TOP 30

OOPs

INTERVIEW QUESTION

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

Ans : Object Oriented Programming (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.

Q 2. Why OOPs?

Ans :

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.

Q 3. What is a Class?

Ans : 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.

Q 4. What is an Object?

Ans : 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 in C++ of how an instance of a class (i.e an object) of a class is created

```cpp
#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
```

Q 5. What are the main features of OOPs?

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

- ❖ Encapsulation
- ❖ Data Abstraction
- ❖ Polymorphism
- ❖ Inheritance

The Four Pillars

Q 6. What is Encapsulation?

Ans : 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.
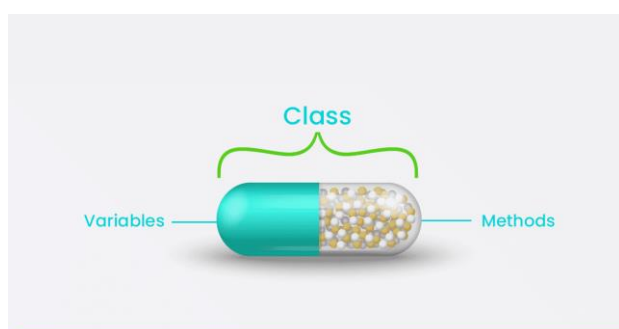
```java
public class Student {

    // 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;

    }

}
```
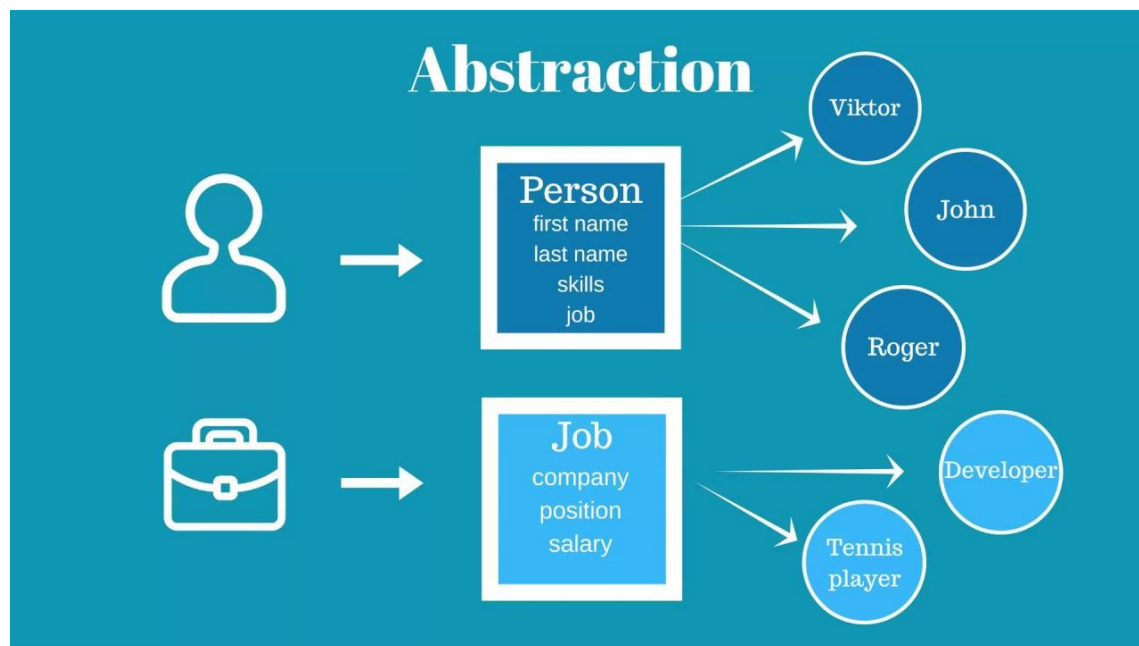
Q 7. What is Abstraction?

Ans : 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.

Q 8. What is Polymorphism?

Ans : 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.

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

Ans : 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.

```
// 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);
}
}
```

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

Ans : 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.

```
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";

}

}
```

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

Ans :

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	The length of the programs is much larger

Advantages of OOPs	Disadvantages of OOPs
redesign resulting in minimizing the complexity of an overall program.	in comparison to the procedural approach.

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

Ans : The programming paradigm is referred to the technique or approach of writing a program.

