a new abstract class called proxyAutomobile in Adapter package. This class will contain all the implementation of any method declared in the interface.

```
package Adapter;
import Model.*;
public abstract class proxyAutomobile {
     private Automobile a1;
}
```

This class contains an instance variable of type Automobile. Variable a1 can be used for handling all operations on Automobile as needed by the interfaces.

        4. So this raises a question – if we are defining all methods declared in interfaces in the abstract class proxyAutomobile then why is proxyAutomobile not implementing the interfaces. We will use inheritance for this purpose. We want proxyAutomobile to be a "hidden" containing all the functionality and expose an empty class for usage with Interfaces. In other words, we will add a new class called BuildAuto in Adapter package that will have no lines of code but will always look like this.

```
package Adapter;

public class BuildAuto
```

```
    extends proxyAutomobile implements
createAuto, updateAuto{

}
```

So whenever a new interface has to be added you can simply update BuildAuto declaration and write all methods in Abstract class called proxyAutomobile.

In a nutshell we have encapsulated access to Automobile instance from the API and also hidden the code (artificially) in the abstract class.

5. Next write a driver to test each of the methods.
    a. Are you able to create and print an Auto instance through CreateAuto interface?
    b. Are you able to update one of OptionSet's name or Option Price for the Auto instance created in previous step.
    c. Chances are that you will not be able to update as the object in proxyAutomobile is not declared as a "static" Object. In other words when you create an instance for BuildAuto (child of proxyAutomobile) a new Automobile is created. This requires variable a1 to be static so it can be shared between objects.

Step 2 - Defensive mechanisms to make software Self-Healing

Our next step to add a custom exception handler to deal with issues in runtime.

For doing this, you will first need to review the custom exception handling classes provided. After you have learned how to write a custom exception class, throw exception and how to catch and fix, you can implement it in this project.

Think of five possible exceptions to fix - Here are some possible example:

1. Missing price for Automobile in Text file.
2. Missing OptionSet data (or part of it)
3. Missing option data
4. Missing filename or wrong filename.

Your Exception class at a minimum should handle and fix at least one exception.

Submitting your work

Pl. review your work against this checklist before submission:

1. Program Specifications / Correctness
    a. No errors, program always works correctly and meets the specification(s).
    b. The code could be reused as a whole or each routine could be reused.
    c. Custom Exception Handler has been implemented with usage of five minimum exceptions.
    d. Abstract Classes constructs utilized to add extensibility
    e. Interfaces utilized to expose Model's (Auto, OptionSet and Option) functionality is meaningful(means the methods in interface are well thought out/useful in context of application).
2. Readability
    a. No errors, code is clean, understandable, and well-organized.
    b. Code has been packaged and authored based on Java Coding Standards.
    c. Text input file is readable and easy to follow.
3. Documentation
    a. The documentation is well written and clearly explains what the code is accomplishing and how.
    b. Class Diagram is provided.
4. Code Efficiency
    a. No errors, code uses the best approach in every case. The code is extremely efficient without sacrificing readability and understanding.
    b. Ability to correct exceptions is added in exception handling class. Ability to log exceptions is added.