**OOPs Interview Questions**

**1. What is Object Oriented Programming (OOPs)?**

O*bject* O*riented* P*rogramming* (also known as OOPs) is a programming paradigm where the complete software operates as a bunch of objects talking to each other. An object is a collection of data and the methods which operate on that data.

**2. Why OOPs?**

The main advantage of OOP is better manageable code that covers the following:

1. The overall understanding of the software is increased as the distance between the language spoken by developers and that spoken by users.
2. Object orientation eases maintenance by the use of encapsulation.  One can easily change the underlying representation by keeping the methods the same.
3. The OOPs paradigm is mainly useful for relatively big software.

**3. What is a Class?**

A **class** is a building block of Object Oriented Programs. It is a user-defined data type that contains the data members and member functions that operate on the data members. It is like a blueprint or template of objects having common properties and methods.

**4. What is an Object?**

An **object** is an instance of a class. Data members and methods of a class cannot be used directly. We need to create an object (or instance) of the class to use them. In simple terms, they are the actual world entities that have a state and behavior.

Eg. The code below shows is an example of how an instance of a class (i.e an object ) of a class is created

* C++
* Java
* Python3
* C#

|  |
| --- |
| #include <iostream>  **using** **namespace** std;    **class** Student{  **private**:      string name;      string surname;  **int** rollNo;    **public**:      Student(string studentName, string studentSurname, **int** studentRollNo){          name = studentName;          surname = studentSurname;          rollNo = studentRollNo;      }    **void** getStudentDetails(){          cout << "The name of the student is " << name << " " << surname << endl;          cout << "The roll no of the student is " << rollNo << endl;      }  };    **int** main() {      Student student1("Vivek", "Yadav", 20);      student1.getStudentDetails();    **return** 0;  } |

**Output**

The name of the student is Vivek Yadav

The roll no of the student is 20

**5. What are the main features of OOPs?**

The main feature of the OOPs, also known as 4 pillars or basic principles of OOPs are as follows:

1. Encapsulation
2. Data Abstraction
3. Polymorphism
4. Inheritance

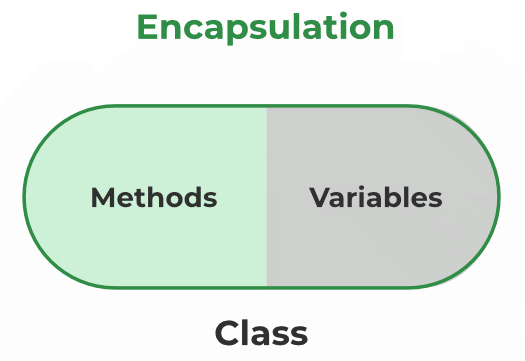


*OOPs Main Features*

**6. What is Encapsulation?**

Encapsulation is the binding of data and methods that manipulate them into a single unit such that the sensitive data is hidden from the users  
It is implemented as the processes mentioned below:

1. **Data hiding:** A language feature to restrict access to members of an object. For example, private and protected members in C++.
2. **Bundling of data and methods together:** Data and methods that operate on that data are bundled together. For example, the data members and member methods that operate on them are wrapped into a single unit known as a class.

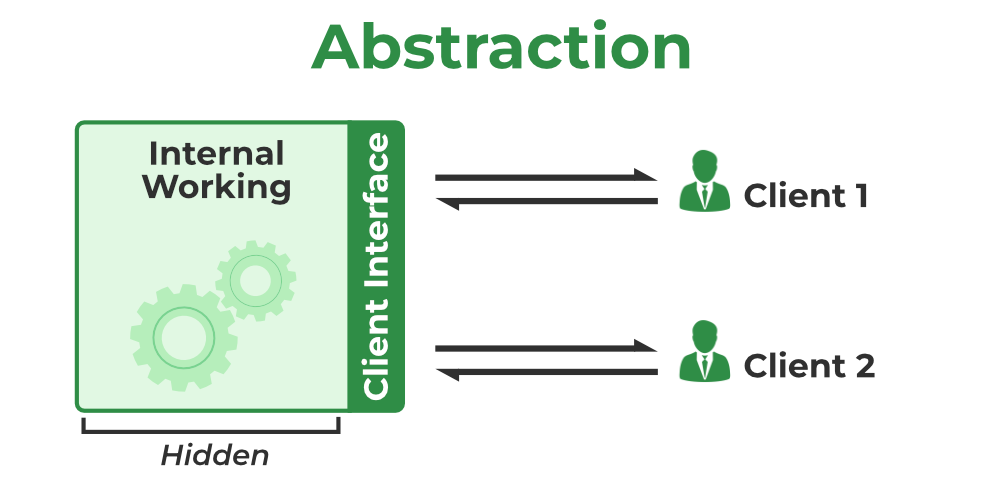


* Java

|  |
| --- |
| //simple demonstration of encapsulation  //It has a private data member and getter and setter methods.  **public** **class** Student{      //private data members  **private** String name;  **private** **int** rollNo;      //public getter method to access the name  **public** String getName(){  **return** name;      }    //public getter method to access rollNo  **public** **int** getRollNo(){  **return** rollNo;      }  //public setter method to set name  **public** **void** setName(String name){  **this**.name=name      }    //public setter method to set rollNo  **public** **void** setRollNo(**int** rollNo){  **this**.rollNo=rollNo;      }  } |

**7. What is Abstraction?**

Abstraction is similar to data encapsulation and is very important in OOP. It means showing only the necessary information and hiding the other irrelevant information from the user. Abstraction is implemented using classes and interfaces.



* Java

|  |
| --- |
| //implementation of abstraction through abstract class  **abstract** **class** Animal {  **abstract** **void** walk();  **void** eat()      {          System.out.println("The animal is eating.");      }      Animal()      {          System.out.println(              "An Animal is going to be created.");      }  }    **class** Cow **extends** Animal {      Cow() { System.out.println("You have created a Cow"); }  **void** walk() { System.out.println("Cow is walking."); }  }    **class** Goat **extends** Animal {      Goat()      {          System.out.println("You have created a Goat");      }  **void** walk() { System.out.println("Goat is walking."); }  }    **public** **class** OOPS {  **public** **static** **void** main(String args[])      {          Cow cow = **new** Cow();          cow.walk();          cow.eat();          Goat goat = **new** Goat();          goat.walk();          goat.eat();      }  } |

**8. What is Polymorphism?**

The word “**Polymorphism**” means having many forms. It is the property of some code to behave differently for different contexts. For example, in C++ language, we can define multiple functions having the same name but different working depending on the context.

Polymorphism can be classified into two types based on the time when the call to the object or function is resolved. They are as follows:

A. Compile Time Polymorphism  
B. Runtime Polymorphism

**A) Compile-Time Polymorphism**

Compile time polymorphism, also known as static polymorphism or early binding is the type of polymorphism where the binding of the call to its code is done at the compile time. Method overloading or operator overloading are examples of compile-time polymorphism.

**B) Runtime Polymorphism**

Also known as dynamic polymorphism or late binding, runtime polymorphism is the type of polymorphism where the actual implementation of the function is determined during the runtime or execution. Method overriding is an example of this method.

* Java

|  |
| --- |
| // An example of method overloading  **class** Student {     String name,surname;  **int** rollNo;    **public** **void** showStudentDetails(String name) {         System.out.println("The name of the student is " + name);     }      **public** **void** showStudentDetails(**int** rollNo) {         System.out.println("the roll no of the student is "+ rollNo);     }      **public** **void** showStudentDetails(String name, String surname, **int** rollNo) {         System.out.println(name);         System.out.println(surname);         System.out.println(age);     }  } |

* Java

|  |
| --- |
| // an example of method oerriding  **class** Student {  **public** **void** read() {         System.out.println("The student is reading");     }  }  **class** SchoolStudent **extends** Student {  **public** **void** read(String book) {         System.out.println("the student is reding "+ book);     }  }  **class** CollegeStudent **extends** Student {  **public** **void** read(String researchPaper , String labJournal) {         System.out.println("the student is reading "+researchPaper +" and "+ labJournal);     }  } |

**9. What is Inheritance? What is its purpose?**

The idea of inheritance is simple, a class is derived from another class and uses data and implementation of that other class. The class which is derived is called child or derived or subclass and the class from which the child class is derived is called parent or base or superclass.

The main purpose of Inheritance is to increase code reusability. It is also used to achieve Runtime Polymorphism.

* Java

|  |
| --- |
| // an example of inheritance  **class** Student {  **public** **void** read() {         System.out.println("The student is reading");     }  }  **class** SchoolStudent **extends** Student {  **public** **void** read(String book) {         System.out.println("the student is reding "+ book);     }  } |

**10. What are access specifiers? What is their significance in OOPs?**

Access specifiers are special types of keywords that are used to specify or control the accessibility of entities like classes, methods, and so on. **Private**, **Public**, and **Protected** are examples of access specifiers or access modifiers.  
The key components of OOPs, encapsulation and data hiding, are largely achieved because of these access specifiers.

* Java

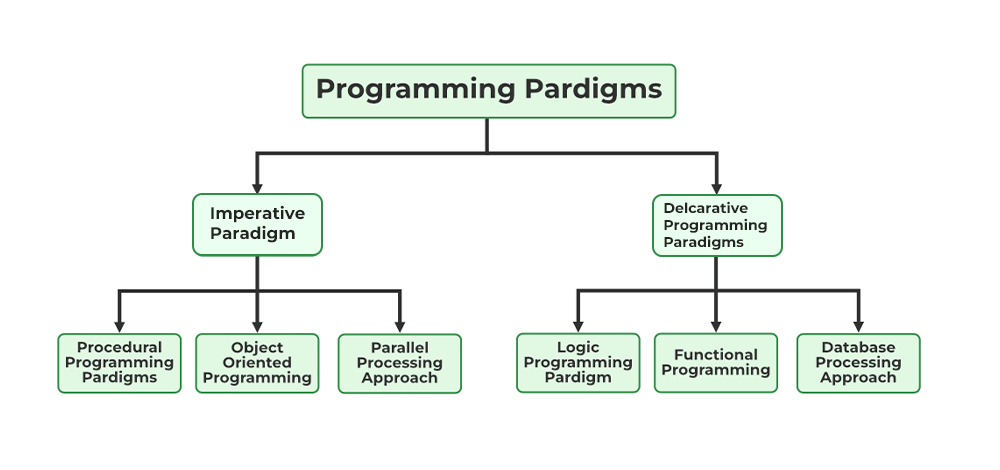
|  |
| --- |
| **class** User {  **public** String userName;  **protected** String userEmail;  **private** String password;      **public** **void** setPassword(String password) {  **this**.password = password;     }  }  **public** **class** OOPS {  **public** **static** **void** main(String args[]) {         User user1 = **new** User();         user1.userName = "Vivek\_Kumar\_Yadav";         user1.setPassword("abcd@12345");         user1.userEmail = "abc@gmail.com";     }  } |

**11. What are the advantages and disadvantages of OOPs?**

| **Advantages of OOPs** | **Disadvantages of OOPs** |
| --- | --- |
| OOPs provides enhanced code reusability. | The programmer should be well-skilled and should have excellent thinking in terms of objects as everything is treated as an object in OOPs. |
| The code is easier to maintain and update. | Proper planning is required because OOPs is a little bit tricky. |
| It provides better data security by restricting data access and avoiding unnecessary exposure. | OOPs concept is not suitable for all kinds of problems. |
| Fast to implement and easy to redesign resulting in minimizing the complexity of an overall program. | The length of the programs is much larger in comparison to the procedural approach. |

**12. What other paradigms of programming exist besides OOPs?**

The programming paradigm is referred to the technique or approach of writing a program. The programming paradigms can be classified into the following types:



**1. Imperative Programming Paradigm**

It is a programming paradigm that works by changing the program state through assignment statements. The main focus in this paradigm is on how to achieve the goal. The following programming paradigms come under this category:

1. **Procedural Programming Paradigm**: This programming paradigm is based on the procedure call concept. Procedures, also known as routines or functions are the basic building blocks of a program in this paradigm.
2. **Object-Oriented Programming or OOP**: In this paradigm, we visualize every entity as an object and try to structure the program based on the state and behavior of that object.
3. **Parallel Programming**: The parallel programming paradigm is the processing of instructions by dividing them into multiple smaller parts and executing them concurrently.

**2. Declarative Programming Paradigm**

Declarative programming focuses on what is to be executed rather than how it should be executed. In this paradigm, we express the logic of a computation without considering its control flow. The declarative paradigm can be further classified into:

1. **Logical Programming Paradigm**: It is based on formal logic where the program statements express the facts and rules about the problem in the logical form.
2. **Functional Programming Paradigm**: Programs are created by applying and composing functions in this paradigm.
3. **Database Programming Paradigm**: To manage data and information organized as fields, records, and files, database programming models are utilized.

**13. What is the difference between Structured Programming and Object Oriented Programming?**

Structured Programming is a technique that is considered a precursor to OOP and usually consists of well-structured and separated modules. It is a subset of procedural programming. The difference between OOPs and Structured Programming is as follows:

| **Object-Oriented Programming** | **Structural Programming** |
| --- | --- |
| Programming that is object-oriented is built on objects having a state and behavior. | A program’s logical structure is provided by structural programming, which divides programs into their corresponding functions. |
| It follows a bottom-to-top approach. | It follows a Top-to-Down approach. |
| Restricts the open flow of data to authorized parts only providing better data security. | No restriction to the flow of data. Anyone can access the data. |
| Enhanced code reusability due to the concepts of polymorphism and inheritance. | Code reusability is achieved by using functions and loops. |
| In this, methods are written globally and code lines are processed one by one i.e., Run sequentially. | In this, the method works dynamically, making calls as per the need of code for a certain time. |
| Modifying and updating the code is easier. | Modifying the code is difficult as compared to OOPs. |
| Data is given more importance in OOPs. | Code is given more importance. |

**14. What are some commonly used Object Oriented Programming Languages?**

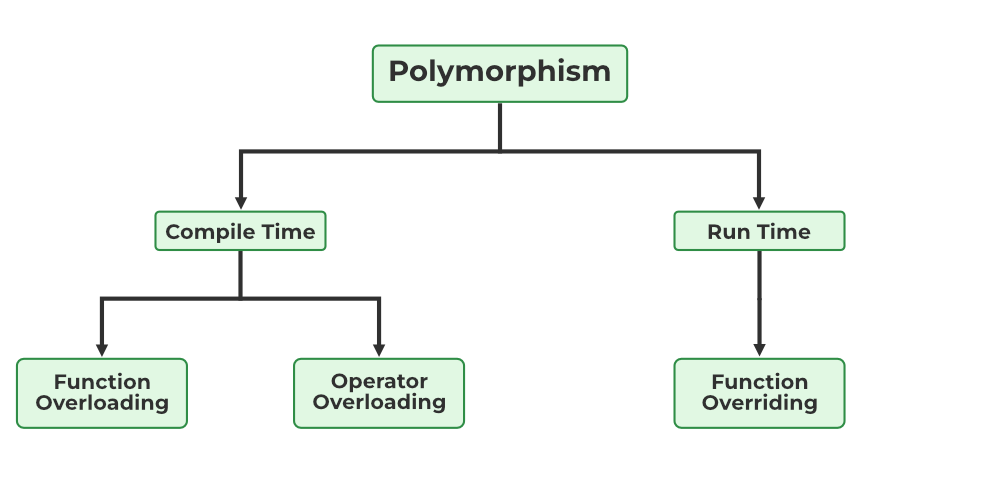
OOPs paradigm is one of the most popular programming paradigms. It is widely used in many popular programming languages such as:

* [C++](https://www.geeksforgeeks.org/c-plus-plus/)
* [Java](https://www.geeksforgeeks.org/java/)
* [Python](https://www.geeksforgeeks.org/python-programming-language/)
* [Javascript](https://www.geeksforgeeks.org/javascript/)
* [C#](https://www.geeksforgeeks.org/csharp-programming-language/)
* [Ruby](https://www.geeksforgeeks.org/ruby-programming-language/)

**15. What are the different types of Polymorphism?**

Polymorphism can be classified into two types based on the time when the call to the object or function is resolved. They are as follows:

1. Compile Time Polymorphism
2. Runtime Polymorphism



*Types of Polymorphism*

**A) Compile-Time Polymorphism**

Compile time polymorphism, also known as static polymorphism or early binding is the type of polymorphism where the binding of the call to its code is done at the compile time. **Method overloading** or **operator overloading** are examples of compile-time polymorphism.

