Project Description

In this project you will build a car configuration application in six units. Each unit provides learning opportunities in Object Oriented Design. You are expected to document these lessons and apply them in next unit. You will notice that the design guidance will taper off as you progress through units in Project 1. You will be expected to design on your own.

Project 1 - Unit 1

In this project 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 start with a proof of concept – so we will first build the underlying object using normal Java Classes and Inner Classes.

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.

Plan of Attack

Learning Objectives:

Object Theory
Inner Classes
File IO
Serialization

Step 1 – Analysis

1. Analyze data and create a set of classes.
2. From the data provided in project description you can try to find patterns in structuring information. You will notice that:
    a. Each model (Ford Wagon ZTW) definition has a set of properties (Color, Transmission etc.)
    b. Each property has a set of values (For color there are different values like Fort Knox Gold Clearcoat Metallic etc.)
    c. Each property has a price.
    d. Model also has a price.
    e. Values for each property tend to be different in quantity.

Step 2 – Designing Classes

1. Using the information in the last step (Analysis) you can start designing classes and how each might relate.
2. We will certainly not create classes for each property as that will not be reusable design.
3. We can create a **Model** class that can hold name of the model, base price and information about all the properties (Let's call it OptionSet)
4. Each **OptionSet** has a set of values. Each value can be defined in its own class (Let's call it **Option**). Each Option can have name and price as properties.
5. Each OptionSet will also have a name.

Step 3 – Define Class relationships for class Model package

1. We can see that Model will contain OptionSet and Each OptionSet will contain Option class.
2. *For practice and learning purposes be sure to make Option class an Inner class of OptionSet.*
3. The structure at a high level might look like this:

```
class Automotive { //This class will represent the Model.
 String name;
 OptionSet opset[] ;
 Model(int size, String n)
 {
       opset = new OptionSet[size]; name = n;
 }
}
class OptionSet {
  Option opt [];
  String name;
  OptionSet(String n, int size)
  {
       opt = new Option[size]; name = n;
  }
}
```

```
}
class Option {
  String name;
  float price;
}
```

4. In the above code you will notice that Each model has a set of OptionSets. Each OptionSet has a set of Options.
5. For each Array object (opset and opt) you will need to instantiate objects to avoid NullPointerException.

```
for(int i=0;i<opt.length;i++)
 optset[i] = new OptionSet();
for(int i=0;i<opt.length;i++)
 opt[i] = new Option();
```

Step 4 – Planning detailed design of Model package

1. For each class (Automotive, OptionSet and Option) make sure you have applied the following criteria to ensure the class definition is complete and properly coded.
   1. Add constructors for every possible meaningful combination.
   2. Write get and set methods for each property.

      For e.g. if OptionSet class has Option array you should write get methods to get one or more Option values.

      Tip:

