# Unit 4 - Building Client-Server model

## Reflection Questions:

You should review the following questions to make sure you understand the outcomes from previous unit and potentially document lessons learned for submission (with final unit of Car Configuration Application). You do not submit these questions for grading.

1. What is the best way to setup multithreading in an Enterprise Class application?
2. What strategy is used for synchronizing so you end up with a scalable application?
3. What implementation strategy can be used for creating a race condition for testing Multithreading?
4. How does Synchronization work in JVM? What are the performance consequences of Synchronizing?

## Requirements

In this assignment you are going to build on previous unit and create a client server application using java.net API.

Server will have the following capabilities:

1. Host all Automobiles in a single data structure - LinkedHashMap.
2. Receive a Properties object from client parses it and builds an Automobile Object on Server.
    a. *Properties object contains Optionsets (including Options) for one Automobile (~ to the contents of Textfile from Assignment 1). It does not contain any car configuration information.*
3. Respond to client request for configuring a car by passing an instance of Automobile object. The object would be serialized to the client.

Client will have the following capabilities:

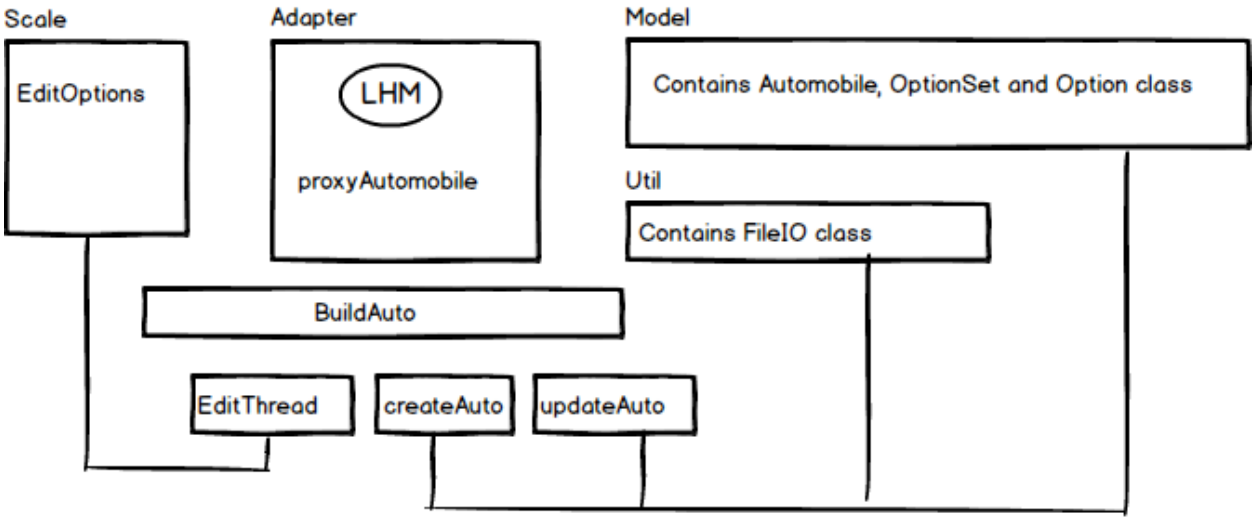
1. Provide a menu of following services when client starts:
    a. Upload a Properties file.
    b. Configure a car.
2. For 1a - A user will be prompted to provide a path to Properties file. Once the path is provided, the client will load the Properties Object using Serialization and send to Server.
    a. *What will the server do in response to this step? Server will parse the Properties file and create an Automobile Object. This object will be added to the static LinkedHashMap automobile object in proxyAutomobile class.*
3. For 1b - A user will be prompted with a list of available models on the Server. A client can select an Automobile and configure a car.
    a. What will the server do in response to this step? First, server will provide a list of available Automobiles. When user has selected an Automobile, then server will serialize that instance to the client. Client can then use that instance and configure the car on the client. Keep in mind, there should be no back and forth communication between client and server for car configuration. The

configured car information only has to be printed to the screen and does not need to be saved on the server or the client.

4. Server in this context only acts as an in-memory storage repository for Automobiles.

Plan of Attack

Let's start with a  recap of what we have built so far:




The above context diagram represents what you have done by end of Unit43.
You have four packages:

1. Model containing Automobile, OptionSet and Option class.
2. Util containing some FileIO class.
3. Adapter containing proxyAutomobile, BuildAuto, createAuto and updateAuto.
4. Scale with EditOptions and Editthread interface.

createAuto and updateAuto are external API's meaning that if we are KBB.com - we would give this to our clients like Toyota, Ford to upload the models and update.
ediitThread is an internal API that can be used by KBB.com for executing operations.

TIP - Pl. keep in mind that external and internal API has nothing to do with public, protected or private scope in Java. It merely represents the exposure of methods from a business perspective.

By creating these separations, it is likely we may have similar operations in both (external and internal) but implemented differently.

Let's start design for Unit 4.

## Step 1:

### Enable the usage of Properties file.

Add a new function in file io class in Util package in which you will parse a Properties file.

How to read data from a  text file using a Properties Object?
Here is a how properties file can be created. and read.

