



[Main Page](#)
[Recent changes](#)
[Random page](#)
[Current events](#)

[Edit this page](#)
[Discuss this page](#)
[Page history](#)
[What links here](#)
[Related changes](#)

[Special pages](#)
[Contact us](#)
[Donations](#)

[Main Page](#) | [Recent changes](#) | [Edit this page](#) | [Page history](#)
[Printable version](#)

Not logged in
[Log in](#) | [Help](#)

[Go](#) [Search](#)

Other languages: [Deutsch](#) | [Español](#) | [Français](#) | [Nederlands](#) | [Polski](#) | [Português](#)

Object-oriented programming

(Redirected from [Object Oriented Programming](#))

Object-oriented programming (OOP) is a computer [programming paradigm](#) that emphasizes the following aspects:

- The use of objects - [objects](#) are used extensively to modularize and structure the computer program.
- [Abstraction](#) - combining multiple smaller operations into a single unit that can be referred to by name.
- [Encapsulation](#) - separating implementation from [interfaces](#).
- [Polymorphism](#) - using the same name to invoke different operations on objects of different [data types](#).
- [Inheritance](#) - defining objects data types as extensions and/or restrictions of other object data types.

Notes: Abstraction is important to but not unique to OOP. [Reusability](#) is a benefit often attributed to OOP.

OOP is often called a paradigm rather than style or type of programming to emphasize the point that OOP can change the way software is developed, by changing the way that programmers and software engineers think about software.

Table of contents [\[hide\]](#)

- [1 Basic](#)
 - [1.1 Inheritance](#)
- [2 OOP with procedural languages](#)
- [3 Definition](#)
- [4 Class-based models](#)
- [5 Object-based models](#)
- [6 History](#)
- [7 Further reading](#)
- [8 See also:](#)
- [9 External link](#)

Basic

The most fundamental aspect of object-oriented programming is that a computer program is composed of a collection of what are called objects. Each object is capable of receiving messages and sending messages to other objects and the user accesses the object via the interface, independent way from the internal structure of the object ([encapsulation](#))--the technique known

as [abstraction with use of objects](#).

In OOP, programmers are supposed to concentrate on distributing responsibility over objects, which is in contrast to the view in traditional paradigms such as [imperative programming](#) where programmers are more concerned about steps of data processes to produce certain results. The proponents of OOP claim that the use of OOP style programming can make programs easier to write, maintain, reuse and prove correct (at least in some cases).

Inheritance

Inheritance: One object's data and/or functionality may be based on those of other objects, from which the former object is said to *inherit*. This allows commonalities among different kinds of objects to be expressed once and reused multiple times. Inheritance is also commonly held to include *subtyping*, whereby one type of object is defined to be a more specialised version of another type (see [Liskov substitution principle](#)), though non-subtyping inheritance is also possible. Inheritance is typically expressed by describing *classes* of objects arranged in an *inheritance hierarchy* reflecting common behavior.

OOP with procedural languages

In procedural languages, OOP often appears as a form where [data types](#) are extended to behave like an object in OOP, very similar to [abstract data type](#) with an extension such as [inheritance](#). Each [method](#) is actually a [subprogram](#) which is syntactically bound to a class.

Definition

The definitions of OOP are disputed. In the most general sense, object-oriented programming refers to the practice of viewing software primarily in terms of the "things" (objects) it manipulates, rather than the actions it performs. Other paradigms such as [functional](#) and [procedural](#) programming focus primarily on the actions, with the objects being secondary considerations; in OOP, the situation is converse.

Widely-used terminology distinguishes *object-oriented* programming from *object-based*. The former is held to include inheritance (described below), while the latter does not. See [Dispute over the definition of object-oriented programming](#)

Class-based models

The most popular and developed model of OOP is a classed based model, as opposed to object-based model. In this model, objects are entities that combine both *state* (i.e., data) and *behavior* (i.e., procedures, or [methods](#)). Objects are defined by a [classes](#), which is a definition, or blueprint, of all

objects of a specific type. An object is considered to be an instance of a class. A class is similar to a structure, with the addition of method pointers, member access control, and an implicit member data type which locates instances of the class (i.e.: actual objects of that class) in the class hierarchy (essential for runtime inheritance features).

Object-based models

[Object-based programming](#) techniques include the concept of an object (encapsulation, and abstraction) but do not include the class-based models of inheritance.

