# Shutfly LTV project

Jinglun Zhang

email: jinglun.usa@gmail.com

July 4, 2017

## 1. Project requirements

Please see the link to the coding challenge:  <https://github.com/sflydwh/code-challenge>

## 2. Usages

ShutflyTest is a class to test ShutflyLTV class with the input files. The input file is a text file with a string steam in JSON format as shown in the events.txt in Sample\_input fold.

The usage is:

*>Java ShutflyTest inputFileName*

Where inputFileName is the input file name in the input fold, it is txt file type and the whole file name is inputFileName.txt, the test output will have the same file name in the output fold. For example, if you issue a command:

*> Java ShutflyTest test1*

There is an input\test1.txt file before testing and it generates output\test1\_*N*.txt, output\test1.log files. Please see Tests section for the output file description. You can also issue a command with multiple inpurt file like:

*> Java ShutflyTest test1 test2 test3 test4*

It generates output\test1\_*N*.txt, output\test1.txt, too.

Please use interfaces Customer.java and DataPool.java for some operations should you use different data structure for ingest result. You can initial a DataPool by calling ShutflyLTV.createDataPool().

## 3. Tests

Each test, there are 3 output files, first input file name without ".txt" plus ".log" , "\_N1.txt" and "\_N2.txt", where N1 is some number of customers less that total number of customers and N2 is some number of customers more that total number of customers.

"*x*.log" file: program running log, including any errors found

"*x\_N1*.txt": Top N1 customers with the highest Life Time Value

"*x\_N2*.txt": All customers with the Life Time Value

1. events.txt: original test file

2. image1.txt, new\_customer1.txt: one event tests

3. site\_visit\_order.txt: two events

4. event\_many*N*.txt: many events, mixed with all verbs, many customers, spanning multiple weeks

5. wrong\_data*N*.txt: incomplete input file or items with wrong data, to test program error checking. If the input file is not formatted as JSON syntax, it will JSON parse error.

## 4. Design and Problem Justification

### 4.1 Data Structure

In the project requirement link, it defines "ingest(e, D)" interface to handle the data feeding into our data warehouse. we generalize the data structure with interface DataPool.java and Customer.java. The main data structures used in the current implementation are treeMap, etc.

Data Structure for DataPool:

first

last

Map<String(for customer\_id), Customer>

Customer objects:

customer\_id

first

last

last\_name

adr\_city

adr\_state

operation TreeMap<event\_time, Map<String, String>>

This data structure is defined to calculation Life Time Value for each customer easily, without sorting, without calculating the largest time duration for all events, while we use *first*: earliest event time and *last*: latest event time for tracking time spanning to save calculation.

In real data warehouse, customers, events, etc., should be stored in database.

### 4.2 Class Structure

*sConstants.java*: define return code, literal messages and properties in a central place to be easier to make globalization and localization.

*ingest(Map, DataPool) |*------ *Customer.java*

*ShutflyTest.java* --------------------------->*ShutflyLTV.java-------------------+*

*topXSimpleLTVCustomers(int, DataPool)* |----*DataPool.java*

While *DataPool.java* is implemented with *MyDataPool.java* and Customer.java is implemented with *MyCustomer.java*.

### 4.3 Problem Justification

1. We assume all event input data is a valid JSON format file.

2. There are a few ***data integrity*** issues and we make loosely handling. For example, for type IMAGE, SITE\_VISIT, and ORDER, if customer doesn't exist, the system will add a customer automatically. Strictly speaking, for type SITE\_VISIT, and ORDER with verb:UPDATE, the customer should exist already. Considering our tests are only a part of the all system operations, we will also add a customer, too.

3. We assume when create and update a customer and a order, upload an image, they are also making a site visit, so we count these events into site visit.

4. For Key and Customer\_ID, we need to check if the ID is consisting of 0-9 and a-f letters and ID length, etc. We made a layer of method to check it, but not really validate it.

5. For event\_time 'Z' letter, we assume it is for Zulu time zone and use 'Z' only in all event data. Please refer to web sites for more information:

https://en.wikipedia.org/wiki/ISO\_8601

https://www.timeanddate.com/time/zones/

6. For the week days, we assume a week is starting from Sunday to Saturday. For first event time, we will get the week's Sunday to calculate.

7. For *topXSimpleLTVCustomers(int, DataPool),* it returns an array of Customer. We print out to output file with JSON format for some key data of the customers.

8. Development and tests are done with Windows 10 and eclipse IDE, use Java JDK 6.

9. For JSON parse, it used json-simple-1.1.1.jar, we need to down load and put it in eclipse project build path.

## 5. Program Algorithm Analysis and Improvement

.Since we use a TreeMap to store events in the order of event\_time, if N is total event number, our *topXSimpleLTVCustomers()* complexity is O(N).

If it is frequently to get Top LTV, we may need to cache order summary data for each customer, which is similar to *Roll Up summary* in data warehouse. Doing it, we can reduce our *topXSimpleLTVCustomers()* complexity is O(C), where C is a constant related to time period, from O(N) (N is total visits). However, during ingest(e, D), we need to do summary for orders. Therefore, the real *topXSimpleLTVCustomers()* complexity is O(N\_orders).

We need to do *more tests* by designing the wrong data cases to make our program robotics.

As we understand, we should use a database to store customer and event data, not an in-memory data structure.

We need to make more generic program by removing the limitations mentioned in section 4.3.