[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html" \l "header)

**Chapter**

**6**

**Object Modeling**

* **6.1**[Object-Oriented Analysis](javascript://)
* **6.2**[Objects](javascript://)
* **6.3**[Attributes](javascript://)
* **6.4**[Methods](javascript://)
* **6.5**[Messages](javascript://)
* **6.6**[Classes](javascript://)
* **6.7**[Relationships Among Objects and Classes](javascript://)
* **6.8**[The Unified Modeling Language (UML)](javascript://)
  + **6.8.1**[Use Case Modeling](javascript://)
  + **6.8.2**[Use Case Diagrams](javascript://)
  + **6.8.3**[Class Diagrams](javascript://)
  + **6.8.4**[Sequence Diagrams](javascript://)
  + **6.8.5**[State Transition Diagrams](javascript://)
  + **6.8.6**[Activity Diagrams](javascript://)
  + **6.8.7**[Business Process Modeling](javascript://)
* **6.9**[Tools](javascript://)
* **6.10**[Summary](javascript://)
* [Chapter Review](javascript://)
  + [Key Terms](javascript://)

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# 6.1Object-Oriented Analysis

As stated in [Chapter 1](javascript://), the most popular systems development options are structured analysis, object-oriented (O-O) analysis, and agile methods. The table in [Figure 1-17](javascript://) shows the three alternatives and describes some pros and cons for each approach. As the table indicates, O-O methodology is popular because it integrates easily with O-O programming languages such as C++, Java, and Python. Programmers also like O-O code because it is modular, reusable, and easy to maintain.

**Object-oriented (O-O) analysis** describes an information system by identifying things called objects. An [**object**](javascript://) represents a real person, place, event, or transaction. For example, when a patient makes an appointment to see a doctor, the patient is an object, the doctor is an object, and the appointment itself is an object.

O-O analysis is a popular approach that sees a system from the viewpoint of the objects themselves as they function and interact. The end product of O-O analysis is an [**object model**](javascript://), which represents the information system in terms of objects and O-O concepts.

[Chapter 4](javascript://) stated that the **Unified Modeling Language (UML)** is a widely used method of visualizing and documenting an information system. In this chapter, the UML is used to develop object models. The first step is to understand basic O-O terms, including objects, attributes, methods, messages, and classes. This chapter shows how systems analysts use those terms to describe an information system.

**Case in Point 6.1**

### TravelBiz

* TravelBiz is a nationwide travel agency that specializes in business travel. It has decided to expand into the vacation travel market by launching a new business division called TravelFun. The IT director assigned two systems analysts to create a flexible and an efficient information system for the new division. One analyst wants to use traditional analysis and modeling techniques for the project, while the other analyst wants to use an O-O methodology. Which approach would you suggest and why?

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**6.2**Objects

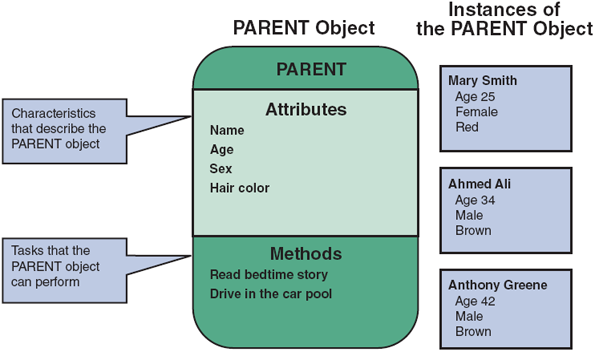
An object represents a person, a place, an event, or a transaction that is significant to the information system. In [Chapter 5](javascript://), DFDs were created that treated data and processes separately. An object, however, includes data *and* the processes that affect that data. For example, a customer object has a name, an address, an account number, and a current balance. Customer objects also can perform specific tasks, such as placing an order, paying a bill, and changing their address.

Consider a simplistic example of how the UML might describe a family with parents and children. UML represents an object as a rectangle with the object name at the top, followed by the object’s attributes and methods.

[Figure 6-1](javascript://) shows a PARENT object with certain attributes such as name, age, sex, and hair color. If there are two parents, then there are two instances of the PARENT object. The PARENT object can perform methods, such as reading a bedtime story, driving the carpool van, or preparing a school lunch. When a PARENT object receives a message, it performs an action, or method.

**Figure 6-1**

The PARENT object has four attributes and two methods. Mary Smith, Ahmed Ali, and Anthony Greene are instances of the PARENT object.



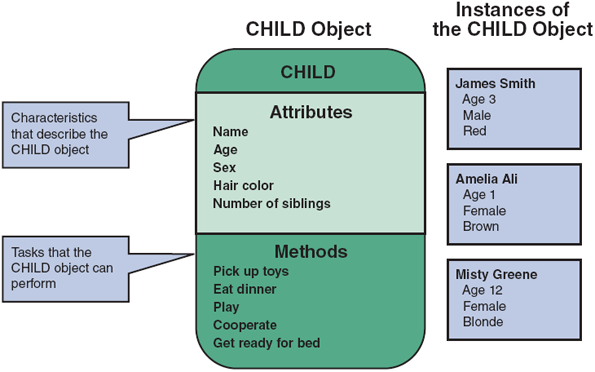
Enlarge Image

For example, the message GOOD NIGHT from a child might tell the PARENT object to read a bedtime story, while the message DRIVE from another parent signals that it is the PARENT object’s turn to drive in the carpool.

