# Modular programming

From Wikipedia, the free encyclopedia

**Modular programming** is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable **modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.

A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface. Modular programming is closely related to structured programming and object-oriented programming, all having the same goal of facilitating construction of large software programs and systems by decomposition into smaller pieces, and all originating around the 1960s. While historically usage of these terms has been inconsistent, today "modular programming" refers to high-level decomposition of the code of an entire program into pieces: structured programming to the low-level code use of structured control flow, and object-oriented programming to the *data* use of objects, a kind of data structure.

In object-oriented programming, the use of interfaces as an architectural pattern to construct modules is known as interface-based programming.

## Contents

## Terminology

The term **assembly** (as in .NET languages like C#, F# or Visual Basic .NET) or **package** (as in Dart, Go or Java) is sometimes used instead of **module**. In other implementations, this is a distinct concept; in Python a package is a collection of modules, while in the upcoming Java 9 the introduction of the new module concept (a collection of packages with enhanced access control) is planned.

Furthermore, the term "package" has other uses in software (for example .NET NuGet packages). A component is a similar concept, but typically refers to a higher level; a component is a piece of a whole system, while a module is a piece of an individual program. The scale of the term "module" varies significantly between languages; in Python it is very small-scale and each file is a module, while in Java 9 it is planned to be large-scale, where a module is a collection of packages, which are in turn collections of files.

Other terms for modules include **unit**, used in Pascal dialects.

## Language support

Languages that formally support the module concept include Ada, Algol, BlitzMax, C#, Clojure, COBOL, D, Dart, eC, Erlang, Elixir, F, F#, Fortran, Go, Haskell, IBM/360 Assembler, IBM i Control Language (CL), IBM RPG, Java,[a] MATLAB, ML, Modula, Modula-2, Modula-3, Morpho, NEWP, Oberon, Oberon-2, Objective-C, OCaml, several derivatives of Pascal (Component Pascal, Object Pascal, Turbo Pascal, UCSD Pascal), Perl, PL/I, PureBasic, Python, Ruby,[2] Rust, JavaScript,[3] Visual Basic .NET and WebDNA.

Conspicuous examples of languages that lack support for modules are C, C++,[4] and Pascal (in its original form). As of 2014, modules have been proposed for C++;[5] modules were added to Objective-C in iOS 7 (2013); and Pascal was superseded by Modula and Oberon, which included modules from the start, and various derivatives that included modules. JavaScript has got native modules since ECMAScript 2015.

Modular programming can be performed even where the programming language lacks explicit syntactic features to support named modules. For example, the IBM System i also uses modules when programming in the Integrated Language Environment (ILE).

## Key aspects

With modular programming, concerns are separated such that modules perform logically discrete functions, interacting through well-defined interfaces. Often modules form a directed acyclic graph (DAG); in this case a cyclic dependency between modules is seen as indicating that these should be a single module. In the case where modules do form a DAG they can be arranged as a hierarchy, where the lowest-level modules are independent, depending on no other modules, and higher-level modules depend on lower-level ones. A particular program or library is a top-level module of its own hierarchy, but can in turn be seen as a lower-level module of a higher-level program, library, or system.

When creating a modular system, instead of creating a monolithic application (where the smallest component is the whole), several smaller modules are written separately so that, when composed together, they construct the executable application program. Typically these are also compiled separately, via separate compilation, and then linked by a linker. A just-in-time compiler may perform some of this construction "on-the-fly" at run time.

This makes modular designed systems, if built correctly, far more reusable than a traditional monolithic design, since all (or many) of these modules may then be reused (without change) in other projects. This also facilitates the "breaking down" of projects into several smaller projects. Theoretically, a modularized software project will be more easily assembled by large teams, since no team members are creating the whole system, or even need to know about the system as a whole. They can focus just on the assigned smaller task (this, it is claimed, counters the key assumption of The Mythical Man Month—making it actually possible to add more developers to a late software project—without making it later still).

