Java Project - Unit 1

Instructor Notes
In this project (over 4 units) you will build a Car Configuration Application. In this unit you will develop a “reference” base object model, read a text file to build the reference base object model and archive it using Serialization.

I would like you to start with a proof of concept – so we will first build the underlying object using normal Java Classes and Inner Classes (if it seems meaningful).

For our proof of concept please consider the following requirements:

We will build Ford's **Focus Wagon ZTW** model with these options:

- **Color** - Fort Knox Gold Clearcoat Metallic, Liquid Grey Clearcoat Metallic, Infra-Red Clearcoat, Grabber Green Clearcoat Metallic, Sangria Red Clearcoat Metallic, French Blue Clearcoat Metallic, Twilight Blue Clearcoat Metallic, CD Silver Clearcoat Metallic, Pitch Black Clearcoat, Cloud 9 White Clearcoat
- **Transmission -** automatic or manual
- **Brakes/Traction Control** - Standard, ABS, or ABS with Advance Trac
- **Side Impact Air Bags** - present or not present
- **Power Moonroof** - present or not present

Configuration options and cost data:

Base Price	$18,445
Color	No additional cost
Transmission	0 for automatic, $-815 for standard (this is a "negative option")
Brakes/Traction Control	$0 for standard, $400 for ABS, $1625 for ABS with Advance Trac
Side Impact Air Bags	$0 for none, $350 if selected
Power Moonroof	$0 for none, $595 if selected

Your Deliverable:
Design and code classes for these requirements and write a driver program to instantiate a Ford Wagon ZTW object and write it to a file. Test your code with a couple of instances of Forward Wagon ZTW.

Classes should be complete with functionality for Creating, Reading, Updating and Deleting information in each class.

Concepts you will need to know.
Object Theory
Inner Classes
File IO
Serialization

Design for Lab 1

The best practice in designing software is to make big changes in small steps. Your first step is to implement our current system, with the functionality mentioned above, but with code that's better designed to handle multiple models.

Not all cars have a moonroof option, or the same brake options, or the same colors. Because of this, the set of options for a car needs to be kept in a collection of some kind, rather than in individual instance variables. Because of its simplicity, we'll use a Java Array for now.

In order to support a more generic handling of options in `Automobile class` you will need to implement two classes:

- `Option`, a generic class to represent an option and its cost, e.g., ABS brakes for $400. Keep in mind that `Option` is an Inner Class of `OptionSet` – you should not be able to instantiate `Option` with creating an instance of `OptionsSet`.
- `OptionSet`, a generic class to represent one set of options for a car, e.g., a set of brake options

Here is the representation for `Option` Class. It's a very simple class that packages a name and a price.

Option

− _name: String
− _price: int
...
+ Option(String name, int price)
+ getName(): String
+ setName(name: String): void
+ getPrice(): int
+ setPrice(price: int): void

`OptionSet` is a bit more complex. The key attribute of `OptionSet` is that it contains an arrayList of `Option` objects:

OptionSet

− _name: String
− _options: Option[] – ArrayList
+ OptionSet()
+ OptionSet(String name)
+ OptionSet(String name, int count)
+ getName(): String
+ setName(s: String): void
+ getOptions(): Option[]

+ setOptions(options: Option[]): void
+ setOption(int i, String name, int price): void
+ getOption(name: String): Option
+ getOptionPrice(name: String): int
− findOption(name: String): int

A few of the methods need some explanation:

- `OptionSet(name, count)` constructs an `OptionSet` with space for `count` `Option` objects. `OptionSet` with no parameters or with only a `name` parameter constructs an `OptionSet` with `_options` left as null.
- `setOption(i, name, price)` sets the `i`th `Option` to the specified `name` and `price`.
- `findOption(name)`, if it's found, returns the index (i.e., the number) of the `Option` with the specified `name`; otherwise it returns -1. This method does most of the work for `getOption` and `getOptionPrice`. Note that `findOption` is private, since it is used only by other members of the class.

We're going to have to define all option using the two classes above. Essentially an OptionSet contains all options. You will need to create an **Automotive** class that has an instance of OptionSet.

Additional Requirements

To make our input more user-friendly, I would like you to create a text file for a car model including all of it options and read it using java.io API – instantiate Automotive object that contains OptionSet and write an instance to the disk (using Serialization). You should read the serialized object and print all attributes to show that properties of all objects are in fact written to disk.

Your deliverable for this part:

1. All coded classes
2. Test program showing the successful implantation of these classes
3. Class diagram