Continuing with the family example, the CHILD object in [Figure 6-2](javascript://) possesses the same attributes as the PARENT object and an additional attribute that shows the number of siblings. A CHILD object performs certain methods, such as picking up toys, eating dinner, playing, cooperating, and getting ready for bed. To signal the CHILD object to perform those tasks, a parent can send certain messages that the CHILD object will understand. For example, the DINNER’S READY message tells a CHILD object to come to the table, while the SHARE WITH YOUR BROTHER/SISTER message tells a CHILD object to cooperate with other CHILD objects.

**Figure 6-2**

The CHILD object has five attributes and five methods. James Smith, Amelia Ali, and Misty Greene are instances of the CHILD object.



Enlarge Image

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**6.3**Attributes

An object has certain [**attributes**](javascript://), which are characteristics that describe the object. If objects are similar to nouns, attributes are similar to adjectives that describe the characteristics of an object. For example, a car has attributes such as make, model, and color. Some objects might have a few attributes; others might have dozens.

Systems analysts define an object’s attributes during the systems design process. In an O-O system, objects can inherit, or acquire, certain attributes from other objects.

Objects can have a specific attribute called a [**state**](javascript://). The state of an object is an adjective that describes the object’s current status. For example, depending on the state, a student can be a future student, a current student, or a past student. Similarly, a bank account can be active, inactive, closed, or frozen.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

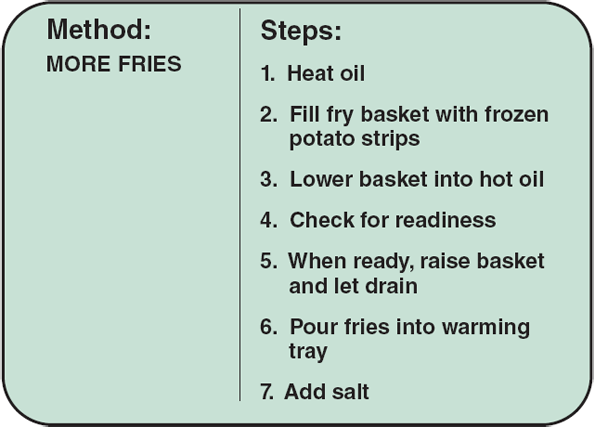
**6.4**Methods

An object also has **methods**, which are tasks or functions that the object performs when it receives a [**message**](javascript://), or command, to do so. For example, a car performs a method called OPERATE WIPERS when it is sent a message with the wiper control, and it can APPLY BRAKES when it is sent a message by pressing the brake pedal. A method defines specific tasks that an object can perform. Just as objects are similar to nouns and attributes are similar to adjectives, methods resemble verbs that describe *what* and *how* an object does something.

Consider a server who prepares fries in a fast-food restaurant. A systems analyst might describe the operation as a method called MORE FRIES, as shown in [Figure 6-3](javascript://). The MORE FRIES method includes the steps required to heat the oil, fill the fry basket with frozen potato strips, lower it into the hot oil, check for readiness, remove the basket when ready and drain the oil, pour the fries into a warming tray, and add salt.

**Figure 6-3**

The MORE FRIES method requires the server to perform seven specific steps.



Enlarge Image

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

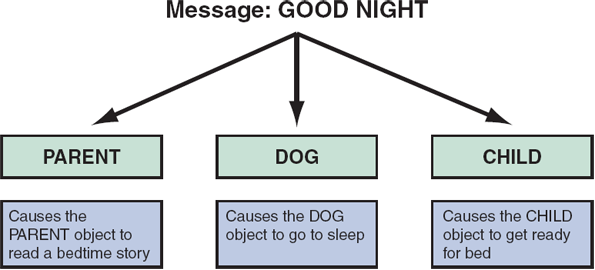
**6.5**Messages

A message is a command that tells an object to perform a certain method. For example, the message PICK UP TOYS directs the CHILD class to perform all the necessary steps to pick up the toys. The CHILD class understands the message and executes the method.

The same message to two different objects can produce different results. The concept that a message gives different meanings to different objects is called [**polymorphism**](javascript://). For example, in [Figure 6-4](javascript://), the message GOOD NIGHT signals the PARENT object to read a bedtime story, but the same message to the CHILD object signals it to get ready for bed. If the family also had a DOG object, the same message would tell the dog to sleep.

**Figure 6-4**

In an example of polymorphism, the message GOOD NIGHT produces different results, depending on which object receives it.



