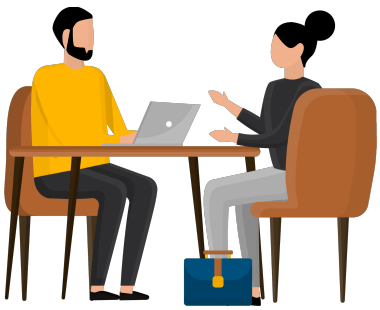
An Approach to Solve a Real-world Problem

Get to know the approach to solving a real-world problem.

Interview process

The object-oriented design interview tests a candidate's understanding of OOP concepts and principles. The candidate is usually provided with a problem statement that the candidate deciphers and solves using their object-oriented design skills.

Generally, all OOD interview questions follow a similar pattern, where an ambiguous problem is presented to the candidate along with a group of constraints. It is for the candidate to understand the problem and identify all the main entities that will play a part and design a workable solution, one that is flexible and adaptable to change in the future.



The OOD interview

To simplify things, you can use the following process to solve any OOD question.

Identify the requirements

The first few minutes (usually around 10 minutes) of the interview are for you to understand the entirety of the problem. You should use this time to gather the system's requirements by asking the interviewer questions. It is easier to work on a solution if there is a well-defined requirement set. You should also ask the interviewer if they expect you to implement the solution fully or give a structural overview. Try to understand and apply the SOLID principle, which covers all the significant aspects of OOD (encapsulation, abstraction, inheritance, polymorphism). It is important to know that the interviewer is looking for the following:

- How good are you at collecting requirements?
- Are you able to effectively scope down the problem?
- Can you produce a design based on your requirements within the 30–45 minutes given to you?

Model the problem

It is important to identify the primary use cases of your system. Talk about it to engage the interviewer. Thinking out loud also helps identify some expectations or ideas you may have missed earlier. The interviewer may sometimes ask you to sketch the use case diagram. Revise the different components of the diagram beforehand, like the system, actors, and use cases.

Establish the classes and their relationships

Identifying classes and their relationships is the highlight and the most important part of the OOD interview. To present a solution, you first have to list all the

part of the OOD interview. To present a solution, you first have to list all the objects that will play some role in the system. For example, if you're designing a parking lot, the potential objects will be vehicles, parking spots, entrances, exits, etc..

Once you have all your objects, the next step is to work out the attributes and operations for each object. Every object has properties and behaviors that allow it to fulfill its role in the system. Recognizing this can help you in creating your classes. An intuitive way to differentiate objects and methods is that the *nouns* in the requirements are possible objects and the *verbs* are the methods.

Next, map out the relationships between different objects that interact with each other. You must also point out and justify whether you will use abstract classes or interfaces. Identify the constraints of each class and point out how OOD concepts like abstraction, encapsulation, inheritance, and polymorphism will play a part here. Make sure to sketch a high-level diagram for the interviewer so they can visualize what you explain to them.

Sequence and activity diagrams

The sequence and activity diagrams are not necessarily an interview requirement. However, sometimes the interviewer asks you to describe a sequence of events or to explain the system flow of control of a certain activity. These diagrams are very useful in visually explaining the concept.

Use design patterns

While designing the problem, remember the different design patterns that symbolize the best practices used by object-oriented software developers. Try to apply the best design patterns applicable to each problem and fully explain your solution using this. This will help your interviewer develop a more positive outlook of you and let them know that you are capable of observing problems through the right lens.

Code

You should also be able to code the classes and relationships you identify in a programming language of your choice. The interviewer expects you to be able to code a high-level structure of the system. Additionally, they may also ask you to code one of the classes at the implementation level. It is important to understand the system's functionality before beginning to code. While coding, prioritize writing the abstract classes and interfaces, followed by the core objects and the internal structure. Ensure that you implement a code that follows the best practices in terms of maintainability.

The sequence of the design problem discussion in the interview

Design approach

We can approach the design for the interview problems by first identifying the possible use cases. For example, for a parking lot problem, this can either be the agent collecting the parking toll of a car or the system showing how many parking spots are available in the parking lot.

We will then note down any constraints present in our system. Our goal in designing our system should be to make it as simple as possible without

compromising on any features. For this step, we can ignore any attributes that do not align with our requirements.

Lastly, we should keep the scalability of our system in mind. For example, in the case of the parking lot problem, if our parking lot only accommodates one floor, we must design our system to ensure that the functionality of multiple floors can be added as per the system's requirements.

Top-down vs. bottom-up approach

The two most popular design approaches are top-down and bottom-up design. The table below highlights a few differences between the two:

Top-down	Bottom-up
This approach constructs high-level objects, and then designs the smaller subcomponents.	This approach identifies the smalles uses those components as a base t components.
It's a backward-looking approach.	It's a forward-looking approach.
It's mainly used in structural programming.	It's mainly used in object-oriented d
It allows for a high amount of data redundancy.	It allows for a minimum data redund

Since most of the problems in this course follow an object-oriented design, it is more suitable to follow the bottom-up design approach to develop them.

Back lesson

Completed Next

Quiz: Design Patterns

Getting Ready: Parking Lot