# OOPS IN JAVA Interview Questions:

**Object-Oriented Programming System**.

**1. What is OOP, and what are its basic principles?**

**Answer:** Object-Oriented Programming (OOP) is a programming paradigm that uses "objects" to design applications and programs. It allows developers to model real-world entities and relationships. The four basic principles of OOP are:

- **Encapsulation:** Wrapping data (variables) and methods that operate on the data into a single unit or class. It restricts access to certain details and only exposes necessary components.
- **Inheritance:** Allows one class (child/subclass) to inherit fields and methods from another class (parent/superclass). This promotes code reuse and establishes a relationship between classes.
- **Polymorphism:** The ability to take many forms. It allows objects of different classes to be treated as objects of a common superclass. Polymorphism is often achieved through method overriding and method overloading.
- **Abstraction:** Hiding the complex implementation details and exposing only the essential features of an object. Abstract classes and interfaces are used to achieve abstraction in Java.

**2. What is the difference between abstraction and encapsulation?**

**Answer:**

- **Abstraction:** Focuses on hiding the implementation details and showing only the essential features of an object. It deals with the design level, providing a clear separation between what an object does and how it does it. Abstraction is implemented using abstract classes and interfaces.
- **Encapsulation:** Focuses on wrapping data (variables) and methods into a single unit or class and restricting access to some components. Encapsulation is more about protecting the data within an object and is implemented using access modifiers like private, protected, and public.

**3. What is the difference between method overloading and method overriding?**

**Answer:**

- **Method Overloading:** Occurs when multiple methods in the same class have the same name but different parameter lists (different number of parameters or different types). It is a compile-time polymorphism feature.

time polymorphism feature.

```java
class Example {
    void display(int a) {
        System.out.println("Integer: " + a);
    }

    void display(String a) {
        System.out.println("String: " + a);
    }
}
```

**Method Overriding:** Occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The method in the subclass must have the same name, return type, and parameters as the method in the parent class. It is a runtime polymorphism feature.

```java
class Parent {
    void display() {
        System.out.println("Parent display");
    }
}

class Child extends Parent {
    @Override
    void display() {
        System.out.println("Child display");
    }
}
```

**4. What is an interface, and how is it different from an abstract class?**

**Answer:**

- **Interface:** An interface in Java is a reference type that can contain only abstract methods (until Java 8) and static constants. From Java 8 onwards, interfaces can also have default and static methods. An interface is used to define a contract that implementing classes must follow.

```java
interface Animal {
    void eat(); // Abstract method
}

class Dog implements Animal {
    @Override
    public void eat() {
        System.out.println("Dog eats");
    }
}
```

**Abstract Class:** An abstract class is a class that cannot be instantiated on its own and can have both abstract and concrete methods. Abstract classes are used when there is a need to share common code among several closely related classes.

```java
abstract class Animal {
    abstract void eat(); // Abstract method

    void sleep() { // Concrete method
        System.out.println("Sleeping");
    }
}

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

**Key Differences:**

- An abstract class can have both abstract and non-abstract methods, while an interface can only have abstract methods (before Java 8).
- A class can implement multiple interfaces, but it can only extend one abstract class.
- Abstract classes can have constructors, while interfaces cannot.

**5. What is the difference between == and equals() method in Java?**

**Answer:**

- **==:** The == operator compares the references (memory addresses) of two objects. It checks whether both references point to the same object in memory.

whether both references point to the same object in memory.

```java
String s1 = new String("Hello");
String s2 = new String("Hello");

System.out.println(s1 == s2); // false
```

- `equals()`: The `equals()` method is used to compare the contents (values) of two objects. The default implementation in the `Object` class compares references, but it is often overridden in classes like `String` to compare actual content.

```java
System.out.println(s1.equals(s2)); // true
```

**6. What is a constructor, and what are the types of constructors in Java?**

**Answer:** A constructor is a special method that is called when an object is instantiated. Constructors are used to initialize the state of an object. In Java, constructors have the same name as the class and do not have a return type.

**Types of Constructors:**

- **Default Constructor:** A constructor with no parameters. If no constructor is defined, Java provides a default constructor.

```java
class Example {
    Example() {
        System.out.println("Default Constructor");
    }
}
```

**Parameterized Constructor:** A constructor that takes arguments to initialize the object with specific values.

```java
class Example {
    int value;

    Example(int value) {
        this.value = value;
    }
}
```

**7. What is the significance of the super keyword in Java?**

**Answer:** The super keyword in Java is used to refer to the immediate parent class object. It can be used to:

- **Call the parent class constructor:**

- Call the parent class constructor:

```java
class Parent {
    Parent() {
        System.out.println("Parent Constructor");
    }
}

class Child extends Parent {
    Child() {
        super(); // Calls Parent constructor
        System.out.println("Child Constructor");
    }
}
```

  - Call the parent class method:

```java
class Parent {
    void display() {
        System.out.println("Parent display");
    }
}

class Child extends Parent {
    @Override
    void display() {
        super.display(); // Calls Parent's display
        System.out.println("Child display");
    }
}
```

**8. Can you explain the concept of this keyword in Java?**

**Answer:** The this keyword in Java is used to refer to the current instance of the class. It can be used to:

- **Refer to the current class instance variable:**

- Refer to the current class instance variable:

```java
class Example {
    int value;

    Example(int value) {
        this.value = value; // Refers to the instance variable
    }
}
```

- Invoke the current class method:

```java
class Example {
    void display() {
        System.out.println("Display method");
    }

    void invoke() {
        this.display(); // Invokes display method
    }
}
```

- Invoke the current class constructor:

```java
class Example {
    Example() {
        this(10); // Calls parameterized constructor
        System.out.println("Default Constructor");
    }

    Example(int value) {
        System.out.println("Parameterized Constructor: " + value);
    }
}
```

**9. What is a static method in Java? Can you override a static method?**

**Answer:** A static method in Java is a method that belongs to the class rather than any instance of the class. It can be called without creating an object of the class. Static methods can only access static variables and static methods directly.

**Example:**

Example:

```java
class Example {
    static void display() {
        System.out.println("Static method");
    }
}

Example.display(); // No need to create an object
```

**Overriding Static Methods:** Static methods cannot be overridden because method overriding is based on dynamic binding at runtime, while static methods are bound at compile time (static binding). If you declare a static method in the subclass with the same signature as in the superclass, it hides the superclass method instead of overriding it.

**10. What is the difference between an abstract class and a concrete class?**

**Answer:**

- **Abstract Class:** An abstract class is a class that cannot be instantiated directly. It can have both abstract methods (without implementation) and concrete methods (with implementation). Abstract classes are meant to be subclassed, and the abstract methods must be implemented in the subclasses.

```java
abstract class Animal {
    abstract void sound(); // Abstract method

    void sleep() { // Concrete method
        System.out.println("Sleeping");
    }
}
```

- **Concrete Class:** A concrete class is a regular class that can be instantiated. It must provide implementations for all abstract methods if it extends an abstract class.

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

Dog dog = new Dog(); // Concrete class can be instantiated
```