EXTENDED LEARNING MODULE G

OBJECT-ORIENTED TECHNOLOGIES

Student Learning Outcomes

1. Explain the primary difference between the traditional technology approach and the object-oriented technology approach.
2. List and describe the five primary object-oriented concepts.
3. Explain how classes and objects are related.
4. Discuss the three fundamental principles of object-oriented technologies.
5. Describe two types of object-oriented technologies.

Introduction

The explosion of object-oriented technologies is radically changing the way businesses view and develop IT systems. As we discussed in Chapter 6, systems development is a time-intensive task, but it can yield the powerful and innovative systems that propel organizations toward achieving their strategic goals and gaining competitive advantages. The entire object-oriented genre aims at increasing both the efficiency and the effectiveness of the systems development process, specifically component-based development (CBD), making the task easier and the outcome more satisfactory.

It's difficult to find a business or IT department today that isn't using some type of object-oriented technology. Software developers everywhere are learning how to write in object-oriented programming languages, create databases using object-oriented database management systems, and design new systems using object-oriented analysis and design techniques.

By reading this module, you'll gain an understanding of object-oriented technologies, how they work, and in what ways they can be beneficial. Even though you may not plan to be a systems developer, it's still important that you learn the fundamentals of important object-oriented concepts and technologies because you'll certainly be involved in analyzing, designing, and using object-oriented systems. Learning about object-oriented concepts and technologies will help prepare you for your future role as a knowledge worker in the information age. This module walks you through

- The traditional technology approach
- The object-oriented technology approach
- The five primary concepts of object-oriented technologies: information, procedures, classes, objects, and messages
- The three fundamental principles of object-oriented technologies: inheritance, encapsulation, and polymorphism
- A detailed business case example
- The three types of object-oriented technologies: object-oriented programming languages, object-oriented databases, and object-oriented technologies and client/server environments

LEARNING OUTCOME 1

Traditional Technology Approach

Let's first take a look at the traditional technology approach to help you understand the object-oriented approach. The ***traditional technology approach*** has two primary views of any computer-based system—*information and procedures*—and it keeps these two views separate and distinct at all times.

INFORMATION VIEW

The ***information view*** in the traditional technology approach includes all of the information stored within a system. You're probably familiar with many terms that describe information, including data, variables, and attributes. All these terms refer to the same thing—information. For example, in a student grading system, information would include *Student Last Name, Student Address,* and *Final Course Grade.* In Figure G.1, you can see all the different types of information that might be required to build a student grading system. When designing and developing this system, your focus in the information view is only on the information itself and not on any of the procedures necessary to

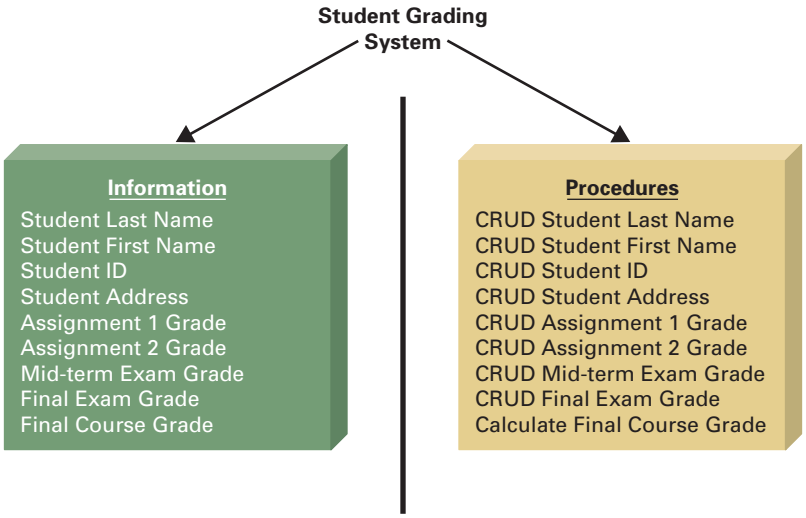


Figure G.1
Student Grading System: Separate Information and Procedure Views

maintain the information. For example, *Final Course Grade* is a function of *Assignment 1 Grade, Assignment 2 Grade,* and so on. But in the information view, you do not know the relative weights of each grade that go into calculating the *Final Course Grade.* Those weights and the appropriate computation are contained in the *procedure view.*

PROCEDURE VIEW

The ***procedure view*** in the traditional technology approach contains all of the procedures within a system. A ***procedure*** manipulates or changes information. You're probably familiar with many terms that describe procedures, including formulas, functions, methods, and routines.

