**SOF & Open Platform - Ex interview**

# **Goals**

This exercise gives you a chance to demonstrate your learning ability, while also giving you a look of a technology very popular in our code base.

# **Instructions**

## **Prerequisites**

You must install the following tools to finish the exam

* Maven (https://maven.apache.org/index.html)
* Java JDK 8+

## **General guidelines**

* This exam has 2 parts. It is recommended that you finish part 1 before jumping to part 2.
* This exam requires you to be somewhat familiar with Dependency Injection design pattern and specifically Dagger 2. Spend some time studying those concepts. You can check the References section at the end of this document for some starting materials.
* You don’t need to read all materials, some of them are overlapped, you can select the form that you are most comfortable with.
* The implementation is not hard once you understand those concepts, so don’t panic if this seems too complicated. Navigating through complex problems is an essential skill to be successful at MoMo.
* Don't give up. It's another highly rewarded skill in MoMo in general and in the SA (SOF & Open Platform) team in particular.
* You can submit your work even when it is not completed. This is only one stage of the whole interview process.

# **How do you design a Coin-Operated Soda Machine?**

1. You need to design a Machine which:
   1. Accepts notes of 10.000, 20.000, 50.000, 100.000, 200.000 VND
   2. Allows user to select products Coke (10.000), Pepsi (10.000), Soda (20.000)
   3. Allows user to receive a refund by canceling the request.
   4. Releases the selected product and remaining change if any
2. Now your Machine is up and running. Next, you need to integrate the promotion scheme as following:
   1. If there are 3 consecutive purchases of the same product, the user will have a 10% chance to receive a product for free.
   2. The limited budget for the program is 50.000 a day.
   3. If the limit cannot be reached for a day, the win rate of the next day will be increased by 50%.

# **Results to be achieved**

* Design documentation: ideas and logical processing flow
* Source code (with Unit Test)

# **References**

* Dependency Injection concept (not related to Dagger 2), recommended if you are not familiar with the concept: <https://www.journaldev.com/2394/java-dependency-injection-design-pattern-example-tutorial>
* Dagger Homepage: <https://dagger.dev/>
* Dagger video: <https://www.youtube.com/watch?v=oK_XtfXPkqw>