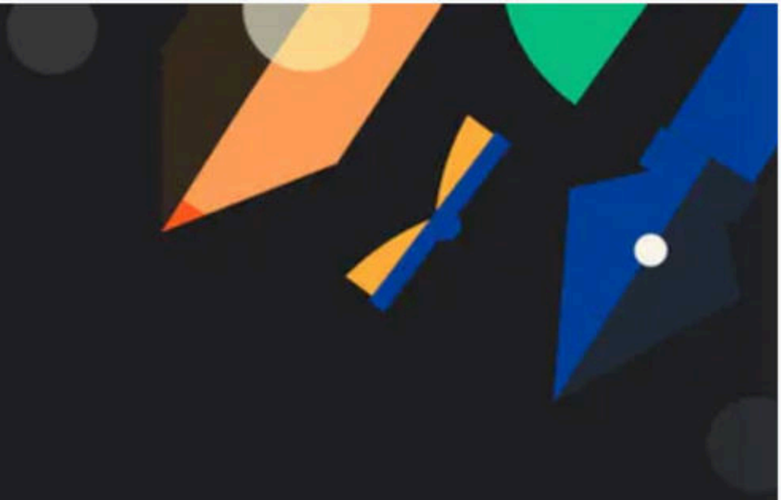

Solving Design Problems - Level I

Course on Solving Real World OOPs Design Problems

Love Babbar • Lesson 1 • June 4, 2022

How to Approach a problem

Planning:

- Understand Problem First
- Ask Clarifying Questions
- Don't directly jump to Implementation, think about the solution first
- Think about classes and possible design patterns/principles that can be applied
- You should be aware of the time too.

Implementation:

- Use Different Modules
- Mark imp fields as Private and use getters/setters.
- You should have a driver class for testing your code
- Always have a plan to finish 30 mins before the time

Review:

- Start with different class u create and relationship between them
- Tell them about the principles/Patterns u have used
- Interviewer can provide u custom test case
- There will be definitely some scope of improvement that Interviewer will find out , So don't panic and do the necessary changes on the go.

This is not Right or Wrong Game, it is a game of discussing and optimising the code.

In-Memory Cache:

Description:
Babbar Bhaiya got a new client for his web-services company. Clients wants to have a efficiently designed In-Memory Cache system, which he/she can invoke with the driver class. Help babbar bhaiya to deliver this task.

Requirements:
- create a user-defined size cache and with Eviction policy as “LRU”
- support fetch/Insert/Delete/Clear functionality in the system.

Other Details:
- Do not use any database or NoSQL store, use in-memory store for now.
- Do not create any UI for the application.
- Write a driver class for demo purposes. Which will execute all the commands at one place in the code and test cases.
- Please prioritize code compilation, execution and completion.
- Please do not access the internet for anything EXCEPT syntax.
- You are free to use the language of your choice.
- All work should be your own. If found otherwise, you may be disqualified.

Expectations:
- Code should be demo-able (very important)
- Complete coding within the duration of 90 minutes.
- Code should be modular, with Object Oriented design.
- Maintain good separation of concerns.
- Code should be extensible. It should be easy to add/remove functionality without rewriting the entire codebase.
- Code should handle edge cases properly and fail gracefully.
- Code should be readable.
- Follow good coding practices: Use intuitive variable names, function names, class names etc. Indent code properly. Once the code is complete please zip the source code and upload it to: xxxyyyzzz.com