History

The concepts of object-oriented programming first took root in [Simula 67](#), a language designed for making simulations, created by [Ole-Johan Dahl](#) and [Kristen Nygaard](#) of the [Norwegian Computing Centre](#) in [Oslo](#). (Reportedly, the story is that they were working on ship simulations, and were confounded by the combinatorial explosion of how the different attributes from different ships could affect one another. The idea occurred to group the different types of ships into different classes of objects, each class of objects being responsible for defining its *own* data and behavior.) They were later refined in [Smalltalk](#), which was developed in Simula at [Xerox PARC](#), but was designed to be a fully dynamic system in which objects could be created and modified "on the fly" rather than having a system based on static programs.

Object-oriented programming "took off" as the dominant programming methodology during the mid-1980s, largely due to the influence of [C++](#), an extension of the [C programming language](#). Its dominance was further cemented by the rising popularity of [Graphical user interfaces](#), for which object-oriented programming is allegedly well-suited. Indeed, the rise of GUIs changed the user focus from the sequential instructions of text-based interfaces to the more dynamic manipulation of tangible components. An example of a closely related dynamic GUI library and OOP language can be found in the [Cocoa](#) frameworks on [Mac OS X](#), written in [Objective C](#), an object-oriented, dynamic messaging extension to C based on Smalltalk.

Object-oriented features were added to many existing languages during that time, including [Ada](#), [BASIC](#), [Lisp](#), [Pascal](#), and others. Adding these features to languages that were not initially designed for them often led to problems with compatibility and maintainability of code. "Pure" object-oriented languages, on the other hand, lacked features that many programmers had come to depend upon. To bridge this gap, many attempts have been made to create new languages based on object-oriented methods but allowing some procedural features in "safe" ways. Bertrand Meyer's [Eiffel](#) was an early and moderately successful language with those goals.

In the past decade [Java](#) has emerged in wide use partially because of its similarity to the [C language] but more importantly because of its implementation using a [virtual machine](#) that theoretically runs code

unchanged on many different platforms, the latter of which makes it the darling of larger development shops with heterogeneous environments.

More recently, a number of languages have emerged that are primarily object-oriented yet compatible with procedural methodology, such as [Python](#) and [Ruby](#). Besides Java, probably the most commercially important recent object-oriented languages are [VB.NET](#) and [C Sharp](#) designed for Microsoft's [.NET](#) platform.

Just as procedural programming led to refinements of technique such as [structured programming](#), modern object-oriented software design methods include refinements such as the use of [design patterns](#), [design by contract](#), and [modelling languages](#) (such as [UML](#)).

Further reading

- Booch, Grady. (1993) [ISBN 0805353402](#) *Object-Oriented Analysis and Design with Applications (Second Edition)*. Addison-Wesley.
- Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. (1995) [ISBN 0201633612](#) *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.
- Meyer, Bertrand. (1997) [ISBN 0136291554](#) *Object-Oriented Software Construction (Second Edition)*. Prentice Hall.
- Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. (1991) [ISBN 0136298419](#) *Object-Oriented Modeling and Design*. Prentice Hall.
- Jacobsen, Ivar. (1994) [ISBN 0201544350](#) *Object-Oriented Software Engineering: A Use Case-Driven Approach*. Addison-Wesley.
- Abelson, Harold, Gerald Jay Sussman with Julie Sussman. (1996) [ISBN 0262011530](#) *Structure and Interpretation of Computer Programs (Second edition)*. The MIT Press

See also:

- [Software component](#)
- [Interface description language](#)
- [class](#)
- [object](#)
- [object-based programming](#)
- [Object-oriented programming language](#)
- [Functional programming](#)
- [Procedural programming](#)
- [Structured programming](#)
- [Post-object programming](#)
- [glossary of object-oriented programming](#)

External link

- [Object-oriented programming FAQ](#)

[Edit this page](#) | [Discuss this page](#) | [Page history](#) | [What links here](#) | [Related changes](#)

Other languages: [Deutsch](#) | [Español](#) | [Français](#) | [Nederlands](#) | [Polski](#) | [Português](#)

[Main Page](#) | [About Wikipedia](#) | [Recent changes](#) |

 Search: 

This page was last modified 18:57, 3 Dec 2003. All text is available under the terms of the [GNU Free Documentation License](#).