There are four primary procedures, or ways, a system can manipulate information: create, read, update, and delete. These four procedures are commonly referred to as ***CRUD*** (**C**reate, **R**ead, **U**pdate, **D**elete). Notice in Figure G.1 that the procedure view maintains CRUD procedures for all the information except *Final Course Grade,* which is maintained by a *Calculate* procedure.

The primary problem with the traditional technology approach is that the two separate views can lead to potential disconnects between the information and procedures. It's possible to have the correct information, for example, but not be able to do anything with it because you don't have the corresponding procedures. Likewise, you could have the correct procedures but not be able to do anything with them because you don't have the corresponding information. In the first instance, it's like having a workbook but no spreadsheet software. In the second instance, it's like having the spreadsheet software but no information to work with.

Such disconnects, if encountered in today's digital world, would lead to numerous problems. People on the move, not working in a central office, need access to both information and procedures. Consumers can't very well use a PDA or other handheld device to participate in m-commerce if they can't access both information (e.g., products they would like to buy) and procedures (the necessary steps for ordering those products). At home, when you use a universal remote control to program your VCR or DVD recording device, the remote control is an *object* in the object-oriented world, and so is your VCR and DVD recording device. By that we mean that each device contains both information and procedures. Together, they create an object-oriented system—one that most of us cannot imagine living without.

Object-Oriented Technology Approach

The object-oriented approach bridges the gap between information and procedures by providing a holistic view of an information system. That is, an ***object-oriented approach*** combines information and procedures into a single view.

INFORMATION AND PROCEDURE VIEWS COMBINED

Let's take a look at the student grading system in Figure G.2, the same student grading system that was shown in Figure G.1, but now taking an object-oriented approach. Notice the diagram in Figure G.2 represents a holistic or single view of the entire system with the information in the middle and the procedures surrounding and bearing upon the information.

When you build an object-oriented system you think of the procedures and information together as such a single interlocking unit (object). For knowledge workers dealing with a system like the student grading system, a diagram such as Figure G.2 reveals which types of information are in the system and which kinds of procedures in the system can be performed on the information. Knowledge workers using the object-oriented approach (as pictured in Figure G.2), looking at both information and procedures combined, find it easier to perform the following three important tasks:

1. Understand the entire system.
2. Determine if any key information is missing that would be required in order to perform all the procedures.
3. Determine if any key procedures are missing which are required to manipulate the information.

When implemented, an object-oriented system allows various users to access information and modify it according to the procedures. For example, if your instructor—using a PDA—wanted to compute the *Final Course Grade* for everyone in your class,

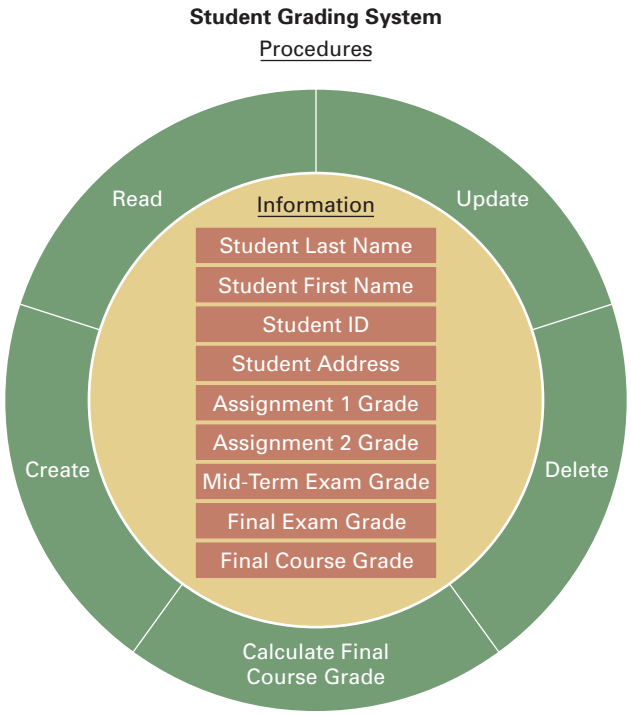


Figure G.2
Student Grading System: Combined Information and Procedures

the system would pass to your instructor all the necessary information for each student (*Assignment 1 Grade, Assignment 2 Grade,* etc.) as well as the procedure *Calculate Final Course Grade.* Once your instructor verified the *Final Course Grade* for each student, his/her PDA would pass the results back to the main system, which would be housed somewhere in your school's IT system.

