

$$\begin{aligned} &= P(x \text{ is even given } x \leq 5) P(x \leq 5) \\ &= (2/5) * (1/2) \\ &= 1/5 \end{aligned}$$

Observe that since $P(A \text{ and } B) = P(B \text{ given } A) P(A) = P(A \text{ given } B) P(B)$, you can express the probability of A given B in terms of the reverse:

$$P(A \text{ given } B) = P(B \text{ given } A) P(A) / P(B)$$

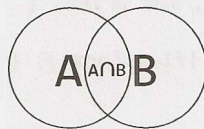
The above equation is called Bayes' Theorem.

Probability of A or B

Now, imagine you wanted to know what the probability of landing in A or B is. If you knew the odds of landing in each individually, and you also knew the odds of landing in their intersection, then you could express the probability as:

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$

Logically, this makes sense. If we simply added their sizes, we would have double-counted their intersection. We need to subtract this out. We can again visualize this through a Venn diagram:



For example, imagine we were picking a number between 1 and 10 (inclusive). What's the probability of picking an even number *or* a number between 1 and 5? We have a 50% probability of picking an even number and a 50% probability of picking a number between 1 and 5. The odds of doing both are 20%. So the odds are:

$$\begin{aligned} &P(x \text{ is even or } x \leq 5) \\ &= P(x \text{ is even}) + P(x \leq 5) - P(x \text{ is even and } x \leq 5) \\ &= \frac{1}{2} + \frac{1}{2} - \frac{1}{5} \\ &= \frac{4}{5} \end{aligned}$$

From here, getting the special case rules for independent events and for mutually exclusive events is easy.

Independence

If A and B are independent (that is, one happening tells you nothing about the other happening), then $P(A \text{ and } B) = P(A) P(B)$. This rule simply comes from recognizing that $P(B \text{ given } A) = P(B)$, since A indicates nothing about B.

Mutual Exclusivity

If A and B are mutually exclusive (that is, if one happens, then the other cannot happen), then $P(A \text{ or } B) = P(A) + P(B)$. This is because $P(A \text{ and } B) = 0$, so this term is removed from the earlier $P(A \text{ or } B)$ equation.

Many people, strangely, mix up the concepts of independence and mutual exclusivity. They are *entirely* different. In fact, two events cannot be both independent and mutually exclusive (provided both have probabilities greater than 0). Why? Because mutual exclusivity means that if one happens then the other cannot. Independence, however, says that one event happening means absolutely *nothing* about the other event. Thus, as long as two events have non-zero probabilities, they will never be both mutually exclusive and independent.

If one or both events have a probability of zero (that is, it is impossible), then the events are both independent and mutually exclusive. This is provable through a simple application of the definitions (that is, the formulas) of independence and mutual exclusivity.

► Start Talking

Don't panic when you get a brainteaser. Like algorithm questions, interviewers want to see how you tackle a problem; they don't expect you to immediately know the answer. Start talking, and show the interviewer how you approach a problem.

► Develop Rules and Patterns

In many cases, you will find it useful to write down "rules" or patterns that you discover while solving the problem. And yes, you really should write these down—it will help you remember them as you solve the problem. Let's demonstrate this approach with an example.

You have two ropes, and each takes exactly one hour to burn. How would you use them to time exactly 15 minutes? Note that the ropes are of uneven densities, so half the rope length-wise does not necessarily take half an hour to burn.

Tip: Stop here and spend some time trying to solve this problem on your own. If you absolutely must, read through this section for hints—but do so slowly. Every paragraph will get you a bit closer to the solution.

From the statement of the problem, we immediately know that we can time one hour. We can also time two hours, by lighting one rope, waiting until it is burnt, and then lighting the second. We can generalize this into a rule.

Rule 1: Given a rope that takes x minutes to burn and another that takes y minutes, we can time $x+y$ minutes.

What else can we do with the rope? We can probably assume that lighting a rope in the middle (or anywhere other than the ends) won't do us much good. The flames would expand in both directions, and we have no idea how long it would take to burn.

However, we can light a rope at both ends. The two flames would meet after 30 minutes.

Rule 2: Given a rope that takes x minutes to burn, we can time $\frac{x}{2}$ minutes.

We now know that we can time 30 minutes using a single rope. This also means that we can remove 30 minutes of burning time from the second rope, by lighting rope 1 on both ends and rope 2 on just one end.

Rule 3: If rope 1 takes x minutes to burn and rope 2 takes y minutes, we can turn rope 2 into a rope that takes $(y-x)$ minutes or $(y - \frac{x}{2})$ minutes.

Now, let's piece all of these together. We can turn rope 2 into a rope with 30 minutes of burn time. If we then light rope 2 on the other end (see rule 2), rope 2 will be done after 15 minutes.

From start to end, our approach is as follows:

1. Light rope 1 at both ends and rope 2 at one end.
2. When the two flames on Rope 1 meet, 30 minutes will have passed. Rope 2 has 30 minutes left of burn-time.

3. At that point, light Rope 2 at the other end.
4. In exactly fifteen minutes, Rope 2 will be completely burnt.

Note how solving this problem is made easier by listing out what you've learned and what "rules" you've discovered.

► Worst Case Shifting

Many brainteasers are worst-case minimization problems, worded either in terms of *minimizing* an action or in doing something at most a specific number of times. A useful technique is to try to "balance" the worst case. That is, if an early decision results in a skewing of the worst case, we can sometimes change the decision to balance out the worst case. This will be clearest when explained with an example.

