# Object Oriented Programming (Interview Questions & Answers)

Collected from different websites. Use for non-commercial purpose.

Sohail Basheer
Lecturer- Computer Science (visiting)
Department of Computer Science & IT (Evening Program)
Federal Urdu University Arts, Science & Technology
Gulshan Campus Karachi

Object-oriented programming (OOP) is a computer science term used to characterize a programming language that began development in the 1960's. The term 'object-oriented programming' was originally coined by Xerox PARC to designate a computer application that describes the methodology of using objects as the foundation for computation. By the 1980's, OOP rose to prominence as the programming language of choice, exemplified by the success of C++. Currently, OOPs such as Java, J2EE, C++, C#, Visual Basic.NET, Python and JavaScript are popular OOP programming languages that any career-oriented Software Engineer or developer should be familiar with.

OOP is widely accepted as being far more flexible than other computer programming languages. OOPs use three basic concepts as the fundamentals for the programming language: classes, objects and methods. Additionally, Inheritance, Abstraction, Polymorphism, Event Handling and Encapsulation.

## Common Questions & Answers

**1) What is meant by Object Oriented Programming?**

OOP is a method of programming in which programs are organized as cooperative collections of objects. Each object is an instance of a class and each class belong to a hierarchy.

**2) What is a Class?**

Class is a template for a set of objects that share a common structure and a common behavior.

**3) What is an Object?**

Object is an instance of a class. It has state, behavior and identity. It is also called as an instance of a class.

**4) What is an Instance?**

An instance has state, behavior and identity. The structure and behavior of similar classes are defined in their common class. An instance is also called as an object.

**5) What are the core OOP's concepts?**

Abstraction, Encapsulation, Inheritance and Polymorphism are the core OOP's concepts.

**6) What is meant by abstraction?**

Abstraction defines the essential characteristics of an object that distinguish it from all other kinds of objects. Abstraction provides crisply-defined conceptual boundaries relative to the perspective of the viewer. Its the process of focusing on the essential characteristics of an object. Abstraction is one of the fundamental elements of the object model.

**7) What is meant by Encapsulation?**

Encapsulation is the process of compartmentalizing the elements of an abstraction that defines the structure and behavior. Encapsulation helps to separate the contractual interface of an abstraction and implementation.

**8) What is meant by Inheritance?**

Inheritance is a relationship among classes, wherein one class shares the structure or behavior defined in another class. This is called Single Inheritance. If a class shares the structure or behavior from multiple classes, then it is called Multiple Inheritance. Inheritance defines "is-a" hierarchy among classes in which one subclass inherits from one or more generalized superclasses.

**9) What is meant by Polymorphism?**

Polymorphism literally means taking more than one form. Polymorphism is a characteristic of being able to assign a different behavior or value in a subclass, to something that was declared in a parent class.

**10) What is an Abstract Class?**

Abstract class is a class that has no instances. An abstract class is written with the expectation that its concrete subclasses will add to its structure and behavior, typically by implementing its abstract operations.

**11) What is an Interface?**

Interface is an outside view of a class or object which emphasizes its abstraction while hiding its structure and secrets of its behavior.

**12) What is a base class?**

Base class is the most generalized class in a class structure. Most applications have such root classes. In Java, Object is the base class for all classes.

**13) What is a subclass?**

Subclass is a class that inherits from one or more classes

**14) What is a superclass?**

superclass is a class from which another class inherits.

**15) What is a constructor?**

Constructor is an operation that creates an object and/or initializes its state.

**16) What is a destructor?**

Destructor is an operation that frees the state of an object and/or destroys the object itself. In Java, there is no concept of destructors. Its taken care by the JVM.

**17) What is meant by Binding?**

Binding denotes association of a name with a class.

**18) What is meant by static binding?**

Static binding is a binding in which the class association is made during compile time. This is also called as Early binding.

**19) What is meant by Dynamic binding?**

Dynamic binding is a binding in which the class association is not made until the object is created at execution time. It is also called as Late binding.

**20) Define Modularity?**

Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

**21) What is meant by Persistence?**

Persistence is the property of an object by which its existence transcends space and time.

**22) What is collaboration?**

Collaboration is a process whereby several objects cooperate to provide some higher level behavior.

### 23) In Java, How to make an object completely encapsulated?

All the instance variables should be declared as private and public getter and setter methods should be provided for accessing the instance variables.

### 24) How is polymorphism achieved in java?

Inheritance, Overloading and Overriding are used to achieve Polymorphism in java

### What is a class?

Class is concrete representation of an entity. It represents a group of objects, which hold similar attributes and behavior. It provides Abstraction and Encapsulations.

### What is an Object? What is Object Oriented Programming?

Object represents/resembles a Physical/real entity. An object is simply something you can give a name. Object Oriented Programming is a Style of programming that represents a program as a system of objects and enables code-reuse.

### What is Encapsulation?

Encapsulation is binding of attributes and behaviors. Hiding the actual implementation and exposing the functionality of any object. Encapsulation is the first step towards OOPS, is the procedure of covering up of data and functions into a single unit (called class). Its main aim is to protect the data from out side world

### What is Abstraction?

Hiding the complexity. It is a process of defining communication interface for the functionality and hiding rest of the things.