**B) Runtime Polymorphism**

Also known asdynamic polymorphismor late binding, runtime polymorphism is the type of polymorphism where the actual implementation of the function is determined during the runtime or execution. **Method overriding** is an example of this method.

**16. What is the difference between overloading and overriding?**

A compile-time polymorphism feature called **overloading** allows an entity to have numerous implementations of the same name. Method overloading and operator overloading are two examples.

**Overriding** is a form of runtime polymorphism where an entity with the same name but a different implementation is executed. It is implemented with the help of virtual functions.

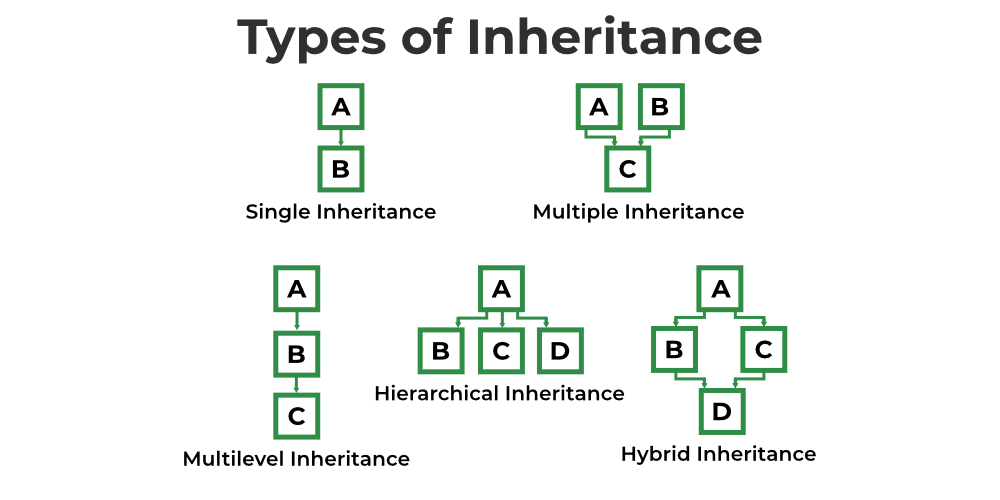
**17. Are there any limitations on Inheritance?**

**Yes,** there are more challenges when you have more authority. Although inheritance is a very strong OOPs feature, it also has significant drawbacks.

* As it must pass through several classes to be implemented, inheritance takes longer to process.
* The base class and the child class, which are both engaged in inheritance, are also closely related to one another (called tightly coupled). Therefore, if changes need to be made, they may need to be made in both classes at the same time.
* Implementing inheritance might be difficult as well. Therefore, if not implemented correctly, this could result in unforeseen mistakes or inaccurate outputs.

**18. What different types of inheritance are there?**

Inheritance can be classified into 5 types which are as follows:



1. **Single Inheritance:** Child class derived directly from the base class
2. **Multiple Inheritance:** Child class derived from multiple base classes.
3. **Multilevel Inheritance:**Child class derived from the class which is also derived from another base class.
4. **Hierarchical Inheritance:** Multiple child classes derived from a single base class.
5. **Hybrid Inheritance:** Inheritance consisting of multiple inheritance types of the above specified.

* Java

|  |
| --- |
| // an example of single inheritance  **class** Father {    // any properties and function specific to father  }  **class** Son **extends** Father {    //inherits the properties and functions of father  } |

* Java

|  |
| --- |
| // an example of hierarchial inheritance  **class** Father {    // any properties and function specific to father  }  **class** Son **extends** Father {    //inherits the properties and functions of father  }  **class** Daughter **extends** Father {    //inherits the properties and functions of father  } |

* Java

|  |
| --- |
| // an example of multilevel inheritance  **class** Father {    // any properties and function specific to father  }  **class** Son **extends** Father {    //inherits the properties and functions of father  }  **class** GrandChild **extends** Son {    } |

**19. What is an interface?**

A unique class type known as an interface contains methods but not their definitions. Inside an interface, only method declaration is permitted. You cannot make objects using an interface. Instead, you must put that interface into use and specify the procedures for doing so.

**20. How is an abstract class different from an interface?**

Both abstract classes and interfaces are special types of classes that just include the declaration of the methods, not their implementation. An abstract class is completely distinct from an interface, though. Following are some major differences between an abstract class and an interface.

| **Abstract Class** | **Interface** |
| --- | --- |
| When an abstract class is inherited, however, the subclass is not required to supply the definition of the abstract method until and unless the subclass actually uses it. | When an interface is implemented, the subclass is required to specify all of the interface’s methods as well as their implementation. |
| A class that is abstract can have both abstract and non-abstract methods. | An interface can only have abstract methods. |
| An abstract class can have final, non-final, static and non-static variables. | The interface has only static and final variables. |
| Abstract class doesn’t support multiple inheritance. | An interface supports multiple inheritance. |

**21. How much memory does a class occupy?**

Classes do not use memory. They merely serve as a template from which items are made. Now, objects actually initialize the class members and methods when they are created, using memory in the process.

**22. Is it always necessary to create objects from class?**

**No.**If the base class includes non-static methods, an object must be constructed. But no objects need to be generated if the class includes static methods. In this instance, you can use the class name to directly call those static methods.

**23. What is the difference between a structure and a class in C++?**

The structure is also a user-defined datatype in C++ similar to the class with the following differences:

* The major difference between a structure and a class is that in a structure, the members are set to public by default while in a class, members are private by default.
* The other difference is that we use **struct** for declaring structure and **class** for declaring a class in C++.

**24. What is Constructor?**

A constructor is a block of code that initializes the newly created object. A constructor resembles an instance method but it’s not a method as it doesn’t have a return type. It generally is the method having the same name as the class but in some languages, it might differ. For example:

In python, a constructor is named **\_\_init\_\_.**

In C++ and Java, the constructor is named the same as the class name.

**Example:**

* C++
* Java
* Python3

|  |
| --- |
| **class** Student {        String name;        String surname;  **int** rollNo;      Student()      {          cout<< "contructor is called";      }  } |

**25. What are the various types of constructors in C++?**

The most common classification of constructors includes:

1. **Default Constructor**
2. **Non-Parameterized Constructor**
3. **Parameterized Constructor**
4. **Copy Constructor**

**1. Default Constructor**

The default constructor is a constructor that doesn’t take any arguments. It is a non-parameterized constructor that is automatically defined by the compiler when no explicit constructor definition is provided.

It initializes the data members to their default values.

**2. Non-Parameterized Constructor**

It is a user-defined constructor having no arguments or parameters.

**Example:**

* C++
* Java
* Python3

|  |
| --- |
| **class** Student {        String name;        String surname;  **int** rollNo;      Student()      {          cout << "Non-parameterized contructor is called" ;      }  } |

**3. Parameterized Constructor**

The constructors that take some arguments are known as parameterized constructors.

**Example:**

* C++
* Java
* Python3

|  |
| --- |
| **class** Student {        String name;        String surname;  **int** rollNo;      Student(String studentName, String studentSurname, **int** studentRollNo)      {          cout << "Constructor with argument is called";      }  } |

**4. Copy Constructor**

A copy constructor is a member function that initializes an object using another object of the same class.

**Example:**

* C++
* Java

|  |
| --- |
| **class** Student {      String name, surname; **int** rollNo;      Student(Student& student) // copy constructor      {          name = student.name;          surname=student.surname;            rollNo= student.rollNo;      }  } |

In Python, we do not have built-in copy constructors like Java and C++ but we can make a workaround using different methods.

**26. What is a destructor?**

A destructor is a method that is automatically called when the object is made of scope or destroyed.

In C++, the destructor name is also the same as the class name but with the (**~**) **tilde symbol** as the prefix.

In Python, the destructor is named **\_\_del\_\_**.

**Example:**

* C++
* Python3

|  |
| --- |
| **class** base {  **public**:      ~base() { cout << "This is a destructor"; }  } |

In Java, the garbage collector automatically deletes the useless objects so there is no concept of destructor in Java. We could have used finalize() method as a workaround for the java destructor but it is also deprecated since Java 9.

**27. Can we overload the constructor in a class?**

We can overload the constructor in a class. In fact, the default constructor, parameterized constructor, and copy constructor are the overloaded forms of the constructor.

**28. Can we overload the destructor in a class?**

No. A destructor cannot be overloaded in a class. The can only be one destructor present in a class.

**29. What is the virtual function?**

A virtual function is a function that is used to override a method of the parent class in the derived class. It is used to provide abstraction in a class.

In C++, a virtual function is declared using the virtual keyword,

In Java, every public, non-static, and non-final method is a virtual function.

Python methods are always virtual.

**Example:**

* C++
* Java
* Python3

|  |
| --- |
| **class** base {  **virtual** **void** print()      {          cout << "This is a virtual function";      }  } |

**30. What is pure virtual function?**

A pure virtual function, also known as an abstract function is a member function that doesn’t contain any statements. This function is defined in the derived class if needed.

**Example:**

* C++
* Java

|  |
| --- |
| **class** base {  **virtual** **void** pureVirFunc() = 0;  } |

In Python, we achieve this using @abstractmethod from the ABC (Abstract Base Class) module.

**Bonus Question**

**What is an abstract class?**

In general terms, an abstract class is a class that is intended to be used for inheritance. It cannot be instantiated. An abstract class can consist of both abstract and non-abstract methods.

In C++, an abstract class is a class that contains at least one pure virtual function.

In Java, an abstract class is declared with an **abstract** keyword.

**Example:**

* Java

|  |
| --- |
| **abstract** **class** Animal {  **abstract** **void** walk();  **void** eat()      {          System.out.println("The animal is eating.");      }      Animal()      {          System.out.println(              "An Animal is going to be created.");      }  }    **class** Cow **extends** Animal {      Cow() { System.out.println("You have created a Cow"); }  **void** walk() { System.out.println("Cow is walking."); }  }    **class** Goat **extends** Animal {      Goat()      {          System.out.println("You have created a Goat");      }  **void** walk() { System.out.println("Goat is walking."); }  }    **public** **class** OOPS {  **public** **static** **void** main(String args[])      {          Cow cow = **new** Cow();          cow.walk();          cow.eat();          Goat goat = **new** Goat();          goat.walk();          goat.eat();      }  } |

In Python, we use ABC (Abstract Base Class) module to create an abstract class.

**Basic OOPs Interview Questions**

**1. What is OOPs?**

Programmers can use objects to represent real-world circumstances thanks to object-oriented programming. Any entity with states and behaviors is an object. While methods define an item’s behaviors, states reflect the characteristics or data of an entity. Objects include students, workers, books, etc. By exchanging messages, these things communicate with one another. A class is also a template for building an object. A class is required in order to generate objects. For instance, there needs to be an Employee class in order to generate an Employee object.

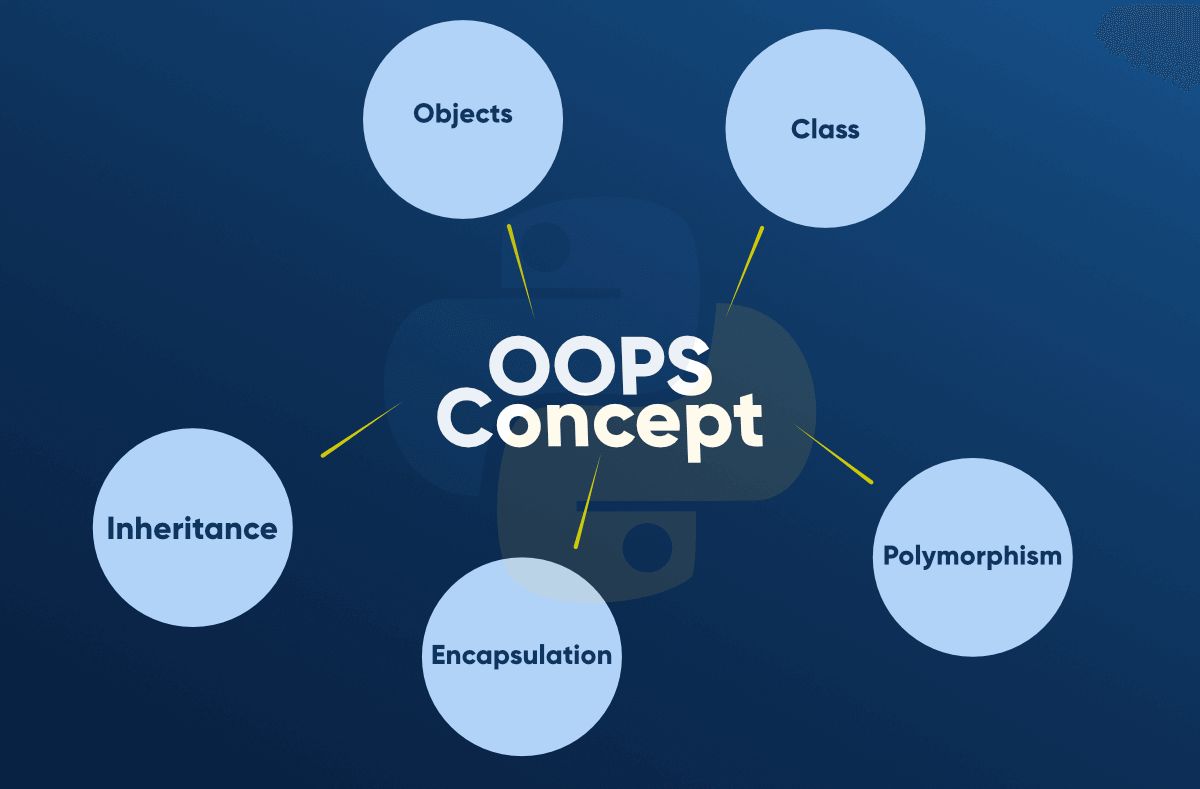
**2. Difference between Procedural programming and OOPs?**

|  |  |
| --- | --- |
| **Procedural Programming** | **Oops** |
| Procedural Programming is based on functions. | Object-oriented programming is based on real-world objects. |
| It shows the data to the entire program. | It encapsulates the data. |
| It does not have a scope for code reuse. | It provides more scope for code reuse. |
| It follows the concept of top-down programming. | It follows a bottom-up programming paradigm. |
| The nature of the language is complicated. | It is less complicated in nature, so it is easier to modify, extend and maintain. |
| It is hard to modify, extend and maintain the code. |  |

**3. Why use OOPs?**

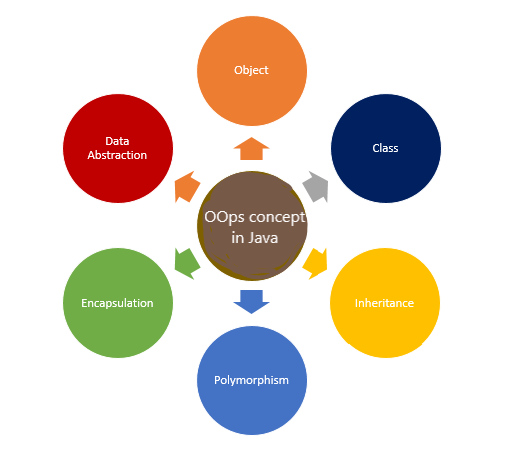
Programming with OOP lets you package together data states and functionality to change those data states while keeping the specifics secret (Consider the analogy of a car, you can only see the steering of the car while driving, the circuitry behind it is hidden from you). As a result, OOP design produces flexible, modular, and abstract code. Because of this, it is very helpful for developing larger programs. Using classes and objects, you may include OOP into your code. The objects you construct will have the states and capabilities of the class to which they belong.

