**What is Object-Oriented Programming?**

Object-oriented programming (OOP) is defined as a programming paradigm (and not a specific language) built on the concept of classes and objects. Each object is an instance of a class, properties (attributes) represent the data of an object, while methods represent the behavior or actions that the object can perform. There are many object-oriented programming languages, including JavaScript, C++, C#, Java, and Python.

A diagram of a car

Description automatically generated

**What are basic principles of Object-Oriented Programming?**

There are four major principles of an object-oriented programming language. These are encapsulation, Data Abstraction, Polymorphism and Inheritance. These are also called four pillars of Object-Oriented programming.

1. **Encapsulation**

**Definition**

Encapsulation is the mechanism of hiding internals details of an object and only exposing a controlled interface. This is done by making instance variable private and providing public methods to access and modify these variables.

It’s like a car where the engine, fuel system, and controls are all enclosed within the car. You don’t see or interact with these internal components directly; instead, you use the car’s controls (like the steering wheel) to drive.

Inside the smartphone, all the components (like the processor, battery, and memory) are neatly packaged together. You can’t access these components directly; instead, you use the smartphone’s touchscreen and buttons to interact with it.

**Purpose**

The goal is to protect the internal state of the object and ensure that it can only be modified in controlled ways through defined methods.

Encapsulation is the practical approach to implementing abstraction by keeping the internal details hidden and only exposing a controlled interface.

**Example:** Consider the following class **`Addition`** using c++**:**

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

In this example:

**Private Variables: `num1` and `num2`** are private, which means they cannot be accessed directly from outside the **`Employee`** class.

**Public Methods: `void input(….)`, `void sum()`** are public, providing controlled accessed to the private variables. These methods provide the interface through which other classes can interact with the object.

**INHERITANCE**

Polymorphism is a core principle of Object-Oriented Programming (OOP)

**Polymorphism**

In OOP, polymorphism allows objects of different classes to be treated as objects of a common superclass.

In Object-Oriented Programming, any object or method has more than one name associated with it. That is nothing but **polymorphism**.

**For Objects –** When any object can hold the reference of its parent object then it is a polymorphism.

**Example of Polymorphism**

To understand polymorphism more clearly, consider the example of televisions from different companies like Samsung, LG, Xiaomi, etc. Regardless of the brand, we generally refer to them as "Television." This illustrates the concept of polymorphism, where objects of different types (Samsung TV, LG TV, Xiaomi TV) can be treated as objects of a common type (Television).

**A diagram of a television

Description automatically generated**

**A white background with text

Description automatically generated with medium confidence**

**A close up of text

Description automatically generated**

**A close up of words

Description automatically generated**

**A black screen with white text

Description automatically generated** **A screen shot of a computer code

Description automatically generated**

A computer screen with text

Description automatically generatedA screen shot of a computer program

Description automatically generated

A close up of text

Description automatically generated

Definition

Polymorphism is a core principle of Object-Oriented Programming (OOP) that allows objects of different classes to be treated as objects of a common superclass.

Purpose

Polymorphism simplifies code and enhances flexibility by allowing the same method to operate differently based on the object's actual type.

Types of Polymorphism

1. Compile-Time Polymorphism
2. Runtime Polymorphism

Compile-Time Polymorphism

Achieved Using Method Overloading:

Definition: Method overloading allows multiple methods with the same name but different parameters or different data types.

How It Works: The compiler decides which method to call based on the method signature (i.e., method name, number of parameters, and data types) at compile time.

When It Happens: During the compilation of the code.

Runtime Polymorphism:

Achieved Using Method Overriding:

Definition: Method overriding allows a derived class to provide a specific implementation of a method that is already defined in its base class.

How It Works: The method that gets called is determined at runtime based on the actual object type.

When It Happens: When the program is running, and the specific method implementation is chosen based on the actual object type.

A computer screen shot of a program

Description automatically generated A computer screen with text on it

Description automatically generated

In this example, the **Calculator** class provides multiple ways to add numbers together, which can be thought of as different "operations" that the calculator can perform based on the number and type of inputs provided.

**A screen shot of a computer code

Description automatically generated**

**A computer screen with text

Description automatically generated** **A screen shot of a computer code

Description automatically generated**

**A computer screen shot of text

Description automatically generated**

In this example, the **Animal** class and its derived classes **Dog** and **Cat** provide specific implementations of the **sound** method, illustrating how different types of animals produce different sounds.

**ABSTRACTION**

Think about your TV remote. When you press the ON button, the TV turns on. Do you need to know exactly how the remote sends signals to the TV? Do you need to understand the technical details of the 0's and 1's it uses? No, pressing the ON button is enough.

Abstraction makes things easier to use. Most people know that pressing the ON button on a remote turns on the TV. They don't need to understand the complex electronics behind it. If we did need to know all that detail, it would be too hard to use TVs.

**Conceptual Abstraction:** You know the car has a steering wheel, pedals, and buttons, and you know how to use them to drive. You don’t need to understand the engine mechanics or the electrical systems to drive the car effectively.

