Object-Oriented Programming (OOPs) in Java

Introduction

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data and code.
The data is in the form of fields (often known as attributes or properties), and the code is in the form of methods (functions).
Java is one of the most popular programming languages that supports OOP principles.

Core Principles of OOP

1. Encapsulation

Encapsulation is the technique of wrapping data (variables) and methods (functions) together as a single unit.
It restricts direct access to some components of an object, which is a good practice for securing the integrity of data.

Key points:
- Use private modifiers to restrict access to variables.
- Use getter and setter methods to provide controlled access.

Example:

```
class Person {
   private String name;
   private int age;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getAge() {
      return age;
   }

   public void setAge(int age) {
      this.age = age;
   }
}
```

2. Inheritance

Inheritance is the mechanism in Java by which one class acquires the properties and behavior of another class.
It is a way to achieve code reusability.

Key points:
- Use the `extends` keyword for inheritance.
- Parent class (superclass) passes its properties to the child class (subclass).

Example:

```
class Animal {
   void eat() {
      System.out.println("This animal eats food.");
   }
}

class Dog extends Animal {
   void bark() {
      System.out.println("This dog barks.");
   }
}
```

3. Polymorphism

Polymorphism means "many forms." It allows one interface to be used for a general class of actions.

Key points:
- Two types: Compile-time (method overloading) and Runtime (method overriding).
- Overloading allows methods with the same name but different parameters.
- Overriding allows a subclass to provide a specific implementation of a method.

Example (Method Overloading):

```
class MathUtils {
   int add(int a, int b) {
      return a + b;
   }
```

```
    double add(double a, double b) {
        return a + b;
    }
}
```

Example (Method Overriding):

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
```

4. Abstraction

Abstraction is the concept of hiding implementation details and showing only functionality to the user.

Key points:
- Achieved using abstract classes and interfaces.
- Abstract class: Use the `abstract` keyword.
- Interface: Use the `interface` keyword.

Example (Abstract Class):

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

Example (Interface):

```
interface Animal {
   void sound();
}

class Cat implements Animal {
   public void sound() {
      System.out.println("Meow");
   }
}
```

Trick Questions and Concepts

1. What is the difference between an abstract class and an interface?

Abstract Class:
- Can have abstract and concrete methods.
- Can have constructors.
- Can contain instance variables.

Interface:
- Only abstract methods (until Java 8).
- Cannot have constructors.
- All fields are public, static, and final by default.

2. Can we override static methods in Java?

No, static methods cannot be overridden because they are bound to the class, not the object.
However, they can be hidden by declaring a method with the same name in the subclass.

3. What is the diamond problem? How is it resolved in Java?

The diamond problem occurs when a class inherits from two classes that have the same method.
Java resolves this issue by not allowing multiple inheritance with classes.
However, with interfaces (since Java 8), the `default` methods can cause conflicts, and the programmer must override them.

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

Method Overloading:
- Same method name but different parameter lists.

- Happens at compile time.

Method Overriding:
- Same method signature in parent and child classes.
- Happens at runtime.

5. What are the key differences between this and super in Java?

- this: Refers to the current instance of the class.
- Used to access instance variables, methods, or constructors within the same class.
- Example: this.name = name;
- super: Refers to the parent class (superclass) of the current object.
- Used to access methods, variables, or constructors of the superclass.
- Example: super.methodName();

6. Can constructors be overridden in Java?

- No, constructors cannot be overridden because they are not inherited by the subclass. However, they can be overloaded in the same class or subclass.

7. What is the purpose of final in Java?

- final variable: Prevents reassignment of the variable.
- final method: Prevents method overriding.
- final class: Prevents the class from being subclassed.

Example:
```
    final class Constants {

            public static final double PI = 3.14159;

        }
```

8. What is the difference between Abstract class and Concrete class?

- Abstract Class:
- Cannot be instantiated.
- Contains abstract methods (without implementation) and concrete methods.
- Concrete Class:
- Fully implemented class that can be instantiated.

9. Can a class be both final and abstract in Java? Why?

- No, because abstract implies the class needs to be extended, and final prevents it from being extended.

10. What are access modifiers in Java? Explain their scope.

- Public: Accessible everywhere.
- Private: Accessible only within the class.
- Protected: Accessible within the package and subclasses.
- Default (no modifier): Accessible only within the package.

11. What is a marker interface? Can you give examples?

- A marker interface is an interface with no methods or fields, used to provide metadata to a class.
- Example: Serializable, Cloneable.

12. How is polymorphism implemented in Java?

- Compile-Time Polymorphism: Achieved using method overloading.
- Runtime Polymorphism: Achieved using method overriding.

13. What is the difference between association, aggregation, and composition?

- Association: A relationship where all objects have their own lifecycle.
- Aggregation: A weak association where the child object can exist without the parent.
- Example: Department and Employee.
- Composition: A strong association where the child cannot exist without the parent.
- Example: Car and Engine.

14. What is the purpose of the Object class in Java?

- The Object class is the root of the Java class hierarchy. Every class implicitly inherits from it.
- Common methods include toString(), equals(), hashCode(), and clone().

15. What is the difference between equals() and ==?

- ==: Compares object references (memory locations).
- equals(): Compares the values/content of objects (can be overridden).

16. What is the difference between shallow copy and deep copy in Java?

- Shallow Copy: Copies only the object’s references, not the actual objects.
- Deep Copy: Copies the object along with its referenced objects.

18. What is dynamic method dispatch?

- Dynamic method dispatch is a mechanism in Java by which a call to an overridden method is resolved at runtime based on the object being referred to.

19. Can an interface extend another interface in Java?

- Yes, interfaces can extend other interfaces using the extends keyword.
- Example:

```
interface A {
     void methodA();
    }
interface B extends A {
      void methodB();
    }
```

20. What happens if you do not override all methods in an abstract class?

If a class does not override all abstract methods, it must itself be declared abstract.