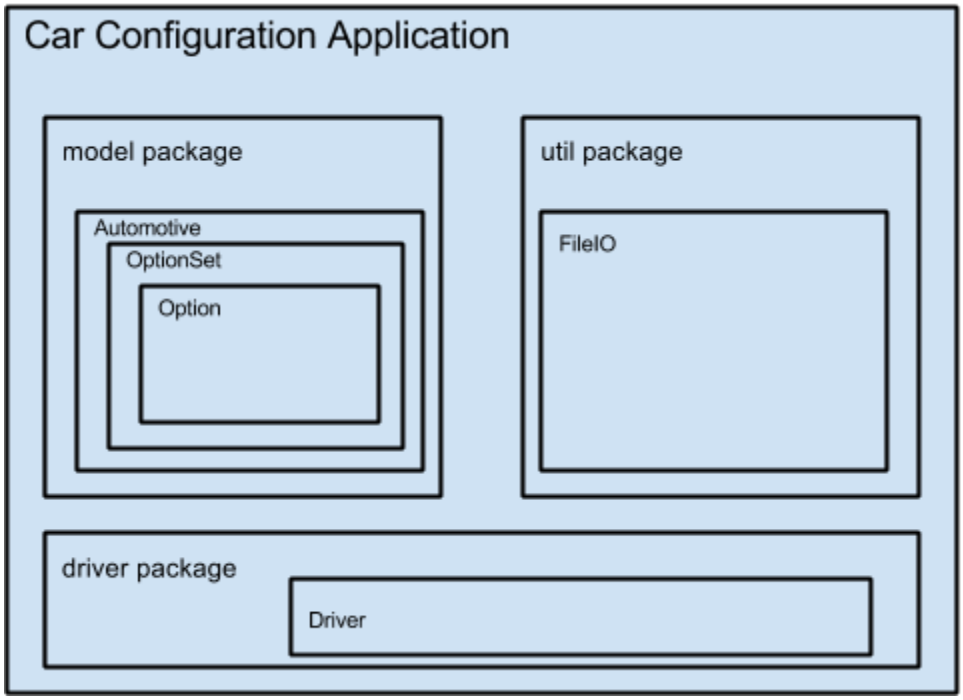
      Be careful that auto generation options in IDE's might not suffice. Additional work will be needed.
   3. For each property in a class make sure you have added operations for Creating (generally done with constructors), Reading (Generally done with find functions and/or getters), Updating (Update functions that will call find functions, Deleting values(Delete value of a property).
   4. As a result of above work your Automotive class structure might look like this:

```
Automotive class
Properties
 name
 baseprice
 OptionSet []

Constructor (create)
Method categories for all the above stated properties.
    Getter (read)
    Find (read)
    Setter (Inserting values(
    Update.
    Delete - deleting an option (1 at a time) or optionset (1 at a time).
```

   5. A more detailed analysis on each category produces following possibilities for each category.
      - Automotive class
        i. name
        ii. baseprice
        iii. OptionSet opset [];
      - Constructor
        i. Automotive()
        ii. Automotive(name, baseprice, OptionSetsize)
      - Getter
        i. Write methods for following criteria:
           1. Get Name of Automotive
           2. Get Automotive Base Price
           3. Get OptionSet (by index value)
      - Find
        i. Find OptionSet with name
        ii. Find Option with name (in context of OptionSet)

- Setter
  i. SetName
  ii. Set Base Price
  iii. Set values of OptionSet
  iv. Set values of Option (in context of OptionSet)
- Delete
  i. Delete an Option
  ii. Delete an OptionSet
- Update (Find and Set)
  i. Update values of OptionSet
  ii. Update values of Options



Step 5 – Constructing Model

1. After all classes have been design you can start doing construction.
2. When writing your code factor in these elements:
   1. Add Automotive, OptionSet and Option class in a package called Model.
   2. Make sure you only have one class per java file.
   3. Make sure the methods in Option and OptionSet class are protected and properties are private. Methods in Automotive class will be public and properties will be private.
   4. You should also follow these coding conventions:
   5. Each class should have grouping of elements as follows:
      a. instance variables
      b. static variables
      c. constructors
      d. getter/setters
      e. instance methods
      f. static methods
   6. All class names start with upper case letter
   7. Every class should be able to print its property through a print() or a toString() method. Do not use String concatenation in toString method. Use StringBuffer/StringBuilder
   8. Do not use any static methods in the Model package.
   9. Do not use ArrayList or any other Collections in this submission.

Step 6 – Populating data in Model Package using a text file.

Getting the data into Model Package

1. Create a text file that contains all the properties and value that are described in requirements. The format of the file is to be created by you. *TA's or Instructor WILL NOT HELP YOU with the structure of this file.*
2. Your file format should use the following criteria when creating a file -
    a. You can make this engineer friendly or user friendly.
    b. When reading file you should do a single scan and populate the automotive object.
    c. Do not create or use a random file or XML file.
    d. When reading file you cannot use a buffer or any data structure like array list to save data temporarily in memory.
3. After creating a file, let's think about where to put the code for reading data from a file to populated objects created from classes in Model package. You should not put anything related to files in Model package. We should do this in package called Util. Doing this will help separate code organization according to class responsibilities.
4. When creating a class/method for reading data from a text file you should:
    a. Put the class in a package called Util.
    b. Use methods from Automotive class ONLY and not use any methods from OptionSet and Option class.
    c. Make sure there is no reference to code related to file IO in Model package.
    d. Method that is responsible for reading data from a file and building an Automotive should be an instance method. This method should not be declared as a static method.
    a. public Automotive buildAutoObject(String filename) { ...
    e. You need to make a decision on the formal parameter(s) and return value of buildAutoObject method. Instructor or TA's will not help you with this decision. This decision will have consequences in the future modules.
        i. Would you:
            1. Pass an instance of Automotive.
                a. `public void buildAutoObject(String filename, Automotive a1)`
            2. Return an instance of Automotive
                a. `public Automotive buildAutoObject(String filename)`
            3. Do both.
                a. `public Automotive buildAutoObject(String filename, Automotive a1)`
    f. You must use FileReader and BufferedReader classes for working with text files.

