OOPS NOTES BY PRADEEP

-OOPS BASIC

-OOPS Interview questions with explaination

Basic Concepts of OOP

Object: An object is a real-world entity that has attributes (data) and behaviors (methods). For example, a dog can be an object with attributes like color and breed, and behaviors like bark() and fetch().

Class: A class is a blueprint for creating objects. It defines the properties and behaviors that the objects created from the class will have. For instance, a Dog class can define what attributes (like color) and methods (like bark()) a dog has.

Encapsulation: This is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit, or class. It also restricts direct access to some of the object's components, which helps prevent unintended interference and misuse.

Inheritance: This allows a new class (child class) to inherit properties and methods from an existing class (parent class). For example, if Animal is a parent class, Dog and Cat can be child classes that inherit from Animal.

Polymorphism: This means "many forms." It allows methods to do different things based on the object that it is acting upon. For example, a makeSound() method can make a dog bark and a cat meow, even though both use the same method name.

FREQUENTLY ASKED QUESTIONS

---

1. What is OOP?

Explanation: OOP stands for Object-Oriented Programming, a programming paradigm that uses "objects" to represent data and functions. This approach helps model real-world entities and relationships.

Example:

```java
class Player {
int health;
void attack() {
System.out.println("Player attacks!");
}
}
```

Here, `Player` is a class representing a player in a game with health as an attribute and an `attack()` method.

---

2. What is a class and an object?

Explanation: A class is a blueprint for creating objects. An object is an instance of a class that contains actual data.

Example:

```java
class Dog {
String breed;
void bark() {
System.out.println("Woof!");
```

```
}
}

Dog myDog = new Dog();
myDog.breed = "Labrador";
myDog.bark(); // Outputs: Woof!
```

Here, `Dog` is a class, and `myDog` is an object of that class.

---

3. What is encapsulation?

Explanation: Encapsulation is the practice of keeping the data (attributes) private within a class and only allowing access through public methods. This protects the integrity of the data.

Example:

```java
class BankAccount {
private double balance; // private variable

public void deposit(double amount) {
balance += amount;
}

public double getBalance() {
return balance;
}
}
```

In this example, the `balance` attribute is private, preventing direct access. Instead, it can only be modified through the `deposit()` method.

---

4. What is inheritance?

Explanation: Inheritance allows one class (child class) to inherit the properties and methods of another class (parent class), promoting code reuse.

Example:

```java
class Animal {
void eat() {
System.out.println("Eating...");
}
}

class Dog extends Animal {
void bark() {
System.out.println("Woof!");
}
}
```

Here, `Dog` inherits from `Animal`, meaning it can use the `eat()` method while also having its own method `bark()`.

---

5. What is polymorphism?

Explanation: Polymorphism allows methods to perform different tasks based on the object that calls them. This can be achieved through method overriding and overloading.

Example:

```java
class Animal {
```

```
void makeSound() {
System.out.println("Animal sound");
}
}

class Dog extends Animal {
void makeSound() {
System.out.println("Woof!");
}
}

class Cat extends Animal {
void makeSound() {
System.out.println("Meow!");
}
}
```

Here, both `Dog` and `Cat` classes override the `makeSound()` method, providing different implementations.

---

6. What is an interface?

Explanation: An interface is a contract that defines methods a class must implement. It allows different classes to implement the same methods, promoting flexibility.

Example:

```java
interface Animal {
void makeSound();
}
```

```
class Dog implements Animal {
public void makeSound() {
System.out.println("Woof!");
}
}

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

Here, both `Dog` and `Cat` implement the `Animal` interface, providing their own versions of `makeSound()`.

---

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

Explanation: A class is a blueprint for creating objects, while an object is an instance of a class containing actual data.

Example:

```java
class Car {
String color;
void drive() {
System.out.println("Car is driving.");
}
}

Car myCar = new Car(); // myCar is an object of the Car class
myCar.color = "Red"; // Setting the attribute
```

```
myCar.drive(); // Calling the method
```

---

8. What are the benefits of OOP?

Explanation: OOP provides several benefits:

- **Code Reusability**: Inheritance allows reuse of existing code.
- **Maintainability**: Changes in one part of the code can be made independently.
- **Scalability**: Easier to extend applications as requirements grow.

---

9. What is method overloading?

Explanation: Method overloading allows multiple methods in the same class to have the same name but different parameters (different type or number).

Example:

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

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

Here, the `add` method is overloaded to handle both integers and doubles.

---

10. What is method overriding?

Explanation: Method overriding allows a subclass to provide a specific implementation of a method that is already defined in its parent class.

Example:

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

class Dog extends Animal {
void sound() {
System.out.println("Woof!"); // Overrides the sound method
}
}
```

In this example, `Dog` overrides the `sound()` method to provide its own implementation.

10.5 ifference between method overloading and method overriding:

Aspect	**Method Overloading**	**Method Overriding**
Definition	Same method name, different parameters	Same method name and parameters in child class
Purpose	To create multiple methods with the same name	To provide a specific implementation of a method
Compile-Time / Run-Time	Resolved at compile time	Resolved at run time

Return Type	Can have different return types	Must have the same return type or subtype
Class Relationship	Can occur within the same class	Occurs in parent-child class relationship
Example	`int add(int a, int b)` and `double add(double a, double b)`	`class Dog extends Animal { void sound() { /*...*/ } }`

11. What is a constructor?

Explanation: A constructor is a special method used to initialize objects. It has the same name as the class and does not have a return type.

Example:

```java
class Dog {
String name;

// Constructor
Dog(String name) {
this.name = name; // 'this' refers to the current object's name
}
}

Dog myDog = new Dog("Buddy"); // Calls the constructor
```

---

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

Explanation: An abstract class can have method implementations and state (attributes), while an interface only declares methods without implementations.

Example:

- **Abstract Class**:

```java
abstract class Animal {
abstract void sound(); // abstract method
void eat() { // concrete method
System.out.println("Eating...");
}
}
```

- **Interface**:

```java
interface Animal {
void sound(); // no implementation
}
```

---

13. What is a static method?

Explanation: A static method belongs to the class rather than any specific object. It can be called without creating an instance of the class.

Example:

```java
class Utility {
static void printMessage() {
System.out.println("Hello, static method!");
}
}

// Calling the static method without creating an object
```

```
Utility.printMessage();
```

---

14. What is the use of `this` keyword?

Explanation: The `this` keyword refers to the current object. It is often used to differentiate between instance variables and parameters with the same name.

Example:

```java
class Dog {
String name;

Dog(String name) {
this.name = name; // 'this.name' refers to the instance variable
}
}
```

---

15. What is the use of `super` keyword?

Explanation: The `super` keyword is used to refer to the parent class. It can call parent class methods and constructors.

Example:

```java
class Animal {
Animal() {
System.out.println("Animal created");
}
```

```
}

class Dog extends Animal {
Dog() {
super(); // Calls the constructor of Animal
System.out.println("Dog created");
}
}
```

Read about access specifiers also
Public, Private, - - - - - - - - -