## History

Modular programming, in the form of subsystems (particularly for I/O) and software libraries, dates to early software systems, where it was used for code reuse. Modular programming per se, with a goal of modularity, developed in the late 1960s and 1970s, as a larger-scale analog of the concept of structured programming (1960s). The term "modular programming" dates at least to the National Symposium on Modular Programming, organized at the Information and Systems Institute in July 1968 by Larry Constantine; other key concepts were information hiding (1972) and separation of concerns (SoC, 1974).

Modules were not included in the original specification for ALGOL 68 (1968), but were included as extensions in early implementations, ALGOL 68-R (1970) and ALGOL 68C (1970), and later formalized.[6] One of the first languages designed from the start for modular programming was the short-lived Modula (1975), by Niklaus Wirth. Another early modular language was Mesa (1970s), by Xerox PARC, and Wirth drew on Mesa as well as the original Modula in its successor, Modula-2 (1978), which influenced later languages, particularly through its successor, Modula-3 (1980s). Modula's use of dot-qualified names, like `M.a` to refer to object `a` from module `M`, coincides with notation to access a field of a record (and similarly for attributes or methods of objects), and is now widespread, seen in C#, Dart, Go, Java, and Python, among others. Modular programming became widespread from the 1980s: the original Pascal language (1970) did not include modules, but later versions, notably UCSD Pascal (1978) and Turbo Pascal (1983) included them in the form of "units", as did the Pascal-influenced Ada (1980). The Extended Pascal ISO 10206:1990 standard kept closer to Modula2 in its modular support. Standard ML (1984)[7] has one of the most complete module systems, including functors (parameterized modules) to map between modules.

In the 1980s and 1990s modular programming was overshadowed by and often conflated with object-oriented programming, particularly due to the popularity of C++ and Java; this was also seen in the failure of Modula-3, which included modules but not objects. For example, the C family of languages had support for objects and classes in C++ (originally C with Classes, 1980) and Objective-C (1983), only supporting modules 30 years or more later. Java (1995) supports modules in the form of packages, though the primary unit of code organization is a class. However, Python (1991) prominently used both modules and objects from the start, using modules as the primary unit of code organization and "packages" as a larger-scale unit; and Perl 5 (1994) includes support for both modules and objects, with a vast array of modules being available from CPAN (1993).

Modular programming is now widespread, and found in virtually all major languages developed since the 1990s. The relative importance of modules varies between languages, and in class-based object-oriented languages there is still overlap and confusion with classes as a unit of organization and encapsulation, but these are both well-established as distinct concepts.

## See also

- Architecture description language
- Cohesion (computer science)
- Component-based software engineering
- Constructionist design methodology, a methodology for creating modular, broad Artificial Intelligence systems
- Conway's law
- Coupling (computer science)
- David Parnas

- Information hiding (encapsulation)
- Library (computing)
- List of system quality attributes
- Plug-in (computing)
- Snippet (programming)
- Structured Design
- Structured programming

## Notes

a. The term "package" is used for the analog of modules in the JLS;[1] — see Java package. "Modules", a kind of collection of packages, are planned for Java 9 as part of Project Jigsaw (http://openjdk.java.net/projects/jigsaw/); these were earlier called "superpackages" and planned for Java 7.

## References

1. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, *The Java Language Specification, Third Edition*, ISBN 0-321-24678-0, 2005. In the Introduction, it is stated "Chapter 7 describes the structure of a program, which is organized into packages similar to the modules of Modula." The word "module" has no special meaning in Java.
2. [1] (http://ruby-doc.org/core-2.0/Module.html)
3. ECMAScript® 2015 Language Specification, 15.2 Modules (http://www.ecma-international.org/ecma-262/6.0/#sec-modules)
4. C and C++ do, however, allow separate compilation and declarative interfaces to be specified using header files
5. N4047: A Module System for C++ (http://isocpp.org/blog/2014/05/n4047) by Gabriel Dos Reis, Mark Hall, Gor Nishanov
6. Lindsey, Charles H. (Feb 1976). "Proposal for a Modules Facility in ALGOL 68" (PDF). *ALGOL Bulletin* (39): 20–29.
7. David MacQueen (August 1984). "Modules for Standard ML, LFP '84 Proceedings of the 1984 ACM Symposium on LISP and functional programming": 198–207.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Modular_programming&oldid=771477202"

Categories: Programming paradigms | Modularity