**Accenture Java Hackathon Instructions**

Version: 0.3

Table of Contents

[problem statement 3](#_Toc536094028)

[technical guidelines 5](#_Toc536094029)

# problem statement

An automobile dealer company, which is new in the industry, wants to setup an Inventory Management system for its customers that would enable tracking inventory levels, placing orders, computing estimated sales and reporting estimated deliveries of cars those are ordered based on certain choices and preferences.

The proposed solution should accept **CarStandingOrders**, an input CSV file, which has list of standing orders those need to be placed towards the dealership. This input file should contain the columns and the associated values given in brackets as follows:

1. Name of Customer
2. State (Maharashtra, Gujarat, Madhya Pradesh, Goa, Karnataka)
3. Vendor (Tata, Maruti Suzuki)
4. Model (Nano, Bolt, Indica, Ciaz, Swift, Wagon R)
5. Variant (Petrol, Diesel)
6. Color (White, Black, Gray)
7. Accessories (Music System, Power Door Locks, Body Cover, Floor Mat, Seat Cover)
8. Motor Insurance (ICICI Lombard, Bajaj Allianz, NA)
9. Personal Protect Plan for Insurance (Yes, No)

The application should read this file and process the functionality with the data that is mentioned for making orders in a concurrent manner so that orders can be placed, in bulk, faster by processing each record or set of records in different threads that are spawned from the application.

Application should check the availability of selected Car and Accessories during each order is being placed.

The availability check of selected Car should happen against **CarInventory**, an input CSV file, which should contain the columns and associated values given in brackets as follows:

1. Vendor (Tata, Maruti Suzuki)
2. Model (Nano, Bolt, Indica, Ciaz, Swift, Wagon R)
3. Variant (Petrol, Diesel)
4. Color (White, Black, Gray)
5. Base Price
6. Quantity Available

Similarly the availability check of selected Accessories should happen against **AccessoryInventory**, an input CSV file, which should contain the columns and associated values given in brackets as follows:

1. Vendor (Tata, Maruti Suzuki)
2. Model (Nano, Bolt, Indica, Ciaz, Swift, Wagon R)
3. Accessories (Music System, Power Door Locks, Body Cover, Floor Mat, Seat Cover)
4. Price
5. Quantity Available

Both of these files should contain records based on combinations of all values mentioned in brackets.

Application should report with an error message if the **CarStandingOrders** file contains values other than the ones present in **CarInventory** and **AccessoryInventory** files.

For each successful order placed, the value of Quantity Available must be decremented by 1 to keep track of the available stock in the Warehouse. If selected car is not available in the Warehouse, an error message should be logged with required details about standing order.

As the dealer company has an alliance with few Motor Insurance providers, the choice is to be given to customers whether they need to purchase Motor Insurance or not. Inclusion of Personal Protect Plan option needs to be also provided. Design should assume that “Bajaj Allianz” doesn’t offer Personal Protect Plan as of now. Following is the **MotorInsuranceProviderInventory**, an input CSV file, which should contain the columns and associated values given in brackets as follows:

1. Motor Insurance Provider (ICICI Lombard, Bajaj Allianz)
2. Personal Protect Plan Offered (Yes, No)
3. First Year Premium (in INR)

Application should not execute the logic associated to **MotorInsuranceProviderInventory** if the **CarStandingOrders** file contains NA for Motor Insurance option in which case the Customer does not want to avail the insurance policy.

Furthermore, Application should report with an error message if the **CarStandingOrders** file contains values and combination of values other than the ones present in Motor Insurance Provider Inventory file.

Once the availability is checked, application should compute the Total Price of selected car depending on the region that is mentioned in the **CarStandingOrders** file. Following is the **RegionalTaxRateConfiguration**, an input CSV file, which should contain the columns and associated values given in brackets as follows:

1. State (Maharashtra, Gujarat, Madhya Pradesh, Goa and Karnataka)
2. Tax Rate (values in %)

Application should report with an error message if the **CarStandingOrders** file contains values, for State, other than the ones present in **RegionalTaxRateConfiguration** file.

Further, the Tax Expense for each order should be calculated as follows:

*Tax Expense = (Base Price + Additional cost due to Accessories) \* Tax Rate*

Application should create **RegionalEstimatedSalesReport**, an output CSV file, which should populate the columns and associated values given in brackets as follows:

1. State (Maharashtra, Gujarat, Madhya Pradesh, Goa, Karnataka)
2. Estimated Sales in Units (*= number of car units estimated to be sold for given region*)
3. Total Estimated Sales [(*= Base Price + Additional cost due to Accessories + First Year Premium + Tax Expense*) for each ordered car)
4. Estimated Net Income (*= Total Estimated Sales - Tax Expense*)

Additionally, application should create console log entries of Total Price of Car for reporting estimated deliveries whose standing orders are confirmed (according to availability). The Total Price for each order should then be calculated as follows.

*Total Price = Base Price + Additional cost due to Accessories + Tax Expense*

Please refer to attached input csv files and all these csv files have to be added in the classpath of eclipse project.



# technical guidelines

While designing and building an application based on the above problem statement, the solution must adhere to the technical guidelines mentioned herewith.

The solution should be based on **Concurrent Core Java application** which must be executable from eclipse based on **Java 8**.

Following **Language Features** must be used:

1. Lambda Expressions
2. Collections including Stream APIs
3. Concurrency related APIs
4. Date-Time APIs
5. Generics
6. Annotation
7. String in Switch Statement
8. Handling more than one type of Exception

Application should exhibit following **Design Principles** and **System Characteristics**:

1. Object oriented design principles
2. Modularity
3. Reusability
4. Error Handling (application should gracefully exit in erroneous scenarios)
5. Logging (The console output can be used to log required information of erroneous and other important events. The use of logging frameworks is not mandatory here.)

Additionally, following **Coding Standards** must be considered:

1. Code Quality should be of highest degree according to Accenture standards defined in SonarQube.
2. Prepare unit test cases with the help of JUnit. Code Coverage should be 80%.
3. Generate necessary Javadocs using the JavaDoc tool.

Furthermore, the **Performance of System** must be measured and reported such that the first statement in the application is to be considered as a starting point of execution and last statement as ending point where the calculation of time taken by the entire application to run must be reported on console with the help of log entries.

**Notes**:

1. The emphasis will not be given on the usage and the number of popular frameworks and libraries used but rather on the devised approach.
2. Please use open library for reading and writing the csv file rather than writing the custom code for it.
3. All of the input CSV files mentioned in the Problem Statement are provided along with Problem Statement.
4. Please do not modify the input csv files. Java code would be evaluated against the standard input csv files
5. Make necessary assumptions wherever required and document them in the code
6. All the errors have to be printed the output file. Please find below for sample of output csv file:-

