Java OOP Concepts – Theory Questions & Answers

Class & Object

Q: What is a class?

Answer: A class is a blueprint or template used to create objects. It defines properties (variables) and behaviors (methods).

Q: What is an object?

Answer: An object is an instance of a class that contains real data and can perform actions defined in the class.

Q: Difference between class and object?

Answer: A class is a blueprint, while an object is the actual entity created from that blueprint.

Q: Can a class exist without objects?

Answer: Yes. A class can exist without creating objects, but it becomes useful only when objects are created.

Q: What is object creation?

Answer: Object creation means allocating memory for a class using the new keyword.

Q: What is memory allocation for objects?

Answer: Memory for objects is allocated in the heap memory when using the new keyword.

Q: What is a reference variable?

Answer: A reference variable stores the address of an object, not the object itself.

Q: What happens when object is created?

Answer: Memory is allocated, constructor is executed, and the reference is returned.

Q: Difference between instance and class variable?

Answer: Instance variables belong to objects, while class (static) variables belong to the class itself.

Q: Why do we need objects?

Answer: Objects allow us to model real-world entities and use class features in programs.

Constructor

Q: What is a constructor?

Answer: A constructor is a special method used to initialize objects when they are created.

Q: Types of constructors?

Answer: Default constructor and parameterized constructor.

Q: Can constructor be private?

Answer: Yes, constructors can be private to restrict object creation.

Q: Constructor vs method?

Answer: Constructor initializes objects and has no return type; methods perform operations.

Q: Can constructor be overloaded?

Answer: Yes, multiple constructors can exist with different parameters.

Q: Default constructor meaning?

Answer: A constructor automatically provided by Java if none is defined.

Q: What if constructor not defined?

Answer: Java provides a default constructor automatically.

Q: Constructor chaining?

Answer: Calling one constructor from another using this() or super().

Q: Can constructor return value?

Answer: No, constructors do not return values.

Q: When constructor is called?

Answer: Constructor is called automatically when an object is created.

Encapsulation

Q: What is encapsulation?

Answer: Encapsulation is wrapping data and methods into a single unit while hiding data using private access.

Q: Why use encapsulation?

Answer: To protect data and control access through methods.

Q: How to achieve encapsulation?

Answer: By making variables private and using getters and setters.

Q: What are getters and setters?

Answer: Methods used to read and modify private variables.

Q: Benefits of encapsulation?

Answer: Security, maintainability, and data control.

Q: Real-life example?

Answer: Bank account where balance is hidden and accessed through methods.

Q: Data hiding meaning?

Answer: Restricting direct access to variables.

Q: Why variables private?

Answer: To prevent unauthorized modification.

Q: Encapsulation vs abstraction?

Answer: Encapsulation hides data; abstraction hides implementation details.

Q: Example in banking system?

Answer: Users cannot directly modify balance; they use deposit/withdraw methods.

Inheritance

Q: What is inheritance?

Answer: Inheritance allows one class to acquire properties and methods of another class.

Q: Why use inheritance?

Answer: To reuse code and build hierarchical relationships.

Q: Types of inheritance?

Answer: Single, multilevel, hierarchical (multiple via interfaces).

Q: Why Java doesn't support multiple inheritance?

Answer: To avoid ambiguity known as the diamond problem.

Q: Superclass meaning?

Answer: Parent class whose properties are inherited.

Q: Subclass meaning?

Answer: Child class that inherits from parent.

Q: IS-A relationship?

Answer: Represents inheritance relation between classes.

Q: Method inheritance?

Answer: Child class inherits methods of parent.

Q: Constructor inheritance?

Answer: Constructors are not inherited but can be invoked using super().

Q: Advantages of inheritance?

Answer: Code reuse, easy maintenance, and hierarchy support.

Polymorphism

Q: What is polymorphism?

Answer: Polymorphism means many forms; the same method behaves differently.

Q: Types of polymorphism?

Answer: Compile-time and runtime polymorphism.

Q: Compile-time polymorphism?

Answer: Achieved using method overloading.

Q: Runtime polymorphism?

Answer: Achieved using method overriding.

Q: Method overloading?

Answer: Multiple methods with same name but different parameters.

Q: Method overriding?

Answer: Child class provides its own method implementation.

Q: Why overriding used?

Answer: To change behavior of parent method.

Q: Rules of overriding?

Answer: Method name, parameters must match; access cannot be more restrictive.

Q: Dynamic method dispatch?

Answer: Method call resolved at runtime based on object type.

Q: Real-world example?

Answer: Different payment methods processing payment differently.

Abstraction

Q: What is abstraction?

Answer: Abstraction hides implementation details and shows only functionality.

Q: Why abstraction used?

Answer: To reduce complexity and improve design.

Q: Abstract class meaning?

Answer: A class that cannot be instantiated and may contain abstract methods.

Q: Abstract method meaning?

Answer: A method without implementation.

Q: Can abstract class have constructor?

Answer: Yes, abstract classes can have constructors.

Q: Can abstract class have methods?

Answer: Yes, both abstract and concrete methods.

Q: Difference abstract & interface?

Answer: Interface supports multiple inheritance; abstract class does not.

Q: When to use abstraction?

Answer: When implementation details should be hidden.

Q: Real-world example?

Answer: ATM machine shows options without exposing internal processing.

Q: Why abstraction important?

Answer: Improves maintainability and scalability.

Interface

Q: What is interface?

Answer: An interface is a blueprint containing abstract methods to be implemented by classes.

Q: Why interface used?

Answer: To achieve abstraction and multiple inheritance.

Q: Interface vs abstract class?

Answer: Interface supports multiple inheritance; abstract class may have concrete methods.

Q: Multiple inheritance using interface?

Answer: A class can implement multiple interfaces.

Q: Default methods in interface?

Answer: Methods with implementation using default keyword.

Q: Functional interface?

Answer: Interface with only one abstract method.

Q: Can interface have variables?

Answer: Yes, variables are public, static, and final by default.

Q: Interface inheritance?

Answer: Interfaces can extend other interfaces.

Q: Marker interface?

Answer: Interface without methods used to mark classes.

Q: SAM interface?

Answer: Single Abstract Method interface.

this Keyword

Q: What is this keyword?

Answer: this refers to the current object instance.

Q: When used?

Answer: To resolve variable conflict or call constructors.

Q: Constructor chaining?

Answer: Calling another constructor using this().

Q: Why needed?

Answer: To differentiate instance variables from parameters.

Q: Difference between variable and this.variable?

Answer: this.variable refers to instance variable, not local variable.

super Keyword

Q: What is super?

Answer: super refers to parent class object.

Q: Access parent variable?

Answer: Yes, using super.variable.

Q: Access parent method?

Answer: Yes, using super.method().

Q: Call parent constructor?

Answer: Yes, using super().

Q: Why needed?

Answer: To access parent class features when overridden.