**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.

OOPS is abbreviated as Object Oriented Programming system in which programs are considered as a collection of objects. Each object is nothing but an instance of a class.

OOPs refers to Object-Oriented Programming. It is the programming paradigm that is defined using objects. Objects can be considered as real-world instances of entities like class, that have some characteristics and behaviors.

**3. What are some major Object Oriented Programming languages?**

The programming languages that use and follow the Object-Oriented Programming paradigm or OOPs, are known as Object-Oriented Programming languages. Some of the major Object-Oriented Programming languages include:

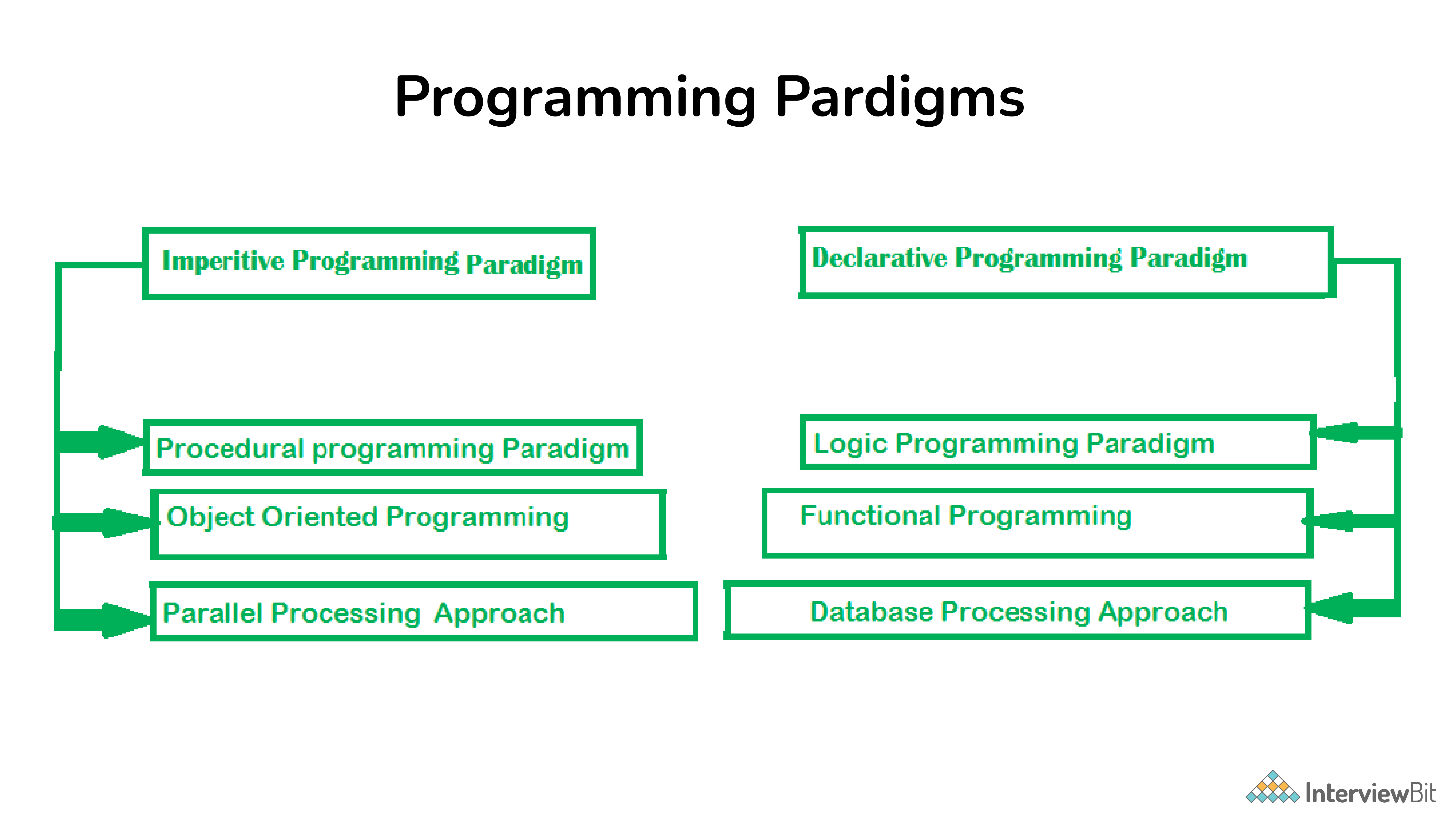
* Java
* C++
* Javascript
* Python
* PHP

