Aryabhatta Knowledge University (AKU)

Electronics and Communications Engineering

Object Oriented Programming

Solved Exam Paper 2019

Question. What is object oriented programming (OOP)? Write the basic concept of OOP.

Answer: Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained.

The organization of an object-oriented program also makes the method beneficial to collaborative development, where projects are divided into groups.

Additional benefits of OOP include code reusability, scalability and efficiency. Even when using micro services, developers should continue to apply the principles of OOP.

The first step in OOP is to collect all of the objects a programmer wants to manipulate and identify how they relate to each other -- an

exercise often known as data modeling.

Examples of an object can range from physical entities, such as a human being who is described by properties like name and address, down to small computer programs, such as widgets.

Once an object is known, it is labeled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects can communicate with well-defined interfaces called messages.

Principles of OOP

Object-oriented programming is based on the following principles:

Encapsulation. The implementation and state of each object are privately held inside a defined boundary, or class. Other objects do not have access to this class or the authority to make changes but are only able to call a list of public functions, or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.

Abstraction. Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. This concept helps developers more easily make changes and additions over time.

Inheritance. Relationships and subclasses between objects can be assigned, allowing developers to reuse a common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.

Polymorphism. Objects can take on more than one form depending on the context. The program will determine which meaning or usage is necessary for each execution of that object, cutting down the need

to duplicate code.

- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Allows for class-specific behavior through polymorphism
- Easier to debug, classes often contain all applicable information to them
- Secure, protects information through encapsulation

Question. What do you mean by class and object?

Answer: Classes are essentially user defined data types. Classes are where we create a blueprint for the structure of methods and attributes. Individual objects are instantiated, or created from this blueprint.

Classes contain fields for attributes, and methods for behaviors. In our Dog class example, attributes include name & birthday, while methods include bark() and update Attendance().

Remember the class is a template for modeling a dog, and an object is instantiated from the class representing an individual real world thing.

Objects

Of course OOP includes objects! Objects are instances of classes created with specific data, for example in the code snippet below Rufus is an instance of the Dog class.

When the new class Dog is called:

A new object is created named rufus

The constructor runs name & birthday arguments, and assigns values

N/A	What is it?	Information Contained	Actions	Exam
Classes	Blueprint	Attributes	Behaviors defined through methods	Dog Temp
Objects	Instance	State, Data	Methods	Rufus Fluff

Question. With an example, explain the terms constructor and destructor

Constructor:

Answer: A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not. It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created. It can be defined manually with arguments or without arguments. There can be many constructors in class. It can be overloaded but it cannot be inherited or virtual. There is a concept of copy constructor which is used to initialize an object from another object.

Syntax:

```
ClassName()
```

```
{

 //Constructor's Body

}
```

Destructor:

Like constructor, deconstructor is also a member function of a class that has the same name as the class name preceded by a tilde (~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it can't be overloaded. It is always called in the reverse order of the constructor. if a class is inherited by another class and both the classes have a destructor then the destructor of the child class is called first, followed by the destructor of the parent or base class.

Syntax:

```
~ClassName()

{

}
```

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be Private.

Question. With an example, explain what virtual function is?

Answer: A virtual function is a member function which is declared within a base class and is re-defined (Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

They are mainly used to achieve Runtime polymorphism

Functions are declared with a virtual keyword in base class.

The resolving of function calls is done at Run-time.

Rules for Virtual Functions

Virtual functions cannot be static.

A virtual function can be a friend function of another class.

Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.

The prototype of virtual functions should be the same in the base as well as derived class.

They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.

A class may have a virtual destructor but it cannot have a virtual constructor.

Question. What do you mean by polymorphism?

Answer: Polymorphism is an important concept of object-oriented programming. It simply means more than one form. That is, the same entity (function or operator) behaves differently in different scenarios. For example,

The + operator in C++ is used to perform two specific functions.