Enlarge Image

An object can be viewed as a **black box**, because a message to the object triggers changes within the object without specifying how the changes must be carried out. A gas pump is an example of a black box. When the economy grade is selected at a pump, it is not necessary to think about how the pump determines the correct price and selects the right fuel, as long as it does so properly.

The black box concept is an example of [**encapsulation**](javascript://), which means that all data and methods are self-contained. A black box does not want or need outside interference. By limiting access to internal processes, an object prevents its internal code from being altered by another object or process. Encapsulation allows objects to be used as modular components anywhere in the system, because objects send and receive messages but do not alter the internal methods of other objects.

O-O designs typically are implemented with O-O programming languages. A major advantage of O-O designs is that systems analysts can save time and avoid errors by using modular objects, and programmers can translate the designs into code, working with reusable program modules that have been tested and verified. For example, in [Figure 6-5](javascript://), an INSTRUCTOR object sends an ENTER GRADE message to an instance of the STUDENT RECORD class. Note that the INSTRUCTOR object and STUDENT RECORD class could be reused, with minor modifications, in other school information systems where many of the attributes and methods would be similar.

**Figure 6-5**

In a school information system, an INSTRUCTOR object sends an ENTER GRADE message to an instance of the STUDENT RECORD class.



Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

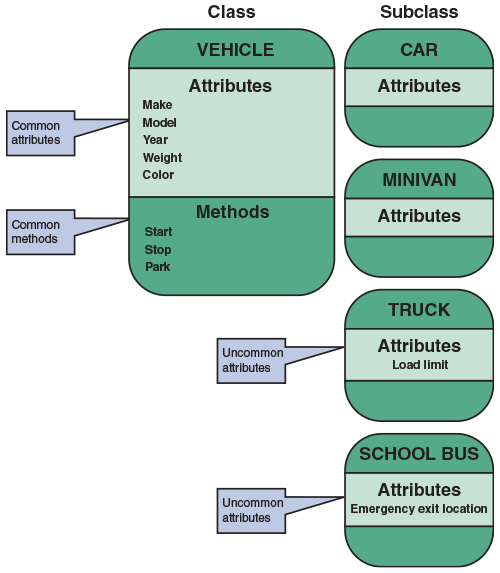
**6.6**Classes

An object belongs to a group or category called a **class**. For example, Ford Fiestas belong to a class called CAR. An [**instance**](javascript://) is a specific member of a class. A Toyota Camry, for example, is an instance of the CAR class. At an auto dealership, many instances of the CAR class may be observed: the TRUCK class, the MINIVAN class, and the SPORT UTILITY VEHICLE class, among others.

All objects within a class share common attributes and methods, so a class is like a blueprint or template for all the objects within the class. Objects within a class can be grouped into [**subclasses**](javascript://), which are more specific categories within a class. For example, TRUCK objects represent a subclass within the VEHICLE class, along with other subclasses called CAR, MINIVAN, and SCHOOL BUS, as shown in [Figure 6-6](javascript://). Note that all four subclasses share common traits of the VEHICLE class, such as make, model, year, weight, and color. Each subclass also can possess traits that are uncommon, such as a load limit for the TRUCK or an emergency exit location for the SCHOOL BUS.

**Figure 6-6**

The VEHICLE class includes common attributes and methods. CAR, TRUCK, MINIVAN, and SCHOOL BUS are instances of the VEHICLE class.



A class can belong to a more general category called a [**superclass**](javascript://). For example, a NOVEL class belongs to a superclass called BOOK, because all novels are books in this example. The NOVEL class can have subclasses called HARDCOVER, PAPERBACK, and DIGITAL.

Similarly, consider a fitness center illustrated in [Figure 6-7](javascript://) that might have students, instructors, class schedules, and a registration process. As shown in [Figure 6-8](javascript://), the EMPLOYEE class belongs to the PERSON superclass, because every employee is a person, and the INSTRUCTOR class is a subclass of EMPLOYEE.

**Figure 6-7**

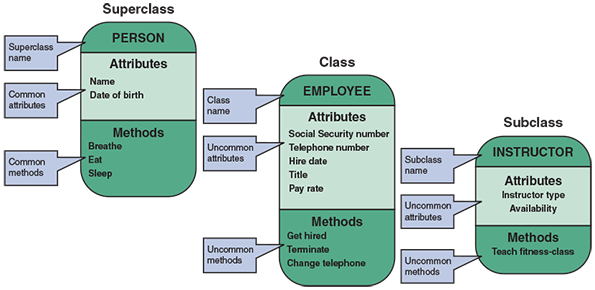
A typical fitness center might have students, instructors, class schedules, and a registration process.



StockLite/ [Shutterstock.com](http://shutterstock.com/" \t "_blank)

**Figure 6-8**

At the fitness center, the PERSON subclass includes common attributes and methods. EMPLOYEE is a class within the PERSON superclass. INSTRUCTOR is a subclass within the EMPLOYEE class.



