135 SOLID interview questions to ask your applicants

Questions

1. Imagine you're building a toy robot. How would you design its code so that adding new actions (like dancing or singing) doesn't require you to rewrite the whole robot's brain?

2. Let's say you have a box of LEGOs. Some are red, some are blue. How would you organize them so you can easily find all the red LEGOs without messing up the blue ones?

3. Pretend you have a magic wand that can do different things like make it rain or make it snow. How would you make sure each spell does only its job and doesn't accidentally change something else?

4. If you have a toy car that can drive and beep, how would you make sure you can change the way it beeps without having to rebuild the whole car?

5. You're building a program to draw shapes. How would you make sure you can add new shapes (like triangles or stars) without breaking the code that draws the existing shapes (like circles and squares)?

6. You have a remote control with buttons for volume, channel, and power. If you want to add a new button for 'record', how would you do it without changing the way the other buttons work?

7. If a class has multiple responsibilities, what problems can arise later in the project?

8. What do you understand by the 'Open/Closed Principle'? Give a real-world example.

9. Why is the 'Liskov Substitution Principle' important when using inheritance?

10. What is the main goal of the 'Interface Segregation Principle'?

11. Explain what 'Dependency Inversion Principle' means to you. Can you provide an example?

12. Can you describe a scenario where a class violates the Single Responsibility Principle, and what refactoring steps would you take?

13. Let's say you have a class that saves data to a database. If you later want to save data to a file instead, how would you design the class to make this change easy, applying SOLID principles?

14. Imagine you have a base class called 'Animal' with a method 'makeSound'. You have subclasses like 'Dog' and 'Cat'. How can you ensure that the 'Liskov Substitution Principle' is followed in this scenario?

15. Suppose you have an interface with many methods, but a class only needs to implement a few of them. How does this violate the Interface Segregation Principle, and how could you fix it?

16. Think of a system where different modules are tightly coupled. What are the disadvantages, and how can the Dependency Inversion Principle help to improve the design?

17. Describe a situation where applying SOLID principles might be overkill. What are the tradeoffs?

18. Have you ever encountered code that was difficult to maintain or extend? How could SOLID principles have helped in that situation?

19. How would you explain the benefits of using SOLID principles to a non-technical person?

20. If you have a class that depends on another class directly, how can you use Dependency Injection to improve the design?

21. How does adhering to SOLID principles affect the testability of your code?

22. Imagine you need to add a new feature to an existing system, and the original code doesn't follow SOLID principles. What are some potential challenges, and how would you approach the task?

23. Imagine you're building a toy car. How would you design it so you can easily swap out the wheels for different types, like big off-road tires or small racing tires, without changing the whole car?

24. Let's say you have a box of LEGOs. You want to build different things, like a house or a car, using the same LEGOs. How do you make sure each thing you build follows its own rules and doesn't mess up the other things?

25. If you have a robot that can do many things, like walk, talk, and dance, what's a good way to organize its abilities so it's easy to add new abilities without breaking the old ones?

26. Think about a set of instructions for making a sandwich. How do you make the instructions clear and simple so anyone can follow them, even if they've never made a sandwich before?

27. Suppose you have a program that prints reports. What are the problems with modifying the same function to print to the console, a file, or a printer?

28. Explain how you would approach designing a class that needs to be extended in the future but should not be modified directly.

29. If a class has too many responsibilities, what are some signs that it needs to be broken down into smaller, more focused classes?

30. How can you ensure that different modules of your code can be easily swapped out or replaced without affecting other parts of the application?

31. Why is it important to avoid tightly coupling different parts of your code, and what are some techniques to achieve loose coupling?

32. If you have a base class with several subclasses, how do you ensure that each subclass can be used wherever the base class is expected without unexpected behavior?

33. Consider a scenario where you have a system for managing different types of employees (e.g., hourly, salaried). How would you structure your code to adhere to the Single Responsibility Principle?

34. How does the Open/Closed Principle relate to writing maintainable and extensible code? Provide an example.

35. Describe a situation where violating the Liskov Substitution Principle could lead to unexpected bugs or errors in your application.

36. How does the Interface Segregation Principle help in designing cleaner and more focused interfaces for classes?

37. Explain the benefits of using Dependency Inversion Principle in managing dependencies between different modules of your code.

38. You have a class that handles both user authentication and session management. How can you refactor it to follow the Single Responsibility Principle?

39. Imagine you have a logging class. How could you design it so it can easily support different logging targets (e.g., file, database, console) without modifying the core class?

40. You have a base class Animal with a method makeSound(). You create a subclass Dog that overrides makeSound() to bark. What could go wrong if a Cat class tried to extend Dog instead of Animal directly?

41. You have an interface with several methods, but a class only needs to implement one of them. How can you apply the Interface Segregation Principle to improve the design?

42. How can you use Dependency Injection to make your classes more testable and less dependent on concrete implementations?

43. You have a class that calculates area of different shapes and the logic is implemented using if/else conditions. What are the SOLID principles being violated and how can it be refactored?

44. Why might it be a bad idea to create a general-purpose class that attempts to handle every possible type of input or situation?