**Free Courses**

[](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-python?gl_blog_id=85199)

**[OOPs in Python](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-python?gl_blog_id=85199" \t "_blank)**

[Land your dream job](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-python?gl_blog_id=85199" \t "_blank)

[](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-java?gl_blog_id=85199)

**[OOPs in Java](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-java?gl_blog_id=85199" \t "_blank)**

[Gain a competitive edge](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-java?gl_blog_id=85199" \t "_blank)

[](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-concepts-in-c?gl_blog_id=85199)

**[OOPs in C++](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-concepts-in-c?gl_blog_id=85199" \t "_blank)**

[Land your dream job](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-concepts-in-c?gl_blog_id=85199" \t "_blank)

**4. What are the basic concepts of OOPs?**

The basic concepts of OOPs are:

* Inheritance
* Encapsulation
* Polymorphism
* Abstraction

**5. What is Encapsulation?**

Encapsulation is also a part of the OOPs concept. It refers to the bundling of data with the methods that operate on that data. It also helps to restrict any direct access to some of an object’s components.

**6. What is Abstraction?**

Abstraction is an OOPs concept to build the structure of real-world objects. It “shows” only essential attributes and “hides” unnecessary information from the outside. The main focus of abstraction is to hide unnecessary details from the users. It is one of the most important concepts of **OOPs**.

**7. What is method overloading?**

There is a concept where two or more methods can have the same name. But they should have different parameters, different numbers of parameters, different types, or both. These methods are known as overloaded methods and this feature is called [**method overloading**](https://www.mygreatlearning.com/blog/method-overloading-in-java/).

**8. What is method overriding?**

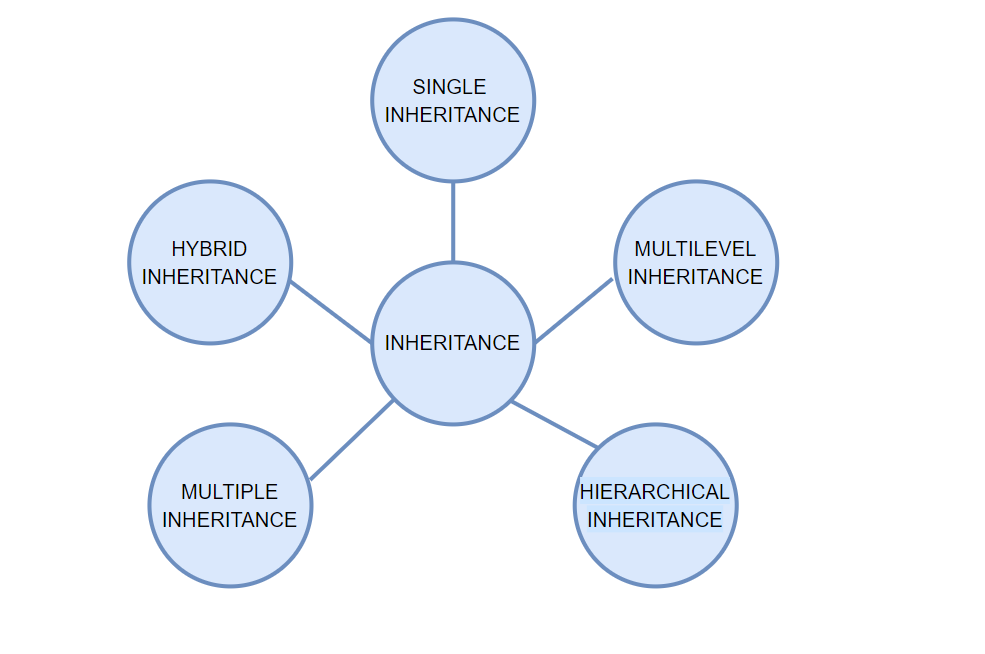
**Method overriding** is a concept of object-oriented programming.

It is a language feature that allows a subclass or child class to provide a specific implementation of a **method** which is already provided by one of its superclasses or parent classes.

**9. Types of Inheritance in OOPS**

Different types of inheritances in OOps are as follows:

* Single Inheritance
* Multiple Inheritance
* Multi-level Inheritance
* Multi-path Inheritance
* Hierarchical Inheritance
* Hybrid Inheritance



**10. What are the main features of OOPs?**

The main features of OOPs are given as follows:

* In OOP, you combine the code into one unit so you can specify the parameters of each piece of data. This process of wrapping up data into a single unit is called encapsulation.
* By using classes, you can generalise your object types and make your application easier to use. This is termed as an abstraction.
* The ability for a class to inherit characteristics and behaviours from another class allows for more code reuse.
* Polymorphism allows for the creation of several objects from a single, adaptable class of code.

**11. Is it possible to call the base class method without creating an instance?**

Yes, we can possibly call the base class method without creating an instance in the following 3 cases:

1. If the method is static
2. Calling the inherited method inside a derived class
3. Calling the method using the base keyword from the sub-classes

The most popular case is that of the static methods.

**12. What are the limitations of OOPs?**

Following are some of the common limitations of OOPs:

* Size exceeds that of other programs.
* It took a lot of work to make, and it runs more slowly than other programs.
* It is inappropriate for certain types of issues.
* It takes some getting used to.

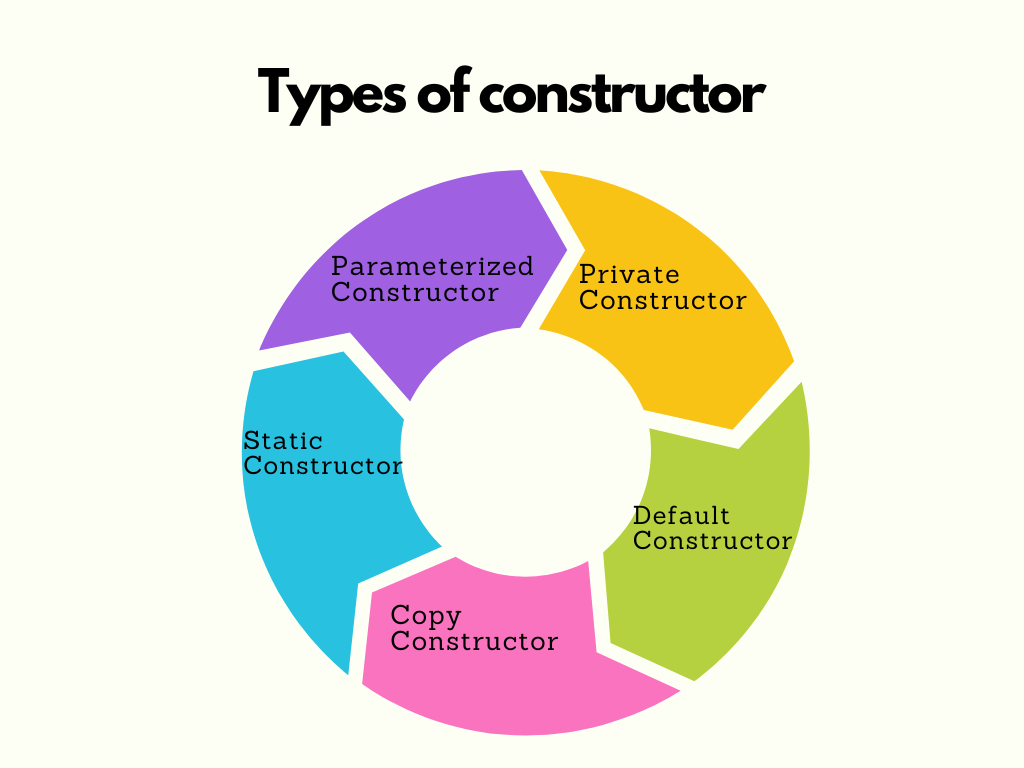
**13. What are constructors?**

The constructor has the same name as the class.  
A constructor is also a special kind of method. It is used to initialize objects of the class.

**14. Types of constructor**

Types of constructors depend upon languages

* Private Constructor
* Default Constructor
* Copy Constructor
* Static Constructor
* Parameterized Constructor



**15. What is the difference between a class and a structure?**

**Class: Class is basically a**User-defined blueprint from which objects are created. It consists of methods ( set of instructions) that are performed on the objects.

**Structure:**A structure is also a user-defined collection of variables. Structures are also different data types.

A user-defined class serves layout or blueprint from which objects can be built. In essence, a class is made up of fields known as attributes and methods known as member functions that define actions. A structure is a grouping of variables of various data kinds under one heading.

**16. What are the access modifiers?**

Access modifiers or access specifiers are the keywords in object-oriented languages.  It helps to set the accessibility of **classes**, **methods**, and other members.

**17. What languages come under the oops concept?**

Simula is known as the first object-oriented programming language, the most popular OOP languages are:

* Java
* JavaScript
* Python
* C++
* Visual Basic
* .NET
* Ruby
* Scala
* PHP

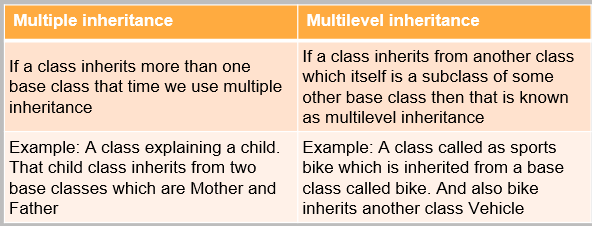
Check out the OOPs concept in Python Video.



**18. What is inheritance?**

Whenever one class is derived from another, it is referred to as inheritance. The child class will inherit all of the parent class’s public and protected properties and methods. Apart from the attributes and methods inherited from the parent class, it can also have its own additional set of features. The’ extends’ keyword is used to specify an inherited class.

If you derive a  class from another class that is known as inheritance. The child class will inherit all the public and protected properties and methods from the parent class. The child class can also have its own properties and methods. An **inherited** class is defined by using the extends keyword.



**19. What is hybrid inheritance?**

The type of inheritance formed by the combination of different types of inheritances like single, multiple inheritances, etc. is classified as hybrid inheritance.

**20. What is hierarchical inheritance?**

In the case of a hierarchical inheritance, multiple subclasses inherit from a parent class. Hierarchical inheritance is a type of inheritance in which multiple classes are descended from a single parent or base class. For example, the fruit class can have ‘apple’, ’mango’, ’banana’, ‘cherry’ etc. as its subclasses.

**21. What are the limitations of inheritance?**

It Increases the execution time and effort. It also requires jumping back and forth between different classes. The parent class and the child class are always tightly coupled. Afford modifications in the program would require changes for the parent and the child’s class. Inheritance requires careful implementation otherwise it would lead to incorrect results.

**22. What is a superclass?**

A superclass is a class from which a subclass or child class is derived. Base class and parent class are other names for a superclass. For example, if Student is a class derived from the Person class, then the Person class will be referred to as the superclass.

A superclass or base class is also a class that works as a parent to some other class/ classes.

For example, the Vehicle class is a superclass of class Bike.

**23. What is a subclass?**

A class that derives from another class is referred to as a subclass. A subclass inherits the properties of its ancestors or parent classes. For example, the class Bike is a subclass or a derivative of the Vehicle class.

**24. What is Polymorphism?**

**Polymorphism** is one of the most used and core concepts in **OOP** languages. It explains the concept of different classes can be used with the same interface. Each of these classes can have its own implementation of the interface.

**25. What is static polymorphism?**

In OOP, static polymorphism determines which method to call at compile time. For the same trigger with static polymorphism, the object might respond differently. Function, constructor and operator overloading are examples of static polymorphism.

**26. What is dynamic polymorphism?**

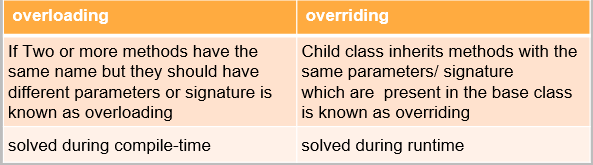
Dynamic polymorphism is a method or process that handles a call to an overridden method during runtime rather than at compile time. It is also referred to as dynamic method dispatch or runtime polymorphism. Using method overriding, we can create dynamic polymorphism. An example of runtime polymorphism: is method overriding.

**27. What is operator overloading?**

The user-defined data type is given a special meaning by the operator using [operator overloading](https://www.mygreatlearning.com/blog/operator-overloading-in-cpp/). It is a compile-time polymorphism.

**28. Differentiate between overloading and overriding.**

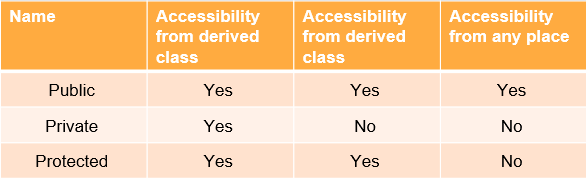
When two or more methods in the same class have the same name but different parameters, this is referred to as overloading. The technique of using the same method signature, i.e., name and parameters, in both the superclass and the child class is known as overriding.



**29. What is encapsulation?**

Encapsulation is used to wrap the data and the code which works in a single unit together. Example: Encapsulation allows data-hiding as the data specified in one class is hidden from other classes.

**30. What is the difference between public, private and protected access modifiers?**



**31. What is data abstraction?**

Data abstraction is one of the most important features of OOPs. It only allows important information to be displayed. It helps to hide the implementation details.

For example, while using a mobile, you know, how can you message or call someone but you don’t know how it actually happens.

This is data abstraction as the implementation details are hidden from the user.

**32. How to achieve data abstraction?**

Data abstraction can be achieved using two ways:

* Abstract class
* Abstract method

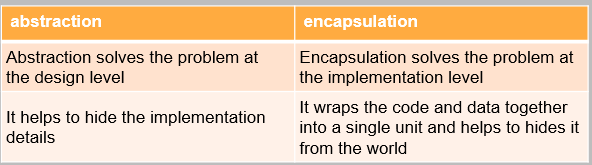
**33. What is an abstract class?**

An abstract class is also a class which is consists of abstract methods.

**So what is an abstract method?**

These methods are basically declared but not defined and If these methods need to be used later in some subclass that time those methods have to be exclusively defined in the subclass.

**34. Differentiate between data abstraction and encapsulation.**



**35. What are virtual functions?**

Virtual functions are also part of the functions which are present in the parent class and they are overridden by the subclass. These functions help to achieve runtime polymorphism.

**36. What is a destructor?**

A destructor is a method that is called automatically when an object is destroyed.

The destructor also recovers the heap space which was allocated to the destroyed object. It also start closing the files and database connections of the object, etc.

**37. What is a copy constructor?**

By copying the members of an existing object, the copy constructor initialises the members of a newly formed object. The argument for the copy constructor is a reference to an object of the same class. Programmers have the option of directly defining the [copy constructor](https://www.mygreatlearning.com/blog/copy-constructor-in-cpp/). The compiler defines the copy constructor if the programmer doesn’t.

**38. What is the use of ‘finalize’?**

Finalize is used to free the unmanaged resources and also help to clean before Garbage Collection(GC). It performs memory management tasks.

**39. What is Garbage Collection(GC)?**

Programming languages like C# and Java include garbage collection (GC) as a memory recovery mechanism. A programming language that supports garbage collection (GC) contains one or more GC engines that automatically release memory space that has been reserved for things the application is no longer using.

**40. What is a final variable?**

A final variable can only receive one explicit initialization. A reference variable that has been marked as final is unchangeable in its object reference. The data included in the object, however, can be modified. As a result, while the object’s state can be altered, its reference cannot.

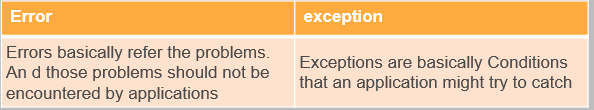
**41. What is an exception?**

An exception is a kind of message that interrupts and comes up when there is an issue with the normal execution of a program. Exceptions provide an error and transfer it to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

**42. What is exception handling?**

Exception handling in Object-Oriented Programming is the most important concept. It is used to manage errors. An exception handler help to throw errors and then catch the error in order to solve them.