**4. What are some other programming paradigms other than OOPs?**

Programming paradigms refers to the method of classification of programming languages based on their features. There are mainly two types of Programming Paradigms:

* Imperative Programming Paradigm
* Declarative Programming Paradigm

Now, these paradigms can be further classified based:  
**1. Imperative Programming Paradigm**: Imperative programming focuses on HOW to execute program logic and defines control flow as statements that change a program state. This can be further classified as:  
a) Procedural Programming Paradigm: Procedural programming specifies the steps a program must take to reach the desired state, usually read in order from top to bottom.  
b) Object-Oriented Programming or OOP: Object-oriented programming (OOP) organizes programs as objects, that contain some data and have some behavior.  
c) Parallel Programming: Parallel programming paradigm breaks a task into subtasks and focuses on executing them simultaneously at the same time.  
  
**2. Declarative Programming Paradigm**: Declarative programming focuses on WHAT to execute and defines program logic, but not a detailed control flow. Declarative paradigm can be further classified into:  
a) Logical Programming Paradigm: Logical programming paradigm is based on formal logic, which refers to a set of sentences expressing facts and rules about how to solve a problem  
b) Functional Programming Paradigm: Functional programming is a programming paradigm where programs are constructed by applying and composing functions.  
c) Database Programming Paradigm: Database programming model is used to manage data and information structured as fields, records, and files.



**5. What is meant by Structured Programming?**

**Structured Programming** refers to the method of programming which consists of a completely structured control flow. Here structure refers to a block, which contains a set of rules, and has a definitive control flow, such as (if/then/else), (while and for), block structures, and subroutines.

Nearly all programming paradigms include Structured programming, including the OOPs model.

**6. 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

There are many reasons why OOPs is mostly preferred, but the most important among them are:

* OOPs helps users to understand the software easily, although they don’t know the actual implementation.
* With OOPs, the readability, understandability, and maintainability of the code increase multifold.
* Even very big software can be easily written and managed easily using OOPs.

OOPs programming paradigm is considered as a better style of programming. Not only it helps in writing a complex piece of code easily, but it also allows users to handle and maintain them easily as well. Not only that, the main pillar of OOPs - Data Abstraction, Encapsulation, Inheritance, and Polymorphism, makes it easy for programmers to solve complex scenarios. As a result of these, OOPs is so popular.

**7. What are some advantages of using OOPs?**

* OOPs is very helpful in solving very complex level of problems.
* Highly complex programs can be created, handled, and maintained easily using object-oriented programming.
* OOPs, promote code reuse, thereby reducing redundancy.
* OOPs also helps to hide the unnecessary details with the help of Data Abstraction.
* OOPs, are based on a bottom-up approach, unlike the Structural programming paradigm, which uses a top-down approach.
* Polymorphism offers a lot of flexibility in OOPs.

**8. 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

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

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

OOPs or Object Oriented Programming mainly comprises of the below four features, and make sure you don't miss any of these:

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

**10. Write basic concepts of OOPS?**

Following are the concepts of OOPS:

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

**11. 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.

An object refers to the instance of the class, which contains the instance of the members and behaviors defined in the class template. In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object. So the objects consume space and have some characteristic behavior.  
For example, a specific car.

