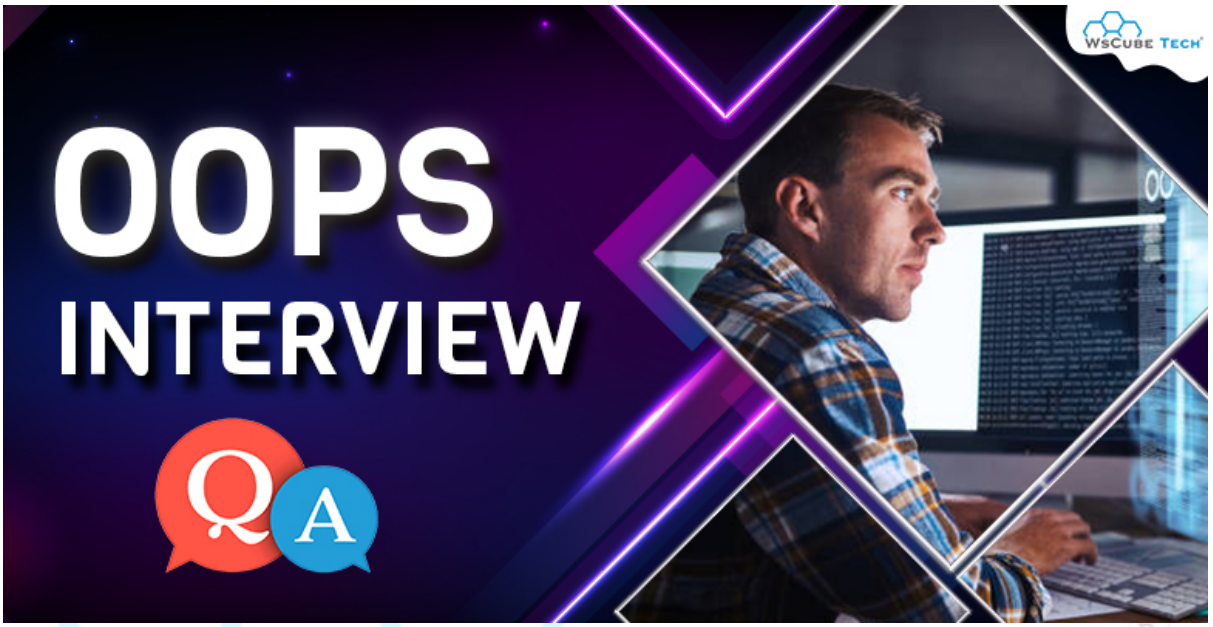
Top OOPs Interview Questions and Answers PDF



Interview Questions on OOPs for Freshers

1. What is OOPs?

It is a programming paradigm that is used to develop software applications by creating objects that interact with each other.

In simple terms, OOP is a way of writing software that models real-world objects, such as people, cars, or buildings, as software objects with specific properties and behaviour.

The main idea behind OOP is to create reusable code that can be modified and extended with ease. It allows programmers to write code that is easier to read, understand, and maintain.

2. What are the key principles of OOPs?

It is one of the most asked interview questions on OOPs concepts. You must know that there are four principles of OOP using which developers can write code more efficiently and maintain it with ease.

a) Encapsulation

It is used to hide the internal details of an object and its behaviour from the outside world.

This allows objects to be used and manipulated without revealing their internal workings. Encapsulation is achieved by defining the object's attributes as private or protected, and providing public methods to access and modify those attributes.

b) Inheritance

It is the process of creating new classes from existing classes. It allows a new class to inherit properties and behaviour from an existing class.

The existing class is known as the superclass or parent class. The new class is known as the subclass or child class.

c) Polymorphism

It is the ability of an object to take on multiple forms. It allows different objects to be used interchangeably, even though they may have different implementations. Polymorphism can be achieved through method overloading and method overriding.

d) Abstraction

It is the process of simplifying complex systems by modelling them at a high level. In OOP, abstraction is achieved by creating abstract classes and interfaces that define the methods that must be implemented by their subclasses.