Enlarge Image

Change font size

[**help**](javascript://)

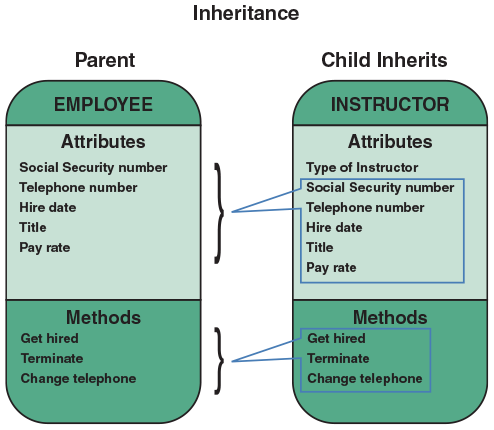
[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**6.7**Relationships Among Objects and Classes

[**Relationships**](javascript://) enable objects to communicate and interact as they perform business functions and transactions required by the system. Relationships describe what objects need to know about each other, how objects respond to changes in other objects, and the effects of membership in classes, superclasses, and subclasses. Some relationships are stronger than others (just as a relationship between family members is stronger than one between casual acquaintances). The strongest relationship is called inheritance. [**Inheritance**](javascript://) enables an object, called a [**child**](javascript://), to derive one or more of its attributes from another object, called a [**parent**](javascript://). In the example in [Figure 6-9](javascript://), the INSTRUCTOR object (child) inherits many traits from the EMPLOYEE object (parent), including SOCIAL SECURITY NUMBER, TELEPHONE NUMBER, and HIRE DATE. The INSTRUCTOR object also can possess additional attributes, such as TYPE OF INSTRUCTOR. Because all employees share certain attributes, those attributes are assumed through inheritance and do not need to be repeated in the INSTRUCTOR object.

**Figure 6-9**

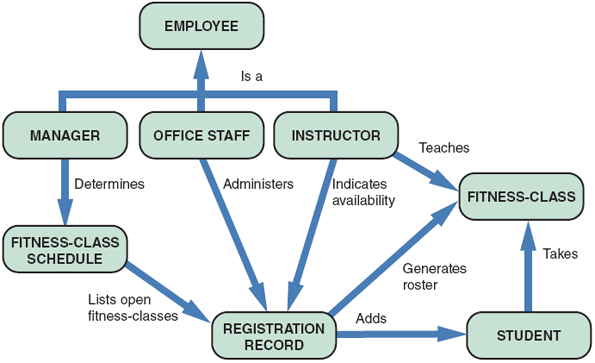
An inheritance relationship exists between the INSTRUCTOR and EMPLOYEE objects. The INSTRUCTOR (child) object inherits characteristics from the EMPLOYEE (parent) class, and can have additional attributes of its own.



After objects, classes, and relationships have been identified, an object relationship diagram can be prepared to provide an overview of the system. That model is used as a guide to continue to develop additional diagrams and documentation. [Figure 6-10](javascript://) shows an object relationship diagram for a fitness center. Note that the model shows the objects and how they interact to perform business functions and transactions.

**Figure 6-10**

An object relationship diagram for a fitness center.



Enlarge Image

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**6.8**The Unified Modeling Language (UML)

Just as structured analysis uses DFDs to model data and processes, systems analysts use UML to describe O-O systems. [Chapter 4](javascript://) explained that UML is a popular technique for documenting and modeling a system. UML uses a set of symbols to represent graphically the various components and relationships within a system. Although the UML can be used for business process modeling and requirements modeling, it is mainly used to support O-O system analysis and to develop object models.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.1Use Case Modeling

A [**use case**](javascript://) represents the steps in a specific business function or process. An external entity, called an **actor**, initiates a use case by requesting the system to perform a function or process. For example, in a medical office system, a PATIENT (actor) can MAKE APPOINTMENT (use case), as shown in [Figure 6-11](javascript://).

**Figure 6-11**

In a medical office system, a PATIENT (actor) can MAKE APPOINTMENT (use case). The UML symbol for a use case is an oval. The actor is shown as a stick figure.

Note that the UML symbol for a use case is an oval with a label that describes the action or event. The actor is shown as a stick figure, with a label that identifies the actor’s role. The line from the actor to the use case is called an association, because it links a particular actor to a use case.

Use cases also can interact with other use cases. When the outcome of one use case is incorporated by another use case, we say that the second case uses the first case. UML indicates the relationship with an arrow that points at the use case being used. [Figure 6-12](javascript://) shows an example where a student adds a fitness class and PRODUCE FITNESS-CLASS ROSTER uses the results of ADD FITNESS-CLASS to generate a new fitness-class roster. Similarly, if an instructor changes his or her availability, UPDATE INSTRUCTOR INFORMATION uses the CHANGE AVAILABILITY use case to update the instructor’s information.

