1. What is encapsulation in OOP?

Answer: Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit known as a class.

2. Why is encapsulation important in OOP?

Answer: Encapsulation helps protect data from unauthorized access, ensuring that the internal state of an object can be maintained and controlled.

3. How is encapsulation achieved in Java?

Answer: Encapsulation in Java is achieved by declaring class variables as private and providing public methods (getters and setters) to access and modify those variables.

4. Explain the principle of inheritance in OOP.

Answer: Inheritance is the process by which a new class (subclass or derived class) can inherit properties and behaviors from an existing class (superclass or base class).

5. What are the advantages of inheritance in OOP?

Answer: Inheritance promotes code reuse, establishes a hierarchical relationship between classes, and allows the extension and modification of existing classes.

6. Differentiate between "is-a" and "has-a" relationships in the context of inheritance.

Answer: "Is-a" relationship represents inheritance, where a subclass is a type of its superclass. "Has-a" relationship represents composition, where a class contains an instance of another class as a member.

7. Explain the "super" keyword in Java.

Answer: The "super" keyword is used in Java to access the members (fields or methods) of the superclass from a subclass.

8. What is polymorphism in OOP?

Answer: Polymorphism is the ability of objects of different classes to be treated as objects of a common superclass. It allows methods to be called on objects of different classes with the same name, producing behavior based on the actual class of the object.

9. What are the two types of polymorphism in Java?

Answer: There are two types of polymorphism in Java: compile-time (method overloading) and runtime (method overriding).

10. Explain method overloading in Java.

Answer: Method overloading is the practice of defining multiple methods in the same class with the same name but different parameter lists.

11. What is dynamic or runtime polymorphism?

Answer: Dynamic or runtime polymorphism occurs when the method that should be invoked is determined at runtime based on the actual object type.

12. What is an abstract class in Java?

Answer: An abstract class is a class that cannot be instantiated and is typically used as a base class for other classes. It may contain abstract methods that are meant to be implemented by its subclasses.

13. Explain the "final" keyword in the context of OOP.

Answer: In OOP, the "final" keyword is used to indicate that a class, method, or variable cannot be further extended, overridden, or modified.

14. What is the "interface" in Java, and how does it relate to OOP?

Answer: An interface in Java defines a contract for classes that implement it. It enforces the implementation of specified methods, promoting a form of polymorphism.

15. Can a class implement multiple interfaces in Java?

Answer: Yes, a class in Java can implement multiple interfaces, allowing it to inherit and define behavior from multiple sources.

16. What is method overriding in Java, and how does it relate to polymorphism?

Answer: Method overriding is the process of providing a specific implementation of a method in a subclass to replace the implementation in the superclass. It's a key element of runtime polymorphism.

17. What is the difference between method overloading and method overriding?

Answer: Method overloading involves defining methods with the same name but different parameter lists within the same class, while method overriding occurs when a subclass provides a specific implementation for a method already defined in the superclass.

18. What is a constructor in Java, and can it be inherited?

Answer: A constructor is a special method used to initialize objects. Constructors are not inherited, but a subclass can call the constructor of its superclass using the "super" keyword.

19. Explain the "instanceof" operator in Java and its use in polymorphism.

Answer: The “instanceof” operator checks if an object is an instance of a specific class or interface. It is commonly used to determine the type of an object before performing operations.

20. What is method hiding in Java, and how does it relate to inheritance?

Answer: Method hiding occurs when a subclass defines a static method with the same signature as a static method in its superclass. This method is not subject to polymorphism.

21. What is composition in OOP, and how does it differ from inheritance?

Answer: Composition is the practice of creating objects of one class inside another class. It differs from inheritance in that it does not establish an “is-a” relationship but rather a “has-a” relationship between classes.

22. Explain the concept of “abstraction” in OOP.

Answer: Abstraction is the process of simplifying complex reality by modeling classes based on their essential properties and behaviors, hiding the irrelevant details.

23. What is a constructor chaining in Java, and how does it work with inheritance?

Answer: Constructor chaining is the practice of calling one constructor from another constructor within the same class or between base and derived classes. Inheritance allows a subclass to call the constructor of its superclass using “super.”

24. What is a base class and a derived class in inheritance?

Answer: The base class (superclass) is the class that is being inherited from, while the derived class (subclass) is the class that inherits the properties and behaviors of the base class.

25. What are “upcasting” and “downcasting” in Java?

Answer: Upcasting is the process of casting an object of a derived class to its base class type. Downcasting is the reverse, casting an object of a base class to its derived class type.

26. What is the “finalizer” method in Java, and how does it relate to OOP?

Answer: The “finalizer” method (finalize()) is used for cleanup operations before an object is garbage collected. It’s related to OOP because it’s a part of object lifecycle management.

27. Explain the principle of composition over inheritance.

Answer: Composition over inheritance suggests that it is often more flexible and maintainable to build classes by composing them with other classes rather than inheriting from them.

28. What is the “protected” access modifier in Java, and when is it typically used in OOP?

Answer: The “protected” access modifier allows access within the same package and by subclasses outside the package. It is typically used when you want to make certain members accessible to subclasses while still restricting access from outside classes.