**43. What is the difference between an error and an exception?**



**44. What is a try/ catch block?**

The terms “try” and “catch” describe how to handle exceptions brought on by coding or data mistakes while a program is running. The section of code where exceptions occur is called a try block. Exceptions from try blocks are caught and handled in a catch block.

**45. What is a finally block?**

Finally designates the section of code that works with the try keyword. It specifies code that is always executed before the method is finished, immediately behind the try and any catch blocks. Regardless of whether an exception is thrown or caught, the finally block is always executed.

**46. Can you call the base class method without creating an instance?**

Yes, you are allowed to call the base class without instantiating it but there are some conditions that are applicable:

* If it is a static method
* The base class is inherited by some other subclass

**47. What is the difference between OOP and SOP?**

The key distinction between structured and object-oriented programming is that the former allows for the creation of programs using a collection of modules or functions, whilst the latter allows for the construction of programs using a collection of objects and their interactions.

Object-oriented programming involves concepts of objects and classes. Everything is considered as an object which has specific properties and behaviours which are represented in a class. Object-oriented programming provides encapsulation and abstraction in the code. Ex: – Java Programming language.

Structure-oriented programming involves the concepts of functions and structures. Everything is considered functionality and structures, represented using functions—Ex: – C Programming language.

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

Any real-world entity is called an object. The object has specific properties and behaviours, and the similar type of objects having similar features and behaviours are grouped as a class. Hence, Class is a blueprint of objects, and an object is an instance of a class.

Ex: -

1. An Animal is a class, and cat, dog, etc., are objects with common properties like name, type, and common behaviors like speaking, walking, running, etc.

2. Mobile is a class, and Nokia, moto, iPhone, etc., are objects with common properties like modal\_no, color, etc., and common behaviors like audio\_calling, video\_calling, music, etc.

**49. What are ‘access specifiers’?**

Access specifiers are the keywords in any programming language used to specify the Class’s, method’s, interface’s and variable’s behaviour concerning its accessibility. The access specifiers in C++ Programming are public, private, and protected.

**50. Can you create an instance of an abstract class?**

No, an instance of the Abstract class cannot be created. To implement the abstract Class, abstract methods, the Abstract Class should be extended by another class, and the object of the implementation class can be created.

**OOPs Interview Questions for Experienced**

**51. What is an interface?**

An interface is a user-defined [data type](https://www.mygreatlearning.com/blog/types-of-data/) and is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. A class describes an object’s attributes and behaviours, and an interface contains behaviours that a class implements. The Class represents “how,” and the interface represents “what’.

**52. What are pure virtual functions?**

A pure virtual function/method is a function whose implementations are not provided in the base class, and only a declaration is provided. The pure virtual function can have its implementation code in the derived class; otherwise, the derived class will also be considered an abstract Class. The Class containing pure virtual functions is abstract.

**53. Differentiate between a class and a method.**

A class is a blueprint of objects, and it consists of the properties and behaviour of the objects.

Methods are programming constructs that perform specific tasks/behaviour.

**54.  Differentiate between an abstract class and an interface?**

An interface can have only abstract methods, but an Abstract class can have abstract and non-abstract methods.

The interface should be used if just the requirement specification is known and nothing about implementation. If the implementation is known, but partially, then an abstract class should be used. If the implementation is known completely, then a concrete Class should be used.

**55. What are the limitations of OOPs?**

1. Larger Program size – Programs can become lengthy if written using [OOps concepts](https://www.mygreatlearning.com/blog/oops-concepts-in-java/" \t "_blank) compared to procedure-oriented programming.
2. Slower execution – As the number of lines of code to be executed is more comparatively, the execution time is also more.
3. Not suitable for all types of Problems.
4. Testing time is also higher for OOP Solutions.

**56. What are the characteristics of an abstract class?**

1. A class having at least one pure virtual function is called an Abstract class.
2. An Abstract class cannot have objects created, i.e., an abstract class cannot be instantiated, but Object references can be created.
3. An Abstract class can have non-abstract functions and pure virtual functions also.
4. The pure virtual function can have its implementation code in the derived class; otherwise, the derived class will also be considered an abstract Class

**57. What is constructor chaining?**

Constructor chaining is a method to call one constructor from another concerning a current object reference. It can be done in two ways: –

1. Using the “this” keyword, the reference can be made to the constructor in the current class.
2. To call the constructor from the base class “super” keyword will be used.

**58. What is Coupling in OOP, and why is it helpful?**

The degree of dependency between the components is called coupling.

**Types of Coupling**

A. **Tight Coupling** – If the dependency between components is high, these components are called tightly coupled.

Ex: –

Below three Classes are highly dependent on each other hence they are tightly coupled.

class P

{

static int a = Q.j;

}

class Q

{

static int j = R.method();

}

class R

{

public static int method(){

return 3;

}

B.  **Loose Coupling** – If the dependency between components is low, it is called loose coupling. Loose coupling is preferred because of the following reasons:-

1. It increases the maintainability of code
2. It provides reusability of code

**59. Name the operators that cannot be overloaded**

All the operators except the + operator cannot be overloaded.

**60. What is Cohesion in OOP?**

The modules having well-defined and specific functionality are called cohesion.

**Advantages**

It improves the maintainability and reusability of code.

**61. What are the levels of data abstraction?**

Highlighting the set of services by hiding internal implementation details is called abstraction.

By using abstract Class and interface, we can implement abstraction

**62. What are the types of variables in OOP?**

Variables are basic units to store data in RAM for Java programs.

Variables should be declared before using them in Java programming. Variable initialization can be static or dynamic. The syntax for variable declaration and static initialization is: –

***Types of variables***

* **Primitive Variables:**It is used to represent primitive values like int, float, etc.
* Reference Variables: It is used to refer to objects in Java.
* **Instance Variables:** Variables whose value varied from object to object are instance variables. For every object, a separate copy of the instance variable is created. Instance variables are declared within the Class and outside any method/block/constructor
* **Static variables:** For static Variables, a single copy of the variable is created, and that copy is shared between every Class object. The static variable is created during class loading and destroyed at class unloading.
* Static variables can be accessed directly from the static and instance area. We are not required to perform initialization explicitly for static variables, and JVM will provide default values.
* **Local Variables:** Variables declared inside a method or block or constructor are local variables. Hence the scope of local variables is the same as the block’s scope in which we declared that variable.

JVM doesn’t provide default values, and before using that variable, the initialization should be performed explicitly.

**63. What do you understand by Garbage Collection in the OOPs world?**

Garbage collection is a memory recovery technique included in programming languages like C# and Java. A GC-enabled programming language contains one or more garbage collectors that automatically free up memory space allocated to objects that are no longer needed by the program.

**64. Is it possible to run a Java application without implementing the OOPs concept?**

No, since Java programmes are founded on the concept of object-oriented programming models, or OOPs, a Java application cannot be implemented without it.

**65. What is the output of the below code?**

|  |
| --- |
| *class Person* *{* *private String show()* *{* *return “This is a person”;* *}* *}* *class Teacher extends Person* *{* *protected String show()* *{* *return “This is a teacher”;* *}* *}* *public class MathsTeacher extends Person* *{* *@Override    public final String show()* *{* *return “This is a Maths teacher”;* *}* *public static void main(String[] name)* *{* *final Person mt = new MathsTeacher();* *System.out.print(mt.show());* *}* *}* |

The output will be: This is a Maths teacher

**66. Find the output of the below code.**

|  |
| --- |
| *class Arithmetic* *{* *public final double var = 5;* *}* *class DeepArith extends Arithmetic* *{* *public final double var = 10;* *}* *public class AdvancedArith extends DeepArith* *{* *public final double secret = 20;* *public static void main(String[] num)* *{* *Arithmetic arith = new AdvancedArith();* *System.out.print(arith.var);* *}* *}* |

The correct output for this code is 5.

**67. Predict the output of the following.**

|  |
| --- |
| *class Parent* *{* *public void display()* *{* *System.out.println(“Parent”);* *}* *}* *class Child extends Parent* *{ private void display()* *{ System.out.println(“Child”);* *}* *}* *public class main* *{* *public static void main(String args[])* *{* *Parent node = new Child(); node.show();* *}* *}* |

Running this code will generate a compile error as a sub-class function overriding a super class function cannot be given more restrictive access.

**68. Implement a Singleton class in Java.**

public class Singleton {

private static Singleton instance;

private Singleton() {

// Private constructor to prevent instantiation.

}

public static Singleton getInstance() {

if (instance == null) {

synchronized (Singleton.class) {

if (instance == null) {

instance = new Singleton();

}

}

}

return instance;

}

}

**69. Implement a Stack data structure using a LinkedList in Python.**

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

class Stack:

def \_\_init\_\_(self):

self.head = None

def is\_empty(self):

return self.head is None

def push(self, data):

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

def pop(self):

if self.is\_empty():

raise Exception("Stack is empty")

popped\_value = self.head.data

self.head = self.head.next

return popped\_value

**70. Implement an abstract class in C# with an abstract method.**

abstract class Shape

{

public abstract double CalculateArea();

}

class Circle : Shape

{

private double radius;

public Circle(double radius)

{

this.radius = radius;

}

public override double CalculateArea()

{

return Math.PI \* radius \* radius;

}

}

Check out OOPs in Java Video



**Frequently Asked OOPs Interview Questions**

**Q: What are the 4 basics of OOP?**

**A:** OOP stands for Object-Oriented Programming, and its four basic principles are Encapsulation, Abstraction, Polymorphism, and Inheritance. OOP enables programmers to consider software development as if they are working with actual entities. In OOP, some objects have a field where data/knowledge can be stored and can do several methods.

**Q: What is the object-oriented programming interview?**

**A:** Object-Oriented Programming, also usually called OOPS, is a kind of programming that is more object-based and not just based on functions or procedures. Individual objects are collected into several classes. Real-world entities such as inheritance, polymorphism, and hiding are implemented by OOPS into programming. It also enables binding data as well as code together.

**Q: What are the 3 principles of OOP?**

**A:** The three main principles of Object-Oriented Programming are Encapsulation, inheritance, and polymorphism.

**Q: What is the concept of OOPS?**

**A:** OOPS or Object-Oriented Programming System is a programming concept that mainly works based on Encapsulation, Abstraction, Polymorphism, and Inheritance. The usual concept of OOPs is to create objects, use them again all through the program, and finally manipulate these objects to fetch our results.

**Q: Why is OOPS used?**

**A:** The main aim of an Object-Oriented Programming System is to implement real-world entities such as polymorphism, inheritance, hiding, and many more in programming. The aim lies in binding together the data as well as functions that work on them so that other parts of the code cannot access the data other than that function.

**Q: What is polymorphism in OOPS?**

**A:** Polymorphism in an Object-Oriented Programming System is a feature of object-based programming languages that enable a particular routine to use variables of several types at different times. It can also be called the ability of a programming language to present the same interface for different primary data types.

**Q: Who is the father of OOPS?**

**A:** The father of the Object-Oriented Programming System is considered to be Alan Kay by some people. He identified some characteristics as basics to OOP Kay 1993:1. He coined OOPs around 1966 or 1967 when he was at grad school.

**Q: What are the main features of OOPS?**

**A:** Some of the main features in OOPS include Classes, Objects, Data Abstraction, Encapsulation, Inheritance, and Polymorphism. OOP is a programming paradigm that is based on the idea of objects.

**Q: What are the advantages of OOPS?**

**A:** Since OOP is one of the main development approaches which is easily accepted, the advantages are many. Some of the advantages of OOPS include Reusability, Data Redundancy, Code Maintenance, Security, Design Benefits, Easy Troubleshooting, Better Productivity, Polymorphism Flexibility, and Problem-solving.

**Basic OOPs Interview Questions for Freshers**

**1. What is the difference between OOP and SOP?**

|  |  |
| --- | --- |
| Object-Oriented Programming | Structural Programming |
| Object-Oriented Programming is a type of programming which is based on objects rather than just functions and procedures | Provides logical structure to a program where programs are divided functions |
| Bottom-up approach | Top-down approach |
| Provides data hiding | Does not provide data hiding |
| Can solve problems of any complexity | Can solve moderate problems |
| Code can be reused thereby reducing redundancy | Does not support code reusability |

**2. What is Object Oriented Programming?**

Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

**3. Why use OOPs?**

* OOPs allows clarity in programming thereby allowing simplicity in solving complex problems
* Code can be reused through inheritance thereby reducing redundancy
* Data and code are bound together by encapsulation
* OOPs allows data hiding, therefore, private data is kept confidential
* Problems can be divided into different parts making it simple to solve
* The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms

**4. What are the main features of OOPs?**

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

To know more about OOPs in JAVA, Python, and C++ you can go through the following blogs:

[[](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=curriculum)](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)

**[Data Science with Python Certification Course](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=curriculum" \t "_blank)

* [**JAVA** OOPs Concepts](https://www.edureka.co/blog/object-oriented-programming/)
* [**Python** OOPs Concepts](https://www.edureka.co/blog/python-class/)
* [**C++** OOPs Concepts](https://www.edureka.co/blog/object-oriented-programming-in-cpp/)

**Classes and Objects OOPs Interview Questions and Answers**

**5. What is an object?**

An object is a real-world entity which is the basic unit of OOPs for example chair, cat, dog, etc. Different objects have different states or attributes, and behaviors.

**6. What is a class?**

A class is a prototype that consists of objects in different states and with different behaviors. It has a number of methods that are common the objects present within that class.

**7. What is the difference between a class and a structure?**

**Class:**User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.

**Structure:**A structure is basically a user-defined collection of variables which are of different data types.

**8. Can you call the base class method without creating an instance?**

Yes, you can call the base class without instantiating it if:

* It is a static method
* The base class is inherited by some other subclass

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

**Data Science Training**

**[DATA SCIENCE AND MACHINE LEARNING INTERNSHIP PROGRAM](https://www.edureka.co/data-science-and-machine-learning-program?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science and Machine Learning Internship Program" \t "_blank)**

**[Data Science and Machine Learning Internship Program](https://www.edureka.co/data-science-and-machine-learning-program?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science and Machine Learning Internship Program" \t "_blank)**

*[Reviews](https://www.edureka.co/data-science-and-machine-learning-program?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science and Machine Learning Internship Program" \t "_blank)*

**[5](https://www.edureka.co/data-science-and-machine-learning-program?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science and Machine Learning Internship Program" \t "_blank)**[(21125)](https://www.edureka.co/data-science-and-machine-learning-program?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science and Machine Learning Internship Program" \t "_blank)

**[PYTHON CERTIFICATION TRAINING COURSE](https://www.edureka.co/python-programming-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Python Certification Training Course" \t "_blank)**

**[Python Certification Training Course](https://www.edureka.co/python-programming-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Python Certification Training Course" \t "_blank)**

*[Reviews](https://www.edureka.co/python-programming-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Python Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/python-programming-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Python Certification Training Course" \t "_blank)**[(50875)](https://www.edureka.co/python-programming-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Python Certification Training Course" \t "_blank)

**[DATA SCIENCE WITH PYTHON CERTIFICATION COURSE](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with Python Certification Course" \t "_blank)**

**[Data Science with Python Certification Course](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with Python Certification Course" \t "_blank)**

*[Reviews](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with Python Certification Course" \t "_blank)*

**[5](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with Python Certification Course" \t "_blank)**[(118646)](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with Python Certification Course" \t "_blank)

**[DATA ANALYTICS WITH R PROGRAMMING CERTIFICATION TRAINING](https://www.edureka.co/data-analytics-with-r-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Analytics with R Programming Certification Training" \t "_blank)**

**[Data Analytics with R Programming Certification Training](https://www.edureka.co/data-analytics-with-r-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Analytics with R Programming Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/data-analytics-with-r-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Analytics with R Programming Certification Training" \t "_blank)*

**[5](https://www.edureka.co/data-analytics-with-r-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Analytics with R Programming Certification Training" \t "_blank)**[(26538)](https://www.edureka.co/data-analytics-with-r-certification-training?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Analytics with R Programming Certification Training" \t "_blank)

