

The Four Pillars of Object-Oriented Programming (OOP)

Object-Oriented Programming is built on four foundational principles—**Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism**. These pillars work together to create modular, maintainable, and scalable software systems. Below is a detailed discussion of each:

Interrelationship of Pillars

- **Encapsulation** and **Abstraction** work together to hide complexity.
- **Inheritance** and **Polymorphism** enable extensibility and dynamic behavior.
- Combined, these pillars make OOP powerful for building modular, maintainable, and scalable software systems.

Encapsulation

Encapsulation is the principle of **bundling data (attributes) and behavior (methods) into a single unit**, typically a class, and restricting direct access to some components. This is achieved through **access modifiers** (e.g., `private`, `protected`, `public`).

- **Purpose:**
 - Protects the internal state of an object from unintended interference.
 - Promotes modularity and maintainability.
- **Example:**
A `BankAccount` class hides its balance behind getter and setter methods, ensuring validation before changes.
- **Key Benefit:**
Reduces coupling and increases cohesion.

Abstraction

Abstraction focuses on **exposing only essential features while hiding unnecessary details**. It allows developers to work at a higher conceptual level without worrying about implementation complexity.

- **Purpose:**
 - Simplifies interaction with complex systems.
 - Provides clear contracts through abstract classes or interfaces.
- **Example:**
An interface `PaymentProcessor` defines methods like `processPayment()`, without specifying how the payment is processed internally.
- **Key Benefit:**
Enhances flexibility and scalability.

Inheritance


Inheritance enables a class (child or subclass) to **reuse and extend the properties and behaviors of another class (parent or superclass)**.

- **Purpose:**
 - Promotes code reuse.
 - Establishes hierarchical relationships.
- **Example:**
A `Car` class inherits from a `Vehicle` class, gaining attributes like `speed` and methods like `move()`.
- **Key Benefit:**
Reduces redundancy and supports polymorphism.
- **Caution:**
Overuse can lead to fragile hierarchies; composition is often preferred for flexibility.

Polymorphism

Polymorphism allows **objects of different classes to be treated uniformly through a common interface**, while each object can respond differently to the same method call.

- **Types:**
 - **Compile-time (Static):** Method overloading.
 - **Run-time (Dynamic):** Method overriding via inheritance.
- **Example:**
A `draw()` method behaves differently for `Circle` and `Rectangle` objects, even when called through a `Shape` reference.
- **Key Benefit:**
Enables extensibility and dynamic behavior without modifying existing code.

 Footer Separator

Markdown Viewer

How to view the markdown files in a browser...

- [Markdown Viewer](#)

Lecture Practices

Here are the lecture Practices...

- [Day 7](#)
- [Day 8](#)
- [Day 9](#)

Lecture Quizzes

Here are the lecture quizzes...

- [Day 7](#)
- [Day 8](#)
- [Day 9](#)

Weekly Topics

Here are the topics for the week...

- [Classes](#)
- [Structs](#)
- [Fields](#)
- [Getters and Setters](#)
- [Constructors](#)
- [Instances](#)
- [Inheritance](#)
- [Polymorphism](#)
- [Pointers](#)
- [Upcasting](#)
- [Misc. Concepts](#)
- [4 Pillars of OOP](#)