29. What is a “sealed class” in Java, and how does it relate to OOP?

Answer: A sealed class restricts which other classes or interfaces can extend or implement it. It enforces a controlled hierarchy, which is useful in maintaining design integrity.

30. What is a “composition relationship” in UML diagrams, and how is it represented?

Answer: A composition relationship represents a strong “part-of” relationship between a whole (composite) and its parts. It is represented by a filled diamond arrow from the whole to the parts.

31. What is the “method signature” in the context of method overloading and overriding?

Answer: The method signature includes the method name and the parameter list (type and number of parameters). It is used to uniquely identify and differentiate methods.

32. How does the “final” keyword impact inheritance and polymorphism in Java?

Answer: The “final” keyword can be used to make a class or method unmodifiable. If a class is declared as “final,” it cannot be extended, and if a method is declared as “final,” it cannot be overridden.

33. What is the “superclass constructor” in Java, and how is it called from a subclass constructor?

Answer: The superclass constructor is called using the “super” keyword in the subclass constructor. It allows you to invoke the constructor of the superclass and initialize its properties.

34. Explain the difference between “static” and “instance” members in classes.

Answer: “Static” members belong to the class itself and are shared among all instances, while “instance” members belong to individual objects created from the class.

35. How does polymorphism enable more flexible and extensible software design?

Answer: Polymorphism allows you to write code that works with objects of different classes in a consistent manner, making your code more adaptable to changes and extensions.

36. What are the three fundamental principles of OOP, and how do they relate to each other?

Answer: The three fundamental principles of OOP are encapsulation, inheritance, and polymorphism. They work together to promote modularity, reusability, and extensibility in software design.

37. What is the purpose of the “abstract” keyword in Java, and how does it facilitate OOP?

Answer: The “abstract” keyword is used to define abstract classes and methods that serve as blueprints for derived classes. It enforces the implementation of certain behaviors in subclasses.

38. What is the Liskov Substitution Principle (LSP), and how does it relate to inheritance in OOP?

Answer: LSP states that objects of a derived class should be able to replace objects of the base class without affecting the correctness of the program. It emphasizes the proper use of inheritance.

39. What is a “virtual function” in C++ and how does it enable dynamic polymorphism?

Answer: In C++, a virtual function is a function declared as “virtual” in a base class and overridden in derived classes. It allows the selection of the appropriate function implementation at runtime, enabling dynamic polymorphism.

40. Explain the concept of “method dispatch” in the context of polymorphism.

Answer: Method dispatch is the process of determining which method to call when a method is invoked on an object. It is based on the actual type of the object, enabling polymorphic behavior.

41. What are the advantages and disadvantages of using inheritance in OOP?

Answer: Advantages of inheritance include code reuse and the establishment of hierarchical relationships. Disadvantages include increased complexity and tight coupling between classes.

42. What is the purpose of the “sealed” keyword in C# and how does it impact inheritance?

Answer: The “sealed” keyword in C# is used to prevent further derivation of a class. It limits the inheritance hierarchy and enforces design constraints.

43. What is the “composition root” in dependency injection and how does it relate to object creation?

Answer: The composition root is the entry point where the entire object graph is constructed and wired together in a dependency injection container. It is responsible for creating and managing objects within the application.

44. What is "multiple inheritance," and how is it implemented in programming languages that support it?

Answer: Multiple inheritance is the ability of a class to inherit from multiple base classes. In languages that support it, it can lead to issues like the "diamond problem." Solutions include virtual inheritance and interfaces.

45. What is the "base class pointer" and "derived class pointer" in C++, and how do they relate to polymorphism?

Answer: A "base class pointer" can point to an object of a derived class. When a base class pointer is used to call a method, it invokes the method of the derived class, enabling polymorphic behavior.

46. What is the "is-a" relationship in the context of inheritance, and how does it affect class hierarchies?

Answer: The "is-a" relationship indicates that a subclass is a type of its superclass. It forms the basis for defining class hierarchies and inheritance relationships.

47. What is the concept of "composition reuse" as an alternative to inheritance in OOP?

Answer: Composition reuse involves creating classes that contain instances of other classes, allowing for greater flexibility and adaptability compared to traditional inheritance.

48. What is the "Diamond Problem" in the context of multiple inheritance, and how is it resolved in some programming languages?

Answer: The "Diamond Problem" occurs when a class inherits from two or more classes with a common base class. It can lead to ambiguity in method resolution. It is resolved in some languages through mechanisms like virtual inheritance.

49. What is "interface segregation" and "dependency inversion" in SOLID principles, and how do they relate to OOP design?

Answer: "Interface segregation" is about creating focused interfaces, and "dependency inversion" suggests that high-level modules should not depend on low-level modules. Both principles emphasize modularity and flexibility in OOP design.

50. What is the "Circle-Ellipse Problem," and how does it relate to violations of the Liskov Substitution Principle in OOP?

Answer: The “Circle-Ellipse Problem” is an example of a violation of the Liskov Substitution Principle, where a Circle class inherits from an Ellipse class. It demonstrates the challenges in preserving invariants in a subclass when it does not adhere to the “is-a” relationship.