**[STATISTICS ESSENTIALS FOR ANALYTICS](https://www.edureka.co/statistics-essentials-for-analytics-self-paced?utm_source=blogbanner&utm_campaign=trending_course" \o "Statistics Essentials for Analytics" \t "_blank)**

**[Statistics Essentials for Analytics](https://www.edureka.co/statistics-essentials-for-analytics-self-paced?utm_source=blogbanner&utm_campaign=trending_course" \o "Statistics Essentials for Analytics" \t "_blank)**

*[Reviews](https://www.edureka.co/statistics-essentials-for-analytics-self-paced?utm_source=blogbanner&utm_campaign=trending_course" \o "Statistics Essentials for Analytics" \t "_blank)*

**[5](https://www.edureka.co/statistics-essentials-for-analytics-self-paced?utm_source=blogbanner&utm_campaign=trending_course" \o "Statistics Essentials for Analytics" \t "_blank)**[(6540)](https://www.edureka.co/statistics-essentials-for-analytics-self-paced?utm_source=blogbanner&utm_campaign=trending_course" \o "Statistics Essentials for Analytics" \t "_blank)

**[SAS TRAINING AND CERTIFICATION](https://www.edureka.co/sas-training?utm_source=blogbanner&utm_campaign=trending_course" \o "SAS Training and Certification" \t "_blank)**

**[SAS Training and Certification](https://www.edureka.co/sas-training?utm_source=blogbanner&utm_campaign=trending_course" \o "SAS Training and Certification" \t "_blank)**

*[Reviews](https://www.edureka.co/sas-training?utm_source=blogbanner&utm_campaign=trending_course" \o "SAS Training and Certification" \t "_blank)*

**[5](https://www.edureka.co/sas-training?utm_source=blogbanner&utm_campaign=trending_course" \o "SAS Training and Certification" \t "_blank)**[(5498)](https://www.edureka.co/sas-training?utm_source=blogbanner&utm_campaign=trending_course" \o "SAS Training and Certification" \t "_blank)

**[DATA SCIENCE WITH R PROGRAMMING CERTIFICATION TRAINING COURSE](https://www.edureka.co/data-science-r-programming-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with R Programming Certification Training Course" \t "_blank)**

**[Data Science with R Programming Certification Training Course](https://www.edureka.co/data-science-r-programming-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with R Programming Certification Training Course" \t "_blank)**

*[Reviews](https://www.edureka.co/data-science-r-programming-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with R Programming Certification Training Course" \t "_blank)*

**[5](https://www.edureka.co/data-science-r-programming-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with R Programming Certification Training Course" \t "_blank)**[(40135)](https://www.edureka.co/data-science-r-programming-certification-course?utm_source=blogbanner&utm_campaign=trending_course" \o "Data Science with R Programming Certification Training Course" \t "_blank)

**[ANALYTICS FOR RETAIL BANKS](https://www.edureka.co/retail-banking?utm_source=blogbanner&utm_campaign=trending_course" \o "Analytics for Retail Banks" \t "_blank)**

**[Analytics for Retail Banks](https://www.edureka.co/retail-banking?utm_source=blogbanner&utm_campaign=trending_course" \o "Analytics for Retail Banks" \t "_blank)**

*[Reviews](https://www.edureka.co/retail-banking?utm_source=blogbanner&utm_campaign=trending_course" \o "Analytics for Retail Banks" \t "_blank)*

**[5](https://www.edureka.co/retail-banking?utm_source=blogbanner&utm_campaign=trending_course" \o "Analytics for Retail Banks" \t "_blank)**[(1653)](https://www.edureka.co/retail-banking?utm_source=blogbanner&utm_campaign=trending_course" \o "Analytics for Retail Banks" \t "_blank)

**[ADVANCED PREDICTIVE MODELLING IN R CERTIFICATION TRAINING](https://www.edureka.co/advanced-predictive-modeling-in-r-sp?utm_source=blogbanner&utm_campaign=trending_course" \o "Advanced Predictive Modelling in R Certification Training" \t "_blank)**

**[Advanced Predictive Modelling in R Certification Training](https://www.edureka.co/advanced-predictive-modeling-in-r-sp?utm_source=blogbanner&utm_campaign=trending_course" \o "Advanced Predictive Modelling in R Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/advanced-predictive-modeling-in-r-sp?utm_source=blogbanner&utm_campaign=trending_course" \o "Advanced Predictive Modelling in R Certification Training" \t "_blank)*

**[4](https://www.edureka.co/advanced-predictive-modeling-in-r-sp?utm_source=blogbanner&utm_campaign=trending_course" \o "Advanced Predictive Modelling in R Certification Training" \t "_blank)**[(4062)](https://www.edureka.co/advanced-predictive-modeling-in-r-sp?utm_source=blogbanner&utm_campaign=trending_course" \o "Advanced Predictive Modelling in R Certification Training" \t "_blank)

Next

|  |  |
| --- | --- |
| Object | Class |
| A real-world entity which is an instance of a class | A class is basically a template or a blueprint within which objects can be created |
| An object acts like a variable of the class | Binds methods and data together into a single unit |
| An object is a physical entity | A class is a logical entity |
| Objects take memory space when they are created | A class does not take memory space when created |
| Objects can be declared as and when required | Classes are declared just once |

To know more about objects and classes in JAVA, Python, and C++ you can go through the following blogs:

* [Objects in **Java**](https://www.edureka.co/blog/java-object/)
* [Class in **Java**](https://www.edureka.co/blog/java-objects-and-classes/)
* [Objects and classes in **Python**](https://www.edureka.co/blog/python-class/)
* [Objects in **C++**](https://www.edureka.co/blog/object-oriented-programming-in-cpp/#Objects)

**OOPs Interview Questions – Inheritance**

**10. What is inheritance?**

Inheritance is a feature of OOPs which allows classes inherit common properties from other classes. For example, if there is a class such as ‘vehicle’, other classes like ‘car’, ‘bike’, etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

**11. What are the different types of inheritance?**

* Single inheritance
* Multiple inheritance
* Multilevel inheritance
* Hierarchical inheritance
* Hybrid inheritance

**12. What is the difference between multiple and multilevel inheritance?**

|  |  |
| --- | --- |
| Multiple Inheritance | Multilevel Inheritance |
| Multiple inheritance comes into picture when a class inherits more than one base class | Multilevel inheritance means a class inherits from another class which itself is a subclass of some other base class |
| Example: A class defining a child inherits from two base classes Mother and Father | Example: A class describing a sports car will inherit from a base class Car which inturn inherits another class Vehicle |

**13. What is hybrid inheritance?**

Hybrid inheritance is a combination of multiple and multi-level inheritance.

**14. What is hierarchical inheritance?**

Hierarchical inheritance refers to inheritance where one base class has more than one subclasses. For example, the vehicle class can have ‘car’, ‘bike’, etc as its subclasses.

**15. What are the limitations of inheritance?**

* Increases the time and effort required to execute a program as it requires jumping back and forth between different classes
* The parent class and the child class get tightly coupled
* Any modifications to the program would require changes both in the parent as well as the child class
* Needs careful implementation else would lead to incorrect results

To know more about inheritance in Java and Python, read the below articles:

* [Inheritance in Java](https://www.edureka.co/blog/inheritance-in-java/)
* [Inheritance in Python](https://www.edureka.co/blog/inheritance-in-python/)

**16. What is a superclass?**

A superclass or base class is a class that acts as a parent to some other class or classes. For example, the Vehicle class is a superclass of class Car.

[[](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=batches)](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=batches" \t "_blank)

**[Data Science with Python Certification Course](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=batches" \t "_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/data-science-python-certification-course?utm_source=blogbanner&utm_campaign=batches" \t "_blank)

**17. What is a subclass?**

A class that inherits from another class is called the subclass. For example, the class Car is a subclass or a derived of Vehicle class.

Want to upskill yourself to get ahead in your Career? Check out this video

**Top 10 Technologies to Learn in 2022| Edureka**



**OOPs Interview Questions – Polymorphism**

**18. What is polymorphism?**

Polymorphism refers to the ability to exist in multiple forms. Multiple definitions can be given to a single interface. For example, if you have a class named Vehicle, it can have a method named speed but you cannot define it because different vehicles have different speed. This method will be defined in the subclasses with different definitions for different vehicles.

**19. What is static polymorphism?**

Static polymorphism (static binding) is a kind of polymorphism that occurs at compile time. An example of compile-time polymorphism is method overloading.

**20. What is dynamic polymorphism?**

Runtime polymorphism or dynamic polymorphism (dynamic binding) is a type of polymorphism which is resolved during runtime. An example of runtime polymorphism is method overriding.

**21. What is method overloading?**

Method overloading is a feature of OOPs which makes it possible to give the same name to more than one methods within a class if the arguments passed differ.

**22. What is method overriding?**

Method overriding is a feature of OOPs by which the child class or the subclass can redefine methods present in the base class or parent class. Here, the method that is overridden has the same name as well as the signature meaning the arguments passed and the return type.

**23. What is operator overloading?**

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it.

**24. Differentiate between overloading and overriding.**

|  |  |
| --- | --- |
| Overloading | Overriding |
| Two or more methods having the same name but different parameters or signature | Child class redefining methods present in the base class with the same parameters/ signature |
| Resolved during compile-time | Resolved during runtime |

To know more about polymorphism in Java and Python, read the below articles:

* [Polymorphism in Java](https://www.edureka.co/blog/polymorphism-in-java/)
* [Polymorphism in Python](https://www.edureka.co/blog/object-oriented-programming-python/#Polymorphism)

**OOPs Interview Questions – Encapsulation**

**25. What is encapsulation?**

Encapsulation refers to binding the data and the code that works on that together in a single unit. For example, a class. Encapsulation also allows data-hiding as the data specified in one class is hidden from other classes.

**26. What are ‘access specifiers’?**

[Access specifiers or access modifiers are keywords](https://www.edureka.co/blog/access-modifiers-in-java/) that determine the accessibility of methods, classes, etc in OOPs. These access specifiers allow the implementation of encapsulation. The most common access specifiers are public, private and protected. However, there are a few more which are specific to the programming languages.

**27. What is the difference between public, private and protected access modifiers?**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Accessibility from own class | Accessibility from derived class | Accessibility from world |
| Public | Yes | Yes | Yes |
| Private | Yes | No | No |
| Protected | Yes | Yes | No |

To know more about encapsulation read along:

* [Encapsulation in Java](https://www.edureka.co/blog/object-oriented-programming/#encapsulation)
* [Encapsulation in C++](https://www.edureka.co/blog/encapsulation-in-cpp/)
* [Encapsulation in Python](https://www.edureka.co/blog/object-oriented-programming-python/#Encapsulation)

**Data abstraction**

**28. What is data abstraction?**

Data abstraction is a very important feature of OOPs that allows displaying only the important information and hiding the implementation details. For example, while riding a bike, you know that if you raise the accelerator, the speed will increase, but you don’t know how it actually happens. This is [data abstraction](https://www.edureka.co/blog/data-abstraction-in-cpp/) as the implementation details are hidden from the rider.

**29. How to achieve data abstraction?**

Data abstraction can be achieved through:

* Abstract class
* Abstract method

**30. What is an abstract class?**

An abstract class is a class that consists of abstract methods. These methods are basically declared but not defined. If these methods are to be used in some subclass, they need to be exclusively defined in the subclass.

**31.** **Can you create an instance of an abstract class?**

No. Instances of an abstract class cannot be created because it does not have a complete implementation. However, instances of subclass inheriting the abstract class can be created.

**32. What is an interface?**

It is a concept of OOPs that allows you to declare methods without defining them. Interfaces, unlike classes, are not blueprints because they do not contain detailed instructions or actions to be performed. Any class that implements an interface defines the [methods of the interface](https://www.edureka.co/blog/java-interface/).

**33. Differentiate between data abstraction and encapsulation.**

|  |  |
| --- | --- |
| Data abstraction | Encapsulation |
| Solves the problem at the design level | Solves the problem at the implementation level |
| Allows showing important aspects while hiding implementation details | Binds code and data together into a single unit and hides it from the world |

To know more about data abstraction, below articles might help you:

* [Abstraction in **Java**](https://www.edureka.co/blog/java-abstraction/)
* [Abstraction in **Python**](https://www.edureka.co/blog/object-oriented-programming-python/#Abstraction)

**Methods and Functions OOPs interview questions**

**34. What are virtual functions?**

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

**35. What are pure virtual functions?**

Pure virtual functions or [abstract functions](https://www.edureka.co/blog/virtual-function-in-cpp/) are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

**36. What is a constructor?**

A constructor is a special type of method that has the same name as the class and is used to initialize objects of that class.

**37. What is a destructor?**

A destructor is a method that is automatically invoked when an object is destroyed. The destructor also recovers the heap space that was allocated to the destroyed object, closes the files and database connections of the object, etc.

**38. Types of constructors**

[Types of constructors](https://www.edureka.co/blog/python-constructors/) differ from language to language. However, all the possible constructors are:

* Default constructor
* Parameterized constructor
* Copy constructor
* Static constructor
* Private constructor

**39. What is a copy constructor?**

A [copy constructor](https://www.edureka.co/blog/constructor-in-java/) creates objects by copying variables from another object of the same class. The main aim of a copy constructor is to create a new object from an existing one.

**40. What is the use of ‘finalize’?**

Finalize as an object method used to free up unmanaged resources and cleanup before Garbage Collection(GC). It performs memory management tasks.

**41. What is Garbage Collection(GC)?**

GC is an implementation of automatic memory management. The Garbage collector frees up space occupied by objects that are no longer in existence.

**42. Differentiate between a class and a method.**

|  |  |
| --- | --- |
| Class | Method |
| A class is basically a template that binds the code and data together into a single unit. Classes consist of methods, variables, etc | Callable set of instructions also called a procedure or function that are to be performed on the given data |

**43. Differentiate between an abstract class and an interface?**

|  |  |  |
| --- | --- | --- |
| Basis for comparison | Abstract Class | Interface |
| Methods | Can have abstract as well as other methods | Only abstract methods |
| Final Variables | May contain final and non-final variables | Variables declared are final by default |
| Accessibility of Data Members | Can be private, public, etc | Public by default |
| Implementation | Can provide the implementation of an interface | Cannot provide the implementation of an abstract class |

**44. What is a final variable?**

A variable whose value does not change. It always refers to the same object by the property of non-transversity.

**OOPs Interview Questions – Exception Handling**

**45. What is an exception?**

An exception is a kind of notification that interrupts the normal execution of a program. Exceptions provide a pattern to the error and transfer the error to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

**46. What is exception handling?**

Exception handling in Object-Oriented Programming is a very important concept that is used to manage errors. An exception handler allows errors to be thrown and caught and implements a centralized mechanism to resolve them.

**47. What is the difference between an error and an exception?**

|  |  |
| --- | --- |
| **Error** | **Exception** |
| Errors are problems that should not be encountered by applications | Conditions that an application might try to catch |

**48. What is a try/ catch block?**

A try/ catch block is used to handle exceptions. The try block defines a set of statements that may lead to an error. The catch block basically catches the exception.

**49. What is a finally block?**

A finally block consists of code that is used to execute important code such as closing a connection, etc. This block executes when the try block exits. It also makes sure that finally block executes even in case some unexpected exception is encountered.

**OOPs Interview Questions – Limitations of OOPs**

**50. What are the limitations of OOPs?**

* Usually not suitable for small problems
* Requires intensive testing
* Takes more time to solve the problem
* Requires proper planning
* The programmer should think of solving a problem in terms of objects

OOPs Interview Questions And Answers

Here are the most frequently asked OOPs Interview Questions and Answers for freshers and experienced for getting programming jobs.

1) What are OOPs?

OOPs(Object Oriented Programming) is a programming concept that creates objects for data and methods. It works on the principles of encapsulation, classes, abstraction, aggregation, polymorphism, and inheritance. OOPs aims to create, re-use, and manipulate objects throughout the program to get results.

