



WIKIPEDIA  
The Free Encyclopedia

WIKIPEDIA

# Object (computer science)

**This is an old revision of this page, as edited by I287 (talk | contribs) at 00:31, 21 September 2022. The present address (URL) is a permanent link to this revision, which may differ significantly from the current revision.**

In computer science, an **object** can be a variable, a data structure, a function, or a method. As regions of memory, they contain value and are referenced by identifiers.

In the object-oriented programming paradigm, an *object* can be a combination of variables, functions, and data structures; in particular in class-based variations of the paradigm it refers to a particular instance of a class.

In the relational model of database management, an object can be a table or column, or an association between data and a database entity (such as relating a person's age to a specific person).<sup>[1]</sup>

## Object-based languages

An important distinction in programming languages is the difference between an object-oriented language and an object-based language. A language is usually considered object-based if it includes the basic capabilities for an object: identity, properties, and attributes. A language is considered object-oriented if it is object-based and also has the capability of polymorphism, inheritance, encapsulation, and, possibly, composition. Polymorphism refers to the ability to overload the name of a function with multiple behaviors based on which object(s) are passed to it. Conventional message passing discriminates only on the first object and considers that to be "sending a message" to that object. However, some object-oriented programming languages such as Flavors and the Common Lisp Object System (CLOS) enable discriminating on more than the first parameter of the function.<sup>[2]</sup> Inheritance is the ability to subclass an object class, to create a new class that is a subclass of an existing one and inherits all the data constraints and behaviors of its parents but also adds new and/or changes one or more of them.<sup>[3][4]</sup>

## Object-oriented programming

Object-oriented programming is an approach to designing modular reusable software systems. The object-oriented approach is an evolution of good design practices that go back to the very

beginning of computer programming. Object-orientation is simply the logical extension of older techniques such as structured programming and abstract data types. An object is an abstract data type with the addition of polymorphism and inheritance.

Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object". An object has state (data) and behavior (code). Objects can correspond to things found in the real world. So for example, a graphics program will have objects such as *circle*, *square*, *menu*. An online shopping system will have objects such as *shopping cart*, *customer*, *product*. The shopping system will support behaviors such as *place order*, *make payment*, and *offer discount*. The objects are designed as class hierarchies. So for example with the shopping system there might be high level classes such as *electronics product*, *kitchen product*, and *book*. There may be further refinements for example under *electronic products*: *CD Player*, *DVD player*, etc. These classes and subclasses correspond to sets and subsets in mathematical logic.<sup>[5][6]</sup>

## Specialized objects

---

An important concept for objects is the design pattern. A design pattern provides a reusable template to address a common problem. The following object descriptions are examples of some of the most common design patterns for objects.<sup>[7]</sup>

- Function object: an object with a single method (in C++, this method would be the function operator, "operator()") that acts much like a function (like a C/C++ pointer to a function).
- Immutable object: an object set up with a fixed state at creation time and which does not change afterward.
- First-class object: an object that can be used without restriction.
- Container object: an object that can contain other objects.
- Factory object: an object whose purpose is to create other objects.
- Metaobject: an object from which other objects can be created (compare with a class, which is not necessarily an object).
- Prototype object: a specialized metaobject from which other objects can be created by copying
- God object: an object that knows or does too much (it is an example of an anti-pattern).
- Singleton object: an object that is the only instance of its class during the lifetime of the program.
- Filter object: an object that receives a stream of data as its input and transforms it into the object's output. Often the input and output streams are streams of characters, but these also may be streams of arbitrary objects. These are generally used in wrappers since they conceal the existing implementation with the abstraction required at the developer side.

## Distributed objects

---

The object-oriented approach is not just a programming model. It can be used equally well as an interface definition language for distributed systems. The objects in a distributed computing

model tend to be larger grained, longer lasting, and more service-oriented than programming objects.