The programming paradigms can be classified into the following types:
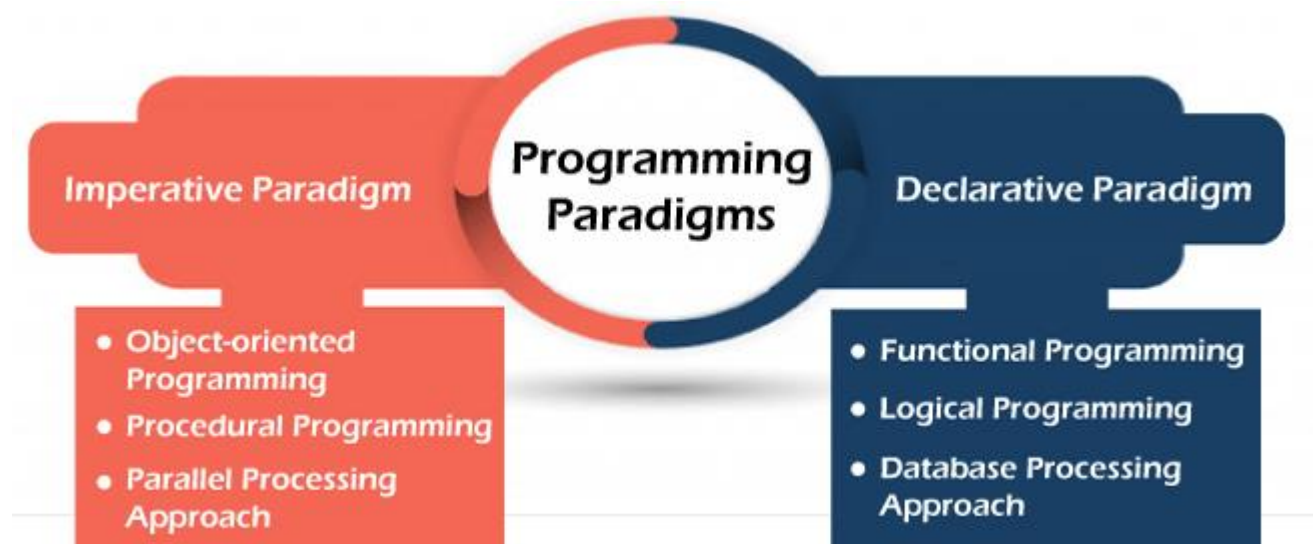
1. Imperative Programming Paradigm

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:

> ➢ **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.

> ➢ **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.

> ➢ **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:

> ➢ **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.

> ➢ **Functional Programming Paradigm**: Programs are created by applying and composing functions in this paradigm.

> ➢ **Database Programming Paradigm**: To manage data and information organized as fields, records, and files, database programming models are utilized.

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

Ans :

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	In this, the method works dynamically,

Object-Oriented Programming	Structural Programming
globally and code lines are processed one by one i.e., Run sequentially.	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.

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

Ans :

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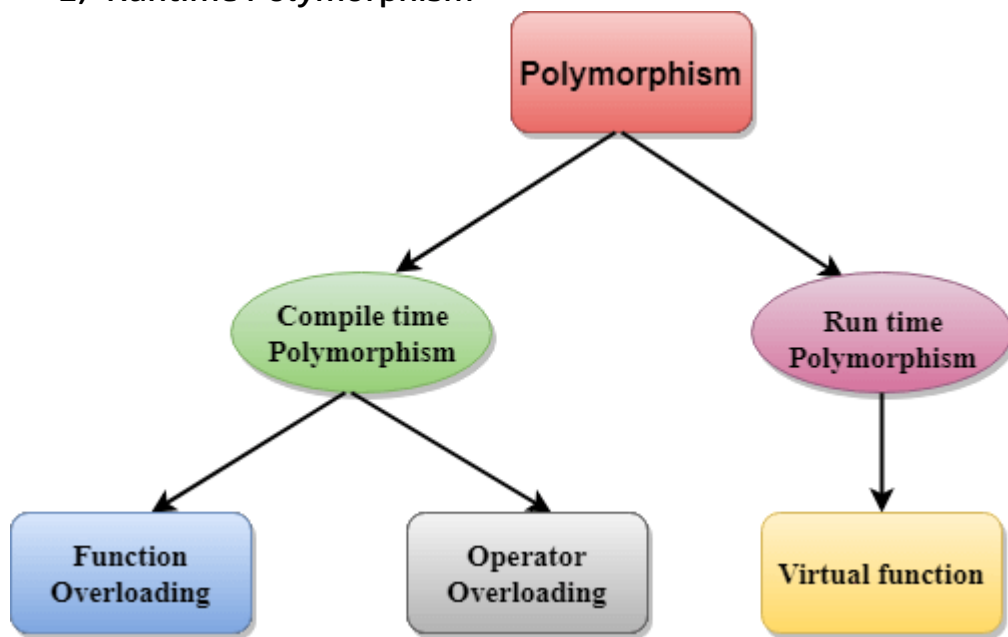
It is widely used in many popular programming languages such as:

- ◆ C++
- ◆ Java
- ◆ Python
- ◆ Javascript
- ◆ C#
- ◆ Ruby

Q 15. What are the different types of Polymorphism?

