Many of the concepts used in object-oriented programming simply extend concepts already familiar to most programmers. Nevertheless, the amount of new terminology encountered by a programmer new to object-oriented programming may be daunting. This section attempts to clarify the meaning of new terms you may have encountered in this guide. All of these terms are also defined

Action message The message that a user interface control object—such as a Button or Slider—sends to its target. An action method is implemented by the target object specifically to respond to this message as the user manipulates the control.

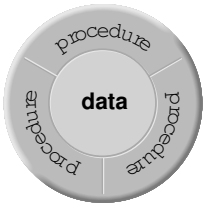
Archiving A general-purpose Objective C storage feature that allows objects to be written to and retrieved from files. Interface Builder uses archiving to save and retrieve the objects in a user interface. In addition to the objects, the user interface archive includes attributes and connections for those objects. Another use of archiving is the simple data storage and retrieval mechanism implemented by YourCall.

Bundles A NeXTSTEP feature for application resource file management. Bundles provide an object-oriented way to access an application's resource files, including the user interface archive, images, and sounds. Bundles also enable multiple-languages within a single application, with the user interface text for each language contained in its own bundle. Bundles free NeXTSTEP applications from dependence on a particular file system.

Class A template for a particular type of object. The class defines both procedures and data for a particular object type. Much as you can define types of data structures in procedural programming, you define classes of objects in object-oriented programming.

Data encapsulation The feature of object-oriented programming that allows access to an object's

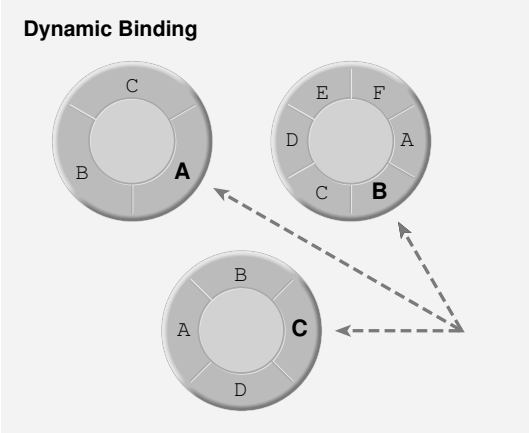
data only through its procedures. Thus an object's data is effectively encapsulated by its procedures.



Database model The representation of a particular database's entities, attributes, and data used by the Database Kit. You use the DBModeler application to create models based on information stored in the database. When you load the model in Interface Builder, it automatically creates Database Kit objects for accessing data in the database.

Delegate A kind of outlet—specifically, an object that acts on the behalf of another. As the name implies, the delegate shares responsibility with the object it connects to. A number of Application Kit classes use delegates to let you synchronize the custom behavior of your application with standard NeXTSTEP behavior.

Dynamic binding The ability of a program to set both the message and the object receiving that message as it runs. This is particularly important in graphical, user-driven applications, where one user command—say Copy or Paste—may apply to any number of user-interface objects.

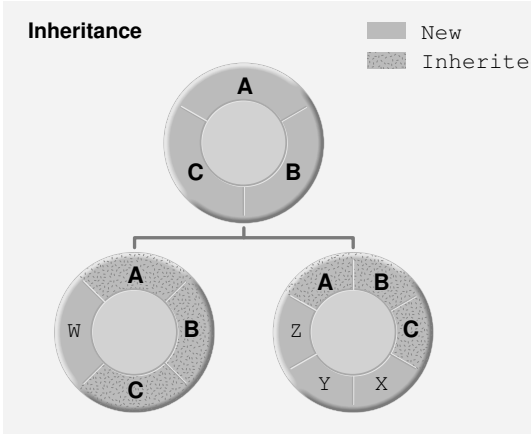


Entity-relationship model A generic database model in which data is organized into fundamental *entities*, each entity defined by its component *attributes*; connections, or *relationships*, are used to link entities. This model applies equally well to relational, flat file, hierarchical, and object-oriented databases. The Database Kit is designed to look at the organization of data as an entity-relationship model.

Event A signal received by a program to indicate a particular action, usually a user-generated action such as a mouse-click or keystroke. In NeXTSTEP, events are sent as messages to user interface objects.

Implementation file The source code file that contains the program code for the class, identified by the class name and a ".m" (for *methods*) extension. The code in the implementation file can include a combination of Objective C, C, and C++ syntax.

Inheritance The object-oriented feature whereby a new class acquires the instance variables and methods of its superclass. Inheritance reduces the amount of code you write and debug to make incremental changes.



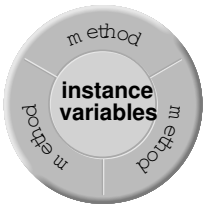
Instance Term describing the relationship of an object to its class. An object belonging to a specific class is referred to as an instance of that class.

Instance variable A data item belonging to an object. A class definition includes a list of instance variables for its objects. Each object has its own set of these instance variables. Only the methods for that object can directly access its instance variables.

Interface file The source code file that declares instance variables and methods for a class, identified by the class name and a ".h" (for *header*) extension. This file is referred to as the interface file in standard C terminology because it presents the public declaration of the class; it's also referred to as the *header* file. A class's interface file is referenced (or *imported*) by other code files that send messages to objects of that class.

Message Code that tells an object to perform one of its methods. Sending a message to an object is analogous to invoking a procedure on a particular data structure in procedural programming.