This allows the programmer to focus on the essential features of the system, while ignoring the irrelevant details.

3. What is a class in OOPs?

It is a blueprint or a template for constructing objects. It is used to define a set of properties and methods that will be present in an object belonging to that class.

It is essentially a user-defined data type that encapsulates data (in the form of fields or properties) and behaviour (in the form of methods). The fields or properties define the characteristics of the object, while the methods define what the object can do.

Example of class

Let's consider a class called "*Car*". This class may have fields such as "*make*", "*model*", "*year*", and "*color*", which define the characteristics of a car object.

It may also have methods such as "*start*", "*stop*", "*accelerate*", and "*brake*", which define the behaviour of a car object.

Once a class has been defined, it can be used to create multiple instances or objects of that class. Each object will have its own set of values for the fields or properties, and will be able to perform the methods defined by the class.

4. What is an object in OOPs?

An object is an instance of a class. It is a concrete entity that can be created based on a class blueprint.

When an object is created, it has its own unique set of properties and methods based on the definition of its class. These properties and methods can be accessed and manipulated through the object's public interface, which consists of its public methods and properties.

Example of object

Consider the class "*Car*". An object of the Car class can be created by instantiating the class, like this:

```
Car myCar = new Car();
```

In this case, *myCar* is an object of the Car class. It has its own set of properties (such as make, model, year, and color) and methods (such as start, stop, accelerate, and brake) that are defined by the Car class. These properties and methods can be accessed and manipulated using dot notation, like this:

```
myCar.color = "red";
myCar.start();
```

5. What is inheritance in OOPs?

This is among the top OOPs questions for interview.

Inheritance is the process of creating a new class (called the subclass or child class) from an existing class (called the superclass or parent class).

The subclass inherits properties and behaviours from its superclass, and can also add new properties and behaviours or modify existing ones.

The basic idea behind inheritance in OOP is to create a hierarchy of classes with increasing levels of specialisation. The most general class in the hierarchy is the superclass, which defines the basic properties and behaviours that are shared by all its subclasses. Each subclass can then add more specific properties and behaviours that are unique to it.

Example of inheritance in OOP

Consider a class hierarchy that includes a superclass called "*Vehicle*" and two subclasses called "*Car*" and "*Motorcycle*". The Vehicle class may have properties such as "*make*", "*model*", "*year*", and "*color*", as well as methods such as "*start*" and "*stop*" that are common to all types of vehicles.

The Car subclass may inherit these properties and methods from the Vehicle class, but also add its own properties, such as "*numDoors*" and methods such as "*getGasMileage*". The Motorcycle subclass may also inherit from Vehicle, but may have its own unique properties such as "*numWheels*" and methods such as "*lean*".

6. What is encapsulation in OOPs?

Encapsulation is the concept of hiding the implementation details of a class from its users, and providing a public interface or API (Application Programming Interface) for interacting with the class.

The idea behind encapsulation is to protect the internal state of an object from being modified directly by external code, and to enforce the use of the public interface for any interactions with the object. This helps to prevent errors and ensure that the object is used correctly.

Example of encapsulation

Consider a class called "*BankAccount*". This class may have private fields such as "*balance*" and "*accountNumber*", which should not be modified directly by external code. Instead, the class may provide public methods such as "*deposit*", "*withdraw*", and "*getBalance*" for interacting with the account.

7. What is abstraction in OOPs?

It is the process used to define a simplified interface or model that represents the essential features of an object, without including unnecessary details or implementation specifics.

Abstraction helps to manage complexity by hiding the underlying details of a system and presenting only the essential features that are relevant to the user. This allows the user to interact with the system at a higher level of abstraction, without having to worry about the low-level details.

An abstract class may contain one or more abstract methods, which are defined but not implemented in the abstract class. The subclasses of the abstract class must implement these abstract methods to provide concrete functionality.

Example
For example, consider a class hierarchy that includes a superclass called "Animal" and two subclasses called "Dog" and "Cat".

