7

Object-Oriented Design

Object-oriented design questions require a candidate to sketch out the classes and methods to implement technical problems or real-life objects. These problems give—or at least are believed to give—an interviewer insight into your coding style.

These questions are not so much about regurgitating design patterns as they are about demonstrating that you understand how to create elegant, maintainable object-oriented code. Poor performance on this type of question may raise serious red flags.

How to Approach

Regardless of whether the object is a physical item or a technical task, object-oriented design questions can be tackled in similar ways. The following approach will work well for many problems.

Step 1: Handle Ambiguity

Object-oriented design (OOD) questions are often intentionally vague in order to test whether you'll make assumptions or if you'll ask clarifying questions. After all, a developer who just codes something without understanding what she is expected to create wastes the company's time and money, and may create much more serious issues.

When being asked an object-oriented design question, you should inquire *who* is going to use it and *how* they are going to use it. Depending on the question, you may even want to go through the "six Ws": who, what, where, when, how, why.

For example, suppose you were asked to describe the object-oriented design for a coffee maker. This seems straightforward enough, right? Not quite.

Your coffee maker might be an industrial machine designed to be used in a massive restaurant servicing hundreds of customers per hour and making ten different kinds of coffee products. Or it might be a very simple machine, designed to be used by the elderly for just simple black coffee. These use cases will significantly impact your design.

Step 2: Define the Core Objects

Now that we understand what we're designing, we should consider what the "core objects" in a system are. For example, suppose we are asked to do the object-oriented design for a restaurant. Our core objects might be things like `Table`, `Guest`, `Party`, `Order`, `Meal`, `Employee`, `Server`, and `Host`.

Step 3: Analyze Relationships

Having more or less decided on our core objects, we now want to analyze the relationships between the objects. Which objects are members of which other objects? Do any objects inherit from any others? Are relationships many-to-many or one-to-many?

For example, in the restaurant question, we may come up with the following design:

- `Party` should have an array of `Guests`.
- `Server` and `Host` inherit from `Employee`.
- Each `Table` has one `Party`, but each `Party` may have multiple `Tables`.
- There is one `Host` for the `Restaurant`.

Be very careful here—you can often make incorrect assumptions. For example, a single `Table` may have multiple `Parties` (as is common in the trendy "communal tables" at some restaurants). You should talk to your interviewer about how general purpose your design should be.

Step 4: Investigate Actions

At this point, you should have the basic outline of your object-oriented design. What remains is to consider the key actions that the objects will take and how they relate to each other. You may find that you have forgotten some objects, and you will need to update your design.

For example, a `Party` walks into the `Restaurant`, and a `Guest` requests a `Table` from the `Host`. The `Host` looks up the `Reservation` and, if it exists, assigns the `Party` to a `Table`. Otherwise, the `Party` is added to the end of the list. When a `Party` leaves, the `Table` is freed and assigned to a new `Party` in the list.

▸ Design Patterns

Because interviewers are trying to test your capabilities and not your knowledge, design patterns are mostly beyond the scope of an interview. However, the Singleton and Factory Method design patterns are widely used in interviews, so we will cover them here.

There are far more design patterns than this book could possibly discuss. A great way to improve your software engineering skills is to pick up a book that focuses on this area specifically.

Be careful you don't fall into a trap of constantly trying to find the "right" design pattern for a particular problem. You should create the design that works for that problem. In some cases it might be an established pattern, but in many other cases it is not.

Singleton Class

The Singleton pattern ensures that a class has only one instance and ensures access to the instance through the application. It can be useful in cases where you have a "global" object with exactly one instance. For example, we may want to implement `Restaurant` such that it has exactly one instance of `Restaurant`.

```
1  public class Restaurant {
2    private static Restaurant _instance = null;
3    protected Restaurant() { ... }
4    public static Restaurant getInstance() {
5      if (_instance == null) {
6        _instance = new Restaurant();
7      }
```

```
8         return _instance;
9      }
10  }
```

It should be noted that many people dislike the Singleton design pattern, even calling it an "anti-pattern." One reason for this is that it can interfere with unit testing.