OOPs, are popularly used in modern programming languages like Java.

|  |
| --- |
| If you would like to Enrich your career with a **Python-certified professional**, then **visit Mindmajix** - A Global online training platform:  “**[Python Training](https://mindmajix.com/python-training" \o "Python Training" \t "_blank)**” Course.  This course will help you to achieve excellence in this domain. |

2) What are the four basics of OOPs?

The four main basics of OOPs in Java are:

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

**Abstraction -**It means using simple things to represent complexity.

**Encapsulation -** It’s a practice of keeping fields in a private class, then accessing through public methods.

**Inheritance -** It’s a unique feature of OOPs that lets users create new classes sharing some of the existing classes’ attributes.

**Polymorphism -**It lets you use the same word to mean different things in different contexts. Method overriding and Method overloading are the two forms of Polymorphism. Method overloading occurs when the code itself implies different meanings. Method Overriding occurs when the values of the supplied variables indicate different meanings.

3) What are the benefits of OOPs?

OOPs is a core development approach used in modern programming languages. Let’s see the advantages of OOPs that it offers:

* Code reusability through inheritance
* [Data redundancy](https://www.techopedia.com/definition/18707/data-redundancy)
* Security
* Code maintenance
* Easy troubleshooting
* Better productivity
* Polymorphism flexibility
* Effective problem solving

4) What are the principles of OOPs?

The five concepts that make up solid principles of OOPs:

* Single Responsibility Principle.
* Liskov Substitution principle.
* Open/Closed principle.
* Dependency Inversion principle.
* Interface Segregation Principle.

5) Who invented OOPs?

Alan Kay put the idea of object orientation in the early 1970s. The concept includes classes, multiple instances of classes, and the message passing between the objects of one class and another.

[[](https://bit.ly/3if9dmk)](https://bit.ly/3if9dmk" \o "Subscribe MindMajix YouTube Channel" \t "_blank)

6) What is a class and object in OOPs?

* A class is a set of instructions or a blueprint to build a specific type of object. It determines what an object will contain and how it behaves.

**Syntax:**

class <class\_name>{

field;

method;

}

* An object is an instance of the class. It’s nothing but a self-contained component consisting of properties and methods to make a particular type of data useful.

**Syntax:**

ClassName ReferenceVariable = new ClassName();

7) What’s the difference between class and Object?

The following table lists the differences between object and class:

|  |  |
| --- | --- |
| **Class** | **Object** |
| Class is a template or blueprint from which objects are created. | The object is a class instance. |
| It’s a group of similar objects. | It’s a real-world entity. |
| Logical entity | Physical entity |
| Declared once | It can be declared many times based on the requirement. |
| It does not allocate memory when its created | Allocates memory when created. |
| The class keyword is the only way used to create a class. | There are many ways to create the object, such as a new keyword, clone() method, newInstance(), factory method, and deserialization. |

8) Explain the use of abstraction in OOPs?

* One of the main concepts of OOPs is Abstraction. It handles the program complexity and improves efficiency by hiding unnecessary information from the user, showing only essential attributes.
* Abstraction is selecting data from a large pool to show only relevant data of the object to the user.
* For example, you want to create a bank app and collect all your customer details.
* The customer details you come up with might include full name, address, contact, tax information, favorite place, etc. But only a few of them are required to create a bank app.
* Relevant data like name, address, etc., make sense for a banking application.
* Since we have fetched/selected/removed the customer information from a larger pool, the process is called abstraction in OOPs.

However, the same information, once extracted, can be used for a wide range of applications. For instance, you can use the same data for job portal applications, hospital applications, a Government database, etc., with little or no modifications. Hence, it becomes your Master Data. That is an advantage of Abstraction in OOPs.

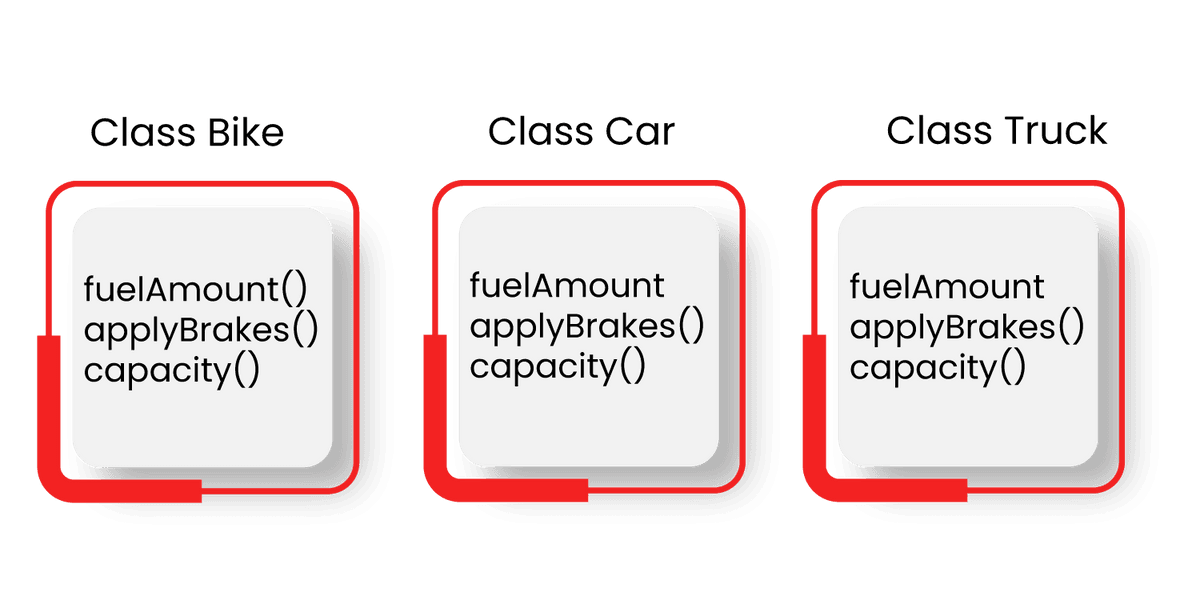
9) What is encapsulation?

* Encapsulation in object-oriented programming is referred to as the bundling of data, along with the methods that operate on that data in a single unit.
* Many programming languages use encapsulation in the form of classes. A class is a program code template that allows you to create objects that contain both variables (data) and behaviors (methods or functions).
* A class consists of data and its methods bundled into a single unit.
* Encapsulation can also hide the internal representation or state of an object from the outside. It can hide both data functions and data members associated with instantiated class or object.

10) Explain the concept of inheritance in OOPs.

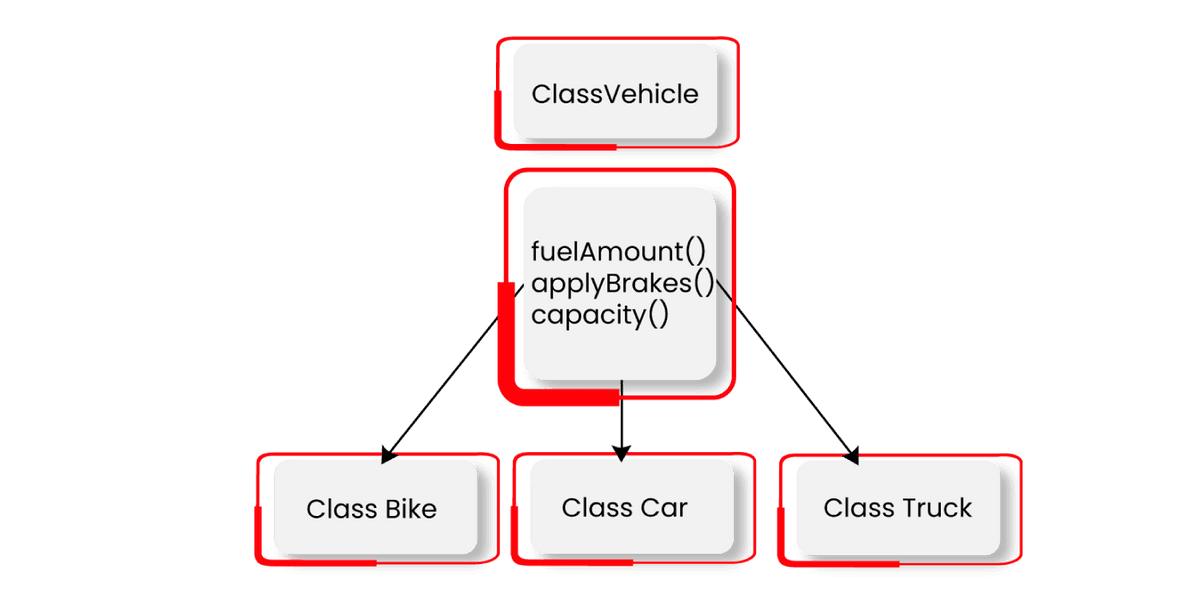
* Inheritance is one of the core concepts of OOPs, in which one class acquires the property of another class in the same way, as a child inherits some attributes from its parents.
* With inheritance, we can reuse the methods and fields of the existing class and facilitate reusability.
* It declares different kinds of exceptions, adds custom logic to the existing framework, and even maps the database’s domain model.
* For example, consider a group of vehicles. For that, you need to create classes for Bike, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be the same for all of the three classes.

If these classes are created avoiding inheritance, then all these functions are written in each of the classes as shown below:



This creates deduplication of the same code three times. This increases data redundancy and the chances of errors. To avoid this, inheritance is used. If you create a class named Vehicle and write these functions in it and inherit the rest of the classes, we can avoid data duplication and improve reusability.

Look at the below diagram to understand how three classes inherited from vehicle class:



OOPs supports different types of inheritance as given below:

* Single inheritance.
* Multiple inheritances.
* Multi-level inheritance.
* Hierarchical Inheritance.
* Multipath inheritance.
* Hybrid Inheritance.

OOPs Basic Interview Questions

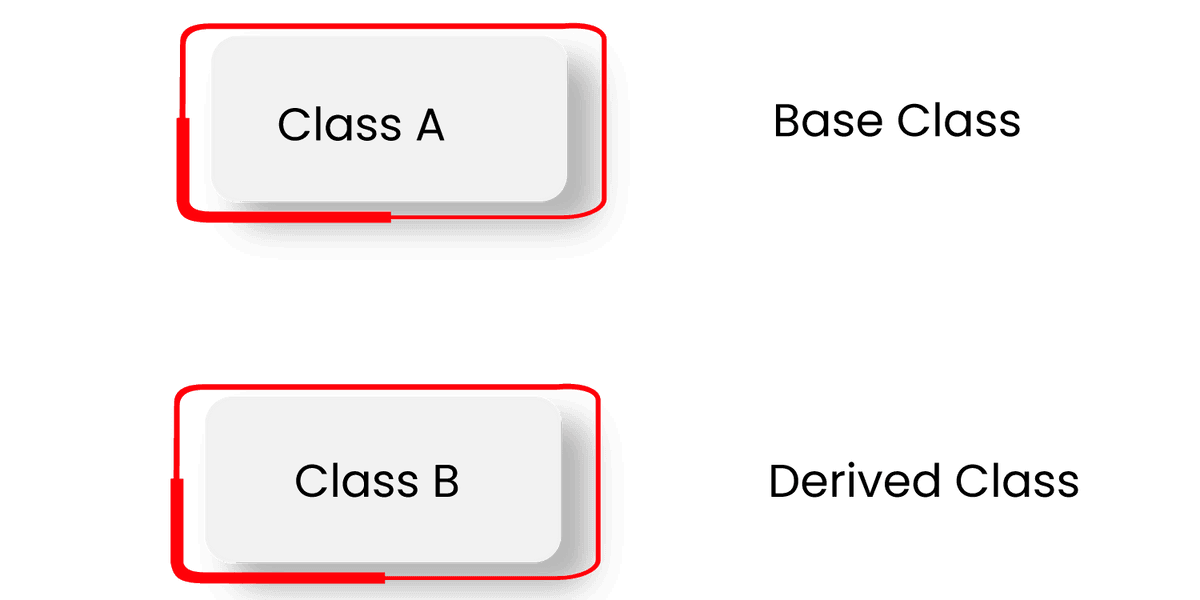
This section covers Basic OOPs interview questions which will help you with different expertise levels to reap maximum benefit.

11) What are the limitations of Inheritance?

* The main disadvantage of using inheritance is two classes get tightly coupled. That means one cannot be used independently of the other. If a method or aggregate is deleted in the Super Class, we have to refactor using that method in SubClass.
* Inherited functions work slower compared to normal functions.
* Need careful implementation otherwise leads to improper solutions.

12) Define Single and multiple inheritances?

* Single inheritance allows a derived class to inherit properties from a single parent class ().



In the above diagram, Class B (subclass) extends only Class A(superclass).

Syntax:

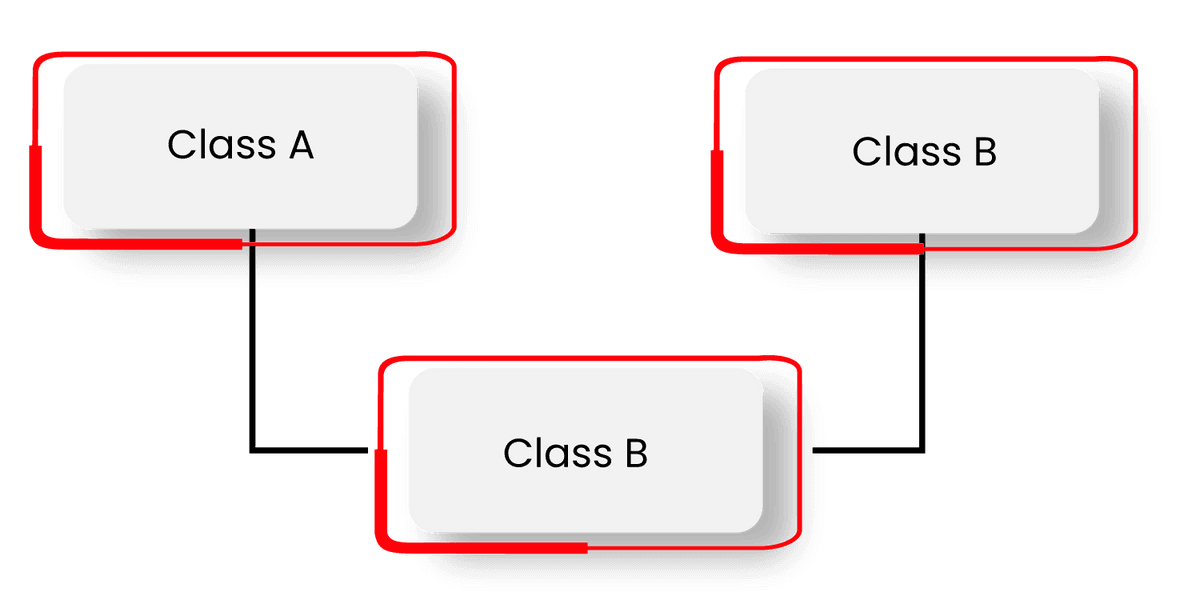
class subclass\_name : access\_mode base\_class

{

//body of subclass

};

* Multiple Inheritance is where a class can inherit from more than one class.



Syntax:

class subclass\_name : access\_mode base\_class1, access\_mode base\_class2, ....

{

//body of subclass

};

13) What is the difference between multiple and multilevel inheritances? Give suitable examples to illustrate both.

|  |  |
| --- | --- |
| **Multiple Inheritance** | **Multilevel Inheritance** |
| Multiple inheritances come into the picture when one class extends more than one class. | Multilevel inheritance comes into the picture when we create a derived from another derived class. |
| Multiple Inheritance | Multilevel Inheritance |

14) What’s the difference between hybrid and hierarchical inheritances?

|  |  |
| --- | --- |
| **Hybrid Inheritance** | **Hierarchical Inheritance** |
| It’s a combination of both single and multiple inheritances. | More than one sub-class inherited from the base class in this type of inheritance means more than one derived class from a single base class. |
| Hybrid Inheritance | Hierarchical Inheritance |

15) What does Polymorphism in OOPs mean?