The "nine balls" question is a classic interview question. You have nine balls. Eight are of the same weight, and one is heavier. You are given a balance which tells you only whether the left side or the right side is heavier. Find the heavy ball in just two uses of the scale.

A first approach is to divide the balls in sets of four, with the ninth ball sitting off to the side. The heavy ball is in the heavier set. If they are the same weight, then we know that the ninth ball is the heavy one. Replicating this approach for the remaining sets would result in a worst case of three weighings—one too many!

This is an imbalance in the worst case: the ninth ball takes just one weighing to discover if it's heavy, whereas others take three. If we *penalize* the ninth ball by putting more balls off to the side, we can lighten the load on the others. This is an example of "worst case balancing."

If we divide the balls into sets of three items each, we will know after just one weighing which set has the heavy one. We can even formalize this into a *rule*: given N balls, where N is divisible by 3, one use of the scale will point us to a set of $\frac{N}{3}$ balls with the heavy ball.

For the final set of three balls, we simply repeat this: put one ball off to the side and weigh two. Pick the heavier of the two. Or, if the balls are the same weight, pick the third one.

► Algorithm Approaches

If you're stuck, consider applying one of the approaches for solving algorithm questions (starting on page 67). Brainteasers are often nothing more than algorithm questions with the technical aspects removed. Base Case and Build and Do It Yourself (DIY) can be especially useful.

Additional Reading: Useful Math (pg 629).

Interview Questions

- 6.1 The Heavy Pill:** You have 20 bottles of pills. 19 bottles have 1.0 gram pills, but one has pills of weight 1.1 grams. Given a scale that provides an exact measurement, how would you find the heavy bottle? You can only use the scale once.

Hints: #186, #252, #319, #387

pg 289

- 6.2 Basketball:** You have a basketball hoop and someone says that you can play one of two games.

Game 1: You get one shot to make the hoop.

Game 2: You get three shots and you have to make two of three shots.

If p is the probability of making a particular shot, for which values of p should you pick one game or the other?

Hints: #181, #239, #284, #323

pg 290

- 6.3 Dominos:** There is an 8x8 chessboard in which two diagonally opposite corners have been cut off. You are given 31 dominos, and a single domino can cover exactly two squares. Can you use the 31 dominos to cover the entire board? Prove your answer (by providing an example or showing why it's impossible).

Hints: #367, #397

pg 291

- 6.4 Ants on a Triangle:** There are three ants on different vertices of a triangle. What is the probability of collision (between any two or all of them) if they start walking on the sides of the triangle? Assume that each ant randomly picks a direction, with either direction being equally likely to be chosen, and that they walk at the same speed.

Similarly, find the probability of collision with n ants on an n -vertex polygon.

Hints: #157, #195, #296

pg 291

- 6.5 Jugs of Water:** You have a five-quart jug, a three-quart jug, and an unlimited supply of water (but no measuring cups). How would you come up with exactly four quarts of water? Note that the jugs are oddly shaped, such that filling up exactly "half" of the jug would be impossible.

Hints: #149, #379, #400

pg 292

- 6.6 Blue-Eyed Island:** A bunch of people are living on an island, when a visitor comes with a strange order: all blue-eyed people must leave the island as soon as possible. There will be a flight out at 8:00 pm every evening. Each person can see everyone else's eye color, but they do not know their own (nor is anyone allowed to tell them). Additionally, they do not know how many people have blue eyes, although they do know that at least one person does. How many days will it take the blue-eyed people to leave?

Hints: #218, #282, #341, #370

pg 293

- 6.7 The Apocalypse:** In the new post-apocalyptic world, the world queen is desperately concerned about the birth rate. Therefore, she decrees that all families should ensure that they have one girl or else they face massive fines. If all families abide by this policy—that is, they have continue to have children until they have one girl, at which point they immediately stop—what will the gender ratio of the new generation be? (Assume that the odds of someone having a boy or a girl on any given pregnancy is equal.) Solve this out logically and then write a computer simulation of it.

Hints: #154, #160, #171, #188, #201

pg 293

- 6.8 The Egg Drop Problem:** There is a building of 100 floors. If an egg drops from the N th floor or above, it will break. If it's dropped from any floor below, it will not break. You're given two eggs. Find N , while minimizing the number of drops for the worst case.

Hints: #156, #233, #294, #333, #357, #374, #395

pg 296

- 6.9 100 Lockers:** There are 100 closed lockers in a hallway. A man begins by opening all 100 lockers. Next, he closes every second locker. Then, on his third pass, he toggles every third locker (closes it if it is open or opens it if it is closed). This process continues for 100 passes, such that on each pass i , the man toggles every i th locker. After his 100th pass in the hallway, in which he toggles only locker #100, how many lockers are open?

Hints: #139, #172, #264, #306

pg 297

- 6.10 Poison:** You have 1000 bottles of soda, and exactly one is poisoned. You have 10 test strips which can be used to detect poison. A single drop of poison will turn the test strip positive permanently. You can put any number of drops on a test strip at once and you can reuse a test strip as many times as you'd like (as long as the results are negative). However, you can only run tests once per day and it takes seven days to return a result. How would you figure out the poisoned bottle in as few days as possible?

FOLLOW UP

Write code to simulate your approach.

Hints: #146, #163, #183, #191, #205, #221, #230, #241, #249

pg 298

Additional Problems: Moderate Problems (#16.5), Hard Problems (#17.19)

Hints start on page 662.

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     }
9 }
```

```

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 $N \times N$ 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.