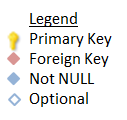
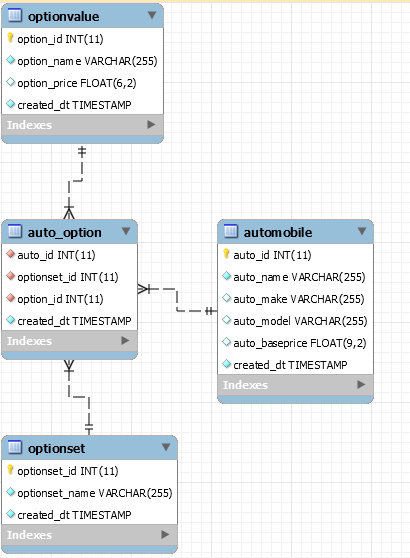
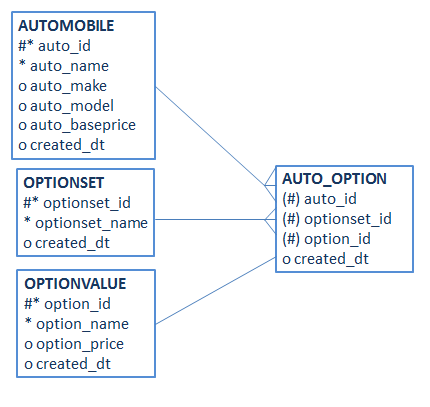
**Project 1 Unit 6**

In this assignment, we built upon the previous one by incorporating MySQL database and persisting our data store. Below is the database