Description:

The following project aim to create a simulation environment for models that can be represented as a sequence of algebraic equations that can be written in JavaScript.

The project has being developed in Java 1.7, and can be started from command prompt calling the main method of the Simulation class.
On a Linux Fedora 20 running on Intel i6 64 bits, with 8 Gb RAM, the simulations takes approximately 45 minutes (9000 test vectors).

Characteristics:

I have intended to create a code to maximize the reusability and extendability of it. Therefore:

1. Data vectors can be easily added/removed/modified and extended in “data.txt” and “datafields.txt”.
2. The model can be easily modified in “equations.txt”, even new fields and intermediate variables can be introduced -they have to be declared in “datafields.txt” as well.
3. Terminating conditions -in our case 15 years run- can be easily introduced, one or more fields/variables can be checked against their value to finish the current vector simulation.
4. Input values can be easily declared and modified -by now just one input value per simulation.

Configuration files:

“data.txt”
Contains the data set to simulate the model with.
The data divided by lines -data vectors- within the coma separated values that represent the following model fields:

```
Agent_Breed,Policy_ID,Age,Social_Grade,Payment_at_Purchase,Attribute_Brand,Attribu
te_Price,Attribute_Promotions,Auto_Renew,Inertia_for_Switch
```

The provide data file contains 8837 of such data rows.

“datafields.txt”
Contains the data field names in the same order used in “data.txt”. As well provides the field type:

```
Agent_Breed,String
Policy_ID,String
Age,Double
Social_Grade,Double
Payment_at_Purchase,Double
Attribute_Brand,Double
Attribute_Price,Double
Attribute_Promotions,Double
Auto_Renew,Boolean
Inertia_for_Switch,Double
Affinity,Double
Year,Double
Change_Breed,Boolean
Number_changes,Double
```

“equations.txt”
Contains the equations to apply to the data vectors, one equation per line. These mus be declared in the sequence order of application. The first field represents the result field, where the resulting equation value will be stored. After the comma, follows the equations itself, that must be a valid Javascript expression. All numbers in Javascript are represented as a floating point format.
Inside the equations has being introduced the '`Number_changes`' required to calculate one of the KPIs required by the Project.

To beautify the formulas has being used:
http://jsbeautifier.org/

To minimize the formulas -as in inside the file- has being used:
http://www.danstools.com/javascript-minify/

Equations:

```
Year,Year+1
Age,Age+1
Affinity,Payment_at_Purchase/Attribute_Price + (2 * Attribute_Promotions *
Inertia_for_Switch)
Change_Breed,function f1(){if(Auto_Renew){return
false;}else{if((Agent_Breed=="Breed_C")&&(Affinity<(Social_Grade*Attribute_Brand))
){return true;}else
if((Agent_Breed=="Breed_NC")&&(Affinity<(Social_Grade*Attribute_Brand*Brand_Factor
))){return true;}else{return false;}}}f1();
Agent_Breed,function f1(){if(Change_Breed){if(Agent_Breed=="Breed_C")
{return"Breed_NC";}else if(Agent_Breed=="Breed_NC"){return"Breed_C";}}else{return
Agent_Breed;}}f1();
Number_changes,function f1(){if(Change_Breed){return
Number_changes+1.0;}else{return Number_changes;}}f1();
```

“finishConditions.txt”
Represents the conditions that will determine when the simulations has to terminate. In our scenario the simulation mus run for 15 years, so the condition will be that the variable Year is 15.

```
Year,15.0
```

“input.txt”
Represents the inputs values to feed the model. Input values are handled in the same way as fields/internal variables of the model.

```
Year,15.0
```

Functional description:

1. The `simulation.simulate()` method iterate all the data vectors to be tested, and call the method `engine.simulateTestVector` for each of them.
Once the simulation is finished the results are out printed by `printResults` method.

2. The `engine.simulateTestVector` method loops through all possible input data values and add to the result vector the required values to start the simulation.
The input and the data vector are now prepared; the method `applyEquation will` being called for each of the equations in the definition order, until until the finish condition is true

`-isFinishCondition` mehod.

3. The method `applyEquation`, applies the corresponding equations to the data vector, get the result and assign it back to the corresponding variable/field.

Future improvements:

1. The “input.txt” now declares each of the possible input values, and just one per time. As an improvement will be the possibility of declare formulas, for example: `Brand_Factor, 0.1, 2.9, 0.1`, (input_field, starting_value, final_value, increment).
Also the possibility to declare several input fields, for example:

```
0,0,Brand_Factor, 0.1, 2.9, 0.1
0,1,random_factor, 0.1, 1.0, 0.1, random
etc.
```

2. Replace the object `ScriptEngineManager` and utilize a pure Java scripting engine that can use Java beans to link to the project code. Use native Java Object types to increase performance.
User java threading capabilities to increase the parallelism of the simulation and allow it to run on a multiple machine environment.

3. Logging possibilities. Currently just the standard output stream is being used.

4. Exception handling. Currently the manage of exceptions is very limited to the mandatory ones. These should be enhanced to handle errors in the configuration files.

5. Test cases are not being written. Because of time limit I have not included any unit test. Test cases must be added to ensure the code functionality.