An object is an instance of a class. It has its own state, behavior, and identity.

**12. 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.

A class is simply a representation of a type of object. It is the blueprint/plan/template that describes the details of an object.

A class can be understood as a template or a blueprint, which contains some values, known as member data or member, and some set of rules, known as behaviors or functions. So when an object is created, it automatically takes the data and functions that are defined in the class.  
Therefore the class is basically a template or blueprint for objects. Also one can create as many objects as they want based on a class.

For example, first, a car’s template is created. Then multiple units of car are created based on that template.

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

Classes do not consume any memory. They are just a blueprint based on which objects are created. Now when objects are created, they actually initialize the class members and methods and therefore consume memory.

**14. 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.

No, class and structure are not the same. Though they appear to be similar, they have differences that make them apart. For example, the structure is saved in the stack memory, whereas the class is saved in the heap memory. Also, Data Abstraction cannot be achieved with the help of structure, but with class, Abstraction is majorly used.

The default access type of a Structure is public, but class access type is private. A structure is used for grouping data, whereas a class can be used for grouping data and methods. Structures are exclusively used for data, and it doesn’t require strict validation, but classes are used to encapsulate and inherent data, which requires strict validation.

**15. 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

Yes, it is possible to call the base method without creating an instance. And that method should be “Static method.”

Doing Inheritance from that class.-Use Base Keyword from a derived class.

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

No. An object is necessary to be created if the base class has non-static methods. But if the class has static methods, then objects don’t need to be created. You can call the class method directly in this case, using the class name.

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

|  |  |
| --- | --- |
| 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 |

An object is an instance of a class. Objects hold multiple information, but classes don’t have any information. Definition of properties and functions can be done in class and can be used by the object.

A class can have sub-classes, while an object doesn’t have sub-objects.

**18. 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.

Inheritance is one of the major features of object-oriented programming, by which an entity inherits some characteristics and behaviors of some other entity and makes them their own. Inheritance helps to improve and facilitate code reuse.

Let me explain to you with a common example. Let's take three different vehicles - a car, truck, or bus. These three are entirely different from one another with their own specific characteristics and behavior. But. in all three, you will find some common elements, like steering wheel, accelerator, clutch, brakes, etc. Though these elements are used in different vehicles, still they have their own features which are common among all vehicles. This is achieved with inheritance. The car, the truck, and the bus have all inherited the features like steering wheel, accelerator, clutch, brakes, etc, and used them as their own. Due to this, they did not have to create these components from scratch, thereby facilitating code reuse.

The term “inheritance” means “receiving some quality or behavior from a parent to an offspring.” In object-oriented programming, inheritance is the mechanism by which an object or class (referred to as a child) is created using the definition of another object or class (referred to as a parent). Inheritance not only helps to keep the implementation simpler but also helps to facilitate code reuse.

Inheritance is a concept where one class shares the structure and behavior defined in another class. If Inheritance applied to one class is called Single Inheritance, and if it depends on multiple classes, then it is called multiple Inheritance.

Inheritance OOPS concept is used as a reuse mechanism.

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

The various types of inheritance include:

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

**20. 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 |

**21. What is hybrid inheritance?**

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

**22. 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.

**23. 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

Yes, with more powers comes more complications. Inheritance is a very powerful feature in OOPs, but it has some limitations too. Inheritance needs more time to process, as it needs to navigate through multiple classes for its implementation. Also, the classes involved in Inheritance - the base class and the child class, are very tightly coupled together. So if one needs to make some changes, they might need to do nested changes in both classes. Inheritance might be complex for implementation, as well. So if not correctly implemented, this might lead to unexpected errors or incorrect outputs.

**24. What are manipulators?**

Manipulators are the functions which can be used in conjunction with the insertion (<<) and extraction (>>) operators on an object. Examples are endl and setw.

**25. 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.

Superclass is also a part of Inheritance. The superclass is an entity, which allows subclasses or child classes to inherit from itself.