TIP - How to read data from a text file?

```
import java.io.*;

public class ReadSource {
    public static void main(String[] arguments) {
        try {
            FileReader file = new
                FileReader("ReadSource.java");
            BufferedReader buff = new
                BufferedReader(file);
            boolean eof = false;
            while (!eof) {
                String line = buff.readLine();
                if (line == null)
                   eof = true;
                else
                    System.out.println(line);
            }
```

```
                buff.close();
            } catch (IOException e) {
                System.out.println("Error -- " + e.toString());
            }
        }
    }
```

Step 7 – Learn Serialization and implement it (useful for testing)

1. Once you have read a file and built an instance of Automotive class you should archive the instance using Serialization. To do this you will need to decide which classes should implement the Serializable interface.
2. After the file has been written to disk be sure to read it back and print the properties to console
3. When working with serialization you should write one object per file. To write > 1 object use an array or a collection.

TIP - How to write and read a serialized file?

```
class A implements Serializable {… }
class B extends A {...}
class ObjectFileTest
{  public static void main(String[] args)
   {  try
      {
         A [] staff = new A[3];
         staff[0] = new A();
         staff[1] = new B();
         staff[2] = new A();

         ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("A.dat"));
         out.writeObject(staff);
         out.close();

        ObjectInputStream in =  new ObjectInputStream(new FileInputStream("A.dat"));
        A[] newStaff = (A[]) in.readObject();
      }
      catch(Exception e)
      {  System.out.print("Error: " + e);
         System.exit(1);
      }
   }
}
```

Step 8 – Test and Submit

1. Use the following driver (or something similar) for testing entire project.

```
class Driver
{
       public static void main(String [] args)
       {
              //Build Automobile Object from a file.
              Automobile FordZTW = (Some instance method in a class of Util
package).readFile("FordZTW.txt");
              //Print attributes before serialization
              FordZTW.print();
              //Serialize the object
              Lab1.autoutil.FileIO.serializeAuto(FordZTW);
              //Deserialize the object and read it into memory.
              Automobile newFordZTW = Lab1.autoutil.FileIO.DeserializeAuto("auto.ser");
              //Print new attributes.
              newFordZTW.print();
```

```
        }
}
```

2. Turn in your submission using the criteria specified in syllabus.

Submitting your work

- Please review your work using the following Grading criteria.
- Program Specification / Correctness
    - No errors, program always works correctly and meets the specification(s)
    - The code could be reused as a whole or each routine could be reused.
    - Classes for Model and FileIO are in separate packages. Model has Auto, OptionSet and Option Class.
    - Java coding conventions are followed.
    - Automobile class completed with access method for all CRUD operations for Option, OptionSet and Automotive class. Properties Option and OptionSet class are protected. Data is populated into Automotive instance using a text file. Serialization and Deserialization is demonstrated. Collections like ArrayList are not applied.
- Readability
    - No errors, code is readable, and well-organized.
    - Code has been packaged and authored based on Java Coding Standards.
    - Text input file is readable and easy to follow.
    - A single text file is used for data input.
    - Code for file IO class should be placed in Util folder.
- Documentation
    - The documentation is well written and clearly explains what the code is accomplishing. Class Diagram is provided.
- Code Efficiency
    - No errors, code uses the best approach in every case. The code is extremely efficient without sacrificing readability and understanding.
    - FileIO routines are not embedded in AutoModel including instantiation of FileReader Objects in AutoModel.
- Assignment Specification
    - No errors. Text file is read to build a Automodel. Automodel is serialized to a file, read back and then properties printed.