schema design represented by an ER Diagram and a crow notation diagram:

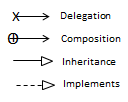
**DB ER Diagram:**



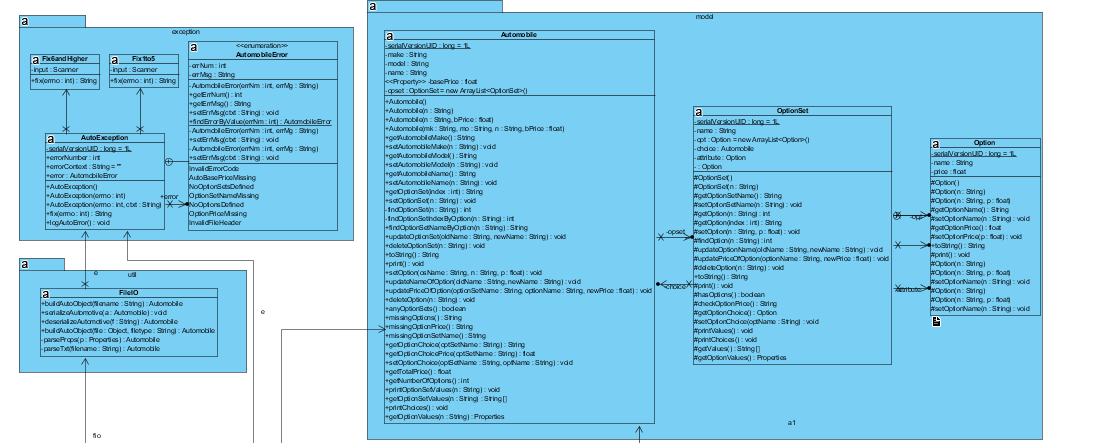
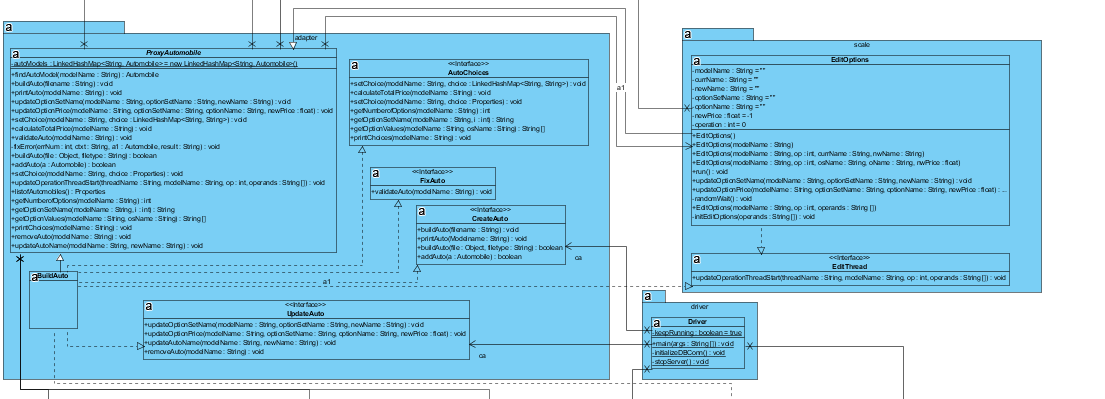
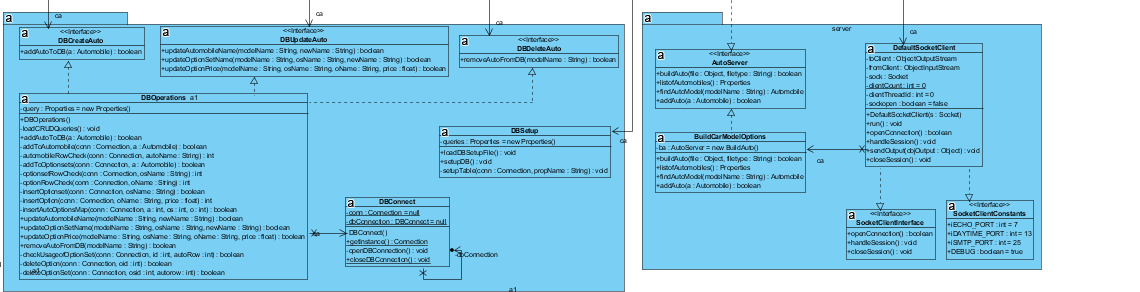
**DB Crow Notation:**