**26. 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.

The subclass is a part of Inheritance. The subclass is an entity, which inherits from another class. It is also known as the child class.

**27. What are a base class, subclass, and superclass?**

The base class is the most generalized class, and it is said to be a root class.

A Subclass is a class that inherits from one or more base classes.

The superclass is the parent class from which another class inherits.

**28. 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.

Polymorphism is composed of two words - “poly” which means “many”, and “morph” which means “shapes”. Therefore Polymorphism refers to something that has many shapes.

In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms in OOPs languages.

Polymorphism is nothing but assigning behavior or value in a subclass to something that was already declared in the main class. Simply, polymorphism takes more than one form.

**29. What is dynamic or run time polymorphism?**

Dynamic or Run time polymorphism is also known as method overriding in which call to an overridden function is resolved during run time, not at the compile time. It means having two or more methods with the same name, same signature but with different implementation.

**30. What is Compile time Polymorphism and how is it different from Runtime Polymorphism?**

Compile Time Polymorphism: Compile time polymorphism, also known as Static Polymorphism, refers to the type of Polymorphism that happens at compile time. What it means is that the compiler decides what shape or value has to be taken by the entity in the picture.

Example:

// In this program, we will see how multiple functions are created with the same name,

// but the compiler decides which function to call easily at the compile time itself.

**class** **CompileTimePolymorphism**{

// 1st method with name add

**public** **int** **add**(**int** x, **int** y){

**return** x+y;

}

// 2nd method with name add

**public** **int** **add**(**int** x, **int** y, **int** z){

**return** x+y+z;

}

// 3rd method with name add

**public** **int** **add**(**double** x, **int** y){

**return** (**int**)x+y;

}

// 4th method with name add

**public** **int** **add**(**int** x, **double** y){

**return** x+(**int**)y;

}

}

**class** **Test**{

**public** **static** **void** **main**(String[] args){

CompileTimePolymorphism demo=**new** CompileTimePolymorphism();

// In the below statement, the Compiler looks at the argument types and decides to call method 1

System.out.println(demo.add(2,3));

// Similarly, in the below statement, the compiler calls method 2

System.out.println(demo.add(2,3,4));

// Similarly, in the below statement, the compiler calls method 4

System.out.println(demo.add(2,3.4));

// Similarly, in the below statement, the compiler calls method 3

System.out.println(demo.add(2.5,3));

}

}

In the above example, there are four versions of add methods. The first method takes two parameters while the second one takes three. For the third and fourth methods, there is a change of order of parameters. The compiler looks at the method signature and decides which method to invoke for a particular method call at compile time.  
**Runtime Polymorphism**: Runtime polymorphism, also known as Dynamic Polymorphism, refers to the type of Polymorphism that happens at the run time. What it means is it can't be decided by the compiler. Therefore what shape or value has to be taken depends upon the execution. Hence the name Runtime Polymorphism.

Example:

**class** **AnyVehicle**{

**public** **void** **move**(){

System.out.println(“Any vehicle should move!!”);

}

}

**class** **Bike** **extends** **AnyVehicle**{

**public** **void** **move**(){

System.out.println(“Bike can move too!!”);

}

}

**class** **Test**{

**public** **static** **void** **main**(String[] args){

AnyVehicle vehicle = **new** Bike();

// In the above statement, as you can see, the object vehicle is of type AnyVehicle

// But the output of the below statement will be “Bike can move too!!”,

// because the actual implementation of object ‘vehicle’ is decided during runtime vehicle.move();

vehicle = **new** AnyVehicle();

// Now, the output of the below statement will be “Any vehicle should move!!”,

vehicle.move();

}

}

As the method to call is determined at runtime, as shown in the above code, this is called runtime polymorphism.

