Using the Code

```
01  FSJVehModelsTable vehiclesTable = new FSJVehModelsTable(vehiclesFile);
02  FSJVehicleModelParameters vehModel = vehiclesTable.getVehicleModel(idInTable);
03
04  FASTSimJ2 fsim = new FASTSimJ2(new FSJSimConstants());
05  fsim.runC(              //Compact Run
06          vehModel,       //Vehicle Model Parameters
07          timeSec,        //Array of Time Values (sec) [optional]
08          speedMPH,       //Array of Vehicle Speed Values (mph)
09          roadGrade,      //Array of Road Grade Values [optional]
10          auxKW,          //Array of Auxiliary Power Demand (kW) [optional]
11          initRelSoC);    //Relative Initial Battery State of Charge
12
13  System.out.println("Trip miles    = "+fsim.lastSimSummary().miles);
14  System.out.println("Miles per Gal = "+fsim.lastSimSummary().mpg());
15  System.out.println("kWh per Mile  = "+fsim.lastSimSummary().kwhpm());
```

Three Main Steps:

1. Set the vehicle model parameters. This is done in line 1, where a FSJVehModelsTable class is used for loading a number of vehicle models from a file on disk (referred to by its folder and full file name including extension in the String variable vehiclesFile), and then selecting one of the vehicle models (line 2) from among the models that have been loaded.
2. Initializing the main simulations class FASTSimJ2 with simulation constants (as per line 3). If some of the simulation constants (e.g. air density) need to be changed from default values, the class FSJSimConstants can be instantiated separately, and the desired constant adjusted.
3. Invoking one of the fuel economy simulations functions (line 5-11). The default function for that is the compact memory management run function runC(), which takes six inputs:
    - Data object for the vehicle model parameters (which had been set in lines 1 and 2).
    - Optional entry of an array of the time values for speed record (next entry in the function) of the vehicle. If this variable is null, then the code treats all speed entry values as being at exactly one second interval apart.
    - Array of vehicle speed values [mph] for the trip to be simulated.
    - Optional entry of an array of the road grade values. If this variable is null, then the code assumes the simulated trip is happening on a flat terrain (road grade of zero).
    - Optional entry of an array of additional auxiliary power values. While the vehicle model parameters already include an entry for a base value of auxiliary power (which is constant throughout every trip), this input variable allows for time-varying auxiliary power such as due to initial warm-up HVAC load during cold climate. If this variable is null, then the code assumes that there are no additional auxiliary loads aside from the constant value in the vehicle modeling parameters.
    - Initial state of charge (as a relative value between 1 for fully charged to 0 for empty). This value only affects electrified powertrain. Optionally, an entry of -1 can be input in this variable in order to invoke a specific default for different powertrain types. For BEVs and PHEVs, the default is that the state of charge continues from the last value that had been reached in a previous trip. For HEVs however, the default value is to conduct a search that seeks to estimate the fuel-only equivalent simulation for the trip (i.e. an estimate of the HEV fuel economy when there is zero net electric energy consumption in the trip)

Listing of Main Classes

Programmer's Interface

`FSJVehicleModelParameters`	Data class for holding FASTSim vehicle modeling parameters (same 54 data values as in FASTSim-Excel)
`FSJVehModelsTable`	Class for reading a set of vehicle models from a text file in comma separated format (CSV). Once the file is read, vehicle modeling parameters for a vehicle models can be recalled by the vehicle ID in the file.
`FSJSimConstants`	Data class for holding default constants (e.g. air density, energy content in Gasoline, etc.)
`FASTSimJ2`	Main simulation class that performs the fuel economy simulations.

Component Models

`VehicleState`	Data class that holds details of the vehicle state (speed, engine-on/off, various power consumption terms.) at some time instant. Also includes code modules to advance the state by one time step increment to a new desired speed.
`FuelConverterModel`	Data and code modules for generic models for an ICE
`ElectMotorModel`	Data and code modules for generic model of an electric motor.

Type Enumerations

`VehicleDriveTrainType`	Powertrain type {CV, HEV, PHEV or BEV}
`FuelConverterEffType`	Type of ICE {Spark-ignition, Atkins or Diesel
`EnergyMgmtControlStrategy`	Generic model for engine on/off management in HEVs & PHEVs

Example Implementations

`FASTSimJ2GUI`	Windows-based graphical user interfaces that utilizes FASTSim-Java
`FASTSimJ2Run`	Example of an entry point *main()* function that launches a graphical interface or runs in batch mode (depending on number of entries in the command line)

Further Details

Please refer to SAE Publication 2018-01-0412:

> Karim Hamza, Kenneth P. Laberteaux, John Willard. "A Java Implementation of Future Automotive Systems Technology Simulator (FASTSim) Fuel Economy Simulation Code Modules," SAE World Congress 2018, Detroit, MI, USA