As the word suggests, poly means ‘many’ and morph points at ‘forms’. On the whole, it means property of many forms. In OOPs, polymorphism processes objects of different types and classes through a single, uniform interface. It allows us to perform a single action in different ways.  It implements the concept of overriding, overloading, and virtual functions. Also, it can be used for inheritance in programming.

For example, if you define a class called Vehicle, it can have a speed method but cannot be defined as different vehicles with different speeds. We define this method in the subclasses with different definitions for different vehicles.

16) What are the different types of Polymorphism?

OOPs supports two different types of Polymorphism as below:

1. Static Binding (or Compile time) Polymorphism
2. [Dynamic Binding](https://en.wikipedia.org/wiki/Dynamic_binding) (or Runtime) Polymorphism

Static Binding or Compile time polymorphism

This polymorphism type uses method overloading or function overloading. Certain conditions are conducive for static polymorphism as below:

* Parameter types should be different.
* The sequence of parameters can be different.
* A number of parameters for one method should be different from another method.
* The matching type and number of arguments of static polymorphism invoke the overloaded functions.
* Dynamic Binding or Runtime polymorphism
* This polymorphism type uses method overriding. Through pointers and virtual functions, we can achieve overriding.
* When a derived class has a definition for one of the base class’s member functions, that base class is method overridden.

17) Describe method overloading.

In OOPs, method overloading is a feature that allows a class to have one or more methods that have the same name with different arguments lists. Overloading is related to compile-time (or static) polymorphism.

Method Overloading is achieved either through:

* Changing the number of arguments

                                             Or

* By changing the data type of parameters

Following operators cannot be overloaded:

* Member selection through a pointer to function (.\*)
* Scope Resolution (::)
* Member Selection (.)

18) What is method overriding?

Method Overriding in OOPs is a feature that allows a child class or subclass to provide a specific implementation of a method that is already provided by one of its parent class or superclass. Method overriding is related to runtime polymorphism.

Conditions for method overriding:

* The method must have the same name as in the parent class.
* There must be an IS-A relationship (inheritance).

19) Abstraction vs. encapsulation: What are the differences between both of them?

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| Abstraction shows only useful data by providing required details. | Encapsulation wraps code and data for required information. |
| It solves problems at the design or interface level. | It solves problems at the implementation level. |
| It’s a method to hide unwanted information. | It’s a method to hide and protect data from inside and outside. |
| Abstract classes and interfaces implement abstraction. | You can implement encapsulation using access modifiers: public, private, and protected. |
| Objects that perform abstraction can encapsulate. | Objects that perform encapsulation cannot perform abstraction. |
| In abstraction, the implementation complexities hide using abstract classes and interfaces. | While in encapsulation, the data hides using methods of getters and setters. |

20) Can you have polymorphism without inheritance?

* Both inheritance and polymorphism are independent of each other but are related entities.
* If you use a language ([Java](https://mindmajix.com/java-tutorial), C++, C#) that requires variables to have a specific type, then the concepts of inheritance and polymorphism are linked.
* If you use a language (javascript, [python](https://mindmajix.com/python-tutorial), ruby, vb.net ) with a generic type of declaration, then the concepts of polymorphism and inheritance are separated.

|  |
| --- |
| **Explore**[**Frequency Asked Core Java Interview Questions**](https://mindmajix.com/core-java-interview-questions) |

OOPs Advanced Interview Questions

The following section discusses the most advanced OOPs interview questions that you should absolutely know to crack the OOPs-related interviews.

21) Inheritance vs. Polymorphism: What are the differences between inheritance and polymorphism?

|  |  |
| --- | --- |
| **Inheritance** | **Polymorphism** |
| Inheritance is one in which a derived class inherits the already existing class’s features. | Polymorphism is one that you can define in different forms. |
| Applied to classes | Applied to functions or methods |
| Used for pattern designing | Used for pattern designing |
| Inheritance can be single, hybrid, multiple, hierarchical, multipath, and multilevel inheritances. | Polymorphism can be compiled-time (overload) and run-time polymorphism (overriding). |
| Reduces code length and supports code reusability | Allows the object to decide which form of the function to implement at run-time (overriding) and compile-time (overloading). |

22) What’s the difference between association and aggregation?

|  |  |
| --- | --- |
| **Association** | **Aggregation** |
| Association refers to "has a" relationship between two classes that use each other. | Aggregation "has a"+ relationship between two classes where one contains the collection of other class objects. |
| Linkage is needed to maintain the association. | The linkage between objects is not mandatory. |
| Lines represent the association. | Diamond shape next to assembly class is used to represent the aggregation relationship. |
| Inflexible in nature | Flexible in nature |

23) What are access modifiers in OOPs?

In OOPs, access modifiers are keywords that set the accessibility of methods, classes, constructors, and other members. You can change the access level of fields, methods, class, and constructor by applying the access modifier on it.

Access modifiers can also be called access specifiers.

24) What are the different types of access modifiers?

OOPs supports four types of access modifiers:

1. **Public:**You can access this modifier from everywhere. This modifier’s access level is within the package, outside the package, within, and outside the class.
2. **Private:**You can access this modifier outside the class. The access level of this modifier is only within the class.
3. **Protected:** This modifier’s access level is within the package, through the child class, and outside the package. If you don’t define child class, you cannot access it from outside the package.
4. **Default:** This modifier’s access level is only within the package, and you cannot access it outside the package. If the access level is not specified, it will be the default.

25) What is a constructor?

A constructor is a particular type of subroutine called to create a project in object-oriented programming. It resembles an instance method but differs as it has no explicit return type.

A constructor is different from regular functions in the following ways:

* Constructors don’t have a return type.
* The constructor has the same name as the class itself.
* A constructor is called when you create an object.

26) Do we require a parameter for constructors?

No, you don’t require a parameter for constructors.

27) What are the different types of constructors?

* Parameterized constructors
* Default constructors
* Copy constructors
* Conversion constructors
* Move constructors

28) Can the constructor be private?

Yes, you can declare a constructor as private. Once a constructor is declared private, you cannot create an object of a class. We use private constructors in the Singleton Design Patterns.

Rules for private constructors:

* The private constructor does not allow object creation outside the class and a class to be subclassed.
* We can use a private constructor when all the methods are static.
* If we extend a class containing a private constructor, then a compile-time error will occur.

29) Why is the constructor not overridden?

You cannot override the constructor because it looks like a method, but it is not. It doesn’t have the return type, and the name is the same as the class name. If you treat it as a method and write a super class's constructor in the sub-class compiler expecting a return type, it will generate a compile-time error.

30) Can you use this () and super () both in a constructor?

* Both super() and this() functions are used to make constructor calls. this() is used to call the current class constructor, while super() is used to call the base class’s constructor.
* But we can’t use both this() and super() together in a constructor, as it will give a compile-time error.
* Because super() and this() must be the first executable statement, if anyone is written first, the other will become the second statement and vice-versa. That's why we can't use this() and super() together.

OOP's Interview Questions For Experienced

In this section, we introduce you to the most commonly asked questions for experienced professionals in OOPs interviews for the year 2021.

31) Can the constructor be static or final?

No, the constructor can’t be made static or final.

32) Why do you use an abstract class in oops?

Declaring a class as an abstract means that it cannot be directly instantiated, which means the object cannot be created from it. It protects the code from being misused. Abstract classes need subclasses to define attributes for individual instantiation further.

33) How do you access an abstract class?

The only way to access the non-static method of an abstract class is to extend it, implement the abstract methods in it (if any), and then use the subclass object you need to invoke the required methods.

34) Why do you need a virtual function?

A virtual function is a member function declared within a base class and redefined by a derived class. When a class containing a virtual function is inherited, the derived class can redefine the virtual function to satisfy its requirements.

35) What are the final variables or methods?

We use a final keyword for declaring an entity. A variable declared with the final keyword means its value can't be changed, essentially, a constant. A method declared with the final keyword cannot? be overridden or hidden by subclasses.

36) What is the difference between virtual and pure virtual functions?

A virtual function is a member function of a base class that a derived class can redefine. A pure virtual function is a member function of a base class whose declaration is provided only in the base class and defined in the derived class, or else the derived class can also become abstract.

37) What is the purpose of a virtual destructor?

Virtual destructor helps to destruct the resources correctly when you delete a base class pointer pointing to the derived class object.

38) Can you override a final method?

No, you can’t override the final method.

39) Can you inherit a final class?

If you make a class final, then no class can inherit the final class feature. You cannot extend a final class.

40) Explain Garbage collections

Garbage collection(GC) is reclaiming unused runtime memory automatically. The GC frees up the space occupied by the objects that are no longer in existence.

Different types of Garbage collectors are:

* CMS Garbage Collector.
* Serial Garbage Collector.
* Parallel Garbage Collector.
* G1 Garbage Collector.

Python OOP's Interview Questions

Following are the most frequently asked OOPs interview questions in Python for both freshers as well as experienced professionals.

41) What is an Interface?

The interface is an essential concept of OOPs that allows you to declare methods without defining them. Unlike classes, interfaces are not blueprints because they don’t contain specific instructions or actions to be performed. Any class that implements an interface defines the methods of the interface.

42) What is an exception in object-oriented programming?

An exception is a type of notification that interrupts the usual program execution. Exceptions help you to detect and react to unexpected events. The program’s state is saved when an exception arises, and control is passed to an exception handler. The exceptions are thrown or raised by programming code that must send a signal to the executing program about an error or an unusual situation.

For example, when you want to open a file that doesn’t exist, the code responsible for opening the file will detect this and throw an exception with a proper error message. Exceptions are of two types in OOPs, such as checked exceptions and unchecked exceptions.

* **Checked exception -** The classes which inherit compile-time exceptions are known as checked exceptions.
* **Unchecked exception -** The classes which inherit Runtime exceptions are known as unchecked exceptions.

43) What’s the difference between error and exception?

|  |  |
| --- | --- |
| **Error** | **Exception** |
| Classified as an unchecked type | Classified as a checked and unchecked type |
| Error recovery is not possible. | Exception recovery is possible using a try-catch block or throwing exceptions back to the caller. |
| Errors are caused by the program's running environment. | The program itself causes exceptions. |
| It occurs at runtime and is not known to the compiler. | It occurs at runtime but checked exceptions are known to the compiler while unchecked is not. |

44) What is exception handling?

In OOPs, exception handling is used to handle errors. It allows errors to be thrown and caught and implements a mechanism to resolve them.

|  |
| --- |
| Read Top [**Python Interview Questions and Answers**](https://mindmajix.com/python-interview-questions) |

45) Can a base class be called without creating an instance?

Yes, you can call a base call without instantiating it if:

* Base class instantiated by some other subclass.
* It’s a static method.

46) What is the use of try/ catch block?

You can use the try/ catch block to handle exceptions. Try statements define the statements that lead to error. Catch block catches the exception.

47) What is a ternary operator?

The ternary operator is also called a conditional operator. It’s an operator that takes three arguments. Results and arguments are of different data types depending on the function.

|  |
| --- |
| Explore [**Python Sample Resumes**](https://mindmajix.com/python-sample-resumes) Download & Edit, Get Noticed by Top Employers! |

48) What is early and late Binding?

Early binding refers to the assignment of values during design time to variables. On the other hand, late binding refers to the assignment of values during run time to variables.

49) What is the difference between override and new?

The override modifier overrides the base class function. Simultaneously, a new modifier instructs the compiler to use a new implementation instead of the base class function.

50) What are the disadvantages of OOPs?

The disadvantages of object-oriented programming are:

* **Steep learning curve -**The concepts involved with OOPs are challenging to comprehend initially and can take a long time to get used to them.
* **Slower programs -** OOPs are typically slower than procedure-based programs as it requires more instructions for execution.
* **Larger program size -** OOPs programs are much larger than other programs. It involves more lines of code.
* **Not suitable for all types of problems -** Some problems lend themselves well to logic-programming style, functional-programming style, or procedure-based programming style, and applying OOPs in those situations will result in inefficient programs.

**Top OOPs Interview Questions and Answers for 2023**

**Q1. What is Object-Oriented Programming?**

**Ans.** [Object-Oriented Programming(OOPs)](https://www.shiksha.com/online-courses/articles/web-stories/learn-oops-with-analogy/) is a [computer programming](https://www.shiksha.com/online-courses/what-is-programming-st619) model based on objects instead of functions and procedures. The individual objects are grouped into classes. In OOP, the data and the functions are closely bound together so that the data cannot be accessed by any other part of the code except that function. Object-oriented programming breaks down complex things as simple structures. This paradigm offers many concepts, such as [inheritance](https://www.shiksha.com/online-courses/articles/inheritance-in-java/), [abstraction](https://www.shiksha.com/online-courses/articles/abstraction-in-python/), [encapsulation](https://www.shiksha.com/online-courses/articles/encapsulation-in-python/), and [polymorphism](https://www.shiksha.com/online-courses/articles/polymorphism-in-java/).

***Check out this***[***OOPs explanation with an Analogy***](https://www.shiksha.com/online-courses/articles/web-stories/learn-oops-with-analogy/)***.***

**Compare Top courses:**

|  |  |
| --- | --- |
| [**Top Programming Courses**](https://www.shiksha.com/online-courses/programming-courses-certification-training-st619) | [**Free Java Courses**](https://www.shiksha.com/online-courses/free-java-courses-certification-training-st619-tg17) |
| [**Free C++ Courses**](https://www.shiksha.com/online-courses/free-c-plus-plus-courses-certification-training-st619-tg1441) | [**Free Python Courses**](https://www.shiksha.com/online-courses/free-python-courses-certification-training-st619-tg21) |

**Q2. Name the main features of OOPs.**

**Ans.** The main features of Object Oriented Programming Language are:

* [Classes and Objects](https://www.shiksha.com/online-courses/articles/classes-in-c-plus-plus/)
* [Data Abstraction](https://www.shiksha.com/online-courses/articles/data-abstraction-in-java/)
* [Encapsulation](https://www.shiksha.com/online-courses/articles/encapsulation-in-c-plus-plus-with-a-real-life-example/)
* [Inheritance](https://www.shiksha.com/online-courses/articles/inheritance-in-python/)
* [Polymorphism](https://www.shiksha.com/online-courses/articles/polymorphism-in-java/)

**Q3. What is a Class in OOP?**

**Ans.** A [class](https://www.shiksha.com/online-courses/articles/classes-in-c-plus-plus/) is like a template or blueprint of a house. Just like using blueprint you can build as many houses and modify each of them with different interiors, similarly using a class one can use it to describe one or multiple objects. A class is a collection of variables and methods which are common to all objects of a particular kind.

**Q4. What is an Object in OOP?**

**Ans.** Each house you build (or instantiate) is an [object](https://www.shiksha.com/online-courses/articles/classes-in-c-plus-plus/), also known as an instance of class. An object is the basic unit of Object-Oriented Programming. The term “object” in object-oriented programming refers to a specific instance of a class which can be a combination of variables, [functions](https://www.shiksha.com/online-courses/articles/functions-in-python/), and [data structures](https://www.shiksha.com/online-courses/articles/data-structures-and-algorithms-in-python-all-you-need-to-know/)within in a class.