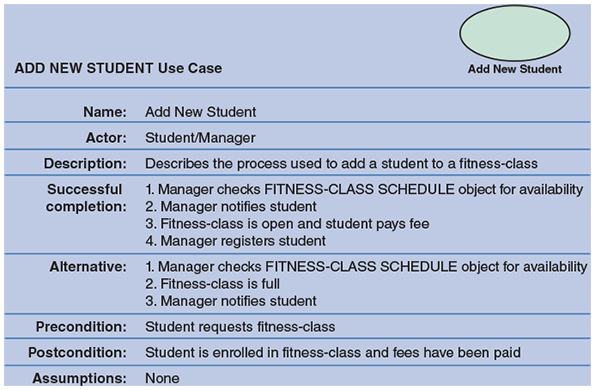
**Figure 6-12**

When a student adds a class, PRODUCE FITNESS-CLASS ROSTER uses the results of ADD FITNESS-CLASS to generate a new roster. When an instructor changes his or her availability, UPDATE INSTRUCTOR INFORMATION uses the CHANGE AVAILABILITY use case to update the instructor’s information.

To create use cases, start by reviewing the information that gathered during the requirements modeling phase. The objective is to identify the actors and the functions or transactions they initiate. For each use case, develop a [**use case description**](javascript://) in the form of a table. A use case description documents the name of the use case, the actor, a description of the use case, a step-by-step list of the tasks and actions required for successful completion, a description of alternative courses of action, preconditions, postconditions, and assumptions. [Figure 6-13](javascript://) shows an example of the ADD NEW STUDENT use case for the fitness center.

**Figure 6-13**

The ADD NEW STUDENT use case description documents the process used to add a current student into an existing class at the fitness center.



Enlarge Image

When use cases are identified, all the related transactions should be grouped into a single use case. For example, when a hotel customer reserves a room, the reservation system blocks a room, updates the occupancy forecast, and sends the customer a confirmation. Those events are all part of a single use case called RESERVE ROOM, and the specific actions are step-by-step tasks within the use case.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.2Use Case Diagrams

A **use case diagram** is a visual summary of several related use cases within a system or subsystem. Consider a typical auto service department, as shown in [Figure 6-14](javascript://). The service department involves customers, service writers who prepare work orders and invoices, and mechanics who perform the work. [Figure 6-15](javascript://) shows a possible use case diagram for the auto service department.

**Figure 6-14**

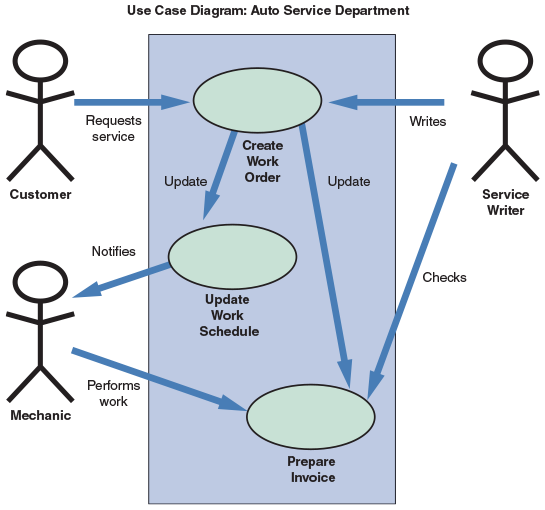
A typical auto service department might involve customers, service writers who prepare work orders and invoices, and mechanics who perform the work.



michaeljung/ [Shutterstock.com](http://shutterstock.com/" \t "_blank)

**Figure 6-15**

A use case diagram to handle work at an auto service department.



When a use case diagram is created, the first step is to identify the system boundary, which is represented by a rectangle. The [**system boundary**](javascript://) shows what is included in the system (inside the rectangle) and what is not included in the system (outside the rectangle). After the system boundary is identified, use cases are placed on the diagram, the actors are added, and the relationships shown.

**Case in Point 6.2**

### Hilltop Motors

* You were hired by Hilltop Motors as a consultant to help the company plan a new information system. Hilltop is an old-line dealership, and the prior owner was slow to change. A new management team has taken over, and they are eager to develop a first-class system. Right now, you are reviewing the service department, which is going through a major expansion. You decide to create a model of the service department in the form of a use case diagram. The main actors in the service operation are customers, service writers who prepare work orders and invoices, and mechanics who perform the work. You are meeting with the management team tomorrow morning. How would you create a draft of the diagram to present to them?

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.3Class Diagrams

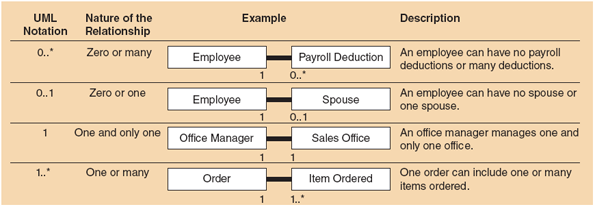
A [**class diagram**](javascript://) shows the object classes and relationships involved in a use case. Like a DFD, a class diagram is a logical model, which evolves into a physical model and finally becomes a functioning information system. In structured analysis, entities, data stores, and processes are transformed into data structures and program code. Similarly, class diagrams evolve into code modules, data objects, and other system components.