45. Let's say you have a function that needs to perform several steps, such as reading data from a file, processing the data, and writing the results to a database. How might you apply the Single Responsibility Principle to this function?

46. You have a class that sends notifications to users, but it currently only supports email. How can you design it so that it can easily support other notification methods like SMS or push notifications in the future, following the Open/Closed Principle?

47. You have a base class called Shape with methods to calculate area and perimeter. You create a subclass called Rectangle. What problems might arise if you later create a subclass called Circle that inherits from Shape?

48. You have an interface called Worker with methods for working, eating, and sleeping. However, not all workers need to eat or sleep. How can you improve this design using the Interface Segregation Principle?

49. How can you use a Dependency Injection container to manage dependencies between different parts of your application, and what are the benefits of doing so?

50. What are some practical ways you can determine if a class is violating the Single Responsibility Principle in a real-world project?

51. Explain a scenario where using inheritance might not be the best approach, and how you could achieve the same result using composition, and how does it relate to SOLID principles?

52. Explain the Liskov Substitution Principle with a real-world analogy, focusing on a scenario where violating it leads to unexpected behavior.

53. How can you identify potential violations of the Dependency Inversion Principle in existing code, and what refactoring strategies can you use to address them?

54. Describe a situation where applying the Single Responsibility Principle might lead to a proliferation of small classes, and how you would manage that complexity.

55. Discuss the trade-offs between adhering strictly to the Open/Closed Principle and the need for timely feature delivery.

56. Explain how the SOLID principles can contribute to building a more maintainable microservices architecture.

57. How does understanding SOLID principles aid in debugging and troubleshooting complex software systems?

58. Describe a design pattern that complements or reinforces one of the SOLID principles, and explain how they work together.

59. How can you use unit testing to verify adherence to the SOLID principles in your code?

60. Explain how the SOLID principles relate to the concept of code cohesion and coupling.

61. How can you balance the benefits of SOLID principles with the potential for over-engineering in a project?

62. Illustrate the impact of the Interface Segregation Principle on client code when dealing with a complex interface.

63. Explain a scenario where applying the Open/Closed Principle might introduce unnecessary abstraction.

64. How can the Dependency Inversion Principle improve the testability of a software component?

65. Describe how violating the Liskov Substitution Principle can lead to unexpected runtime errors.

66. Discuss strategies for educating a development team about the SOLID principles and promoting their adoption.

67. How can you use static analysis tools to help enforce the SOLID principles in a codebase?

68. Explain how the SOLID principles relate to the concept of code reusability.

69. How can you refactor a class that violates the Single Responsibility Principle into multiple, more focused classes?

70. Describe a situation where applying the Interface Segregation Principle might increase code complexity.

71. Discuss the role of SOLID principles in agile software development methodologies.

72. Explain the relationship between SOLID principles and design patterns.

73. How does the Single Responsibility Principle aid in parallel development by multiple developers?

74. Explain a time when you had to make a judgement call to deviate from a SOLID principle. What considerations led to that decision?

75. Let's say you have an anti-corruption layer. How can the SOLID principles be applied "within" that layer to improve maintainability.

76. How do the SOLID principles apply in a dynamically typed language compared to a statically typed language?

77. Imagine you inherit a large legacy system. What's your approach for applying SOLID principles gradually without disrupting existing functionality?

78. How do SOLID principles relate to Domain-Driven Design (DDD) concepts like Entities, Value Objects, and Aggregates?

79. What are the challenges of applying SOLID to data-heavy applications that rely heavily on database interactions and object-relational mapping (ORM)?

80. Let's say you have a class that requires multiple dependencies. What are some strategies, in line with SOLID, to manage that class's dependencies effectively?

81. Explain how SOLID principles can guide the design of RESTful APIs and promote their long-term evolution.

82. How have you used SOLID principles to refactor legacy code, and what were the biggest challenges you faced?

83. Describe a time when you intentionally violated a SOLID principle and why.

84. Explain how you would design a system to be highly extensible using SOLID principles, providing specific examples.

85. What are some potential drawbacks of strictly adhering to SOLID principles in every situation?

86. How do SOLID principles relate to other design patterns, such as strategy or template method?

87. Explain how you would test code that adheres to SOLID principles versus code that does not.

88. Describe a scenario where applying the Interface Segregation Principle improved the maintainability of a project.

89. How do you ensure that your team understands and applies SOLID principles consistently?

90. Explain how the Liskov Substitution Principle can prevent unexpected behavior in a system.

91. Discuss how SOLID principles contribute to reducing technical debt in a software project.

92. How would you approach a code review to identify violations of SOLID principles?

93. Describe a situation where applying the Dependency Inversion Principle made testing easier.

94. Explain how you balance SOLID principles with other important considerations like performance and time constraints.

95. Discuss how SOLID principles relate to microservices architecture.

96. How do you handle situations where different SOLID principles seem to conflict with each other?

97. Explain how you would convince a team to adopt SOLID principles if they are resistant to change.

98. Describe how you have used SOLID principles in the context of a specific project you worked on, detailing the before and after.

99. How would you explain the Open/Closed Principle to a junior developer in a way that is easy to understand?

100. Explain how SOLID principles can help in designing a RESTful API.