**Explore:**[**Data Structure Interview Question**](https://www.shiksha.com/online-courses/articles/top-universities-offering-free-online-courses-for-programmers/)

**Q5. What is the difference between a Class and a Structure?**

**Ans.** The differences between a class and a structure are:

|  |  |
| --- | --- |
| **Class** | **Structure** |
| Class is a collection of objects with common properties | The structure is a group of different data types |
| It is a reference data type | It is a value data type |
| A Class uses the keyword “class” | A Structure uses the keyword “struct” |
| Supports Inheritance | Does not support Inheritance |
| An object is created in the heap memory | An object is created in the stack memory |
| Members are private by default | Members are public by default |
| An instance of a ‘class’ is ‘object’ | An instance of ‘structure’ is ‘structure variable’ |
| **Declaration syntax:** class class\_name{ data member; member function; }; | **Declaration syntax:** struct structure\_name{ type struct\_element 1; type struct\_element 2; type struct\_element 3; . }; |

**Also read:**[**Difference between Class and Structure**](https://www.shiksha.com/online-courses/articles/difference-between-class-and-structure/)

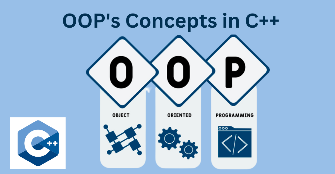
**Q6. Which are the widely used OOP languages?**

**Ans.** Some of the commonly asked OOP languages include:

* [Java](https://www.shiksha.com/online-courses/what-is-java-st619-tg17)
* [Python](https://www.shiksha.com/online-courses/what-is-python-st619-tg21)
* [C++](https://www.shiksha.com/online-courses/what-is-c-plus-plus-st619-tg1441#description)
* Go
* C#
* Ruby

**[Object Oriented Programming (OOPs) Concept in Java](https://www.shiksha.com/online-courses/articles/oops-concepts-in-java/)**

[Learn about Java OOPs Concepts with examples, oops features, abstraction, encapsulation, inheritance, and Polymorphism.](https://www.shiksha.com/online-courses/articles/oops-concepts-in-java/)

[[](https://www.shiksha.com/online-courses/articles/oops-concepts-in-cpp/)](https://www.shiksha.com/online-courses/articles/oops-concepts-in-cpp/)

**[OOPs Concepts in C++](https://www.shiksha.com/online-courses/articles/oops-concepts-in-cpp/)**

[This blog will make it easy for you to learn OOPs concepts with a real-life analogy. Also, you will learn](https://www.shiksha.com/online-courses/articles/oops-concepts-in-cpp/)**[...read more](https://www.shiksha.com/online-courses/articles/oops-concepts-in-cpp/)**

**[Learn Object Oriented Programming Structure - OOPS with Analogy](https://www.shiksha.com/online-courses/articles/web-stories/learn-oops-with-analogy/)**

[Understand OOPS in simple terms with analogies from Squid Game. Learn about inheritance, polymorphism, abstraction, and encapsulation in simple terms](https://www.shiksha.com/online-courses/articles/web-stories/learn-oops-with-analogy/)**[...read more](https://www.shiksha.com/online-courses/articles/web-stories/learn-oops-with-analogy/)**

**Q7. What are the advantages of OOP?**

**Ans.** The advantages of OOP are:

**1. Reusability of code –**

In Object-oriented programming, we can write a class once and use it many times by creating the object for the class.

**2. Code Maintenance –**

The existing code can be easily maintained and modified. New objects can be created by making small changes to existing codes.

**3. Data Redundancy –**

If we require a similar functionality in multiple classes, we can write common class definitions for similar functionalities and inherit them.

**4. Security –**

Using data hiding and abstraction, we can filter out limited data to view or exposure. This helps in maintaining security.

**5. Design Benefits**

Designers go through an extensive design phase for object-oriented programs. This results in providing better designs and fewer flaws.

All these features enable programmers to achieve higher productivity, reduce maintenance costs, and provide better quality software.

**Q8. What are the limitations of OOP?**

**Ans.** The limitations of OOP are:

1. Object-oriented programming has a steep learning curve and it can take some time to gain a good understanding of it. It may be a little hard to understand the OOP code initially without the corresponding class documentation.
2. The size of the OOP programs may be larger than the programs created using procedure-oriented programming.
3. Object-oriented programs are usually slower as they need more instructions to be executed.
4. OOP programs can use a large amount of memory.
5. Intensive testing processes are required.

**Also Read:**[**Top Python Interview Questions and Answers**](https://www.shiksha.com/online-courses/articles/python-interview-questions-answers/)

Q9. What is a constructor? Name the different types of constructors.

**Ans.** A[**constructor**](https://www.shiksha.com/online-courses/articles/constructors-in-cpp/) is a special type of method that is called to create an object. The name of the constructors is the same name as the class.

Different types of constructors in object-oriented programming are:

* Default constructor
* Static constructor
* Copy constructor
* Private constructor
* Parameterized constructor

Q10. Name some operators that cannot be overloaded.

**Ans.** Some of the [**operators**](https://www.shiksha.com/online-courses/articles/operators-in-cpp/) that cannot be overloaded are:

Dot or Member access operator – ( . )

Scope resolution operator – ( :: )

Ternary operator – ( ? : )

Pointer to member operator – ( .\* )

Object size operator – (sizeof)

Object type operator – (typeid)

**Must read:**[**New and delete operators in C++**](https://www.shiksha.com/online-courses/articles/new-and-delete-operators-in-cpp/)

Q11. What are the differences between procedural programming and object-oriented programming?

**Ans.** This is one of the most important **OOPs basic interview questions**.

The differences between procedural programming and object-oriented programming are:

|  |  |
| --- | --- |
| **Procedural-Oriented Programming** | **Object-Oriented Programming** |
| Procedural Oriented Programming follows a step-by-step approach for dividing a task into a set of variables and routines through instructions. | OOP uses classes and objects to create models instead of functions and logic. |
| A program is broken down into small parts called functions. | A program is broken down into small parts called objects. |
| It follows the top-down approach. | It follows a bottom-up approach. |
| No modifiers are introduced in Procedural Programming. | OOP has access modifiers such as private, public, and protected. |
| It is less secure as it does not have any proper way of hiding data. | OOP is more secure as it allows data hiding. |
| Overloading is not possible. | Overloading is possible. |
| Does not have the Inheritance feature. | It has the Inheritance feature. |
| Not suitable for complex problems. | Suitable for complex problems. |
| Examples: C, Pascal, FORTRAN. | Examples: Java, C++, Python, C#, Ruby. |

**Also Read:**[**Top Paying IT Certifications**](https://www.shiksha.com/online-courses/articles/top-paying-certifications/)

Q12. What is Inheritance?

**Ans.** [Inheritance](https://www.shiksha.com/online-courses/articles/introduction-to-inheritance-in-cpp/) is a fundamental object-oriented programming concept in which a new class is created that inherits the properties of the already existing class. It supports “code reusability” and reduces the length of the code. The code written in one class can be extended into another class. Inheritance has a base class, which is inherited by the derived class.

Different types of Inheritance:

* Single inheritance
* Multi-level inheritance
* Hybrid inheritance
* Multiple inheritance
* Hierarchical inheritance

Q13. What is Polymorphism?

**Ans.** [**Polymorphism**](https://www.shiksha.com/online-courses/articles/polymorphism-in-cpp/)is a concept in the object-oriented programming language that means ‘one function, multiple forms’. It performs different things according to the object’s class, which calls it. Polymorphism is applied to the functions or methods. It enables the object to decide which form of the function to implement at compile-time and run-time.

Different types of Polymorphism:

* Compile-time polymorphism (Method Overloading)
* Run-time polymorphism (Method Overriding)

**Check out the top frequently asked Java OOPs interview questions – read our**[**blog Java Interview Questions**](https://www.shiksha.com/online-courses/articles/java-interview-questions-answers/)**.**

Q14. What is Data Abstraction? What are the levels of data abstraction?

**Ans.** Data abstraction is an important concept of object-oriented programming that displays only the important information and unnecessary details or implementation from the users. This concept helps in reducing programming complexity and efforts.

Data Abstraction has three levels:

* Physical Level (Lowest level of data abstraction): It tells how the data is stored in memory.
* Logical Level: It has the information that is stored in the database in the form of tables. The relationship among the data entities in relatively simple structures is also stored.
* View Level (Highest level of data abstraction): Only a part of the actual database is visible to the user.

**Must Check:** [Polymorphism in C++](https://www.shiksha.com/online-courses/articles/polymorphism-in-cpp/)

**Must check:**[**Difference between encapsulation and abstraction**](https://www.shiksha.com/online-courses/articles/difference-between-encapsulation-and-abstraction/)

**Explore popular**[**Databases Course**](https://www.shiksha.com/online-courses/databases-courses-certification-training-st605)

Q15. What is encapsulation?

**Ans.** In object-oriented programming, [**Encapsulation**](https://www.shiksha.com/online-courses/articles/encapsulation-in-cpp/)refers to wrapping up the data and the code that operates on that data in a single unit. It is a protective shield that allows data-hiding as the data specified in one class is hidden from other classes. Example: Class. A class consists of data and methods that are bundled together into a single unit. Many programming languages use Encapsulation in the form of classes.

Q16. What is hybrid inheritance?

**Ans.** Hybrid inheritance combines different kinds of inheritance, such as multiple inheritance, simple inheritance, and hierarchical inheritance. Hybrid inheritance is the combination of one or more types of inheritance. For example, a combination of simple inheritance and hierarchical inheritance. Therefore, hybrid inheritance is also called multipath inheritance.

Q17. How can we achieve data abstraction?

**Ans.** Data abstraction is achieved with abstract methods or abstract classes.

Q18. What is a copy constructor?

**Ans.** [**Copy Constructor**](https://www.shiksha.com/online-courses/articles/copy-constructor-in-cpp/)is a type of constructor. It initializes an object using another object within the same class. There are two types of copy constructors:

* Default Copy Constructor
* User-Defined Copy Constructor

Q19. What is a destructor?

**Ans.** A destructor is a member function that is automatically called when an object goes out of scope or is being destroyed. A destructor has the same name as the class with a tilde ~ sign as a prefix to it. Destructors never have any arguments.

**Destructor Example:** the destructor for class String is declared as ~String().

Q20. What are the access modifiers? Name the different types of access modifiers.

**Ans.** [**Access modifiers**](https://www.shiksha.com/online-courses/articles/understanding-cpp-modifiers/) in OOP are keywords that set the accessibility of classes, methods, and other members. They determine the scope of the method or variables that can be accessed from other objects or classes. The different types of access modifiers in OOPs are:

* Private
* Protected
* Public
* Friend

Q21. Is it possible to call the base class method without creating any instance?

**Ans.** Yes, it is possible to call the base class without instantiating it is a static method and if the base class is inherited by some subclass.

Q22. What is static polymorphism?

**Ans.** Static polymorphism is a type of polymorphism that occurs during compile time. Overloading is an example of static polymorphism.

Q23. Is it possible to run a Java application without the implementation of the OOPs concept?

**Ans.** No, it is not possible to run a Java application without the implementation of the OOPs concept since these applications are based on the OOPs model.

Q24. What is dynamic polymorphism?

**Ans.** Dynamic polymorphism implies runtime resolution of function calls. This is implied by overriding, followed by inheritance in c++. Dynamic polymorphism occurs at runtime, is based on object orientation, and allows separation of implementation of interfaces and class hierarchies. Late binding, dynamic dispatch, or getting dispatch at runtime requires two things: virtuality and indirection, such as pointers and references. Overriding is a type of dynamic polymorphism.

Q25. What is method overloading?

**Ans.** Method overloading is an OOPs feature of OOPs. Whenever the arguments that have been passed differ, it allows giving the same name to more than one method in a class.Method overloading is the process of overloading methods with the same name but different parameters. C++ provides a way to overload this function. Method overloading allows users to use the same name for different methods, but the parameters passed to the methods must be different. The return types of the methods can be the same or different.

**Also check:**[**Difference Between Overloading And Overriding**](https://www.shiksha.com/online-courses/articles/difference-between-overloading-and-overriding/)

Q26. Is it possible to run a C++ application without the implementation of the OOPs concept?

**Ans.** Yes, it is possible to implement C++ applications without OOPs

Q27. Explain the diamond problem in multiple inheritance?

The diamond problem is a complication that arises in object-oriented programming languages that support multiple inheritance. Let’s see an example:

1. **Setup**:
   * Suppose you have a base class A.
   * Classes B and C inherit from A.
   * Class D inherits from both B and C.
2. **Problem**:
   * If there is a method m() in class A and none in B and C, then there won’t be any ambiguity if this method is called by an object of class D since it would be inherited from A.
   * However, if the method m() is overridden in both B and C, then the object of class D will have ambiguity. It will be confused about which method to inherit: either from B or from C.

This is called the diamond problem because the class structure in this scenario looks like a diamond:

A

/ \

B C

\ /

D

Languages like Java and C# do not support multiple inheritance of classes primarily due to this ambiguity. However, C++ supports multiple inheritance and uses virtual inheritance to solve this problem.

Q28. What is a virtual function? Why is it used?

In [C++](https://www.shiksha.com/online-courses/what-is-c-plus-plus-st619-tg1441), a function is declared as virtual in the base class with the intention of being overridden or redefined in a derived class. When a base class pointer points to a derived class object and a virtual function is called through that pointer, the derived class’s version of the function is invoked. Without the virtual keyword, the base class’s function would be called, even if it’s overridden in the derived class.

**It is used because :**

* Achieve Runtime Polymorphism
* Ensure Consistency
* Provide a Base Implementation
* Work with Abstract Classes

**Let’s see an example :**

class Shape {

public:

virtual void draw() {

cout << "Drawing a shape" << endl;

}

};

class Circle : public Shape {

public:

void draw() override {

cout << "Drawing a circle" << endl;

}

};

int main() {

Shape\* shape = new Circle();

shape->draw(); *// Outputs: Drawing a circle*

delete shape;

return 0;

}

**Copy code**

**In the example above**, even though the pointer shape is of type Shape, it points to an object of type Circle. When the draw method is called on this pointer, the Circle class’s draw method is invoked, thanks to the virtual keyword in the base class.

***Without the virtual keyword, the base class’s draw method would have been called, which would not have given the expected behavior in the context of polymorphism.***

**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 behavior.

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 behavior 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 behavior 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 modeling 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 behavior (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 behavior 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 the 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 behaviors from its superclass, and can also add new properties and behaviors or modify existing ones.

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

Suggested Reading: [SQL Interview Questions 2023](https://www.wscubetech.com/blog/sql-interview-questions/)

**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 an 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 the 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 initializes 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 initializes 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.

**Interview Questions for You to Prepare for Jobs**

|  |  |
| --- | --- |
| [DBMS Interview Questions](https://www.wscubetech.com/blog/dbms-interview-questions/) | [Power BI Interview Questions](https://www.wscubetech.com/blog/power-bi-interview-questions/) |
| [Java Interview Questions](https://www.wscubetech.com/blog/java-interview-questions-answers/) | [JavaScript Interview Questions](https://www.wscubetech.com/blog/javascript-interview-questions/) |
| [CSS Interview Questions](https://www.wscubetech.com/blog/css-interview-questions-answers/) | [Flutter Interview Questions](https://www.wscubetech.com/blog/flutter-interview-questions/) |
| [HTML Interview Questions](https://www.wscubetech.com/blog/html-interview-questions-answers/) | [NodeJS Interview Questions](https://www.wscubetech.com/blog/nodejs-interview-questions/) |
| [MySQL Interview Questions](https://www.wscubetech.com/blog/mysql-interview-questions-answers/) | [ReactJS Interview Questions](https://www.wscubetech.com/blog/reactjs-interview-questions-answers/) |
| [Python Interview Questions](https://www.wscubetech.com/blog/python-interview-questions-and-answers/) | [C Programming Interview Questions](https://www.wscubetech.com/blog/c-programming-interview-questions-answers/) |
| [OOPS Interview Questions](https://www.wscubetech.com/blog/oops-interview-questions/) | [Data Structure Interview Questions](https://www.wscubetech.com/blog/data-structures-interview-questions/) |

**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 organize 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. |

**Suggested Reading:**[**Angular Interview Questions and Answers**](https://www.wscubetech.com/blog/angular-interview-questions/)

**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 initialized. It is also called a constant. A final variable can be initialized during declaration or in a constructor. Once initialized, 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 the number, type, and order of the arguments passed to the method.

**b) Dynamic polymorphism**

Also known as runtime polymorphism, 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 behavior to be added to an individual object, either statically or dynamically, without affecting the behavior 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 a 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 parameterized 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.

**Suggested Reading:**[**DBMS Interview Questions and Answers**](https://www.wscubetech.com/blog/dbms-interview-questions/)

**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 |