OBJECT-ORIENTED APPROACH AND THE REAL WORLD

The object-oriented approach not only makes a knowledge worker's job easier but also improves overall systems development because the approach more closely models the real world. In the real world, you actually view a given business process as a combination of information and the procedures you need to act on that information. Have you ever purchased a product that required "some assembly," such as a mountain bike or a gas grill? Upon opening the box and spreading out the contents you probably immediately reached for the instruction booklet. In the instructions you found a detailed set of steps concerning assembly along with a description of the various parts, both the information and procedures provided together, which is similar to an object-oriented approach. You didn't find the set of instructions (procedures) in a separate booklet from the description of the parts; they were combined together because this makes logical sense.

If you viewed only the procedures in the student grading system example, would they make any sense? Probably not. It would be difficult for you to understand the procedures without understanding the information. For example, it would be impossible for you to understand the *Calculate Final Course Grade* procedure if you didn't know what type of information was used in this calculation.

Let's take a look at an inventory tracking process, a good example of how the object-oriented approach models the real world. In any business process such as tracking inventory, you can identify several key pieces of required information, in this case, *Part Number, Part Name, Part Manufacturer, Quantity on Hand, Reorder Point,* and *Cost.* At the same time, you can also identify how the information needs to be manipulated to perform the process, operations such as *Calculate Quantity to Order, Add a New Part, Update a Part Cost,* and so on, all examples of procedures, in the case of an inventory system. In short, information and procedures are integrated in the track inventory process. This is logical and effective. Object-oriented concepts mirror a real-world view in which information and procedures are necessarily combined together.

Five Primary Concepts of Object-Oriented Technologies

LEARNING OUTCOME 2

There are five basic object-oriented concepts that you as a knowledge worker should understand. We introduced you to two of these in the previous section—*information* and *procedures.* Although we've already discussed these concepts, we'll briefly review them in this section. The remaining three basic object-oriented concepts are *classes, objects,* and *messages.*

INFORMATION

Information comprises key characteristics stored within a system. You're already familiar with the different types of information stored in the student grading system such as *Student Last Name* and *Final Course Grade,* so let's take a look at information stored in a different system. Imagine you're building a member tracking system for an athletic club.

What types of information would the system need in order to track different members? The answer to this can be any key characteristic you can think of related to a member including *Member Name, Member ID, Address, Phone, Weight, Height, Membership Type,* and so forth. These are all examples of different types of information the member tracking system could store.

PROCEDURES

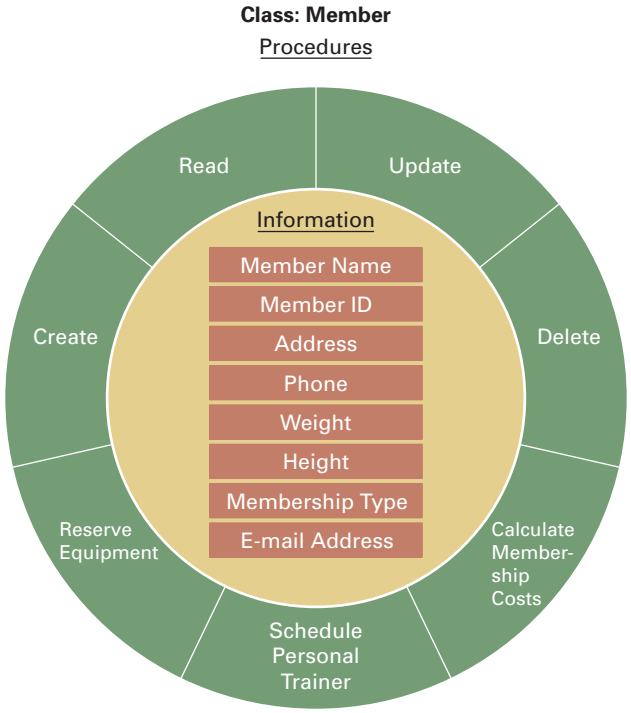
As we said earlier, a *procedure* manipulates or changes information. Again, you're already familiar with the procedures in the student grading system, including *CRUD Student Last Name* and *Calculate Final Course Grade.* Let's define some different procedures required to build the member tracking system for an athletic club. What types of procedures does the system need? The answer to this can once again be anything you think you'll need in order to manipulate the member information, including *CRUD Member Name* and *CRUD Member ID.* It's also important to understand that procedures are used not only for CRUD on information, but also to perform other functions or operations. In the member tracking system, other procedures might include *Schedule Personal Trainer, Cancel Membership, Schedule a Workout Time,* and so on.