**Practical Implementation:** When you drive, you interact with the car through its controls, which abstract away the complexities of how the car works internally. The car’s design and engineering implement this abstraction.

**Design Phase:**

During the design phase of software development, abstraction helps you decide what features and details to expose and what to hide. For example, when designing a class, you decide what methods and properties are public (for interaction) and what details remain private (hidden from the user).

**Implementation:**

In practice, abstraction is implemented through classes, interfaces, and abstract methods in object-oriented programming. These constructs help you build systems where users interact with simplified interfaces while the complexity is managed internally.

**Code Examples:**

**Class Interfaces:** In C#, you use classes with public and private members to implement abstraction. The public methods provide a simplified interface, while the private members handle the internal details.

**Abstract Classes and Interfaces:** Languages like C# support abstract classes and interfaces, which are concrete implementations of abstraction. They define what methods a class should have, without specifying how those methods are implemented.

**Definition**

* Abstraction is about hiding the complex details and showing only the essential features of an object or system.
* It’s like using a TV remote: you press the ON button to turn on the TV without needing to know how the remote sends the signal or how the TV receives it.
* Think of a smartphone. You see a screen with apps and buttons, but you don’t need to know how the phone’s hardware or software works internally to use it. You interact with the phone through its simple interface.

**Purpose**

* The goal is to simplify how you interact with a system by focusing only on the necessary information and ignoring the complex internal workings.
* Abstraction gives you a way to think about and use objects by focusing on what they do rather than how they do it.
* Abstraction is about what you see and interact with (the user interface or functionality).

**Example:** Consider the following class **`Vehicle`** using c++**:**

* the **virtual** keyword is used to define virtual functions, which can be overridden in derived classes. When a virtual function is set to 0, it becomes a pure virtual function, making the class an abstract class.
* A **pure virtual function** is a function that has no definition in the base class and must be overridden in any derived class. A class containing at least one pure virtual function is considered an abstract class and cannot be instantiated directly.
* In summary, using the **virtual** keyword with **= 0** to create **pure virtual** functions in C++ serves a similar purpose to the **abstract keyword** in other programming languages, enabling abstraction by defining a common interface while hiding the implementation details.

**A computer screen shot of a program code

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

In this example:

In the vehicle management system, the abstract class **Vehicle** exposes methods like **start()** and **stop()** to the user. The user doesn't need to know how a car engine works or how a bicycle's braking system operates. They only need to know how to start and stop the vehicle.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Building blocks of OOP**

* Classes
* Objects
* Methods
* Attributes

**How Object-Oriented programming is different from Procedure Oriented programming?**

Object-Oriented Programming (OOP):

* Focuses on objects and their interactions.
* Programs are organized around objects, encapsulating data and behavior.
* Supports encapsulation, inheritance, polymorphism, and abstraction.
* Encourages code reusability and modularity through classes and objects.
* There is a possibility of data hiding.
* Data is encapsulated within objects, and access to this data is controlled through methods.
* follows a bottom-up approach.
* has access specifiers like private, public, protected, etc.
* used for designing large and complex programs.
* Example: .NET, C#, Python, Java, VB.NET, and C++.

Procedure-Oriented Programming (POP):

* Focuses on functions that manipulate data.
* Programs are organized around functions, breaking down large programs into smaller functions
* Lacks support for encapsulation, inheritance, and polymorphism.
* Code is generally less modular, and reusable compared to OOP.
* There is not any proper way for data hiding.
* Functions share the same data, which can lead to problems if multiple functions change the data.
* follows a top-down approach.
* no access specifier
* May be suitable for smaller, less complex projects or tasks.
* Example: C, Fortran, Pascal, and VB.

Key => interaction, organization, features, modularity and reusability, data hiding, data sharing, approach, access specifiers, suitable for projects, example.

**Feature / Advantages / Need of Object Programming Language**

**Modularity:** Promotes modularity by breaking down complex systems into smaller, enhancing code organization, reuse, maintain.

**Encapsulation:** Data (attributes/properties) and methods are bundled together inside objects, hiding them from the outside world, and exposing only the required methods/functions.

**Inheritance:** Allows classes to inherit attributes and methods from other classes, promoting code reusability and maintainability.

**Polymorphism:** Enables objects of different classes to be treated as objects of a common superclass, facilitating dynamic method binding and providing flexibility.

**Abstraction:** Shows only required data or information to the user, hiding implementation details and promoting simplicity.

**Code Reusability:** Enables the reuse of code (objects, classes) across programs, speeding up development and easy maintenance by changing in the existing components.

**Hierarchical Organization:** Facilitates the creation of class hierarchies, enhancing code organization and structure.

**Flexibility and Scalability:** Allows for adding or modifying features without altering existing code, adapting to changing demands effectively.

**Easy Troubleshooting:** Facilitates easier debugging and testing because we can test each part separately. This saves time and effort in finding and fixing problems.

**Handling Complex Problems:** Particularly effective for designing large and complex programs.

**Mention advantages of OOPS approach over functional/procedural programming?**

**Modularity:** OOP promotes modularity by breaking down complex systems into smaller, enhancing code readability and maintainability. In contrast, functional/procedural programming might lead to longer, less modular code that's harder to maintain and extend.