**31. 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.

Static Polymorphism is commonly known as the Compile time polymorphism. Static polymorphism is the feature by which an object is linked with the respective function or operator based on the values during

**32. 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.

Dynamic Polymorphism or Runtime polymorphism refers to the type of Polymorphism in OOPs, by which the actual implementation of the function is decided during the runtime or execution. The dynamic or runtime polymorphism can be achieved with the help of method overriding.

**33. 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.

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.

Method overriding is a feature that allows a subclass to provide the implementation of a method that overrides in the main class. It will override the implementation in the superclass by providing the same method name, same parameter, and same return type.

**34. What is function overloading?**

Function overloading is a regular function, but it is assigned with multiple parameters. It allows the creation of several methods with the same name which differ from each other by the type of input and output of the function.

Example

void add(int& a, int& b);

void add(double& a, double& b);

void add(struct bob& a, struct bob& b);

**35. What is operator overloading?**

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

Operator overloading is a function where different operators are applied and depends on the arguments. Operator,-,\* can be used to pass through the function, and it has its own precedence to execute

**36. What are all the operators that cannot be overloaded?**

Following are the operators that cannot be overloaded -.

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

**37. What is a ternary operator?**

The ternary operator is said to be an operator which takes three arguments. Arguments and results are of different data types, and it depends on the function. The ternary operator is also called a conditional operator.

**38. 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 |

Overloading is a compile-time polymorphism feature in which an entity has multiple implementations with the same name. For example, Method overloading and Operator overloading.

Whereas Overriding is a runtime polymorphism feature in which an entity has the same name, but its implementation changes during execution. For example, Method overriding.

Overloading is static Binding, whereas Overriding is dynamic Binding. Overloading is nothing but the same method with different arguments, and it may or may not return the equal value in the same class itself.

Overriding is the same method names with the same arguments and return types associated with the class and its child class.

**39. Which keyword can be used for overloading?**

Operator keyword is used for overloading.

**40. What is the difference between new and override?**

The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

**41. 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.

One can visualize Encapsulation as the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. What it means is that by Encapsulation, all the necessary data and methods are bind together and all the unnecessary details are hidden to the normal user. So Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.  
  
Encapsulation can also be defined in two different ways:  
  
1) **Data hiding:** Encapsulation is the process of hiding unwanted information, such as restricting access to any member of an object.  
  
2) **Data binding:** Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

Encapsulation is an attribute of an object, and it contains all data which is hidden. That hidden data can be restricted to the members of that class.

Levels are Public, Protected, Private, Internal, and Protected Internal.

Encapsulation is the OOPS concept exposes only the necessary information to the calling functions.

**42. 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.

Access specifiers, as the name suggests, are a special type of keywords, which are used to control or specify the accessibility of entities like classes, methods, etc. Some of the access specifiers or access modifiers include “private”, “public”, etc. These access specifiers also play a very vital role in achieving Encapsulation - one of the major features of OOPs.

Private access specifier is used in a class definition.

**43. 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 |

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

Access modifiers determine the scope of the method or variables that can be accessed from other various objects or classes. There are five types of access modifiers, and they are as follows:

* Private
* Protected
* Public
* Friend
* Protected Friend

The default access modifier of a class is Internal and the default access modifier of a class member is Private.

**45. What are sealed modifiers?**

Sealed modifiers are the access modifiers where the methods can not inherit it. Sealed modifiers can also be applied to properties, events, and methods. This modifier cannot be used to static members.

**46. What is Abstraction?**

If you are a user, and you have a problem statement, you don't want to know how the components of the software work, or how it's made. You only want to know how the software solves your problem. Abstraction is the method of hiding unnecessary details from the necessary ones. It is one of the main features of OOPs.   
For example, consider a car. You only need to know how to run a car, and not how the wires are connected inside it. This is obtained using Abstraction.

Abstraction is a useful feature of OOPS, and it shows only the necessary details to the client of an object. Meaning, it shows only required details for an object, not the inner constructors, of an object. Example – When you want to switch on the television, it is not necessary to know the inner circuitry/mechanism needed to switch on the TV. Whatever is required to switch on TV will be shown by using an abstract class.