### What is Overloading?

Adding a new method with the same name in same/derived class but with different number/types of parameters. It implements Polymorphism.

### What is Overriding

A process of creating different implementation of a method having a same name as base class, in a derived class. It implements Inheritance.

### What is Shadowing?

When the method is defined as Final/sealed in base class and not override able and we need to provide different implementation for the same. This process is known as shadowing, uses shadows/new keyword.

### What is Inheritance?

It is a process of acquiring attributes and behaviors from another object (normally a class or interface).

### What is an Abstract class?

An abstract class is a special kind of class that cannot be instantiated. It normally contains one or more abstract methods or abstract properties. It provides body to a class.

## What is an Interface?

An interface has no implementation; it only has the signature or in other words, just the definition of the methods without the body.

## What is Polymorphism?

Mean by more than one form. Ability to provide different implementation based on different number/type of parameters.

## What is Pure-Polymorphism?

When a method is declared as abstract/virtual method in a base class and which is overridden in a base class. If we create a variable of a type of a base class and assign an object of a derived class to it, it will be decided at a run time, which implementation of a method is to be called. This is known as Pure-Polymorphism or Late-Binding.

## What is a Constructor?

A special function Always called whenever an instance of the class is created.

- Same name as class name
- No return type
- Automatically call when object of class is created
- Used to initialize the members of class

```
class Test
{
 int a,b;
 Test()
  {
   a=9;
   b=8;
   }
};
```

Here Test() is the constructor of Class Test.

## What is copy constructor?

Constructor which initializes  it's object member variables ( by shallow copying) with another object of the same class. If you don't implement one in your class then compiler implements one for you.

for example:

Test t1(10); // calling Test constructor

Test t2(t1); // calling Test copy constructor

Test t2 = t1;// calling Test copy constructor

Copy constructors are called in following cases:

- when a function returns an object of that class by value
- when the object of that class is passed by value as an argument to a function
- when you construct an object based on another object of the same class
- When compiler generates a temporary object

## What is default Constructor?

Constructor with no arguments or all the arguments has default values. In Above Question Test() is a default constructor

## What is a Destructor?

A special method called by GC. just before object is being reclaimed by GC.

## How a base class method is hidden?

Hiding a base class method by declaring a method in derived class with keyword new. This will override the base class method and old method will be suppressed.

## What Command is used to implement properties in C#?

get & set access modifiers are used to implement properties in c#.

## What is method overloading?

Method overloading is having methods with same name but carrying different signature, this is useful when you want a method to behave differently depending upon a data passed to it.

## Can constructors have parameters?

Yes, constructors can have parameters. so we can overload it.

## What are Static Assembly and Dynamic Assembly?

Static assemblies can include .NET Framework types (interfaces and classes) as well as resources for the assembly (bitmaps, JPEG files, resource files, and so forth). Static assemblies are stored on disk. Dynamic assemblies run directly from memory and are not saved to disk before execution.

## Describe the functionality of an assembly.

It is the smallest unit that has version control. All types and resources in the same assembly are versioned as a unit and support side by side execution. Assemblies contain the metadata and other identities which allow the common language runtime to execute. They are the boundaries providing the type check. They the unit where security permissions are requested and granted.

## What is serialization?

Serialization is the process of converting an object into a stream of bytes. De-serialization is the opposite process of creating an object from a stream of bytes. Serialization/De-serialization is mostly used to transport objects (e.g. during remoting), or to persist objects (e.g. to a file or database). There are two separate mechanisms provided by the .NET class library for serialization - XmlSerializer and

SoapFormatter and BinaryFormatter. Microsoft uses XmlSerializer for Web Services, and uses SoapFormatter/BinaryFormatter for remoting.

## What are C++ storage classes?

auto:the default. Variables are automatically created and initialized when they are defined and are destroyed at the end of the block containing their definition(when they are out of scope). They are not visible outside that block

```
auto int x;
 int y; Both are same statement. auto is defalt
```

register:a type of auto variable. a suggestion to the compiler to use a CPU register for performance(Generally Used in loops)

static:a variable that is known only within the function that contains its definition but is never destroyed and retains its value between calls to that function. It exists from the time the program begins execution

```
void example()
{
   static int x = 0; // static variable
   x++;
   cout << x <<endl;
}
```

If this function is called 10 times, the output will be 1,2,3,4..etc., The value of the variable x is preserved through function calls. If this static variable is declared as a member of a class, then it will preserve the value for all the objects of the class.i.e, one copy of this data variable will be shared by all objects of the class

Note: if we will declare variable like this int x=0; in the above example. then it will print 1 every it will be called extern:a static variable whose definition and placement is determined when all object and librar modules are combined (linked) to form the executable code file. It can be visible outside the file where it is defined.

## What is RTTI?

Runtime type identification (RTTI) lets you find the dynamic type of an object when you have only a pointer or a reference to the base type. RTTI is the official way in standard C++ to discover the type of an object and to convert the type of a pointer or reference (that is, dynamic typing). The need came from practical experience with C++. RTTI replaces many homegrown versions with a solid, consistent approach.