High level UML diagram for Unit 6 has been provided below (details of individual zoomed-in boxes provided separately later in the document):

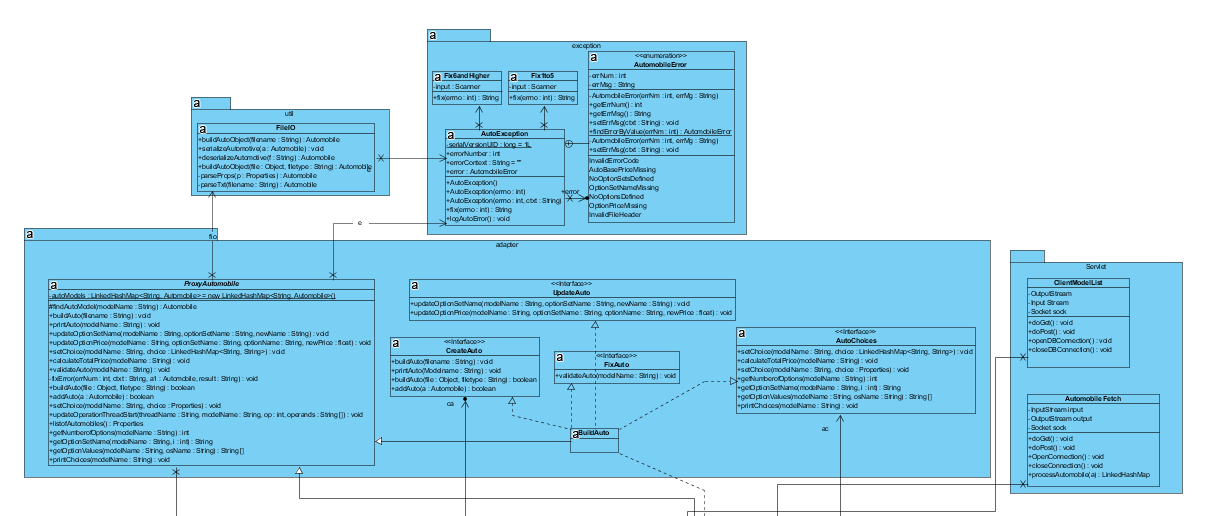


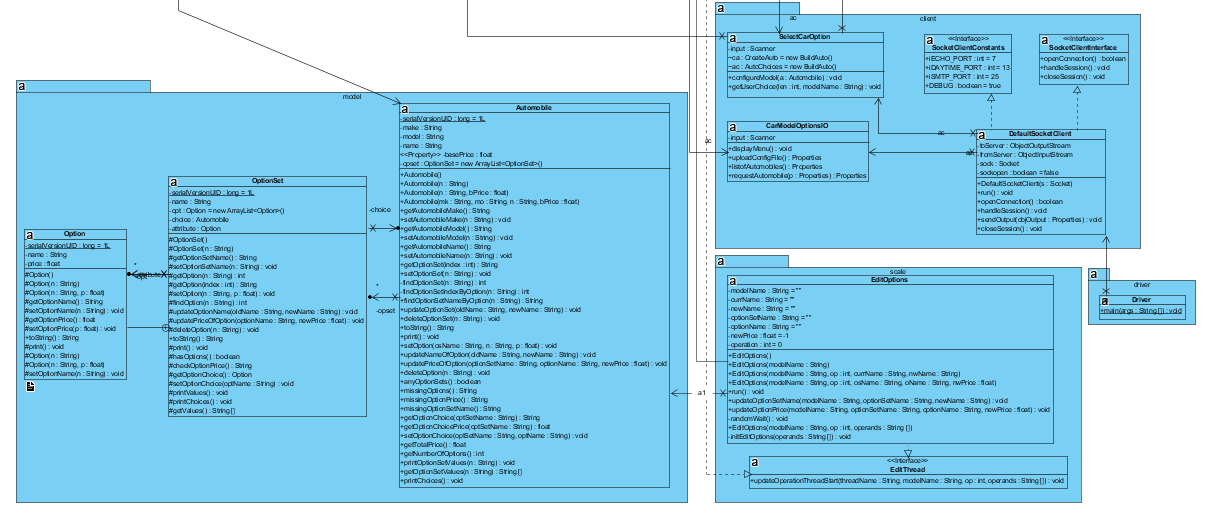
**Server Project (added DB Package in Unit 6):**

**Client Project (Resides on WebServer as ROOT folder):**

* Same as Unit 5 (No Changes made in unit 6).





In Project 1 Unit 6, we incorporated MySQL Database into our project as described below:

**Client:**

The client has not changed since Unit 5. It remains the same allowing the user to upload a car config file through the client console to setup the automobile object on the server. The following files can be used to setup the Automobile from the client side:

* FocusWagonZTW.prop
* ToyotaCamry.prop

**Server:**

On the server side, new files have been added in the Database package and a few changes have been made in the ProxyAuto class to use the DB package. ProxyAuto executes all CRUD operations through the following three interfaces: DBCreateAuto, DBUpdateAuto and DBDeleteAuto.

**Setting up the Database (on Server start):**

A method called initializeDBConn has been added in the driver class to register the MySQL DB driver (com.mysql.jdbc.driver in lib folder) with the system. In this method the DBSetup class from the database package is instantiated and its methods loadDBSetupFile() and setupTable() are called to load the DBConfig and DBSetup properties file with all the schema and table creation queries and then connect and run these queries to create the schema and tables if they do not already exist.

