50 Object Oriented Programming Interview QnA

Made By:

Yadneyesh (Curious Coder)
CodWithCurious.com

1. **What is Object-Oriented Programming (OOP)?**
   Object-Oriented Programming (OOP) is a programming paradigm that organizes code around objects, which are instances of classes. It emphasizes the concepts of encapsulation, inheritance, and polymorphism.
2. **What is a class in OOP?**
   In OOP, a class is a blueprint or template that defines the properties (attributes) and behaviors (methods) that objects of that class will have. It serves as a blueprint for creating objects.
3. **What is an object in OOP?**
   An object is an instance of a class. It represents a real-world entity and encapsulates both data (attributes) and behavior (methods).
4. **What is encapsulation in OOP?**
   Encapsulation is the process of bundling data and methods together within a class, hiding the internal details of how an object works. It provides data abstraction and protects the data from external access.
5. **What is inheritance in OOP?**
   Inheritance is a mechanism in OOP that allows a class (subclass) to inherit the properties and methods of another class (superclass). It promotes code reusability and supports the "is-a" relationship between classes.
6. **What is polymorphism in OOP?**
   Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method to be used with objects of different classes, providing flexibility and extensibility.
7. **What are the four pillars of OOP?**
   The four pillars of OOP are:
   a. Encapsulation: Bundling data and methods within a class.
   b. Inheritance: Inheriting properties and methods from a superclass.
   c. Polymorphism: Treating objects of different classes as objects of a common superclass.

d. Abstraction: Simplifying complex systems by modeling relevant objects.

8. **What is abstraction in OOP?**
   Abstraction is the process of simplifying complex systems by modeling relevant objects and ignoring unnecessary details. It focuses on the essential features and behavior of an object.
9. **What is a constructor in OOP?**
   A constructor is a special method in a class that is called when an object of that class is created. It is used to initialize the object's state and perform any necessary setup.
10. **What is method overriding in OOP?**
    Method overriding is the ability of a subclass to provide a different implementation of a method that is already defined in its superclass. It allows the subclass to modify the behavior of the inherited method.
11. **What is method overloading in OOP?**
    Method overloading is the ability to define multiple methods with the same name but different parameters in a class. It provides flexibility in invoking methods with different argument combinations.
12. **What is the difference between method overriding and method overloading?**
    Method overriding involves providing a different implementation of a method in a subclass, whereas method overloading involves defining multiple methods with the same name but different parameters within a class.
13. **What is a static method in OOP?**
    A static method is a method that belongs to the class itself rather than an instance of the class. It can be called directly on the class without creating an object.
14. **What is a static variable in OOP?**
    A static variable is a variable that is shared among all instances of a class. It is associated with the class rather than specific instances, and its value remains the same across different objects.
15. **What is an instance variable in OOP?**
    An instance variable is a variable that is unique to each instance (object) of a class. Each object has its own copy of instance variables, and their values can vary from object to object.
16. **What is a constructor overloading?**
    Constructor overloading is the ability to define multiple constructors within a class with different parameter combinations. It allows objects to be created with different initialization options.
17. **What is an abstract class in OOP?**
    An abstract class is a class that cannot be instantiated and serves as a base for

deriving subclasses. It may contain abstract methods (without implementation) and can define common behavior for its subclasses.

18. **What is an interface in OOP?**
    An interface is a collection of abstract methods that define a contract for classes that implement it. It specifies the methods that a class must implement but does not provide any implementation itself.
19. **What is multiple inheritance in OOP?**
    Multiple inheritance is the ability of a class to inherit properties and methods from multiple parent classes. However, most programming languages do not support multiple inheritance to avoid ambiguity and complexity.
20. **What is a package in OOP?**
    A package is a mechanism for organizing related classes and interfaces into a single namespace. It helps avoid naming conflicts and provides a modular structure to manage code effectively.
21. **What is a namespace in OOP?**
    A namespace is a container that holds a set of related classes, interfaces, and other objects. It provides a way to group and organize code elements and avoids naming conflicts.
22. **What is the difference between a class and an object in OOP?**
    A class is a blueprint or template that defines the properties and behaviors of objects, whereas an object is an instance of a class. A class represents the general characteristics, while an object represents a specific instance.
23. **What is the difference between a superclass and a subclass in OOP?**
    A superclass is a class from which other classes (subclasses) inherit properties and methods. It is more general and provides common behavior. A subclass is a class that inherits from a superclass and can extend or modify its behavior.
24. **What is the difference between composition and inheritance?**
    Composition is a design technique where a class is composed of objects of other classes as members. It emphasizes "has-a" relationships. Inheritance is a mechanism where a class inherits properties and methods from another class, emphasizing "is-a" relationships.
25. **What is the SOLID principle in OOP?**
    SOLID is an acronym for a set of design principles:
    a. Single Responsibility Principle (SRP)
    b. Open/Closed Principle (OCP)
    c. Liskov Substitution Principle (LSP)
    d. Interface Segregation Principle (ISP)
    e. Dependency Inversion Principle (DIP)
       These principles aim to make software designs more modular, maintainable, and

extensible.