LEARNING OUTCOME 3

CLASSES

A ***class*** contains information and procedures and acts as a template to create objects. A class is the combination of information and procedures as displayed in Figure G.3. It sometimes helps to think of classes as similar to a definition in a dictionary. Let's return to our member tracking system. If you look up the definition of the word *member* in a dictionary, you'll find out what a member is and perhaps what a member's role is. A class does exactly the same thing. If you look at the class Member in Figure G.3, you can quickly understand all the information required to describe a member and many of the procedures the Member class can perform including *Calculate Membership Costs* and *Reserve Equipment.*

Figure G.3
Class Member



MULTIPLE CLASSES So far, for the sake of simplicity, for both the student grading system and the member tracking system, we've identified everything as only a single class. Classes become a little more difficult to handle when you start to think of having 50 or 500 different classes in a single system.

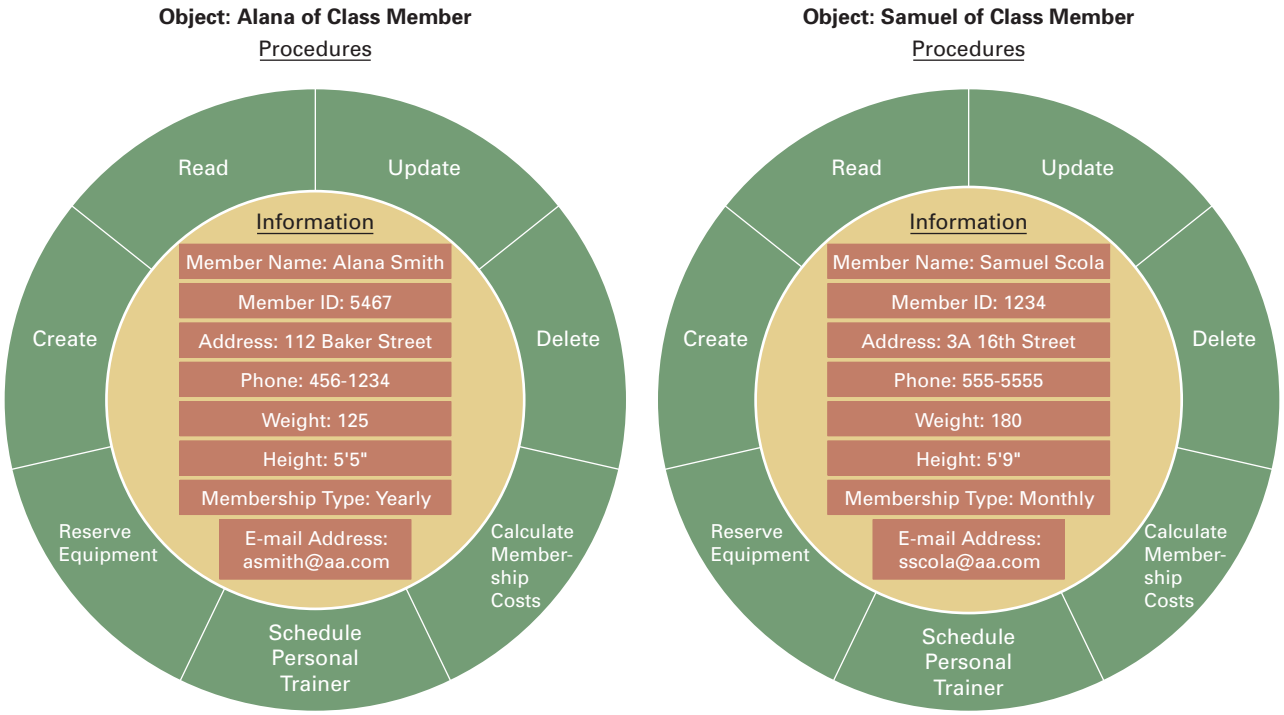
Almost all object-oriented IT systems have multiple classes, instead of just one. It's easy to see why. Imagine if you put all the information for the member tracking system into a single class. This class would be enormous. Just think of all the different types of information it might contain. There could be equipment preferences, workout length and frequency, nutritional habits, and length of membership, to name just a few. If all this information was in a single class, the class would quickly become unmanageable. Determining which types of information belonged to which procedures would be impossible. Breaking down the information and procedures for ease of use and understandability, or practicing ***information decomposition,*** is a better way to structure your information system. Practicing information decomposition makes IT specialists' and knowledge workers' jobs easier because the information and associated procedures are in understandable pieces. There certainly isn't anything stopping you from putting all the information and procedures for the entire system into a single class, but this is a bad system design and would lead to multiple system problems.