The Animal class may have abstract methods such as "makeSound" and "move", which are implemented differently in each subclass. The Dog subclass may implement the "makeSound" method as "bark" and the "move" method as "run", while the Cat subclass may implement the "makeSound" method as "meow" and the "move" method as "walk".

8. What is polymorphism in OOPs?

Polymorphism is a fundamental concept in Object-Oriented Programming (OOP) that allows objects of different classes to be treated as if they are objects of a common superclass or interface.

It enables the same code to work with different objects in different ways, depending on their actual type or class.

9. What is method overloading? Explain with an example.

It is a form of static polymorphism that allows multiple methods to have the same name, but with different parameters or argument types.

When an overloaded method is called, the compiler determines which method to call based on the number, type, and order of the arguments passed to the method. The method name and the number, type, and order of the parameters must be different in each overloaded method.

Example of method overloading in OOPs
For example, consider a class called "Math" that contains two overloaded methods called "add".

The first method takes two integers as arguments and returns their sum, while the second method takes two doubles as arguments and returns their sum.

When we call the "add" method with two integers, the first method is called, and when we call the "add" method with two doubles, the second method is called.

10. What is method overriding? Explain with an example.

It is a form of dynamic polymorphism that allows a subclass to provide its own implementation of a method that is already defined in its superclass.

When a method is called on an object, the runtime environment determines which implementation of the method to call based on the actual type of the object at runtime. The method name, return type, and parameter types must be the same in both the superclass and the subclass.

Example of method overriding in OOPs
For example, consider a class hierarchy that includes a superclass called "Animal" and a subclass called "Dog". The Animal class may have a method called "speak" that returns a string, while the Dog subclass may override the "speak" method to return "woof".

When we call the "speak" method on a Dog object, the overridden method in the Dog class is called, and when we call the "speak" method on an Animal object, the original method in the Animal class is called.

11. What is a constructor?

A constructor in OOP is a special method that is called when an object is created. Its main purpose is to initialise the object's state or data members with some initial values.

In Java, a constructor has the same name as the class and does not have a return type, not even void. It is called implicitly when an object is created using the "new" keyword, and it can also be called explicitly like any other method.

12. What are different types of constructors?

There are two types of constructors:

a) Default constructor

This constructor is provided by Java if a class does not have any constructors explicitly defined. It takes no parameters and initialises all the data members to their default values (e.g., null for object references, 0 for integers, etc.).

b) Parameterized constructor

This constructor is defined by the programmer and takes one or more parameters. It initialises the data members with the values passed as arguments.

13. What is a destructor?

In Java and some other object-oriented programming languages, there is no such thing as a destructor. Instead, these languages rely on garbage collection to automatically reclaim memory that is no longer being used by objects.

Garbage collection is a process by which the Java Virtual Machine (JVM) automatically frees up memory that is no longer being used by objects in the program. The JVM keeps track of all objects that are still being referenced by the program and automatically deletes those that are no longer being used.

In other programming languages, such as C++, destructors are used to free up memory that was allocated using the "new" operator. When an object is created using the "new" operator, memory is allocated for it on the heap. When the object is no longer needed, its destructor is called, and the memory is freed up.

14. What is a static method?

A static method in Java is a method that belongs to the class rather than to any instance of the class. This means that you can call a static method without creating an instance of the class first. You can call it directly on the class itself using the class name.

Example of a static method in Java:

```
public class MyClass {
    private static int count = 0;

    public static void incrementCount() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}
```

Here, the "incrementCount" and "getCount" methods are declared as static. This means that you can call them on the "MyClass" class without creating an instance of it first.

You can call these methods like this:

```
MyClass.incrementCount();
int count = MyClass.getCount();
```

Static methods are often used to provide utility methods that don't require an instance of the class to be created. They can also be used to define constants that are associated with the class rather than with any particular instance of the class.

Static methods cannot access non-static variables or methods because they are not associated with any instance of the class.

15. What is a static variable?

In Java, a static variable is a variable that belongs to the class rather than to any instance of the class. This means that all instances of the class share the same static variable, and changes to the static variable by one instance will be visible to all other instances.