Factory Method

The Factory Method offers an interface for creating an instance of a class, with its subclasses deciding which class to instantiate. You might want to implement this with the creator class being abstract and not providing an implementation for the Factory method. Or, you could have the Creator class be a concrete class that provides an implementation for the Factory method. In this case, the Factory method would take a parameter representing which class to instantiate.

```
1   public class CardGame {
2      public static CardGame createCardGame(GameType type) {
3         if (type == GameType.Poker) {
4            return new PokerGame();
5         } else if (type == GameType.BlackJack) {
6            return new BlackJackGame();
7         }
8         return null;
9      }
10  }
```

Interview Questions

7.1 **Deck of Cards:** Design the data structures for a generic deck of cards. Explain how you would subclass the data structures to implement blackjack.

Hints: #153, #275

pg 305

7.2 **Call Center:** Imagine you have a call center with three levels of employees: respondent, manager, and director. An incoming telephone call must be first allocated to a respondent who is free. If the respondent can't handle the call, he or she must escalate the call to a manager. If the manager is not free or not able to handle it, then the call should be escalated to a director. Design the classes and data structures for this problem. Implement a method `dispatchCall()` which assigns a call to the first available employee.

Hints: #363

pg 307

7.3 **Jukebox:** Design a musical jukebox using object-oriented principles.

Hints: #198

pg 310

7.4 **Parking Lot:** Design a parking lot using object-oriented principles.

Hints: #258

pg 312

7.5 **Online Book Reader:** Design the data structures for an online book reader system.

Hints: #344

pg 318

7.6 **Jigsaw:** Implement an NxN jigsaw puzzle. Design the data structures and explain an algorithm to solve the puzzle. You can assume that you have a `fitsWith` method which, when passed two puzzle edges, returns true if the two edges belong together.

Hints: #192, #238, #283

pg 318

7.7 **Chat Server:** Explain how you would design a chat server. In particular, provide details about the various backend components, classes, and methods. What would be the hardest problems to solve?

Hints: #213, #245, #271

pg 326

7.8 **Othello:** Othello is played as follows: Each Othello piece is white on one side and black on the other. When a piece is surrounded by its opponents on both the left and right sides, or both the top and bottom, it is said to be captured and its color is flipped. On your turn, you must capture at least one of your opponent's pieces. The game ends when either user has no more valid moves. The win is assigned to the person with the most pieces. Implement the object-oriented design for Othello.

Hints: #179, #228

pg 326

7.9 **Circular Array:** Implement a `CircularArray` class that supports an array-like data structure which can be efficiently rotated. If possible, the class should use a generic type (also called a template), and should support iteration via the standard `for (Obj o : circularArray)` notation.

Hints: #389

pg 329

7.10 **Minesweeper:** Design and implement a text-based Minesweeper game. Minesweeper is the classic single-player computer game where an NxN grid has B mines (or bombs) hidden across the grid. The remaining cells are either blank or have a number behind them. The numbers reflect the number of bombs in the surrounding eight cells. The user then uncovers a cell. If it is a bomb, the player loses. If it is a number, the number is exposed. If it is a blank cell, this cell and all adjacent blank cells (up to and including the surrounding numeric cells) are exposed. The player wins when all non-bomb cells are exposed. The player can also flag certain places as potential bombs. This doesn't affect game play, other than to block the user from accidentally clicking a cell that is thought to have a bomb. (Tip for the reader: if you're not familiar with this game, please play a few rounds online first.)

This is a fully exposed board with 3 bombs. This is not shown to the user.

	1	1	1			
	1	*	1			
	2	2	2			
	1	*	1			
	1	1	1			
			1	1	1	
			1	*	1	

The player initially sees a board with nothing exposed.

?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?

Clicking on cell (row = 1, col = 0) would expose this:

	1	?	?	?	?	?
	1	?	?	?	?	?
	2	?	?	?	?	?
	1	?	?	?	?	?
	1	1	1	?	?	?
			1	?	?	?
			1	?	?	?

The user wins when everything other than bombs has been exposed.

	1	1	1			
	1	?	1			
	2	2	2			
	1	?	1			
	1	1	1			
			1	1	1	
			1	?	1	

Hints: #351, #361, #377, #386, #399

pg 332

7.11 **File System:** Explain the data structures and algorithms that you would use to design an in-memory file system. Illustrate with an example in code where possible.

Hints: #141, #216

pg 337

7.12 **Hash Table:** Design and implement a hash table which uses chaining (linked lists) to handle collisions.

Hints: #287, #307

pg 339

Additional Questions: Threads and Locks (#16.3)

Hints start on page 662.