26. **What is the Single Responsibility Principle (SRP)?**
    The Single Responsibility Principle states that a class should have only one reason to change. It means that a class should have a single responsibility or purpose, encapsulating a single concept or functionality.
27. **What is the Open/Closed Principle (OCP)?**
    The Open/Closed Principle states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. It promotes the use of abstraction and inheritance to allow behavior to be extended without modifying existing code.
28. **What is the Liskov Substitution Principle (LSP)?**
    The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. It ensures that subclasses adhere to the contract defined by the superclass.
29. **What is the Interface Segregation Principle (ISP)?**
    The Interface Segregation Principle states that clients should not be forced to depend on interfaces they do not use. It promotes the creation of small, specific interfaces rather than large general-purpose interfaces.
30. **What is the Dependency Inversion Principle (DIP)?**
    The Dependency Inversion Principle states that high-level modules/classes should not depend on low-level modules/classes directly. Instead, they should depend on abstractions (interfaces or abstract classes). It promotes loose coupling and easier substitution of implementations.
31. **What is the difference between composition and aggregation?**
    Composition and aggregation are both forms of association between classes:
    a. Composition implies a strong relationship where the lifetime of the contained objects is controlled by the container object.
    b. Aggregation implies a weaker relationship where the contained objects can exist independently of the container object.
32. **What is method hiding in OOP?**
    Method hiding occurs when a subclass defines a static method with the same name as a static method in its superclass. It hides the superclass method, and the method resolution is determined by the type of the reference at compile time, not the actual object at runtime.
33. **What is the difference between early binding and late binding?**
    Early binding (static binding) is the process of binding a method call to its implementation at compile time based on the type of the reference. Late binding

(dynamic binding) is the process of determining the method implementation at runtime based on the actual object type.

34. **What is the difference between shallow copying and deep copying?**
    Shallow copying creates a new object that references the same memory locations as the original object. Changes to the copied object may affect the original object. Deep copying creates a new object and recursively copies all referenced objects, ensuring complete independence.
35. **What is a design pattern in OOP?**
    A design pattern is a reusable solution to a commonly occurring problem in software design. It provides a proven approach to solve specific design challenges and promotes code reuse, modularity, and maintainability.
36. **What is the Observer pattern?**
    The Observer pattern is a behavioral design pattern where objects (observers) are notified automatically when the state of another object (subject) changes. It establishes a one-to-many dependency between objects, allowing efficient communication and loose coupling.
37. **What is the Factory pattern?**
    The Factory pattern is a creational design pattern that provides an interface for creating objects, but delegates the responsibility of instantiation to subclasses. It allows the creation of objects without specifying their concrete classes, promoting flexibility and encapsulation.
38. **What is the Singleton pattern?**
    The Singleton pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to it. It is often used when there should be exactly one instance of a class, such as a database connection or a logger.
39. **What is the Builder pattern?**
    The Builder pattern is a creational design pattern that separates the construction of complex objects from their representation. It allows the step-by-step creation of objects with different configurations, ensuring a clean and readable construction process.
40. **What is the Prototype pattern?**
    The Prototype pattern is a creational design pattern that allows objects to be copied or cloned. It provides a way to create new objects by cloning existing objects, eliminating the need for repeated construction and initialization.
41. **What is the MVC pattern?**
    The Model-View-Controller (MVC) pattern is an architectural pattern that separates an application into three main components: the Model (data and business logic), the

View (presentation layer), and the Controller (handles user input and coordinates the Model and View).

42. **What is the MVVM pattern?**
    The Model-View-ViewModel (MVVM) pattern is an architectural pattern that enhances the MVC pattern by introducing a ViewModel. The ViewModel acts as an intermediary between the View and Model, providing data-binding capabilities and facilitating UI updates.
43. **What is the Dependency Injection pattern?**
    The Dependency Injection pattern is a design pattern that allows the inversion of control, where dependencies of a class are provided externally rather than created internally. It promotes loose coupling, testability, and flexibility in component composition.
44. **What is the Command pattern?**
    The Command pattern is a behavioral design pattern that encapsulates a request as an object, allowing parameterization of clients with different requests, queueing or logging of requests, and support for undoable operations.
45. **What is the Template Method pattern?**
    The Template Method pattern is a behavioral design pattern that defines the skeleton of an algorithm in a base class but delegates the implementation of certain steps to subclasses. It allows subclasses to redefine certain steps while preserving the overall algorithm structure.
46. **What is the Iterator pattern?**
    The Iterator pattern is a behavioral design pattern that provides a way to access elements of an aggregate object sequentially without exposing its underlying representation. It separates the traversal logic from the object being traversed.
47. **What is the Composite pattern?**
    The Composite pattern is a structural design pattern that allows you to compose objects into tree structures and treat individual objects and compositions uniformly. It represents part-whole hierarchies and simplifies the client's interaction with complex object structures.
48. **What is the Decorator pattern?**
    The Decorator pattern is a structural design pattern that allows behavior to be added to an object dynamically. It provides an alternative to subclassing for extending functionality and promotes the principle of "open for extension, closed for modification."
49. **What is the Adapter pattern?**
    The Adapter pattern is a structural design pattern that allows incompatible interfaces of different classes to work together. It acts as a bridge between two

incompatible interfaces, converting the interface of one class into another interface that clients expect.

50. **What is the Proxy pattern?**
    The Proxy pattern is a structural design pattern that provides a surrogate or placeholder for another object to control access to it. It allows additional functionality to be added when accessing the original object, such as lazy initialization, access control, or caching.