Example of a static variable in Java:

```
public class MyClass {
    private static int count = 0;

    public void incrementCount() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Here, the "count" variable is declared as static. This means that it belongs to the "MyClass" class rather than to any instance of the class.

You can access this variable like this:

```
MyClass obj1 = new MyClass();
MyClass obj2 = new MyClass();
obj1.incrementCount();
obj2.incrementCount();
System.out.println(obj1.getCount()); // prints 2
System.out.println(obj2.getCount()); // prints 2
```

Here, both "obj1" and "obj2" share the same "count" variable. When "incrementCount" is called on each instance, the value of "count" is incremented. When "getCount" is called on each instance, it returns the same value because they are both accessing the same static variable.

16. What is the difference between an instance variable and a class variable?

In object-oriented programming, instance variables and class variables are two types of variables that can be declared within a class.

The main difference between them is that instance variables are associated with an instance of a class, while class variables are associated with the class itself.

Key differences between instance variables and class variables:

a) Scope

Instance variables are only accessible within the instance of the class in which they are defined, whereas class variables are accessible throughout the entire class.

b) Lifetime

Instance variables are created when an instance of the class is created and are destroyed when the instance is destroyed, whereas class variables exist for the entire lifetime of the program.

c) Memory allocation

Instance variables are allocated memory each time an instance of the class is created, whereas class variables are allocated memory only once when the class is loaded.

d) Sharing

Each instance of a class has its own copy of instance variables, while class variables are shared by all instances of the class.

17. What is a package in Java?

In Java, a package is a mechanism for organising related classes and interfaces into a single unit. It provides a way to group related classes and interfaces together in a logical manner, and also provides a way to control access to the members of the classes and interfaces.

A package is declared at the beginning of a Java source file, using the "package" keyword followed by the name of the package. For example, the following code declares a package named "com.example":

```
package com.example;
```

Classes and interfaces can then be declared within this package, like this:

```
package com.example;
public class MyClass {
```

```
    // class members here
}
interface MyInterface {
    // interface members here
}
```

To use a class or interface from a package in another Java source file, you need to import it using the "import" keyword.

For example, to use the "MyClass" class from the "com.example" package in another file, you would add the following import statement:

import com.example.MyClass;

Packages can also contain sub-packages, which can be used to further organise classes and interfaces. For example, the following code declares a package named "com.example.util" which is a sub-package of the "com.example" package:

package com.example.util;

Packages can also be used to control access to the members of classes and interfaces. By default, all members of a class or interface are accessible from within the same package. However, you can use access modifiers such as "public", "private", and "protected" to control the visibility of members from outside the package.

OOPs Interview Questions for Experienced

18. What are the four fundamental principles of OOPs?

The four fundamental principles of Object-Oriented Programming (OOPs) are:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

19. What is the difference between abstraction and encapsulation?

Abstraction and Encapsulation are two important concepts in Object-Oriented Programming (OOPs). They are often confused with each other as they both deal with hiding complexity, but they are different in nature.

Here are the differences between the two:

Feature	Abstraction	Encapsulation
Definition	Abstraction is the process of identifying the essential features of an object and ignoring the non-essential ones.	Encapsulation is the process of hiding the internal implementation details of an object from the outside world and exposing only the necessary information through a public interface.
Purpose	Abstraction helps to manage complexity by breaking down a complex system into smaller, more manageable parts.	Encapsulation helps to achieve data hiding and data abstraction by hiding the implementation details of an object from the outside world.
Level of Focus	Abstraction focuses on the behaviour and characteristics of an object without worrying about the implementation details.	Encapsulation focuses on the implementation details of an object and restricts access to its internal state.
Implementation	Abstraction can be implemented using abstract classes and interfaces.	Encapsulation can be implemented using access modifiers like public, private, and protected.
Relationship	Abstraction is closely related to inheritance and polymorphism.	Encapsulation is closely related to data hiding and information hiding.

20. What is the difference between inheritance and polymorphism?

Inheritance and Polymorphism are two important concepts in Object-Oriented Programming (OOPs). They are often used together to create reusable and extensible code. The differences between the two are:

Feature	Inheritance	Polymorphism
Definition	Inheritance is the mechanism by which one class acquires the properties and behaviour of another class.	Polymorphism is the ability of an object to take on multiple forms or behaviours.

Purpose	Inheritance helps in creating new classes that are modified versions of existing classes without having to rewrite the entire code.	Polymorphism helps in creating flexible and extensible code that can work with objects of different classes.
Types	There are two types of inheritance: single-level and multi-level inheritance.	There are two types of polymorphism: compile-time and runtime polymorphism.
Implementation	Inheritance is implemented using the extends keyword in Java and C++.	Polymorphism is implemented using method overloading and method overriding.
Relationship	Inheritance is closely related to the concept of parent and child classes.	Polymorphism is closely related to the concept of interfaces and abstract classes.

21. What is the role of the final keyword in Java?

The final keyword is used to define entities (variables, methods, and classes) that cannot be changed or overridden.

Here are the different uses of the final keyword in Java:

a) Final variables:

A final variable is a variable whose value cannot be changed once it is initialised. It is also called a constant. A final variable can be initialised during declaration or in a constructor. Once initialised, its value cannot be modified.

Example:

final int MAX_VALUE = 100;

b) Final methods:

A final method is a method that cannot be overridden by a subclass. When a method is marked as final, its implementation is fixed and cannot be changed by a subclass.

Example:

```
public class Parent {
    public final void show() {
        System.out.println("This is a final method");
    }
}
public class Child extends Parent {
    // This will give a compilation error as final method cannot be overridden
    public void show() {
        System.out.println("This method cannot be overridden");
    }
}
```

c) Final classes:

A final class is a class that cannot be extended by any subclass. When a class is marked as final, it cannot be subclassed.

Example:

```
final public class FinalClass {
    // This class cannot be extended
}
```

22. Share an example that shows the difference between instance and class variables in OOPs.

Here's an example that illustrates the difference between instance variables and class variables:

```
public class MyClass {
    private int instanceVar;
    private static int classVar;
    public MyClass(int instanceVar) {
        this.instanceVar = instanceVar;
    }
    public void setInstanceVar(int instanceVar) {
        this.instanceVar = instanceVar;
    }
    public void setClassVar(int classVar) {
        MyClass.classVar = classVar;
    }
    public int getInstanceVar() {
```

```
        return instanceVar;
    }
    public int getClassVar() {
        return classVar;
    }
}
```

Here, "instanceVar" is an instance variable and "classVar" is a class variable. The "setInstanceVar" method sets the value of the instance variable, while the "setClassVar" method sets the value of the class variable. The "getInstanceVar" and "getClassVar" methods return the values of the instance and class variables, respectively.

When you create an instance of this class, each instance will have its own copy of "instanceVar", but they will all share the same "classVar". For example:

```
MyClass obj1 = new MyClass(1);
MyClass obj2 = new MyClass(2);
obj1.setClassVar(10);
System.out.println(obj1.getInstanceVar()); // prints 1
System.out.println(obj1.getClassVar()); // prints 10
System.out.println(obj2.getInstanceVar()); // prints 2
System.out.println(obj2.getClassVar()); // prints 10
```

Here, "obj1" and "obj2" have different values for their instance variable "instanceVar", but they share the same value for their class variable "classVar". When "setClassVar" is called on "obj1", it sets the value of the class variable to 10, which is then shared by all instances of the class.

23. What are the different types of polymorphism?

There are two types of polymorphism: static and dynamic.

a)Static polymorphism

Also known as compile-time polymorphism, the static polymorphism in OOP is achieved through method overloading.

Method overloading allows multiple methods to have the same name, but with different parameters or argument types. The compiler figures out which method to call on the basis of number, type, and order of the arguments passed to the method.

b) Dynamic polymorphism

Also known as runtime polymorphism, the dynamic polymorphism is achieved through method overriding.

Method overriding allows a subclass to provide its own implementation of a method that is already defined in its superclass. When a method is called on an object, the runtime environment determines which implementation of the method to call based on the actual type of the object at runtime.

24. What is the difference between a static method and an instance method?

Static methods and instance methods are two different types of methods in Java. Here is a tabular comparison of static and instance methods:

Feature	Static Method	Instance Method
Definition	A static method is a method that belongs to a class rather than an instance of a class.	An instance method is a method that belongs to an instance of a class.
Access	A static method can be called using the class name without creating an instance of the class.	An instance method can only be called using an instance of the class.
Memory Allocation	A static method is stored in a permanent memory location and is loaded into memory when the class is loaded.	An instance method is stored in memory when an object is created.
Use of this keyword	A static method cannot use the this keyword as it does not belong to an instance of a class.	An instance method can use the this keyword to refer to the current instance of the class.
Accessing Instance Variables	A static method cannot access instance variables directly. It can only access static variables.	An instance method can access both static and instance variables directly.

Overriding	A static method cannot be overridden in Java.	An instance method can be overridden in Java.

25. What is a design pattern, and can you name a few commonly used design patterns?

A design pattern is a general, reusable solution to a commonly occurring problem in software design. Design patterns provide a template for solving problems in software development and help to create flexible, maintainable, and reusable software architectures.

Here are a few commonly used design patterns:

a) Singleton Pattern:

This pattern ensures that a class has only one instance, and provides a global point of access to that instance.

b) Factory Pattern:

This pattern provides an interface for creating objects, but allows subclasses to alter the type of objects that will be created.

c) Observer Pattern:

This pattern defines a one-to-many relationship between objects, so that when one object changes state, all its dependents are notified and updated automatically.

d) Decorator Pattern:

This pattern allows behaviour to be added to an individual object, either statically or dynamically, without affecting the behaviour of other objects from the same class.

e) Strategy Pattern:

This pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable.

f) Adapter Pattern:

This pattern converts the interface of a class into another interface that clients expect. It allows incompatible classes to work together by wrapping one class with another class that has a compatible interface.

g) Facade Pattern:

This pattern provides a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that makes the subsystem easier to use.

26. What is a singleton pattern, and when is it used?

Singleton Pattern is a creational design pattern that is used when we want to ensure that only one instance of a class is created and that the instance can be easily accessed throughout the program. The Singleton Pattern ensures that a class has only one instance and provides a global point of access to that instance.

Here are the characteristics of a Singleton Pattern:

- It has a private constructor to prevent the creation of new instances of the class from outside the class.
- It has a private static instance variable that holds the single instance of the class.
- It has a public static method that returns the single instance of the class.

Example of Singleton Pattern:

```
public class Singleton {
   private static Singleton instance;
   private Singleton() {
      // private constructor to prevent creation of new instances
   }
   public static Singleton getInstance() {
      if (instance == null) {
         instance = new Singleton();
      }
      return instance;
   }
}
```

Here, the Singleton class has a private constructor to prevent the creation of new instances of the class. It has a private static instance variable that holds the single instance of the class, and a public static method getInstance() that returns the single instance of the class.

The getInstance() method checks if an instance of the class has already been created. If it hasn't, it creates a new instance and returns it. If an instance already exists, it returns the existing instance.

27. Give an example of parameterised constructor in Java.

Example of a parameterized constructor in Java:

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

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // getters and setters for name and age
}
```

Here, the "Person" class has a constructor that takes two parameters: "name" (a string) and "age" (an integer). When an object of the "Person" class is created using this constructor, its "name" and "age" data members are initialised with the values passed as arguments.

28. What is a factory pattern, and when is it used?

Factory Pattern is a creational design pattern that provides an interface for creating objects, but allows subclasses to alter the type of objects that will be created. It is used when we want to create objects without exposing the instantiation logic to the client and when we want to provide a common interface for creating objects of different types.

Here are the characteristics of a Factory Pattern:

- It has a Factory interface that declares a factory method for creating objects.
- It has a ConcreteFactory class that implements the Factory interface and creates objects of a particular type.
- It has a Product interface that defines the interface for the objects that the factory method creates.
- It has ConcreteProduct classes that implement the Product interface and provide the implementation for the objects that the factory method creates.

