OOP Interview Preparation Guide

Memory updated

To cover **100% interview preparation for Object-Oriented Programming (OOP)** concepts in big tech MNCs, here are the **most common questions** and the problems associated with them, grouped by key OOP principles. These cover the fundamental topics interviewers focus on:

---

1. Four Pillars of OOP

- **Encapsulation**:
  - **Question**: What is encapsulation, and why is it important?
  - **Problem**: Design a class that represents a `BankAccount` with encapsulated attributes like `balance` and methods to `deposit` and `withdraw`. Ensure balance can't be modified directly.

- **Abstraction**:
  - **Question**: Explain abstraction with an example.
  - **Problem**: Create an abstract class `Shape` with methods `area()` and `perimeter()`. Implement derived classes for `Circle`, `Rectangle`, and `Triangle`.

- **Inheritance**:
  - **Question**: What is inheritance in OOP? How does it promote code reusability?
  - **Problem**: Implement a base class `Vehicle` and derive classes like `Car`, `Bike`, and `Truck`. Show how each derived class inherits common functionality and adds its own features.

- **Polymorphism**:
  - **Question**: What is polymorphism, and how is it achieved in OOP?
  - **Problem**: Implement method overloading and overriding in a class hierarchy of `Animal` with methods like `makeSound()`. Show how different animals (`Dog`, `Cat`, etc.) make different sounds using polymorphism.

---

2. Classes and Objects

- **Question**: What is the difference between a class and an object in OOP?
- **Problem**: Write a `Student` class with attributes like `name`, `rollNumber`, and `marks`. Create objects of this class and show how they represent individual students.

---

3. Constructors and Destructors

- **Question**: What is the role of constructors and destructors in a class?
- **Problem**: Design a `Book` class with attributes `title`, `author`, and `price`. Implement parameterized constructors to initialize the values and a destructor to clean up resources.

4. Access Specifiers (public, private, protected)

- **Question**: How do access specifiers work in OOP?
- **Problem**: Design a `Person` class with private attributes like `name` and `age`, and public methods to access and modify these values.

---

5. Composition vs Inheritance

- **Question**: What is the difference between composition and inheritance? Provide examples.
- **Problem**: Create a `Library` class that contains a collection of `Book` objects (composition). Also, demonstrate inheritance by extending a `PrintedBook` class from `Book`.

---

6. Static vs Instance Members

- **Question**: What are static members in a class? How are they different from instance members?
- **Problem**: Create a `Counter` class with a static variable that keeps track of how many objects have been created from the class.

---

7. Operator Overloading (C++)

- **Question**: What is operator overloading, and how is it used in C++?
- **Problem**: Overload the `+` operator for a `ComplexNumber` class to add two complex numbers.

---

8. Virtual Functions and Pure Virtual Functions (C++)

- **Question**: What is the difference between a virtual function and a pure virtual function?
- **Problem**: Design a class `Animal` with a virtual function `makeSound()`. Create derived classes `Dog` and `Cat`, each implementing `makeSound()` differently. Also, make `Animal` a pure abstract class by making `makeSound()` a pure virtual function.

---

9. Interface vs Abstract Class (Java)

- **Question**: What is the difference between an interface and an abstract class?
- **Problem**: Create an interface `Movable` with methods `move()` and `stop()`. Implement this interface in two classes, `Car` and `Robot`.

---

10. Multiple Inheritance (C++ vs Java)

- **Question**: How is multiple inheritance handled in C++ and Java? What are the issues with multiple inheritance, and how does Java solve them?
- **Problem**: Implement a scenario in C++ where two base classes are inherited by a derived class using multiple inheritance. Show how the diamond problem occurs and how to resolve it using virtual inheritance.

---

11. Exception Handling

- **Question**: How is exception handling implemented in OOP?
- **Problem**: Implement a class `Division` that divides two numbers. Use exception handling to catch divide-by-zero errors.

---

12. Design Patterns

- **Question**: What are design patterns? Can you name a few important ones?
  - **Singleton**: Design a `DatabaseConnection` class where only one instance of the connection is allowed (Singleton pattern).
  - **Factory**: Implement a `ShapeFactory` that creates objects of different `Shape` classes (Factory pattern).
  - **Observer**: Implement a `StockObserver` that monitors changes in stock prices and notifies subscribers.

---

13. SOLID Principles

- **Question**: What are the SOLID principles of object-oriented design?
  - **Single Responsibility Principle**: Design a class `ReportGenerator` that has the single responsibility of generating a report.
  - **Open/Closed Principle**: Show how a `PaymentProcessor` class can be extended to support new payment methods without modifying the existing code.

---

14. Memory Management (C++)

- **Question**: How is memory managed in C++? Explain the role of destructors, new/delete, and smart pointers.
- **Problem**: Implement a class that dynamically allocates memory for a data member using `new`, and ensures proper cleanup using a destructor and smart pointers.

---

15. Copy Constructor and Assignment Operator (C++)

- **Question**: What is the purpose of a copy constructor and assignment operator? How do they differ?
- **Problem**: Implement a `Person` class with deep copy functionality by overriding the copy constructor and assignment operator.

These topics will give you comprehensive coverage of OOP for big tech interviews. Dive deep into each area, practice writing code, and be prepared to explain concepts clearly in interviews.