Object-oriented terminology is fraught with ambiguity and contradictions. Different programming languages might use the same term to mean different things. Furthermore, a programmer might use object-oriented terms from language A to describe features in language B, even if language B already uses some of those terms to mean something different.

The definitions below are meant to be as language-agnostic as possible. These definitions aren’t the only ones that a person might use for object-oriented concepts. Experts can, should, and do differ on the definition of these terms. However, the definitions below are ones that many people would agree on, and they have been tailored to meet the needs of CS 42.

**Terms that aren’t unique to object-oriented programming**

- **software**: data + behavior, expressed in code
- **syntax**: what software looks like (i.e., what the programmer writes)
- **semantics**: what software means (i.e., what happens when a program runs)
- **interface**: what a piece of software can do
- **implementation**: how a piece of software does it

**Terms we use to talk about large programs**

- **modularity**: the ability to use software in a new context (e.g., as part of a larger program), without knowing how it works
- **extensibility**: the ability to modify software (typically to add data/behavior)
- **reusability**: an umbrella term for modularity / extensibility
- **component**: a reusable piece of software

**Object-oriented terms**

- **object**: a self-referential component
  
  So, an object:
  
  - combines data and behavior
  - can be used without knowing how it works
  - can be extended, to add new data / behavior
  - knows about itself, so that it can use its own data / behavior

**Objects as modular: interface vs implementation**

- **type**: a description of an object’s interface
  
  *In Java, a type is most like an interface or an abstract class.*

- **class**: a description of an object’s implementation

- **encapsulation**: the conventions that programmers use to increase modularity, by keeping an object’s interface distinct from its implementation.

  *Some programming languages check whether the conventions have been followed; others do not.*

  *Python programmers use an underscore (_) and the language does not enforce it. Java programmers use public / private, and the language “enforces” it.*
Objects as software: data and behavior

**data member** a piece of data (e.g., part of the state of an object)
*Java programmers also call it an “instance variable.”*

**property** a description of a data member’s interface (e.g., the name used to access data)

**field** a description of a data member’s implementation (e.g., the way in which data is stored)
*Many programmers don’t make a distinction between the interface and the implementation of a data member.*

**method** a behavior
*C++ programmers call it a “member function.”*

**method signature** a description of a behavior’s interface: its name, inputs, and outputs

**method body** a description of a behavior’s implementation

**member** either a data member or a method

**constructor** a way to initialize an instance of a class
*Usually, the constructor is described as if it were a method, where the method body initializes the object’s fields.*

Objects as extensible

**subtype** a type that extends the interface of another type (its supertype)

**subclass** a class that extends the implementation of another class (its superclass)

**method overloading** extending an interface by adding a behavior with the same name (but different parameters) than an existing method

**method overriding** extending an implementation by modifying an existing behavior

**polymorphism** a generic word that can refer to subtyping—where a more detailed and specific interface can substituted for a more general one (in which case, it’s sometimes called *subtype polymorphism*), or to method overloading (in which case, it’s sometimes called *ad hoc polymorphism*), or to higher-order types (in which case it’s sometimes called *parametric polymorphism* or *generics*)

**inheritance** extending the implementation of a self-referential component
*In many (but not all) object-oriented languages, if B "inherits" from A, then B can be both a subclass and a subtype of A. In other words, inheritance can be used to extend an interface, an implementation, or both an interface and an implementation.*