OBJECTS

An ***object*** is an instance of a class. To put it another way, an object is an actual item representing the class.

Let's take a look at the two objects (Alana and Samuel) of the class Member represented in Figure G.4. Can you describe the primary difference between the Member class diagram in Figure G.3 and the Member object diagrams in Figure G.4? Sure, you noticed the Member object diagrams contain the information representing the actual members that the system tracks. Using Figure G.4, you can describe the first object, Alana. Alana Smith is a yearly member who lives at 112 Baker Street. You can also describe the

Figure G.4
Alana and Samuel: Two Instances (Objects) of Class Member



second object, Samuel. Samuel Scola is a monthly member who lives at 3A 16th Street. Every member the system tracks is a separate object, and every member object is an instance of (or created from) the class Member. Alana and Samuel are both examples of objects of the class Member. Each of their respective objects contains information unique to them and also the necessary procedures for manipulating that information.

Remember that classes are a template of information and procedures to create objects. Essentially, the class is a blank template that defines all the different types of information the system can store about an object and the procedures to manipulate that information. Once you create an object from the class, you can fill in the template with the actual information. For example, the Member class can give you a high-level definition of a member object, such as *Member Name,* but it can't tell you the actual name of the member. The class will tell you only that it can store information for the member's name. The object stores the actual information and can tell you that the member's name is Alana (or Samuel). It also gives you the ability to manipulate that information with the *CRUD Member Name* procedure.

MESSAGES

Now that you know something about objects, you may be wondering if objects somehow need to communicate with each other and, if so, how. These are important questions, and once you start asking them you're quickly becoming an object-oriented expert.

Indeed, an object can send a message to another object asking it to perform a certain procedure. ***Messages*** are how objects communicate with each other. In general, systems are created by developing many different classes that work together to perform tasks. Let's refer back to our student grading system. If you build this system, you might have one class representing students, a second class representing courses, and a third class representing instructors. Objects from each of the classes communicate with each other in order to enter the student's course grades. The student object passes a request to enroll in courses to the course object. The instructor object passes a request to teach specific courses to the course object. The instructor object also passes the student's final course grades to the course object. Combining these three classes together via such messaging gives you a fully functioning student grading system.

HOW THE FIVE PRIMARY CONCEPTS INTERACT

Let's review information, procedures, classes, objects, and messages as important object-oriented concepts and how they relate to each other:

- Information and procedures create classes.
- Classes create objects.
- Objects communicate with other objects via messages.

Now, remembering from Chapter 1 that people are the key IT resource, let's reinforce your understanding of these relationships by thinking of the individuals who build the systems and the individuals who use the systems. This should solidify your perspective on how the object-oriented concepts relate to one another.

System developers are the individuals who build the system and are responsible for building the classes which contain information and procedures. System users are the individuals who use the system and are responsible for inputting the actual information, or creating the objects. If you think about the member tracking system, you can easily identify which individuals are responsible for the different tasks. The system developer would be responsible for building the Member class and the associated information and procedures. The athletic club employee, or system user, would be responsible for creating the objects, or inputting the actual member information.

Because you now understand the important concepts of information, procedures, classes, objects, and messages, you've mastered the basics of object-oriented technologies. Next, let's look at some real-world examples of object-oriented systems.