## What is friend function?

As the name suggests, the function acts as a friend to a class. As a friend of a class, it can access its private and protected members. A friend function is not a member of the class. But it must be listed in the class definition.

## What is a scope resolution operator?

A scope resolution operator (::), can be used to define the member functions of a class outside the class.

## What do you mean by pure virtual functions?

A pure virtual member function is a member function that the base class forces derived classes to provide. Normally these member functions have no implementation. Pure virtual functions are equated to zero. class Shape

```
{
public: virtual void draw() = 0;
};
```

## What is the difference between declaration and definition?

The declaration tells the compiler that at some later point we plan to present the definition of this declaration.

E.g.: void stars () //function declaration

The definition contains the actual implementation.

E.g.:

```
void stars ()
{
for(int j=10; j > =0; j--) //function body
cout << *; cout << endl;
}
```

## What are the advantages of inheritance?

It permits code reusability. Reusability saves time in program development. It encourages the reuse of proven and debugged high-quality software, thus reducing problem after a system becomes functional.

## Association

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

## Aggregation

Aggregation is a specialize form of Association where all object have their own lifecycle but there is ownership and child object can not belongs to another parent object. Let's take an example of Department and teacher. A single teacher can not belongs to multiple departments, but if we delete the department teacher object will not destroy. We can think about "has-a" relationship.

## Composition

Composition is again specialize form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted. Let’s take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different house if we delete the house room will automatically delete. Let’s take another example relationship between Questions and options. Single questions can have multiple options and option can not belong to multiple questions. If we delete questions options will automatically delete.

How to Implement (Code Example)

```
Class Circle
{
Point point;
};
Class Point
{
int x;
int y;   };
```

Here Circle is composed of Point.... you can't make a circle without point (Strong dependency)

## Distinguish between the terms fatal error and non–fatal error. Why might you prefer to experience a fatal error rather than a non–fatal error?

A fatal error causes a program to terminate prematurely. A nonfatal error occurs when the logic of the program is incorrect, and the program does not work properly. A fatal error is preferred for debugging purposes. A fatal error immediately lets you know there is a problem with the program, whereas a nonfatal error can be subtle and possibly go undetected.

## What are virtual functions? Describe a circumstance in which virtual functions would be appropriate

Virtual functions are functions with the same function prototype that are defined throughout a class hierarchy. At least the base class occurrence of the function is preceded by the keyword virtual. Virtual functions are used to enable generic processing of an entire class hierarchy of objects through a base class pointer. For example, in a shape hierarchy, all shapes can be drawn. If all shapes are derived from a base class Shape which contains a virtual draw function, then generic processing of the hierarchy can be performed by calling every shape’s draw generically through a base class Shape pointer.

## Given that constructors cannot be virtual, describe a scheme for how you might achieve a similar effect

Create a virtual function called initialize that the constructor invokes.

**How is it that polymorphism enables you to program "in the general" rather than "in the specific." Discuss the key advantages of programming "in the general."**

Polymorphism enables the programmer to concentrate on the processing of common operations that are applied to all data types in the system without going into the individual details of each data type. The general processing capabilities are separated from the internal details of each type.

**Discuss the problems of programming with switch logic. Explain why polymorphism is an effective alternative to using switch logic.**

The main problem with programming using the switch structure is extensibility and maintainability of the program. A program containing many switch structures is difficult to modify. Many, but not necessarily all, switch structures will need to add or remove cases for a specified type. Note: switch logic includes if/else structures which are more flexible than the switch structure.

**Distinguish between static binding and dynamic binding. Explain the use of virtual functions and the vtable in dynamic binding.**

Static binding is performed at compile-time when a function is called via a specific object or via a pointer to an object. Dynamic binding is performed at run-time when a virtual function is called via a base class pointer to a derived class object (the object can be of any derived class). The virtual functions table (vtable) is used at run-time to enable the proper function to be called for the object to which the base class pointer "points". Each class containing virtual functions has its own vtable that specifies where the virtual functions for that class are located. Every object of a class with virtual functions contains a hidden pointer to the class's vtable. When a virtual function is called via a base class pointer, the hidden pointer is dereferenced to locate the vtable, then the vtable is searched for the proper function call.

**Distinguish between inheriting interface and inheriting implementation. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?**

When a class inherits implementation, it inherits previously defined functionality from another class. When a class inherits interface, it inherits the definition of what the interface to the new class type should be. The implementation is then provided by the programmer defining the new class type. Inheritance hierarchies designed for inheriting implementation are used to reduce the amount of new code that is being written. Such hierarchies are used to facilitate software reusability. Inheritance hierarchies designed for inheriting interface are used to write programs that perform generic processing of many class types. Such hierarchies are commonly used to facilitate software extensibility (i.e., new types can be added to the hierarchy without changing the generic processing capabilities of the program.)

**Distinguish between virtual functions and pure virtual functions**

A virtual function must have a definition in the class in which it is declared. A pure virtual function does not provide a definition. Classes derived directly from the abstract class must provide definitions for the inherited pure virtual functions in order to avoid becoming an abstract base class