A standard method to package distributed objects is via an Interface Definition Language (IDL). An IDL shields the client of all of the details of the distributed server object. Details such as which computer the object resides on, what programming language it uses, what operating system, and other platform-specific issues. The IDL is also usually part of a distributed environment that provides services such as transactions and persistence to all objects in a uniform manner. Two of the most popular standards for distributed objects are the Object Management Group's CORBA standard and Microsoft's DCOM.<sup>[8]</sup>

In addition to distributed objects, a number of other extensions to the basic concept of an object have been proposed to enable distributed computing:

- *Protocol objects* are components of a protocol stack that enclose network communication within an object-oriented interface.
- *Replicated objects* are groups of distributed objects (called *replicas*) that run a distributed multi-party protocol to achieve high consistency between their internal states, and that respond to requests in a coordinated way. Examples include fault-tolerant CORBA objects.
- *Live distributed objects* (or simply *live objects*)<sup>[9]</sup> generalize the *replicated object* concept to groups of replicas that might internally use any distributed protocol, perhaps resulting in only a weak consistency between their local states.

Some of these extensions, such as *distributed objects* and *protocol objects*, are domain-specific terms for special types of "ordinary" objects used in a certain context (such as remote method invocation or protocol composition). Others, such as *replicated objects* and *live distributed objects*, are more non-standard, in that they abandon the usual case that an object resides in a single location at a time, and apply the concept to groups of entities (replicas) that might span across multiple locations, might have only weakly consistent state, and whose membership might dynamically change.

## The Semantic Web

---

The Semantic Web is essentially a distributed-objects framework. Two key technologies in the Semantic Web are the Web Ontology Language (OWL) and the Resource Description Framework (RDF). RDF provides the capability to define basic objects—names, properties, attributes, relations—that are accessible via the Internet. OWL adds a richer object model, based on set theory, that provides additional modeling capabilities such as multiple inheritance.

OWL objects are not like standard large-grained distributed objects accessed via an Interface Definition Language. Such an approach would not be appropriate for the Internet because the Internet is constantly evolving and standardization on one set of interfaces is difficult to achieve. OWL objects tend to be similar to the kinds of objects used to define application domain models in programming languages such as Java and C++.

However, there are important distinctions between OWL objects and traditional object-oriented programming objects. Traditional objects get compiled into static hierarchies usually with single inheritance, but OWL objects are dynamic. An OWL object can change its structure at run time and can become an instance of new or different classes.

Another critical difference is the way the model treats information that is currently not in the system. Programming objects and most database systems use the "closed-world assumption". If a fact is not known to the system that fact is assumed to be false. Semantic Web objects use the open-world assumption, a statement is only considered false if there is actual relevant information that it is false, otherwise it is assumed to be unknown, neither true nor false.

OWL objects are actually most like objects in artificial intelligence frame languages such as KL-ONE and Loom.

The following table contrasts traditional objects from Object-Oriented programming languages such as Java or C++ with Semantic Web Objects:<sup>[10][11]</sup>

| OOP Objects                                                                                                 | Semantic Web Objects                                                                                                    |
|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Classes are regarded as types for instances.                                                                | Classes are regarded as sets of individuals.                                                                            |
| Instances can not change their type at runtime.                                                             | Class membership may change at runtime.                                                                                 |
| The list of classes is fully known at compile-time and cannot change after that.                            | Classes can be created and changed at runtime.                                                                          |
| Compilers are used at build-time. Compile-time errors indicate problems.                                    | Reasoners can be used for classification and consistency checking at runtime or build-time.                             |
| Classes encode much of their meaning and behavior through imperative functions and methods.                 | Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.                     |
| Instances are anonymous insofar that they cannot easily be addressed from outside of an executing program.  | All named RDF and OWL resources have a unique URI under which they can be referenced.                                   |
| Closed world: If there is not enough information to prove a statement true, then it is assumed to be false. | Open world: If there is not enough information to prove a statement true, then it may be true or false. <sup>[12]</sup> |

## See also

---

- Object lifetime
- Object copy
- Design pattern (computer science)
- Business object (computer science)
- Actor model

## References

---

- Oppel, Andy (2005). *SQL Demystified*. McGraw Hill. p. 7. ISBN 0-07-226224-9.