Example of Factory Pattern in Java:

```
public interface Animal {
    void makeSound();
}
public class Dog implements Animal {
    public void makeSound() {
        System.out.println("Woof!");
    }
}
public class Cat implements Animal {
    public void makeSound() {
        System.out.println("Meow!");
    }
}
public interface AnimalFactory {
    Animal createAnimal();
}
public class DogFactory implements AnimalFactory {
    public Animal createAnimal() {
        return new Dog();
    }
}
public class CatFactory implements AnimalFactory {
    public Animal createAnimal() {
        return new Cat();
    }
}
```

Here, the Animal interface defines the interface for the objects that the factory method creates. The Dog and Cat classes implement the Animal interface and provide the implementation for the objects that the factory method creates.

The AnimalFactory interface declares a factory method for creating objects, and the DogFactory and CatFactory classes implement the AnimalFactory interface and create objects of the Dog and Cat classes, respectively.

29. What is the difference between private, protected, and public access modifiers in OOPs?

Access modifiers in OOP are used to control the visibility and accessibility of class members (variables, methods, and nested classes) from other classes.

The three most commonly used access modifiers are private, protected, and public.

a) Private:

Members declared as private can only be accessed within the same class. They are not visible to any other class, including subclasses.

This is the most restrictive access level and is often used to protect sensitive data or to prevent unintended modifications to class members.

b) Protected:

Members declared as protected can be accessed within the same class, subclasses, and other classes in the same package.

They are not visible to classes in different packages, except through inheritance. This access level is less restrictive than private but more restrictive than public.

c) Public:

Members declared as public can be accessed from any class or package. This is the least restrictive access level and is often used for methods and variables that are part of the class's public interface.

Access Modifier	Visibility within the Same Class	Visibility within Subclasses	Visibility within the Same Package	Visibility from Different Packages
Private	Yes	No	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

List of Java OOPs Interview Questions

Here are the top 15 interview questions related to OOPs in Java:

1. What is object-oriented programming in Java?
2. What are the four fundamental principles of OOP?
3. What is inheritance in Java? How does it work?

4. What is polymorphism in Java? How is it implemented?
5. What is encapsulation in Java? How is it implemented?
6. What is abstraction in Java? How is it implemented?
7. What is the difference between abstract classes and interfaces in Java?
8. What is a constructor in Java? How is it used?
9. What is the difference between method overloading and method overriding in Java?
10. What is a static method in Java? How is it used?
11. What is the final keyword in Java? How is it used?
12. What is a singleton pattern in Java? When is it used?
13. What is a factory pattern in Java? When is it used?
14. What is the difference between composition and inheritance in Java?
15. What is the difference between checked and unchecked exceptions in Java?

List of OOPs Python Interview Questions

Here are the top 15 interview questions related to OOPs in Python:

1. What is object-oriented programming (OOP) in Python?
2. What are the four fundamental principles of OOP?
3. What is inheritance in Python? How does it work?
4. What is polymorphism in Python? How is it implemented?
5. What is encapsulation in Python? How is it implemented?
6. What is abstraction in Python? How is it implemented?
7. What is the difference between abstract classes and interfaces in Python?
8. What is a constructor in Python? How is it used?
9. What is the difference between method overloading and method overriding in Python?
10. What is a static method in Python? How is it used?
11. What is a decorator in Python? How is it used?
12. What is a generator in Python? How is it used?
13. What is the difference between classmethod and staticmethod in Python?
14. What is multiple inheritance in Python? How does it work?
15. What is the difference between composition and inheritance in Python?

List of OOPs PHP Interview Questions

Here is the list of top interview questions on OOPs in PHP:

1. What is object-oriented programming (OOP) in PHP?
2. What are the four fundamental principles of OOP?
3. What is inheritance in PHP? How does it work?
4. What is polymorphism in PHP? How is it implemented?