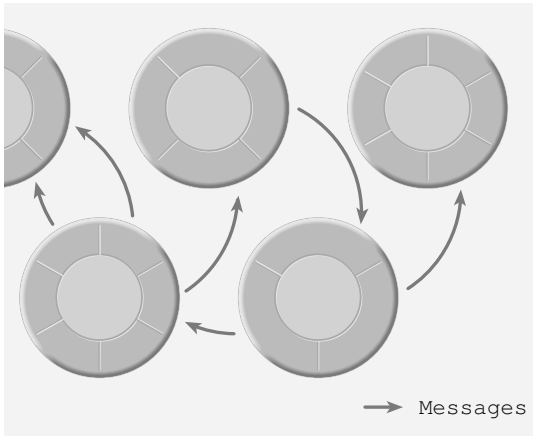
Method A procedure defined for an object by its class. An object's methods implement any functionality, including setting and retrieving the object's instance variables, sending messages to other objects, and so on. A method is invoked by sending a message to an object. An object can be thought of as a set of instance variables encapsulated by methods.



Modularity In object-oriented programming, the ability to divide applications into distinct objects for specific data and specific tasks. Like objects in the physical world, objects in a program have identifying characteristics and behavior. For example, a user interface object such as a button has both an appearance on the screen and a well-defined programmatic response to user action. Modular structure means simpler debugging, since errant behavior can be traced directly to the responsible object.

Object A single programming component, combining both code *and* data. Another way of looking at an object is as a specific instance of a particular class. Much as you create instances of data structures in procedural programming, you create objects in object-oriented programming.

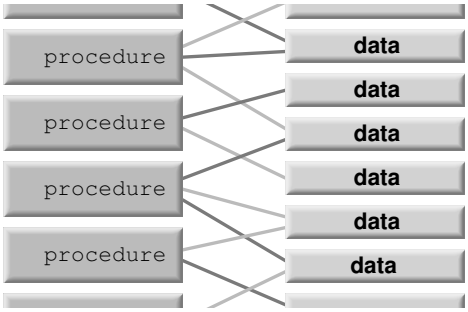
Object-oriented program A network of objects that interact by sending messages to one another. In NeXTSTEP, the basic network of an object-oriented program is defined for you. To create a unique application, you simply create objects that implement your unique behavior and plug them into the existing network.



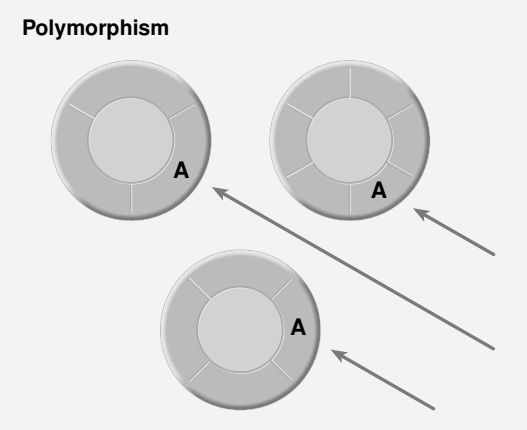
Outlet An instance variable that one object uses to identify another. For example, when you connect one TextField as the nextText outlet of another in Interface Builder, you're actually setting an instance variable named nextText. When the user presses the Tab key in one field, that field sends a standard message to its nextText object, telling that object to begin editing. Similarly, when an object you design needs to communicate with another object, you provide the connection by declaring an outlet instance variable.

Overriding In a subclass, changing the behavior of an inherited method by supplying new code for that method. When overriding a method, the subclass can either implement entirely new behavior, or it can keep the original behavior and expand on it.

Procedural programs Programs made up of two fundamental components: *data* and *code*. The data represents what the user needs to manipulate, while the code does the manipulation. To improve project management and maintenance, code is compartmentalized into *procedures*. However, most data is global, and each procedure may manipulate any part of the data directly.



Polymorphism The ability of different classes of objects to respond to the same message in their own ways. Polymorphism effectively increases program flexibility while maintaining code simplicity.



Protocol An Objective C mechanism for declaring methods outside the context of a particular class. Protocols are used to conceal the class implementing a particular method from others; for example, in distributed objects a server application publishes a protocol to describe methods that its vended object responds to. Protocols are also used to specify methods that must be implemented by a particular kind of object; for example, to work with the NeXTSTEP Text object, an object that performs spell checking must conform to NeXTSTEP's spell checking protocols.

Proxy In the Distributed Objects system, an object created and used by the client application as a stand-in for an object in the server.

Subclass A class created from another class. To createthe new class, you start with a class whose behavior is closest to that which you want to implement, then create the subclass by adding new methods and instance variables. The new class is the *subclass* of the original; the original class is the *superclass* of the new. A new class *inherits* all the methods and instance variables defined for the superclass.

Superclass (see *Subclass*)

Target A special kind of outlet used by NeXTSTEP control objects—Buttons, Sliders, TextFields, MenuCells, and so on—to identify the objects that they send action messages to as the user manipulates them.

Typed stream A NeXTSTEP data buffer type that includes information about the data types stored within it. A typed stream can be copied in memory and written to or read from a file using NeXTSTEP's typed stream functions. The object archiving facility takes advantage of typed streams to store and retrieve objects. When storing a set of objects, a typed stream is opened, each object is sent a write: message to write its instance variables to the typed stream, then the stream is written to a file and closed. To retrieve, a typed stream is opened on a file, each object in the file is sent a read: message to read its instance variables from the stream, then the stream is closed.

Unparsing The process for creating template source files from a class specification created in Interface Builder.