2. Gabriel, Richard; Linda G. DeMichie (1987). "The Common Lisp Object System: An Overview" (<http://www.dreamsongs.com/NewFiles/ECOOP.pdf>) (PDF). *Lucid Inc. Technical Report*. Lecture Notes in Computer Science. **276**: 151. Bibcode:1987LNCS..276..151D (<https://ui.adsabs.harvard.edu/abs/1987LNCS..276..151D>). doi:10.1007/3-540-47891-4\_15 ([https://doi.org/10.1007%2F3-540-47891-4\\_15](https://doi.org/10.1007%2F3-540-47891-4_15)). ISBN 978-3-540-18353-2.
3. Wegner, Peter (December 1987). Meyrowitz, Norman (ed.). "Dimensions of Object-Based Language Design". *OOPSLA'87 Conference Proceedings*. **22** (12): 168–182. doi:10.1145/38807.38823 (<https://doi.org/10.1145%2F38807.38823>).
4. Barbey, S; M. Kempe; A. Strohmeier (1993). "Object-Oriented Programming with Ada 9X" (<http://www.adahome.com/9X/OOP-Ada9X.html>). *Draft Technical Report*. Swiss Federal Institute of Technology in Lausanne Software Engineering Laboratory. Retrieved 15 December 2013. "Ada 83 itself is generally not considered to be object-oriented; rather, according to the terminology of Wegner [Weg 87], it is said to be object-based, since it provides only a restricted form of inheritance and it lacks polymorphism."
5. Jacobsen, Ivar; Magnus Christerson; Patrik Jonsson; Gunnar Overgaard (1992). *Object Oriented Software Engineering* (<https://archive.org/details/objectorientedso00jaco>). Addison-Wesley ACM Press. ISBN 0-201-54435-0.
6. "What is an Object?" (<http://docs.oracle.com/javase/tutorial/java/concepts/object.html>). *oracle.com*. Oracle Corporation. Retrieved 13 December 2013.
7. Martin, Robert C. "Design Principles and Design Patterns" ([https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles\\_and\\_Patterns.pdf](https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf)) (PDF). Archived from the original ([http://www.objectmentor.com/resources/articles/Principles\\_and\\_Patterns.pdf](http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf)) (PDF) on September 6, 2015. Retrieved 28 April 2017.
8. Orfali, Robert (1996). *The Essential Client/Server Survival Guide* (<https://archive.org/details/essentialclients00orfa/page/399>). New York: Wiley Computer Publishing. pp. 399–403 (<https://archive.org/details/essentialclients00orfa/page/399>). ISBN 0-471-15325-7.
9. Ostrowski, K., Birman, K., Dolev, D., and Ahnn, J. (2008). "Programming with Live Distributed Objects", *Proceedings of the 22nd European Conference on Object-Oriented Programming*, Paphos, Cyprus, July 07–11, 2008, J. Vitek, Ed., *Lecture Notes in Computer Science*, vol. 5142, Springer-Verlag, Berlin, Heidelberg, 463-489, <http://portal.acm.org/citation.cfm?id=1428508.1428536>.
10. Berners-Lee, Tim; James Hendler; Ora Lassila (May 17, 2001). "The Semantic Web A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities" (<https://web.archive.org/web/20130424071228/http://www.cs.umd.edu/~golbeck/LBSC690/SemanticWeb.html>). *Scientific American*. **284**: 34–43. doi:10.1038/scientificamerican0501-34 (<https://doi.org/10.1038%2Fscientificamerican0501-34>). Archived from the original (<http://www.cs.umd.edu/~golbeck/LBSC690/SemanticWeb.html>) on April 24, 2013.
11. Knublauch, Holger; Oberle, Daniel; Tetlow, Phil; Wallace, Evan (2006-03-09). "A Semantic Web Primer for Object-Oriented Software Developers" (<http://www.w3.org/2001/sw/BestPractices/SE/ODSD/>). *W3C*. Retrieved 2008-07-30.
12. Table excerpted from tables in: <http://www.w3.org/2001/sw/BestPractices/SE/ODSD/>

## External links

---

- *What Is an Object?* (<http://java.sun.com/docs/books/tutorial/java/concepts/object.html>) from *The Java Tutorials*

---

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Object\\_\(computer\\_science\)&oldid=1111438032](https://en.wikipedia.org/w/index.php?title=Object_(computer_science)&oldid=1111438032)"