Unit 2 – Part A

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

I would like you to expand you 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 designwith 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
* (Optional) Enhance your design and code to create any abstract classes so your code is extensible and reusable.

**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

Unit 2 - Part B

Your first step is to re-implement our current system, with the same functionality, but with code that's better designed to handle multiple models.   
  
Let's reorganize the code to make Automotive more generic so that it can represent cars in general. Automotive should define things that are true for all cars. It should not have anything specific to any particular car. For example all cars have a make, model, and base price, so Automotive should have that.   
Technical Requirement – Set of Models(Automotive) should be saved using LinkedHashMap. Set of OptionSet in each Model and respective Options can be saved in an ArrayList.

In addition, both Automotive and OptionSet will need some methods for keeping track of which options a user has chosen. To try and keep straight which methods are for defining options, and which are for choosing options, I've put Choice in the name of the new methods related to tracking user choices.  
  
Here's a UML class diagram for the updated Automotive:  
Automotive   
− \_basePrice: int  
− \_make: String  
− \_model: String  
− \_optionSets: LinkedHashMap  
+ Automotive  ()  
+ getMake(): String  
+ setMake(make: String): void  
+ getModel(): String  
+ setModel(model: String): void  
+ getName(): String  
+ getBasePrice(): int  
+ setBasePrice(price: int): void  
+ addOptionSet(setName: String, count: int): void  
+ getOptionSet(setName: String): OptionSet  
+ getOption(setName: String, String optionName): Option  
+ setOption(setName: String, index: int, optionName: String, price: int): void  
+ getOptionSetNamesIterator(): Iterator  
+ getOptionChoice(setName: String): String  
+ getOptionChoicePrice(setName: String): int  
+ setOptionChoice(setName: String, optionName: String): void  
+ getTotalPrice(): int  
  
Some of these methods should be obvious. Of the new ones, most are for defining and accessing options and option sets. For example,  
  
addOptionSet("transmission", 2);  
would create an option set to hold 2 transmission options. Then  
  
setOption("transmission", 0, "standard", -815);  
would define the name and price of the first option. Note that options are stored using zero-based indexes. You could then use getOptionSet() to get an option set, or getOptionChoicePrice() to get the price of an individual option.  
  
getOptionSetNamesIterator() is for code that needs to find all the options and option sets a car model has. getOptionSetNamesIterator() returns an iterator that can be used to iterate through the option set names that have been defined for a model. With that information, code can then get each option set, and hence each option in an option set. It's good modern design to return an iterator not the actual internal collection. This lets other code get the information it needs while still allowing us to change later how information is stored internally.  
  
The method setOptionChoice() is for choosing a particular option in an option set. E.g.,  
  
setOptionChoice("transmission", "standard");  
would choose the standard transmission option. After the above choice is set,   
  
getOptionChoice("transmission") would return "standard" and getOptionChoicePrice("transmission") would return -815.  
  
To make it easy to define the Automotive  methods for tracking option choices, add the following methods to OptionSet:  
OptionSet  
...  
...  
  
+ getOptionChoice(): Option  
  
+ setOptionChoice(optionName: String): void  
setOptionChoice(), given the name of an option, would save that choice inside the option set. getOptionChoice() would return the option chosen, if any, otherwise it should return null.  
  
To test this new Automotive class, write a driver program to read your text input file, populate an instance of Automotive class and print the OptionSet's and their respective options.   
  
Your application should work just as it did before, but will be much closer to being able to handle multiple models. (This means that you can still select an optionset with their respective options and calculate the car price, given user's selection.)  
  
Working with LinkedHashMap –   
Note that to access elements in LinkedHashMap you will need to associate an Iterator and then use it to find the elements in LinkedHashMap.