Ans : 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

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.

Q16. What is the difference between overloading and overriding?

Ans : 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.

Q 17. Are there any limitations on Inheritance?

Ans :

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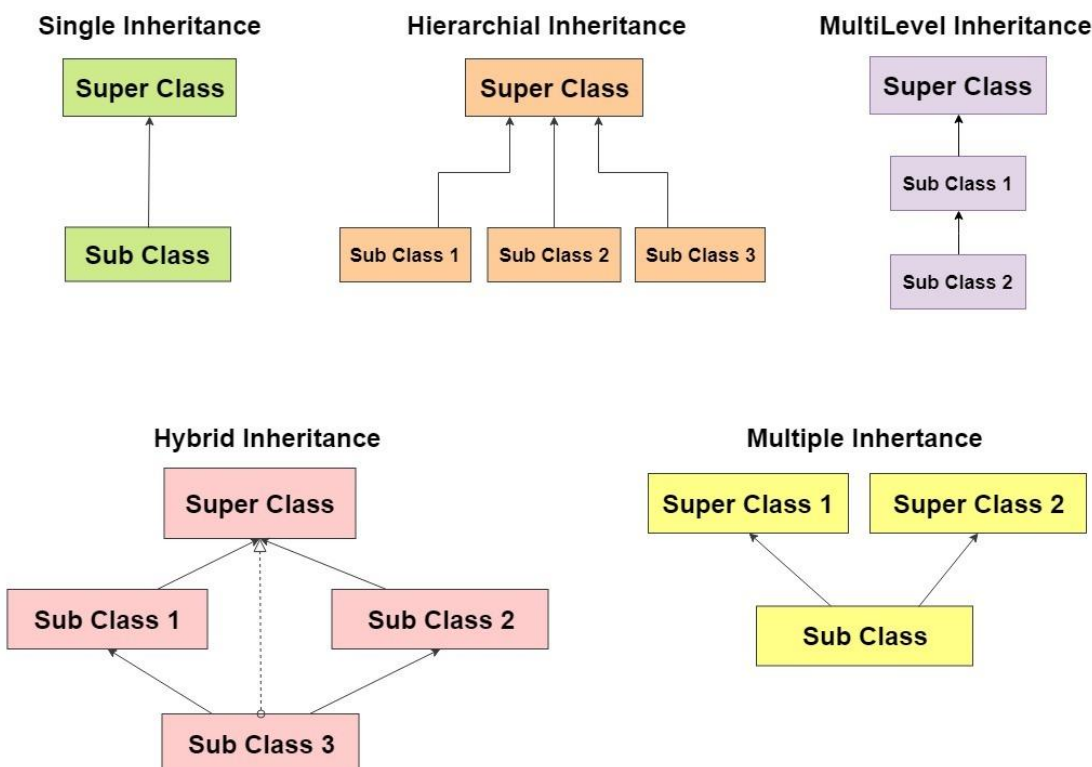
Q 18 . What different types of inheritance are there?

Ans :

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.

Single Inheritance

Super Class

↑

Sub Class

Hierarchial Inheritance

Super Class

↑

Sub Class 1   Sub Class 2   Sub Class 3

MultiLevel Inheritance

Super Class

↑

Sub Class 1

↑

Sub Class 2

Hybrid Inheritance

Super Class

Sub Class 1   Sub Class 2

Sub Class 3

Multiple Inheritance

Super Class 1   Super Class 2

Sub Class

Q 19. What is an interface?

Ans : 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.

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

Ans : 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-	The interface has only static and

Abstract Class	Interface
final, static and non-static variables.	final variables.
Abstract class doesn't support multiple inheritance.	An interface supports multiple inheritance.

Q 21. How much memory does a class occupy?

Ans : 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.

Q22. Is it always necessary to create objects from class?

Ans :

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.

Q23. What is the difference between a structure and a class in C++?

Ans :

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++.

Q 24. What is Constructor?

Ans :

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.

```
class Student {
    String name;
    int rollNo;
    Student()
    {
        System.out.println("contructor is called");
    }
}
```

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

Ans :

The most common classification of constructors includes:

1) **Default Constructor**
2) **Parameterized Constructor**
3) **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. Parameterized Constructor
- It is a user-defined constructor having arguments or parameters.

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

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

Q 26. What is a destructor?

Ans : 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__**.

Q27. Can we overload the constructor in a class?

Ans : 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.

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

Ans :

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

Q 29. What is the virtual function?

Ans : 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.

Q 30. What is pure virtual function?

Ans :

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.

```
abstract class base {

        abstract void prVirFunc();

}
```

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like

Comment

Repost