When it is used with numbers (integers and floating-point numbers), it performs addition.

```
int a = 5;

int b = 6;

int sum = a + b;    // sum = 11
```

And when we use the + operator with strings, it performs string concatenation. For example,

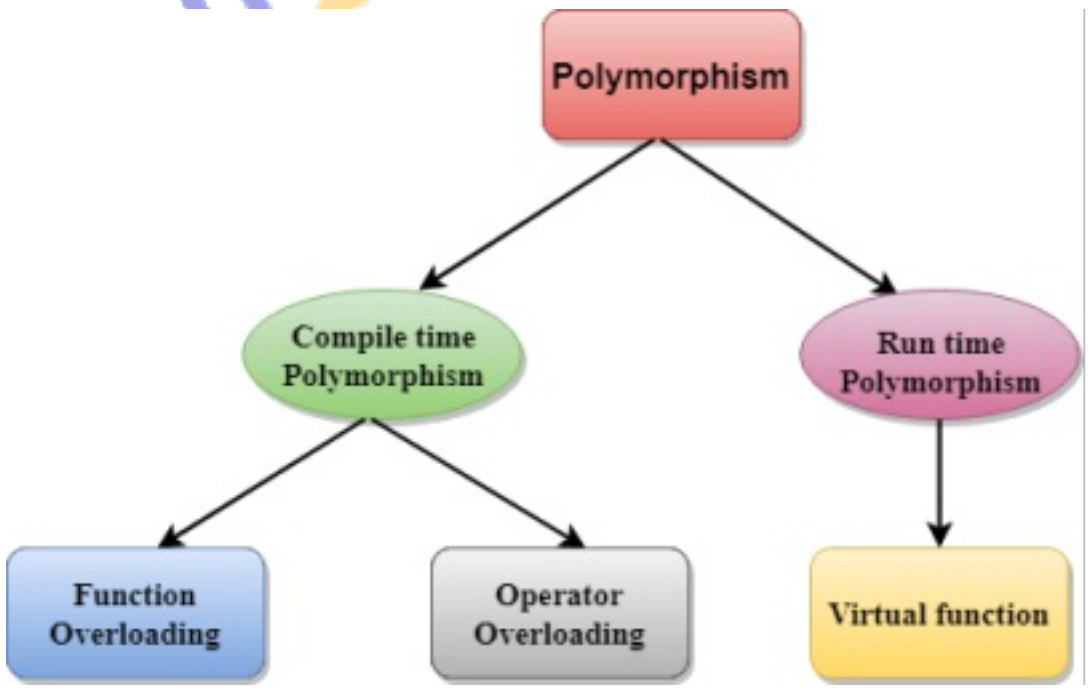
```
string firstName = "abc ";

string lastName = "xyz";



// name = "abc xyz"

string name = firstName + lastName;
```



Compile time polymorphism: The overloaded functions are invoked

by matching the type and number of arguments. This information is available at the compile time and, therefore, the compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding. Now, let's consider the case where function name and prototype are the same.

Runtime polymorphism: Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

We can implement polymorphism in C++ using the following ways:

1. Function overloading
2. Operator overloading
3. Function overriding
4. Virtual functions

Question. With an example, differentiate between run-time and compile-time polymorphism.

Answer:

Compile time polymorphism	Runtime polymorphism
The function to be invoked is known at the compile time.	The function to be invol known at the run time.
It is also known as overloading, early binding and static binding.	It is also known as over Dynamic binding and binding.

Overloading is a compile time polymorphism where more than one method is having the same name but with the different number of parameters or the type of the parameters.	Overriding is a run polymorphism where mor one method is having the name, number of paramete the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual fur and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as a things execute at the run ti

Question. What is a friend function?

Answer: A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend as follows −

```
class Box {
```

```
   double width;


   public:
      double length;
      friend void printWidth( Box box );
      void setWidth( double wid );
};
```

Question. What is virtual function and rules for virtual function?

Answer: A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.

It is used to tell the compiler to perform dynamic linkage or late binding on the function.

There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when the base class pointer contains the address of the derived class object, it always executes the base class function. This issue can only be resolved by using the 'virtual' function.

A 'virtual' is a keyword preceding the normal declaration of a function.

When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Rules of Virtual Function

Virtual functions must be members of some class.

Virtual functions cannot be static members.

They are accessed through object pointers.

They can be a friend of another class.

A virtual function must be defined in the base class, even though it is not used.

The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.

We cannot have a virtual constructor, but we can have a virtual destructor

Consider the situation when we don't use the virtual keyword.

Question. What is abstract class? Write a program to illustrate. Also outline the advantages of abstract class

Answer: An abstract class is a class in C++ which have at least one pure virtual function.

