OBJECT-ORIENTED PROGRAMMING (OOP) — TEST

SECTION A: MULTIPLE CHOICE

(Choose ONE correct answer)

1. What best describes Object-Oriented Programming (OOP)?
   A. A programming style that focuses on functions and logic
   B. A programming paradigm that organizes code around objects with state and behavior
   C. A low-level programming approach close to hardware
   D. A way to write code without data

2. Which problem does OOP primarily solve compared to procedural programming?
   A. Faster execution speed
   B. Reduced memory usage
   C. Better management of large and complex systems
   D. Elimination of bugs

3. What happens if you do NOT define a constructor in a Java class?
   A. The program will not compile
   B. The object cannot be created
   C. Java provides a default constructor
   D. The class becomes abstract

4. Which access modifier best supports encapsulation?
   A. public
   B. default
   C. protected
   D. private

5. Which statement about interfaces is TRUE?
   A. Interfaces can store instance variables
   B. A class can extend multiple interfaces
   C. A class can implement multiple interfaces
   D. Interfaces can have constructors

SECTION B: SHORT ANSWER

6. What is the difference between a class and an object?

7. Explain the difference between abstraction and encapsulation in one or two sentences.

8. What does an “is-a” relationship mean? Give one example.

9. Why is composition often preferred over inheritance? List TWO reasons.

10. What is the fragile base class problem?

SECTION C: CODE & CONCEPTS

11. What will the following code print, and why?

```
Animal a = new Dog();
a.sound();
```

12. Is the following an example of method overloading or method overriding? Explain briefly.

```
int add(int a, int b)
double add(double a, double b)
```

13. What happens if a subclass defines a method with the same name as a parent method, but with a different parameter list?

14. Why can static methods not be overridden?

SECTION D: SCENARIO-BASED QUESTIONS

15. You are designing a system with Car, Bike, and Bus.
    All need a move() method but do not share internal data.
    Should you use an abstract class or an interface? Why?

16. A Car class needs an Engine.
    Should Car inherit from Engine or use composition? Explain.

17. Give a real-world example (not animals) that demonstrates polymorphism.

SECTION E: TRUE / FALSE

(If false, explain why)

18. A class must have a constructor to work.

19. Method overloading is resolved at runtime.

20. Interfaces help achieve loose coupling.

SECTION F: DESIGN & ARCHITECTURE

21. Define tight coupling and loose coupling in your own words.

22. Why are interfaces especially useful in large team-based systems? List TWO reasons.

23. When does inheritance become a bad design choice?

BONUS (INTERVIEW-LEVEL)

24. Explain Object-Oriented Programming to a non-technical manager in about 30 seconds.

ANSWER KEY

1. B
2. C
3. C
4. D
5. C
6. A class is a blueprint that defines properties and methods. An object is an instance of a class created in memory with actual values and behavior.
7. Abstraction hides complexity by exposing only what an object does, while encapsulation protects internal data by controlling access through methods.
8. An “is-a” relationship means a subclass can be treated as its parent type. Example: Dog is an Animal.
9. Reduces tight coupling, More flexible and easier to change, Avoids fragile base class problems, Allows behavior to be swapped at runtime. (Any two earn full credit.)
10. When changes in a parent class unintentionally break subclasses, even though the subclass code did not change.
11. It prints the Dog implementation of sound() (e.g., “Dog barks”). Reason: Runtime polymorphism—method calls are resolved at runtime using dynamic dispatch.
12. This is method overloading because the methods have the same name but different parameter types and are resolved at compile time.
13. This results in method overloading, not overriding. Polymorphism does not occur.
14. Static methods cannot be overridden because they belong to the class, not to instances. They are resolved at compile time.
15. Use an interface because the classes share behavior (move()) but not state, and each class can have its own implementation.
16. Use composition. A Car has an Engine, not is an Engine. Composition is more flexible and avoids tight coupling.
17. A payment system where pay() behaves differently for credit card, PayPal, or Apple Pay.
18. **False**. A class does not need a constructor; Java provides a default one if none is defined.
19. **False**. Overloading is resolved at compile time, not runtime.
20. **True**. Interfaces promote loose coupling by allowing code to depend on abstractions instead of implementations.
21. Tight coupling means classes depend on concrete implementations. Loose coupling means classes depend on abstractions, making systems more flexible.
22. Enable parallel team development, Improve testability (mocking), Provide clear contracts, Reduce dependency between components. (Any two earn full credit.)
23. When the relationship is not truly “is-a,” behavior varies significantly, or flexibility is required—composition is usually better.
24. OOP models software using real-world objects with responsibilities, making systems easier to understand, maintain, and scale.