In a class diagram, each class appears as a rectangle, with the class name at the top, followed by the class’s attributes and methods. Lines show relationships between classes and have labels identifying the action that relates the two classes. To create a class diagram, review the use case and identify the classes that participate in the underlying business process.

The class diagram also includes a concept called [**cardinality**](javascript://), which describes how instances of one class relate to instances of another class. For example, an employee might have earned no vacation days or one vacation day or many vacation days. Similarly, an employee might have no spouse or one spouse. [Figure 6-16](javascript://) shows various UML notations and cardinality examples. Note that in [Figure 6-16](javascript://), the first column shows a UML notation symbol that identifies the relationship shown in the second column. The third column provides a typical example of the relationship, which is described in the last column. In the first row of the figure, the UML notation 0..\* identifies a zero or many relation. The example is that an employee can have no payroll deductions or many deductions.

**Figure 6-16**

Examples of UML notations that indicate the nature of the relationship between instances of one class and instances of another class.

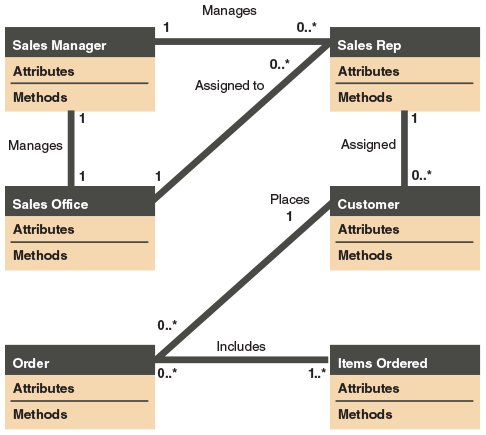


Enlarge Image

[Figure 6-17](javascript://) shows a class diagram for a sales order use case. Note that the sales office has one sales manager who can have anywhere from zero to many sales reps. Each sales rep can have anywhere from zero to many customers, but each customer has only one sales rep.

**Figure 6-17**

Class diagram for a sales order use case (attributes and methods omitted for clarity).



**Case in Point 6.3**

### Train the Trainers, Inc.

* Train the Trainer develops seminars and workshops for corporate training managers, who in turn train their employees. Your job at Train the Trainer is to put together the actual training materials. Right now, you are up against a deadline. The new object modeling seminar has a chapter on cardinality, and the client wants you to come up with at least three more examples for each of the four cardinality categories listed in [Figure 6-16](javascript://). The four categories are zero or many, zero or one, one and only one, and one or many. Even though you are under pressure, you are determined to use examples that are realistic and familiar to the students. What examples will you submit?

Change font size

[**help**](javascript://)

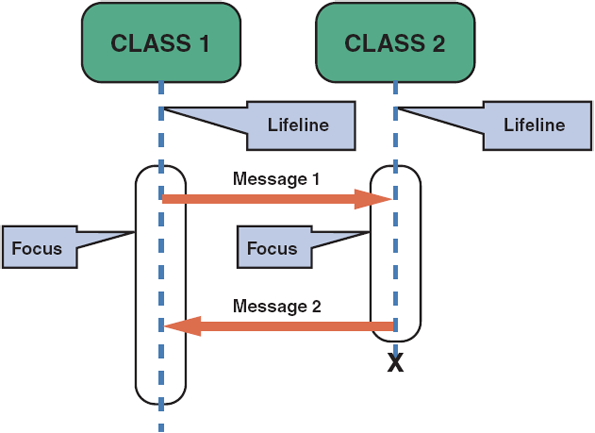
[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.4Sequence Diagrams

A **sequence diagram** is a dynamic model of a use case, showing the interaction among classes during a specified time period. A sequence diagram graphically documents the use case by showing the classes, the messages, and the timing of the messages. Sequence diagrams include symbols that represent classes, lifelines, messages, and focuses. These symbols are shown in [Figure 6-18](javascript://).

**Figure 6-18**

A sequence diagram with two classes. Notice the X that indicates the end of the CLASS 2 lifeline. Also notice that each message is represented by a line with a label that describes the message, and that each class has a focus that shows the period when messages are sent or received.



Enlarge Image

### Classes

A class is identified by a rectangle with the name inside. Classes that send or receive messages are shown at the top of the sequence diagram.

### Lifelines

A lifeline is identified by a dashed line. The [**lifeline**](javascript://) represents the time during which the object above it is able to interact with the other objects in the use case. An X marks the end of the lifeline.

### Messages

A message is identified by a line showing direction that runs between two objects. The label shows the name of the message and can include additional information about the contents.