**Encapsulation for Data Protection:** In OOP, data (attributes/properties) and methods are bundled together within objects, hiding them from the outside world. This enhances data protection and security, preventing unauthorized access and modification, which can be more difficult to achieve in functional or procedural programming.

**Inheritance and Polymorphism for Flexibility and Extensibility:** In OOP, classes inherit attributes and methods from other classes, promoting code reuse and maintenance. Polymorphism allows different class objects to be treated as if they're of a common superclass, allowing for flexible and extensible behavior. Functional and procedural programming styles generally lack these features, making it harder to achieve.

**Abstraction for Simplified Complexity:** In OOP, abstraction hides complex details, making systems easier to understand and modify. This contrasts with functional programming, where data is often shared among functions, leading to potential data integrity issues due to simultaneous modifications of the same data.

**Code Reusability:** In OOP, allows you to reuse code (like objects and classes) in different programs, which speeds up development and makes maintenance easier by using components that already exist. While functional or procedural programming also supports code reuse, OOP typically provides a more structured and organized approach to code reuse.

**Hierarchical Organization:** Facilitates the creation of class hierarchies, enhancing code organization and structure, result maintainability in large and complex projects, which might be more challenging to achieve in functional or procedural programming.

**Flexibility and Scalability:** OOP allows for adding or modifying features without altering existing code, adapting to new needs. This flexibility enables OOP systems to evolve over time while maintaining stability and integrity, which might be more difficult to achieve in functional or procedural programming.

**Easy Troubleshooting:** Facilitates easier debugging and testing because we can test each part separately. This saves time and effort in finding and fixing problems, compared to procedural or functional programming, where troubleshooting can be more complex.

**Discuss how data and functions organized in an object-oriented paradigm are. List major areas of application of OOP.**

In OOP, data and functions are grouped into objects. These objects contain both data (attributes/properties) and functions (methods/behaviors) that work with the data.

**Data Organization:** Data is stored within objects as attributes or properties. These properties define the characteristics of the object. For example, in a "Car" object, attributes might include things like "color," "make," and "model."

**Function Organization:** Functions are defined within objects as methods. These methods represent the actions or behaviors that the object can perform. Continuing with the "Car" example, methods might include "start," "stop," and "accelerate."

**Encapsulation for Data Protection:** In OOP, data (attributes/properties) and methods are bundled together within objects, hiding them from the outside world. This enhances data protection and security, preventing unauthorized access and modification.

**Inheritance and Polymorphism for Flexibility and Extensibility:** In OOP, classes inherit attributes and methods from other classes, promoting code reuse and maintenance. Polymorphism allows different class objects to be treated as if they're of a common superclass, allowing for flexible and extensible behavior.

**Abstraction for Simplified Complexity:** In OOP, abstraction hides complex details, making systems easier to understand and modify.

**Application of OOPs**

**Software Development:** OOP makes it easier to create, reuse, and manage code for large and complicated software projects.

**Graphical User Interfaces (GUIs):** OOP is commonly used in GUI development to represent graphical elements like windows, buttons, and menus as objects with their own properties and functions.

**Web Development:** Many web development application/frameworks/libraries such as C#, Python, Java are built on OOP principles. OOP is used to model web application as collection of objects, like users, pages, content.

**Game Development:** OOP is well-suited for game development because its represent game entities such as character (active player/non-active player), objects (gun, bag, skin, color) and environments (levels, maps, location) as objects with associated behavior. For example, a character object may have attributes like health, position, and speed, along with methods like move (), jump (), and attack ().

**Database System:** OOP concept like encapsulation is used in object-relational mapping (ORM) frameworks to map database entities to objects in the application code.

**Simulation and Modeling:** It is used to represent real-world system as objects with associated behaviors.

* Because of these simulations allow researchers, engineers, and scientists to study and analyze the behavior of complex systems under various conditions without the need for physical experiments.
* Flight Simulation: Flight simulators replicate the behavior of aircraft in various flight conditions, including takeoff, landing, and in-flight maneuvers. Pilots and aviation engineers use these simulations for training, aircraft design, and safety testing.

**Embedded Systems:** OOP allows for modular and structured code in embedded systems, enhancing efficiency and maintainability.

* Modular and Structured code => Each component (or object) encapsulates both data and functionality, making the code easier to understand, maintain, and reuse.
* Efficiency => Instead of writing the same functionality multiple times, developers can create object-oriented modules that can be reused across different parts of the system, saving both time and memory resources.
* Maintainability => OOP principles such as encapsulation, inheritance, and polymorphism promote code maintainability in embedded systems.
* **A common example of embedded systems** in real life is automotive electronics, where microcontrollers and sensors are used to control various functions within a vehicle. OOP is widely used in automotive embedded systems to manage tasks such as engine control, braking, steering, and infotainment.

**Mobile App Development:** It provides structure approach for development, through code reusability, readability, maintainability, and testability.

**AI development:** AI makes code easier to manage by breaking tasks into smaller parts. Principles like modularity, encapsulation, and inheritance to improve efficiency and collaboration of AI system.