**Sample Properties file.**

```
CarModel=Prius
CarMake=Toyota
Option1=Transmission
OptionValue1a=Manual
OptionValue1b=Automatic
Option2=Brakes
OptionValue2a=Regular
OptionValue2b=ABS
```

Here is the code for reading a properties file and reads into some local variables.

```
      Properties props= new Properties(); //
      FileInputStream in = new FileInputStream(filename);
      props.load(in); //This loads the entire file in memory.

      String CarMake = props.getProperty("CarMake"); //this is how you read a
property. It is like gettting a value from HashTable.
      if(!CarMake.equals(null))
      {
            String CarModel = props.getProperty("CarModel");
            String Option1 = props.getProperty("Option1");
            String OptionValue1a = props.getProperty("OptionValue1a");
            ……
      }
```

Enable access to this function using the existing create function for Automobile in createAuto function. You should add an additional parameter called fileType in this function.

## Step 2:

Before working on this step make sure you have learned the creation of Client Server interaction by reviewing KnokKnockServer example I covered in class. (Also uploaded on class site).

## Create a Server

Create a package called ***server*** that contains the following:

1. Create a new interface called AutoServer implemented by BuildAuto. You will need to add methods in the Server interface based on functionality depicted below.
2. Add a class BuildCarModelOptions class that has methods to do the following:
    a. Accept a properties object from client socket over a ObjectStream and create an Automobile. Then add that created Automobile to the LinkedHashMap. This method will be declared in the AutoServer interface. AutoServer interface should be implemented in BuildAuto and BuildCarModelOptions class. Based on the current structure, this method will be implemented in proxyAutomobile class and called in a method of BuildCarModelOptions

3. Setup a Java ServerSocket that will run an instance of BuildCarModelOption class to build Automobile using the Properties file.

## Step 3:

## Create a Client

Create a package called **client** that will contain the following:

1. Design and create a class called CarModelOptionsIO that can:
    a. Read data from a Properties file, create a properties object using the load method an transfers the object from the client to server using Object Streams.
    b. Receive a response from the Server verifying that the Car Model object is created successfully.
    c. Use CreateAuto interface to build Automobile. To handle different type of files you might consider passing in filetype.

Test the client and server interaction. Make sure that properties file created in client is uploaded, parsed in server and then added in LinkHashMap object.
You should upload multiple models for testing purposes.

Please be sure to incorporate usage of DefaultSocketClient class in both Server and client implementation. The code for this class can be borrowed from the powerpoint deck on Socket Programming.

## Step 4:

## Enhancing the client:

In the **client** package design and create a class called SelectCarOption that can:

1. Prompt the user of available models
    a. Tip: Use Socket Class to interact with Server to find the available models.
2. Allow user to select a model and enter its respective options
3. Display the selected options for a class

As part of this enhancement you will need to add new methods in **AutoServer** interface to:

1. Provide client a list of available models.
2. When a user selects an Automobile then server should send the object (using Serialization) to client.

TIP - You should make two projects (one for client and another for server) that each contain the following packages - model, adapter, scale and util. Additionally project for server will contain a **server** package. and project for client will contain the **client** package.

**Your deliverables for this project:**

1. Updated classes
2. Class Diagram
3. Test program showing the successful implementation of these classes

# Tips

## How to use DefaultSocket Client class in Server class?

1. Building a Server:

You can pretty much follow the KnockKnockServer example for building your server.
Write a constructor that instantiates ServerSocket class.

```
ServerSocket serverSocket = null;
 try {
     serverSocket = new ServerSocket(4444);
 } catch (IOException e) {
     System.err.println("Could not listen on port: 4444.");
     System.exit(1);
 }
```

2. Then, you need to write a method that can take an incoming request and process it in its own thread.
For this you can use DefaultSocketClient class

```
DefaultSocketClient clientSocket = null;
 try {
     clientSocket = serverSocket.accept();
 } catch (IOException e) {
     System.err.println("Accept failed.");
     System.exit(1);
 }
```

You will need to setup (create references) stream type in DefaultSocketClient class that are needed. You will be using ObjectStreams. This can be best done as follows:

3. Like in KnockKnockServer (example below) you will need to link to the stream objects of the incoming requests.

```
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
                        new InputStreamReader(
                        clientSocket.getInputStream()));
```

4. Next, you will need to override handleSession method in DefaultSocketClient class to handle the incoming request.

## How to use DefaultSocketClient class for client?

You can simply modify DefaultSocketClient class and add the necessary Object Streams for communication.

## Grading your submission

## Total points 40

1. Program Specification/Correctness (25 points)
    a. Class diagram is provided.
    b. No errors, program always works correctly and meets the specification(s).
    c. The code could be reused as a whole or each routine could be reused.
    d. Packages are created as described in the Plan of Attack.
    e. Exposure to LinkedHashMap object is done only through interfaces.
    f. BuildAuto class has no methods defined in it.
    g. Client and Server classes used Object streams only.
    h. Server is not storing any information about class configuration.
    i. DefaultSocketClient class is used in Server implementation. DefaultSocketClient class has to be modified to create a proxy (return value from accept() method) inside the implemented server.
    j. DefaultSocketClient class is used in Client implementation.

2. Readibility(5)
    a. No errors, code is clean, understandable, and well-organized.
    b. Code has been packaged and authored based on Java Coding Standards.

3. Documentation(5)
    a. The documentation is well written and clearly explains what the code is accomplishing and how.
    b. Detailed class diagram is provided.

4. Code Efficiency(10)
    a. No errors, code uses the best approach in every case. The code is extremely efficient without sacrificing readability and understanding.