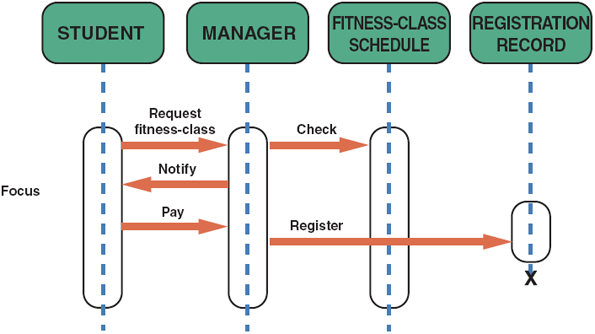
### Focuses

A focus is identified by a narrow vertical shape that covers the lifeline. The [**focus**](javascript://) indicates when an object sends or receives a message.

[Figure 6-19](javascript://) shows a sequence diagram for the ADD NEW STUDENT use case in the fitness center example. Note that the vertical position of each message indicates the timing of the message.

**Figure 6-19**

The sequence diagram for the ADD NEW STUDENT use case. The use case description for ADD NEW STUDENT is shown in [Figure 6-13](javascript://).



Enlarge Image

Change font size

[**help**](javascript://)

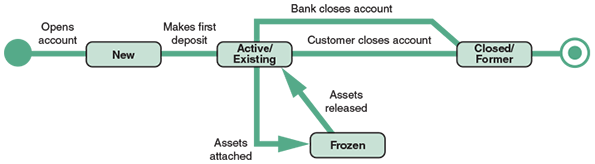
[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.5State Transition Diagrams

Earlier in this chapter, it was explained that state refers to an object’s current status. A [**state transition diagram**](javascript://) shows how an object changes from one state to another, depending on events that affect the object. All possible states must be documented in the state transition diagram, as shown in [Figure 6-20](javascript://). A bank account, for example, could be opened as a NEW account, change to an ACTIVE or EXISTING account, and eventually become a CLOSED or FORMER account. Another possible state for a bank account could be FROZEN, if the account’s assets are legally attached.

**Figure 6-20**

An example of a state transition diagram for a bank account.



Enlarge Image

In a state transition diagram, the states appear as rounded rectangles with the state names inside. The small circle to the left is the initial state or the point where the object first interacts with the system. Reading from left to right, the lines show direction and describe the action or event that causes a transition from one state to another. The circle at the right with a hollow border is the final state.

Change font size

[**help**](javascript://)

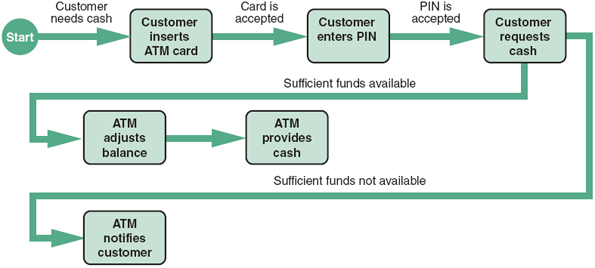
[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 6.8.6Activity Diagrams

An [**activity diagram**](javascript://) resembles a horizontal flowchart that shows the actions and events as they occur. Activity diagrams show the order in which the actions take place and identify the outcomes. [Figure 6-21](javascript://) shows an activity diagram for a cash withdrawal at an ATM machine. Note that the customer initiates the activity by inserting an ATM card and requesting cash. Activity diagrams also can display multiple use cases in the form of a grid, where classes are shown as vertical bars and actions appear as horizontal arrows.

**Figure 6-21**

An activity diagram showing the actions and events involved in withdrawing cash from an ATM.



Enlarge Image

Sequence diagrams, state transition diagrams, and activity diagrams are dynamic modeling tools that can help a systems analyst understand how objects behave and interact with the system.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

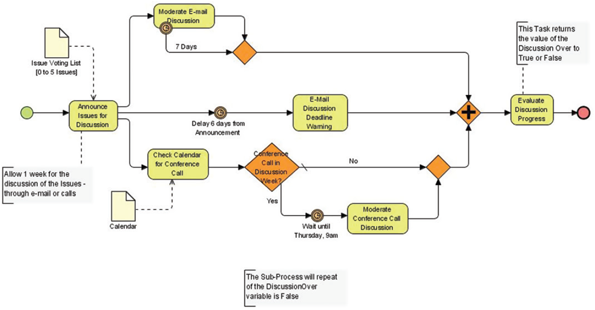
## 6.8.7Business Process Modeling

In addition to sequence diagrams and activity diagrams, business process modeling (BPM) can be used to represent the people, events, and interaction in a system. BPM initially was discussed in [Chapter 4](javascript://) as a requirement diagramming tool, but it can be used anytime during the systems development process. BPM works well with object modeling, because both methods focus on the actors and the way they behave.

There are a number of tools supporting BPM. For example, on Windows, the Bizagi Modeler tool supports business process modeling and simulation using the standard BPM notation. On the Mac, Visual Paradigm supports the creation of BPM diagrams (and several other modeling notations). [Figure 6-22](javascript://) shows a sample BPM for an online discussion cycle.

**Figure 6-22**

A BPM for an online discussion cycle.