**47. 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.

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

Data abstraction can be achieved through:

* Abstract class
* Abstract method

**49. How is data abstraction accomplished?**

Data abstraction is accomplished with the help of abstract methods or abstract classes.

**50. 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 |

**51. 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.

An abstract class is a special class containing abstract methods. The significance of abstract class is that the abstract methods inside it are not implemented and only declared. So as a result, when a subclass inherits the abstract class and needs to use its abstract methods, they need to define and implement them.

An abstract class is a class which cannot be instantiated. Creation of an object is not possible with an abstract class, but it can be inherited. An abstract class can contain only an Abstract method. Java allows only abstract method in abstract class while other languages allow non-abstract method as well.

**52.** **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.

Zero instances will be created for an abstract class. In other words, you cannot create an instance of an Abstract Class.

**53. 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/).

An interface refers to a special type of class, which contains methods, but not their definition. Only the declaration of methods is allowed inside an interface. To use an interface, you cannot create objects. Instead, you need to implement that interface and define the methods for their implementation. 

An interface is a collection of an abstract method. If the class implements an interface, it thereby inherits all the abstract methods of an interface.

Java uses Interface to implement multiple inheritances.

**54. 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.

A virtual function is a member function of a class, and its functionality can be overridden in its derived class. This function can be implemented by using a keyword called virtual, and it can be given during function declaration.

A virtual function can be declared using a token(virtual) in C++. It can be achieved in C/Python Language by using function pointers or pointers to function.

**55. What does the keyword virtual represented in the method definition?**

It means we can override the method.

**56. 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.

A pure virtual function is a function which can be overridden in the derived class but cannot be defined. A virtual function can be declared as Pure by using the operator =0.

Example –

Virtual void function1() // Virtual, Not pure

Virtual void function2() = 0 //Pure virtual

**57. What is an Inline function?**

An inline function is a technique used by the compilers and instructs to insert complete body of the function wherever that function is used in the program source code.

**58. What is a friend function?**

A friend function is a friend of a class that is allowed to access to Public, private, or protected data in that same class. If the function is defined outside the class cannot access such information.

A friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public, or protected.

**59. 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.

Constructors are special methods whose name is the same as the class name. The constructors serve the special purpose of initializing the objects.  
For example, suppose there is a class with the name “MyClass”, then when you instantiate this class, you pass the syntax:  
MyClass myClassObject = new MyClass();

Now here, the method called after “new” keyword - MyClass(), is the constructor of this class. This will help to instantiate the member data and methods and assign them to the object myClassObject.

A constructor is a method used to initialize the state of an object, and it gets invoked at the time of object creation. Rules for constructor are:

* Constructor Name should be the same as a class name.
* A constructor must have no return type.

No, we do not require a parameter for constructors.

**60. 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.

Contrary to constructors, which initialize objects and specify space for them, Destructors are also special methods. But destructors free up the resources and memory occupied by an object. Destructors are automatically called when an object is being destroyed.

A destructor is a method which is automatically called when the object is made of scope or destroyed. Destructor name is also same as class name but with the tilde symbol before the name.

**61. 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

There are three types of constructors:

–  Default Constructor – With no parameters.

–  Parametric Constructor – With Parameters. Create a new instance of a class and also passing arguments simultaneously.

–  Copy Constructor – Which creates a new object as a copy of an existing object.

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

The most common classification of constructors includes:

Default constructor: The default constructor is the constructor which doesn’t take any argument. It has no parameters.

**class** **ABC**

{

**int** x;

ABC()

{

x = 0;

}

}

Parameterized constructor: The constructors that take some arguments are known as parameterized constructors.

