Glassbeam Interview Questions for Platform

Instructions:

1. Please try to attend all the questions.
2. The answers should be sent over email within 48 hours.
3. The difficulty level increases with the question number, for example, Qs Num 1 is easier to solve whereas Qs Num 4 requires more thinking.
4. The preferred programming language to solve these problems, in the same order as mentioned, are
   a) Scala
   b) Java
   c) C++
   d) Python
   e) If candidate doesn't know any of the above mentioned languages, he/she can chose to write the code in his own preferred language.
5. Third Party APIs should not be used.
6. The candidates will be evaluated on the following parameters as how he/she attempts to solve these questions.
   a) Preferred Languages Chosen as mentioned above.
   b) Logical Thinking.
   c) Efficient Coding.
   d) Functional Programming concepts used or not.
   e) Object Oriented Concepts used or not.
   f) Code should be of production quality.
   g) Code should reflect design decisions and assumptions clearly.
   h) Code should have proper commenting but excessive commenting should be avoided.
   i) Including Unit Tests will be encouraged.
   j) Whatever code candidates writes, it must be executable.

Questions:-

1. There is a group of 3 thieves. These thieves carry out thefts in a group of 2 at a time, based on an agreement between them which is mentioned as follows. If you are given a week day, Identify which of the two thieves will be carrying out thefts on that day.

Given :
- Three thieves are A, C and E.
- The 2 thieves who will carry out thefts on a day are identified by applying following
operation to the day.

If the day is divisible by 3, it’s A and E who will be carrying out the thefts.

If the remainder on dividing day by 3 is 1, E and C will be carrying out the thefts.
If the remainder on dividing day with 3 is 2, C and A will be carrying out the thefts.

Input Format
The first line will contain an integer Y.

Output Format
Report which of the two thieves are carrying out thefts on the Yth day. (i. e. you will have to print one of the three combination AE, EC or AC)

2. Given a number, find the number of steps that would be required to determine the final result by multiplying all digits of every intermediate result.

For Example:-
If the given number is n = 4327 then the result would be 4.

The intermediate steps for the above would be

Step 1 - 4*3*2*7 - the product of this would be 168 which is a 3 digit number hence
moving on to the next step.
Step 2 - 1*6*8 - the product of this would be 48 which is a 2 digit number hence
moving on to the next step.
Step 3 - 4*8 - the product of this would be 32 which is again a 2 digit number hence
moving on to the next step.
Step 4 - 3*2 - the product of this is 6 which is a single digit number hence the total
number of steps to reach the single digit is 4.

3. There are two separate infinite sources of data. Data is arriving "simultaneously" into our system as two streams via two channels: Channel 1 and Channel 2. The data could be of three types: R, G and B. Each data element should have two properties: channelNumber and uniqueID. These three types of Data are arriving in a random sequence on the two channels. Write a program which creates pairs of "same types" arriving on two channels in their "order of arrival".
Example:
If a sample sequence is as follows:
Channel 1: R1_1 R1_2 R1_3 B1_4 B1_8 G1_5
Channel 2: B2_6 B2_8 R2_9 G2_10 B2_7 R2_20
output is:
(R1_1, R2_9) (B1_4, B2_6) (B1_8, B2_8) (G1_5, G2_10) (R1_2, R2_20)

4. Gully Cricket:

a. There are two separate infinite sources of data coming to our program as UDP packets. Let's call them: UDP-Channel-1 and UDP-Channel-2

b. Data on UDP-Channel-1 is for new Gully Cricket matches being created and ending of existing matches. The message coming is a JSON with the following shape:

```
{
    "teams": ["team-bangalore", "team-delhi"],
    "location": "BBMP Cricket Grounds"
    "state": "started"
    // this will have "ended" when a match completes
}
```

c. Data on UDP-Channel-2 is for Cricket scores for active matches. To keep the
application simple and not make it complex with rules of Cricket, only one JSON shape is supported on this channel:

```
{
    "teams": ["team-bangalore", "team-delhi"],
    "score": {
        "batting": "team-bangalore",
        "runs": 98,
        "overs": 15,
        "chasing": 0 // this will have a 0 score if the given team is
    batting first.
        A non-zero score if it is batting second
    }
}
```

d. As soon as the application starts, it starts listening on 2 UDP ports to receive messages in the background. And a user should be able to interact with the application on a perpetually running command terminal. The command line interaction will be to find the latest score of a active match by entering the team name like this -

```
        $<run your application>
        Enter the name of one of the teams playing the match
        "team-bangalore"
        ======
        Match between "team-bangalore" & "team-delhi" at "BBMP
    Cricket Grounds".
        "team-bangalore" is batting first and has scored 98 runs in
    15 overs.
        ======
        Enter the name of one of the teams playing the match
        "team-abc"
        ======
        No match with "team-abc" is currently in progress
        ======
```

e. Caveats to this application –

1. We can receive 'n' create match requests for the same 2 teams on UDP-Channel-1. Because there can be 'n' scorers for a single match who are reporting scores. So the application can receive the same create match message on UDP-Channel-1 for a match between "team-bangalore" & "team-delhi" five times - which could mean there are 5 people who may send scores for this match on UDP-Channel-2. In such a case, there should be only 1 match created in the application that is tracked despite multiple create requests

2. Since there could be 'n' scorers uploading scores for a given match, one scorer could be reporting faster (more recent score) while another ends up reporting a slightly old score. The old score should be discarded and the command prompt should always report the latest and most-recent score

3. Following erroneous data could be received and should be logged:

   i. There CANNOT be 2 different ongoing matches for a team. So, if a match is going on between "team-bangalore" & "team-delhi", then, we cannot receive
   another match between "team-bangalore" & "team-chennai". On receipt of 2nd
   such message, error should be logged

   ii. Ending of a match that was not started, or receiving of score for a match that was not created are errors and should be logged.

   iii. Team names are alphanumeric and cannot be empty. Order of team names in both types of messages can be anything and should be supported.

   iv. If the shape of incoming JSON on the 2 channels is not per above examples then log an error with the UDP payload and discard it.

4. Application has to support one create-match request every 5 seconds at max. And the application will receive one new score every second (on channel-2) across all active matches at max. Data velocity will NOT breach these limits.

5. Ensure sufficient unit test coverage. Write a short README describing how to run the program and one sample interaction. You can code in any programming
language that you like.