**Object Oriented Programming System (OOPS)**

Object-oriented programming (OOP) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_%28computer_science%29)", which may contain [data](https://en.wikipedia.org/wiki/Data), in the form of [fields](https://en.wikipedia.org/wiki/Field_%28computer_science%29), often known as *attributes;* and code, in the form of procedures, often known as [*methods*](https://en.wikipedia.org/wiki/Method_%28computer_science%29)*.* A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "[this](https://en.wikipedia.org/wiki/This_%28computer_programming%29)" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. There is significant diversity of OOP languages, but the most popular ones are [class-based](https://en.wikipedia.org/wiki/Class-based_programming), meaning that objects are [instances](https://en.wikipedia.org/wiki/Instance_%28computer_science%29) of [classes](https://en.wikipedia.org/wiki/Class_%28computer_science%29), which typically also determine their [type](https://en.wikipedia.org/wiki/Data_type).

Many of the most widely used programming languages (such as C++, Object Pascal, Java, Python etc.) are [multi-paradigm programming languages](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) that support object-oriented programming to a greater or lesser degree, typically in combination with [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming). Significant object-oriented languages include [Java](https://en.wikipedia.org/wiki/Java_%28programming_language%29), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [C#](https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29), [Python](https://en.wikipedia.org/wiki/Python_%28programming_language%29), [PHP](https://en.wikipedia.org/wiki/PHP), [Ruby](https://en.wikipedia.org/wiki/Ruby_%28programming_language%29), [Perl](https://en.wikipedia.org/wiki/Perl), [Object Pascal](https://en.wikipedia.org/wiki/Object_Pascal), [Objective-C](https://en.wikipedia.org/wiki/Objective-C), [Dart](https://en.wikipedia.org/wiki/Dart_%28programming_language%29), [Swift](https://en.wikipedia.org/wiki/Swift_%28programming_language%29), [Scala](https://en.wikipedia.org/wiki/Scala_%28programming_language%29%22%20%5Co%20%22Scala%20%28programming%20language%29), [Common Lisp](https://en.wikipedia.org/wiki/Lisp_%28programming_language%29), and [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk).

### Encapsulation

Encapsulation is an object-oriented programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of [data hiding](https://en.wikipedia.org/wiki/Information_hiding).

If a class does not allow calling code to access internal object data and permits access through methods only, this is a strong form of abstraction or information hiding known as [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_%28object-oriented_programming%29). Some languages (Java, for example) let classes enforce access restrictions explicitly, for example denoting internal data with the private keyword and *designating m*ethods intended for use by code outside the class with the public keyword. Methods may also be designed public, private, or intermediate levels such as protected (which allows access from the same class and its subclasses, but not objects of a different class). In other languages (like Python) this is enforced only by convention (for example, private methods may have names that start with an [underscore](https://en.wikipedia.org/wiki/Underscore)). Encapsulation prevents external code from being concerned with the internal workings of an object. This facilitates [code refactoring](https://en.wikipedia.org/wiki/Code_refactoring), for example allowing the author of the class to change how objects of that class represent their data internally without changing any external code (as long as "public" method calls work the same way). It also encourages programmers to put all the code that is concerned with a certain set of data in the same class, which organizes it for easy comprehension by other programmers. Encapsulation is a technique that encourages [decoupling](https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29).

### Composition, inheritance, and delegation

Objects can contain other objects in their instance variables; this is known as[object composition](https://en.wikipedia.org/wiki/Object_composition). For example, an object in the Employee class might contain (point to) an object in the Address class, in addition to its own instance variables like "first name" and "position". Object composition is used to represent "has-a" relationships: every employee has an address, so every Employee object has a place to store an Address object.

Languages that support classes almost always support[inheritance](https://en.wikipedia.org/wiki/Inheritance_%28object-oriented_programming%29). This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships. For example, class Employee might inherit from class Person. All the data and methods available to the parent class also appear in the child class with the same names. For example, class Person might define variables "first name" and "last name" with method "make\_full\_name()". These will also be available in class Employee, which might add the variables "position" and "salary". This technique allows easy re-use of the same procedures and data definitions, in addition to potentially mirroring real-world relationships in an intuitive way. Rather than utilizing database tables and programming subroutines, the developer utilizes objects the user may be more familiar with: objects from their application domain.

Subclasses can override the methods defined by super classes.[Multiple inheritances](https://en.wikipedia.org/wiki/Multiple_inheritance)is allowed in some languages, though this can make resolving overrides complicated. Some languages have special support for[mixing](https://en.wikipedia.org/wiki/Mixin), though in any language with multiple inheritance, a mixing is simply a class that does not represent an is-a-type-of relationship. Mixins are typically used to add the same methods to multiple classes. For example, class UnicodeConversionMixin might provide a method unicode\_to\_ascii() when included in class File Reader and class Webpage Scraper, which don't share a common parent.