**class** **ABC**

{

**int** x;

ABC(**int** y)

{

x = y;

}

}

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

**class** **ABC**

{

**int** x;

ABC(**int** y)

{

x = y;

}

// Copy constructor

ABC(ABC abc)

{

x = abc.x;

}

}

**62. 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.

Copy Constructor is a type of constructor, whose purpose is to copy an object to another. What it means is that a copy constructor will clone an object and its values, into another object, is provided that both the objects are of the same class.

This is a special constructor for creating a new object as a copy of an existing object. There will always be only one copy constructor that can be either defined by the user or the system.

**63. 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.

**64. What is the use of finalize method?**

Finalize method helps to perform cleanup operations on the resources which are not currently used. Finalize method is protected, and it is accessible only through this class or by a derived class.

**65. 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.

Object-oriented programming revolves around entities like objects. Each object consumes memory and there can be multiple objects of a class. So if these objects and their memories are not handled properly, then it might lead to certain memory-related errors and the system might fail.

Garbage collection refers to this mechanism of handling the memory in the program. Through garbage collection, the unwanted memory is freed up by removing the objects that are no longer needed.

**66. 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 |

**67. 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 |

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

Interface and abstract class both are special types of classes that contain only the methods declaration and not their implementation. But the interface is entirely different from an abstract class. The main difference between the two is that, when an interface is implemented, the subclass must define all its methods and provide its implementation. Whereas when an abstract class is inherited, the subclass does not need to provide the definition of its abstract method, until and unless the subclass is using it.

Also, an abstract class can contain abstract methods as well as non-abstract methods.

**69. 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.

**70. 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.

An exception can be considered as a special event, which is raised during the execution of a program at runtime, that brings the execution to a halt. The reason for the exception is mainly due to a position in the program, where the user wants to do something for which the program is not specified, like undesirable input.

**71. 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.

No one wants its software to fail or crash. Exceptions are the major reason for software failure. The exceptions can be handled in the program beforehand and prevent the execution from stopping. This is known as exception handling.  
So exception handling is the mechanism for identifying the undesirable states that the program can reach and specifying the desirable outcomes of such states.  
Try-catch is the most common method used for handling exceptions in the program.

An exception is an event that occurs during the execution of a program. Exceptions can be of any type – Runtime exception, Error exceptions. Those exceptions are adequately handled through exception handling mechanism like try, catch, and throw keywords.

**72. 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 |

**73. 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.

**74. 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.

**75. What are the different types of arguments?**

A parameter is a variable used during the declaration of the function or subroutine, and arguments are passed to the function body, and it should match with the parameter defined. There are two types of Arguments.

* Call by Value – Value passed will get modified only inside the function, and it returns the same value whatever it is passed into the function.
* Call by Reference – Value passed will get modified in both inside and outside the functions and it returns the same or different value.

**76. What is the super keyword?**

The super keyword is used to invoke the overridden method, which overrides one of its superclass methods. This keyword allows to access overridden methods and also to access hidden members of the superclass.

It also forwards a call from a constructor, to a constructor in the superclass.

**77. What are tokens?**

A compiler recognizes a token, and it cannot be broken down into component elements. Keywords, identifiers, constants, string literals, and operators are examples of tokens.

Even punctuation characters are also considered as tokens. Example: Brackets, Commas, Braces, and Parentheses.

**78. What is early and late Binding?**

Early binding refers to the assignment of values to variables during design time, whereas late Binding refers to the assignment of values to variables during run time.

**79. What is ‘this’ pointer?**

THIS pointer refers to the current object of a class. THIS keyword is used as a pointer which differentiates between the current object with the global object. It refers to the current object.

**80. Whether static method can use nonstatic members?**

False.

**81. What is static and dynamic Binding?**

Binding is nothing but the association of a name with the class. Static Binding is a binding in which name can be associated with the class during compilation time, and it is also called as early Binding.

Dynamic Binding is a binding in which name can be associated with the class during execution time, and it is also called as Late Binding.