OOPS :-

Object-oriented programming is an approach or a programming sample where the packages are structured around objects rather than functions and logic. It makes the data partitioned into memory areas, i.e., data and functions and helps make the code flexible and modular.
The four pillars of Object-Oriented Programming (OOPS) are:

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

Object-oriented programming mainly focuses on objects which might be required to be manipulated. In OOPs, it may represent data as objects with attributes and functions.

1. Abstraction

Data abstraction is the most essential function of object-oriented programming in C++. Abstraction means displaying only basic information and hiding the details. Data abstraction refers to providing only necessary information about the data to the outside world, hiding the background info or implementation.

2. Encapsulation

In common terms, Encapsulation is defined as wrapping up of Data and information under a single unit. In object-oriented programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Encapsulation also leads to data abstraction or hiding. Using Encapsulation also hides the data.

3. Inheritance

Inheritance is the process in which two classes have a relationship with each other, and objects of one class acquire properties and features of the other class. The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class.

Subclass

The class that inherits properties from another class is called Subclass or Derived class.

Superclass

The class that inherits sub-class properties is called Base class or superclass.

4. Polymorphism

The word polymorphism means having many forms. It is the ability to take more than one form. It is a feature that provides a function or an operator with more than one definition. It can be implemented using function overloading, operator overload, function overriding, virtual function. An operation may show off different behaviors at different times.
C++ supports operator overloading and function overloading.

How to Use the Four Pillars of OOPs in Your Projects

To effectively use the four pillars of Object-Oriented Programming (OOP) in your projects, follow these guidelines:
Encapsulation:

- Define classes that encapsulate data (attributes) and methods (functions) that operate on that data.
- Use access modifiers (e.g., private, protected, public) to control access to class members.
- Encapsulate data with getters and setters to maintain control over data access and manipulation.

Abstraction:

- Identify the essential attributes and behaviors of your objects.
- Create abstract classes and interfaces that define common behaviors and attributes.
- Hide unnecessary implementation details from external code, exposing only what's relevant.

Inheritance:

- Identify relationships and hierarchies between classes. Use inheritance to model these relationships.
- Create a base (super) class that defines common attributes and behaviors shared by subclasses.
- Reuse code and promote consistency by inheriting attributes and methods from a superclass.

Polymorphism:

- Define methods with the same name in different classes to enable method overriding (subclasses providing their implementations).
- Utilize method overloading to define multiple methods with the same name but different parameter lists.
- Take advantage of interfaces and abstract classes to achieve polymorphic behavior.

Overall, the effective use of OOP principles involves proper design and planning. Consider the specific needs of your project and how encapsulation, abstraction, inheritance, and polymorphism can help you create modular, maintainable, and extensible code. Regularly review and refactor your code to ensure it adheres to these principles, improving the quality and readability of your software.

The Benefits of Incorporating the Four Pillars of OOPs

Here are the benefits of incorporating the Four Pillars of OOP:

Encapsulation

- It keeps your data safe and hidden inside the class.
- It makes your code organized and allows independent handling of class-specific tasks.

Abstraction

- It allows using classes without worrying about their complicated internal workings.

- It helps to focus on what the classes do rather than how they do it. It makes code easier to understand.

Inheritance

- It enables the creation of new classes that reuse features from existing classes.
- It saves time and effort by building on existing code.

Polymorphism

- It allows you to treat different objects similarly, even if they work differently internally.
- It provides flexibility, which makes your code more adaptable to changes in different situations.

Exploring the Advantages of the Four Pillars of OOPs

Here are the advantages of the Four Pillars of OOPs:

Encapsulation

- Data hiding for security and integrity.
- Easier maintenance and troubleshooting.
- Reusability of code in different parts of the program.

Inheritance

- Code reusability by inheriting properties and behaviors from a base class.
- The organized hierarchical relationship between classes.
- Reduces redundancy and promotes a consistent codebase.

Polymorphism

- Flexibility and extensibility by adding new classes without modifying existing code.
- Simplifies code by using a common interface for different data types.
- Improves code readability and maintainability.

Abstraction

- Flexibility to change implementations without affecting other parts of the code.
- Better communication among team members with a high-level view of objects.
- It helps manage and understand complex systems more efficiently.

Fundamental Design Principles of the Four Pillars of OOPs

Here are the design principles of the Four Pillars of OOPs:

- **Encapsulation:** This principle involves bundling data (attributes) and methods (functions) that operate on that data within a single unit, known as a class. The class acts as a blueprint for creating objects. Encapsulation hides the class's internal implementation details and provides a clean and controlled interface for interacting with the object.
- **Inheritance:** Inheritance is a mechanism where a class (derived or subclass class) can inherit properties and behaviors (methods and attributes) from another class (superclass class). This allows developers to build on existing classes, saving time and effort.
- **Polymorphism:** Polymorphism allows objects of different classes to be treated as belonging to a common group. You can use the same method names with other classes, and the program will automatically know which version of the method to use based on the object being used. Polymorphism helps write flexible, generic code that works with various object types.
- **Abstraction:** Abstraction simplifies the representation of objects by highlighting only the essential characteristics and behaviors while hiding unnecessary details. It allows developers to create easy-to-use interfaces for working with objects without knowing the complexities of their implementation.