5. What is encapsulation in PHP? How is it implemented?
6. What is abstraction in PHP? How is it implemented?
7. What is the difference between abstract classes and interfaces in PHP?
8. What is a constructor in PHP? How is it used?
9. What is the difference between method overloading and method overriding in PHP?
10. What is a static method in PHP? How is it used?
11. What is a final keyword in PHP? How is it used?
12. What is a singleton pattern in PHP? When is it used?
13. What is a factory pattern in PHP? When is it used?
14. What is the difference between composition and inheritance in PHP?
15. What is the difference between checked and unchecked exceptions in PHP?

OOPs Coding Questions

Below are the top OOPs coding questions that you may encounter in programming interviews:

1. Write a Java program to represent a stack.
2. Write a Java program using class to represent a queue.
3. Implement a class to represent a binary tree.
4. Write a Java program to represent a linked list.
5. Write a Java program using class to represent a hash table.
6. How to represent a binary search tree in Java using OOP?
7. Implement a class to represent a heap.
8. Implement a class to represent a priority queue.
9. Implement a class to represent a queue that also has a max() method to return the maximum element.
10. Write a program to represent a stack that supports push, pop, and min in O(1) time complexity.
11. Write a program to represent a queue that supports enqueue, dequeue, and max in O(1) time complexity.
12. How to represent a binary search tree that supports finding the kth smallest element in O(log n) time complexity.
13. How to represent a graph that can perform depth-first search and breadth-first search.
14. Implement a class to represent a hash table that uses quadratic probing to handle collisions.
15. Write a Java program to represent a priority queue that uses a binary heap.

OOPs MCQs (Quiz)

Find Multiple Choice Questions (MCQs) related to OOP concepts:

1. What is OOPs?

a. Object-Oriented Programming System
b. Object-Oriented Programming Syntax
c. Object-Oriented Programming Standard
d. Object-Oriented Programming Style

Answer: a

2. Which of the following is NOT a fundamental principle of OOPs?

a. Inheritance
b. Encapsulation
c. Polymorphism
d. Abstraction
e. All of the above are fundamental principles of OOPs

Answer: e

3. What is inheritance in OOPs?

a. The ability of an object to take on many forms
b. The ability of a class to be derived from another class
c. The ability of a class to contain many objects
d. The ability of a class to encapsulate its data

Answer: b

4. What is polymorphism in OOPs?

a. The ability of a class to be derived from another class
b. The ability of a class to contain many objects
c. The ability of an object to take on many forms
d. The ability of a class to encapsulate its data

Answer: c

5. What is encapsulation in OOPs?

a. The ability of a class to be derived from another class
b. The ability of a class to contain many objects
c. The ability of an object to take on many forms
d. The ability of a class to encapsulate its data

Answer: d

6. What is abstraction in OOPs?

a. The ability of a class to be derived from another class
b. The ability of a class to contain many objects
c. The ability of an object to take on many forms
d. The ability of a class to encapsulate its data

Answer: c

7. What is the difference between an abstract class and an interface in OOPs?

a. An abstract class can have both concrete and abstract methods, while an interface can only have abstract methods
b. An interface can have both concrete and abstract methods, while an abstract class can only have abstract methods
c. An abstract class cannot be instantiated, while an interface can be instantiated
d. An interface cannot be extended, while an abstract class can be extended

Answer: a

8. What is a constructor in OOPs?

a. A method that is used to create objects of a class
b. A method that is used to destroy objects of a class
c. A method that is used to access private members of a class
d. A method that is used to override the behaviour of a superclass method

Answer: a

9. What is a static method in OOPs?

a. A method that can be called without creating an instance of a class
b. A method that is called automatically when an object is created
c. A method that can only be called from within the same class
d. A method that can be overridden by a subclass

Answer: a

10. What is the final keyword in OOPs?

a. A keyword that is used to make a variable or method unchangeable

b. A keyword that is used to indicate the end of a class definition
c. A keyword that is used to indicate the end of a method definition
d. A keyword that is used to indicate the end of a loop

Answer: a