**DB Schema:** The auto schema has been setup with 4 tables as demonstrated in the ER diagram. The unique row ids of each table (automobile, optionset and optionvalue) have been associated with each other in the bridge table auto\_options. This schema design means that optionsets will be shared across multiple automobiles and when an optionset name is changed it will affect all the automobiles that use that optionset. This is not entirely good practice but it depends on the business use case and what is needed by the business. The option is unique for each automobile as the price for the same option may vary across automobile models.

**Connecting to DB to execute DDL and DML:**

Typically DB connections would be handled by a connection pool managed by the server. However, for the purposes of this project, a singleton class called DBConnect has been created that publishes two static methods: getInstance() and closeDBConnection(). Before executing CRUD operations, the methods get the DB connection instance by calling DBConnect.getInstance(). The getInstance() method calls the private constructor which in turn calls a private openDBConnection() method if a connection doesn’t already exist. At the end of the CRUD operation, the using class calls the closeDBConnection method to close the connection. The openDBConnection method loads the connection details of the DB from a properties file called DBConfig. DB password is setup in this file.

**DB CRUD Operations:**

All DB CRUD operations are handled by the DBOperations class. This class contains methods that perform the following actions by executing SQL queries:

* Setup a new Automobile object in the Database (affects 4 DB tables)
* Update Automobile name (affects one table)
* Update OptionSet name (affects one table)
* Update Option price (affects one table)
* Delete an automobile (affects 4 DB tables)

Each of the methods in this class that perform these actions have been modularized in several smaller helper methods as before making these updates there has to be many select statements to check the current state of the DB regarding the automobile. So my modularizing, reusability was enabled.

Also each transaction affects several tables therefore, each transaction after obtaining a DBConnection starts with turning-off auto-commit and only committing at the end of the transaction once all sub-tasks have been executed successfully. If an error occurs, then the transaction is rolled-back in its entirety.

The SQL statements for usage in the DBOperations class are loaded from a properties file on class instantiation. The variables in the SQL statement are substituted based on method parameters. This allows for maintainability and reuse.

**Interfaces to execute DB Operations:**

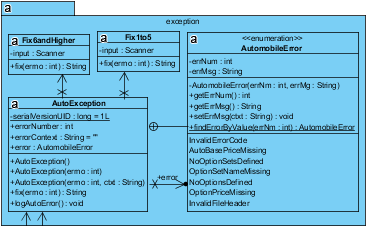
The DBOperations class which contains all the implementation for DB CRUD operations is insulated from the rest of the system by exposing the operations through three interfaces: DBCreateAuto, DBUpdateAuto and DBDeleteAuto. These interfaces are used by ProxyAuto class to execute DB operations everytime the linkedhashmap or automobiles in the linkedhashmap are being updated.

**Project Ouput Explained:**

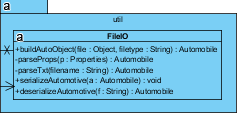
See the other file provided “OutputProject1Unit6” to see screenshots of expected output in the database. Testing has been done for two Automobile objects Focus Wagon ZTW and Camry XLE. In the first case, the automobile has been setup (FocusWagonZTW.txt), updated and deleted using the driver class on the server side. In the second case, the client console has been used to upload a file (ToyotaCamry.prop), build an auto and persist it in the DB on the server side. Hence, persistence of an automobile object in the DB has been demonstrated through both the server side and client side. DB password has been set as “Keshav”, this can be changed in the DBConfig.prop file in the server folder.

Next few sections, just provide expanded view of each package in the UML diagram.