Abstract class can have normal functions and variables along with a pure virtual function.

Abstract class cannot be instantiated, but pointers and references of Abstract class type can be created.

Abstract classes are mainly used for Upcasting, so that its derived classes can use its interface.

If an Abstract Class has derived class, they must implement all pure virtual functions, or else they will become Abstract too.

We can’t create an object of abstract class as we reserve a slot for a pure virtual function in Vtable, but we don’t put any address, so Vtable will remain incomplete.

```
//Abstract base class

class Base

{

   public:

   virtual void show() = 0;   // Pure Virtual Function

};



class Derived:public Base

{

   public:

   void show()

   {

           cout << "Implementation of Virtual Function in Derived class\n";

   }

};
```

```
int main()
{
    Base obj;   //Compile Time Error
    Base *b;
    Derived d;
    b = &d;
    b->show();
}
```

OUTPUT:

Implementation of Virtual Function in Derived class

In the above example Base class is abstract, with pure virtual show () function, hence we cannot create objects of base class.

Question. Differentiate between abstract class and interface

Answer:

Abstract class	Interface
1) Abstract class can have abstract and non-abstract	Interface can have only ab methods. Since Java 8, it can

methods.	default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports m inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and variables.
4) Abstract class can provide the implementation of interfaces.	Interface can't provide implementation of abstract class
5) The abstract keyword is used to declare abstract class.	The interface keyword is us declare an interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend anothe interface only.
7) An abstract class can be extended using the keyword "extends".	An interface can be implen using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are by default.

9)Example:
public abstract class Shape{
public abstract void draw();
	Example:
	public interface Drawable{
	void draw();

}	}

Question. What is an exception? What do you mean by exception handling?

Answer: An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows −

```
try {
```

```
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

Question. With the help of an example program, differentiate between the following: a) overloading vs overriding

Answer:

1) Function Overloading happens in the same class when we declare same functions with different arguments in the same class. Function Overriding is happens in the child class when child class overrides parent class function.

2) In function overloading function signature should be different for all the overloaded functions. In function overriding the signature of both the functions (overriding function and overridden function) should be same.

3) Overloading happens at the compile time thats why it is also known as compile time polymorphism while overriding happens at run time which is why it is known as run time polymorphism.

4) In function overloading we can have any number of overloaded functions. In function overriding we can have only one overriding

function in the child class.

b) Early binding vs Late binding

In early binding, the compiler matches the function call with the correct function definition at compile time. It is also known as **Static Binding** or **Compile-time Binding**. By default, the compiler goes to the function definition which has been called during compile time. So, all the function calls you have studied till now are due to early binding.

You have learned about function overriding in which the base and derived classes have functions with the same name, parameters and return type. In that case also, early binding takes place.

In function overriding, we called the function with the objects of the classes. Now let's try to write the same example but this time calling the functions with the pointer to the base class i.e., refernce to the base class' object.

```
#include <iostream>


using namespace std;


class Animals
{
        public:
                void sound()
                {
                        cout << "This is parent class" << endl;
```

```
        }
};


class Dogs : public Animals
{
    public:
        void sound()
        {
            cout << "Dogs bark" << endl;
        }
};


int main()
{
    Animals *a;
    Dogs d;
    a= &d;
    a -> sound();   //  early binding
    return 0;
}
```

Output: This is a parent class

Now in this example, we created a pointer a to the parent class Animals. Then by writing a= &d , the pointer 'a' started referring to the object d of the class Dogs.

a -> sound(); - On calling the function sound() which is present in both the classes by the pointer 'a', the function of the parent class got called, even if the pointer is referring to the object of the class Dogs.

This is due to Early Binding. We know that a is a pointer of the parent class referring to the object of the child class. Since early binding takes place at compile-time, therefore when the compiler saw that a is a pointer of the parent class, it matched the call with the ‘sound ()' function of the parent class without considering which object the pointer is referring to.

Late Binding

In the case of late binding, the compiler matches the function call with the correct function definition at runtime. It is also known as Dynamic Binding or Runtime Binding.

In late binding, the compiler identifies the type of object at runtime and then matches the function call with the correct function definition.

By default, early binding takes place. So if by any means we tell the compiler to perform late binding, then the problem in the above example can be solved.

This can be achieved by declaring a virtual function.