Enlarge Image

Source: Wikimedia Commons

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# 6.9Tools

Object modeling requires many types of diagrams to represent the proposed system. Creating the diagrams by hand is time consuming and tedious, so systems analysts rely on tools to speed up the process and provide an overall framework for documenting the system components. In addition, tools ensure consistency and provide common links so that once objects are described and used in one part of the design, they can be reused multiple times without further effort.

There are many CASE tools tailored to UML. Tools such as Visio are popular for drawing UML diagrams, but they lack semantics: They are useful for creating visually pleasing figures, but they have no knowledge of the underlying artifacts. This means a systems analyst can create diagrams that appear to be correct but are in fact incorrect when it comes to the rules of the UML.

To overcome this shortcoming, proper systems modeling tools, such as Cameo Systems Modeler or IBM’s Rational product family, are typically used. These tools understand the meaning of the diagrams they help create, which suggests it’s difficult to create a UML diagram that is syntactically incorrect. These tools also provide traceability, an important feature in the SDLC that links design artifacts backward to requirements and forward to development and testing.

### A Question of Ethics

* [iStock.com](https://istock.com/" \t "_blank)/faberfoto\_itYour company sent several staff members for UML training by an outside vendor. Everyone who attended the training received a copy of the instructor’s materials, which included study guides and sample exam questions and solutions.

After completing the training course, you are eligible to sit a certification exam. If you pass the exam, you will be credentialed as a “UML Expert” by an independent agency. You can parlay these credentials into a higher salary and boost your career.

A coworker who did not attend the training asks for a copy of the training materials. He wants to take the exam without “wasting his time in class.” Should you give him a copy of the training materials? If you do, how might this diminish your accomplishments? If you don’t, would you be hurting the team by not helping another member become more knowledgeable about the UML?

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**6.10**Summary

This chapter introduced object modeling, which is a popular technique that describes a system in terms of objects. Objects represent real people, places, events, and transactions. Unlike structured analysis, which treats data and processes separately, objects include data and processes that can affect the data. During the implementation process, systems analysts and programmers transform objects into program code modules that can be optimized, tested, and reused as often as necessary.

O-O terms include attributes, methods, messages, and classes. Attributes are characteristics that describe the object. Methods are tasks or functions that the object performs when it receives a command to do so. Objects can send messages, or commands, that require other objects to perform certain methods, or tasks. The concept that a message gives different meanings to different objects is called polymorphism. An object resembles a black box with encapsulated, or self-contained, data and methods.

Classes include objects that have similar attributes, or characteristics. Individual members of a class are called object instances. Objects within a class can be grouped into subclasses, which are more specific categories within the class. A class also can belong to a more general category called a superclass.

After identifying the objects, classes, and relationships, an object relationship diagram is prepared that shows the objects and how they interact to perform business functions and transactions. The strongest relationship between objects is inheritance.

The UML is a widely used method of visualizing and documenting an information system. UML techniques include use cases, use case diagrams, class diagrams, sequence diagrams, state transition diagrams, and activity diagrams.

A use case describes a business situation initiated by an actor, who interacts with the information system. Each use case represents a specific transaction, or scenario. A use case diagram is a visual summary of related use cases within a system or subsystem. A class diagram represents a detailed view of a single use case, showing the classes that participate in the underlying business transaction, and the relationship among class instances, which is called cardinality. A sequence diagram is a dynamic model of a use case, showing the interaction among classes during a specified time period. Sequence diagrams include lifelines, messages, and focuses. A state transition diagram shows how an object changes from one state to another, depending on events that affect the object. An activity diagram resembles a horizontal flowchart that shows actions and events as they occur in a system.

In addition to object models, business process modeling (BPM) can be used to represent the people, events, and interaction in a system.

CASE tools provide an overall framework for system documentation. CASE tools can speed up the development process, ensure consistency, and provide common links that enable reuse during O-O analysis.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# Chapter Review

## **Key Terms**

* [**activity diagram**](javascript://)
* **actor**
* [**attributes**](javascript://)
* **black box**
* [**cardinality**](javascript://)
* [**child**](javascript://)
* **class**
* [**class diagram**](javascript://)
* [**encapsulation**](javascript://)
* [**focus**](javascript://)
* [**inheritance**](javascript://)
* [**instance**](javascript://)
* [**lifeline**](javascript://)
* [**message**](javascript://)
* [**method**](javascript://)
* [**object**](javascript://)
* [**object model**](javascript://)
* **object-oriented (O-O) analysis**
* [**parent**](javascript://)
* [**polymorphism**](javascript://)
* [**relationships**](javascript://)
* **sequence diagram**
* [**state**](javascript://)
* [**state transition diagram**](javascript://)
* [**subclasses**](javascript://)
* [**superclass**](javascript://)
* [**system boundary**](javascript://)
* **Unified Modeling Language (UML)**
* [**use case**](javascript://)
* [**use case description**](javascript://)
* **use case diagram**

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)