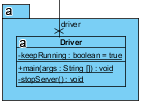
**Exception Package:**



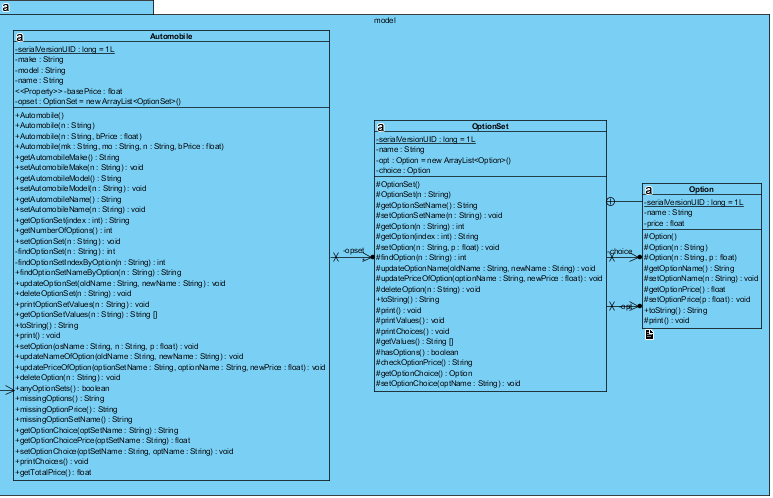
**FileIO Package:**



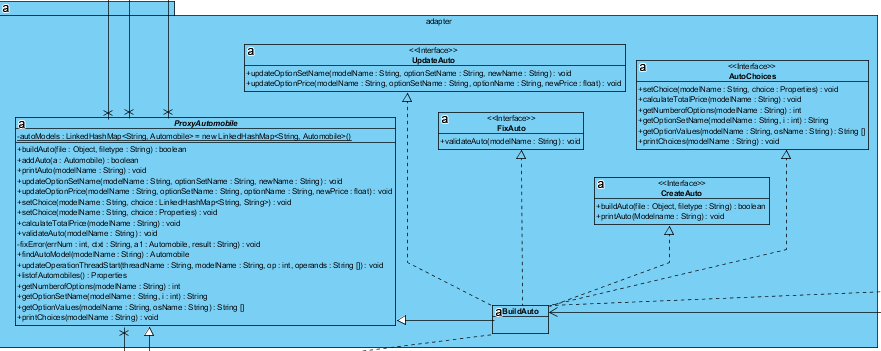
**Driver Server/ Client Package:**

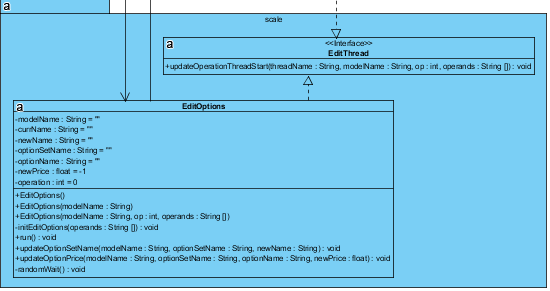
**Model Package:**



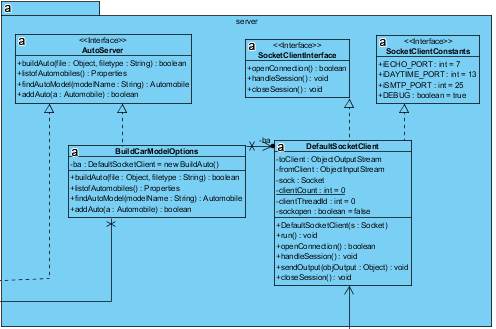
**Adapter Package:**



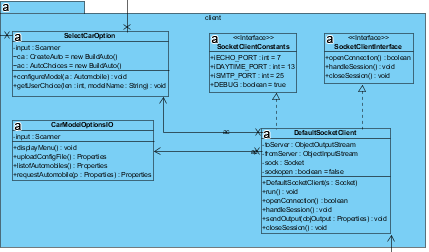
**Scale Package:**



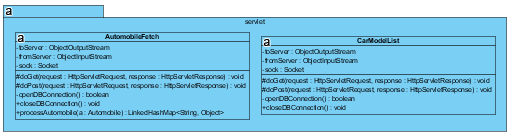
**Server Package:**



**Client Package:**



**Servlet Package:**



**Database Package:**