Real-World Object-Oriented Examples

A home stereo system is a perfect analogy for an object-oriented system (see Figure G.5). If you created classes to represent all the stereo components, they could include some or all of the following classes:

- Amplifier
- CD/DVD player
- Equalizer
- Speakers
- CDs
- DVDs

Think about the information items stored in the CD class. *Title, Artist, Date Recorded,* and *Number of Songs* would be a few. Can you determine how many objects would be created from the CD class? The number would vary depending on how many CDs you owned. Consider the amplifier object—what would you say is its primary procedure? That would be *Set Volume.*

In order for the system to work, objects are created from each class, and each of the objects works with specific information and procedures. The information for the CD/DVD player, for example, could include the *Manufacturer Name, Model Number,* or *Play Speed.* The procedures for the CD/DVD player could include *Play, Fast Forward, Rewind, Skip,* or *Stop.* If you called the *Play* procedure for the CD/DVD player object (using a remote control), do you think you would be able to hear music? No, not instantly. The CD/DVD player could play the CD only if it first sent a message to the amplifier object concerning which song to play, which in turn would send a message (and your music selection) to the speaker object to play the music.

A home stereo system is a true example of a real-world object-oriented system. Each component must work together in order for the system to function. Thus, each component in a home stereo system really is an object. As an object, each component works with only certain information and performs only certain procedures. If one component needs another procedure performed, it must send a message to another object (or component) that can perform that procedure.

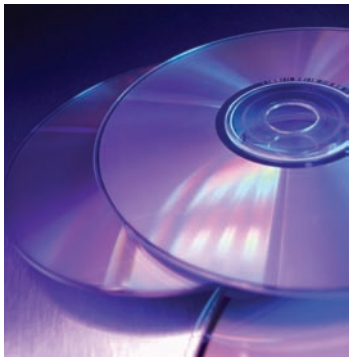


Figure G.5
Home Stereo System

Another real-world object-oriented system is that of an automobile. The steering wheel, tires, and engine are different components or objects that work together in order to accomplish the common goal of driving. In an automobile, there are actually hundreds of objects working together, including the horn, thermostat, air conditioner, anti-lock brake system, headlights, and all the others.

A computer is another example of several different components or objects working together and sending messages to each other in order to accomplish a common task. (See *Extended Learning Module A* for how computer components work together and send messages to each other.) The keyboard, monitor, mouse and operating system all work together to run your applications. When you print a document in Word, for example, your printer receives a message to print the document, various instructions and information including how many copies to print and whether to print in portrait or landscape, and the actual document to be printed. Your printer doesn't determine which document to print (that's a procedure handled by your software); it simply prints the document sent to it via a message.

LEARNING OUTCOME 4

Three Fundamental Principles of Object-Oriented Technologies

Businesses of all types can gain huge IT-enabled competitive advantages in the marketplace by using object-oriented technologies. These advantages derive from the three fundamental principles of object-oriented technologies:

1. Inheritance
2. Encapsulation
3. Polymorphism

INHERITANCE

One of the most powerful principles of object-oriented technologies is inheritance. ***Inheritance*** is the ability to define superclass and subclass relationships among classes. Generalization (parent) and specialization (child) relationships are another way of thinking about superclass and subclass relationships. Take a moment and review Figure G.6 and determine which class is the superclass and which classes are the subclasses. The Car class is the superclass and the Bronco and Porsche classes are the subclasses. Another way to state this relationship is that the Bronco and Porsche subclasses inherit all of the information and procedures of the Car superclass. For example, the CRUD procedures are not defined in the Bronco and Porsche subclasses because they are inherited from the Car superclass.

Defining inheritance is simply a matter of defining generic (generalization) and specific (specialization) information and procedures. *Generic* information and procedures apply to all subclasses, which will inherit the information and procedures from the superclass in which they are defined. *Specific* information and procedures apply only to a particular subclass.

For example, notice that the Car superclass in Figure G.6 contains generic information and procedures that are shared (inherited) by both the Bronco and Porsche subclasses. Both the Bronco and Porsche subclasses have a *Model, Year, Price,* and *Color,* and both can *Honk Horn, Brake,* and *Drive.*

The subclasses contain specific information and procedures that are unique to each particular subclass. The Bronco subclass contains information for *Four-Wheel Drive* and

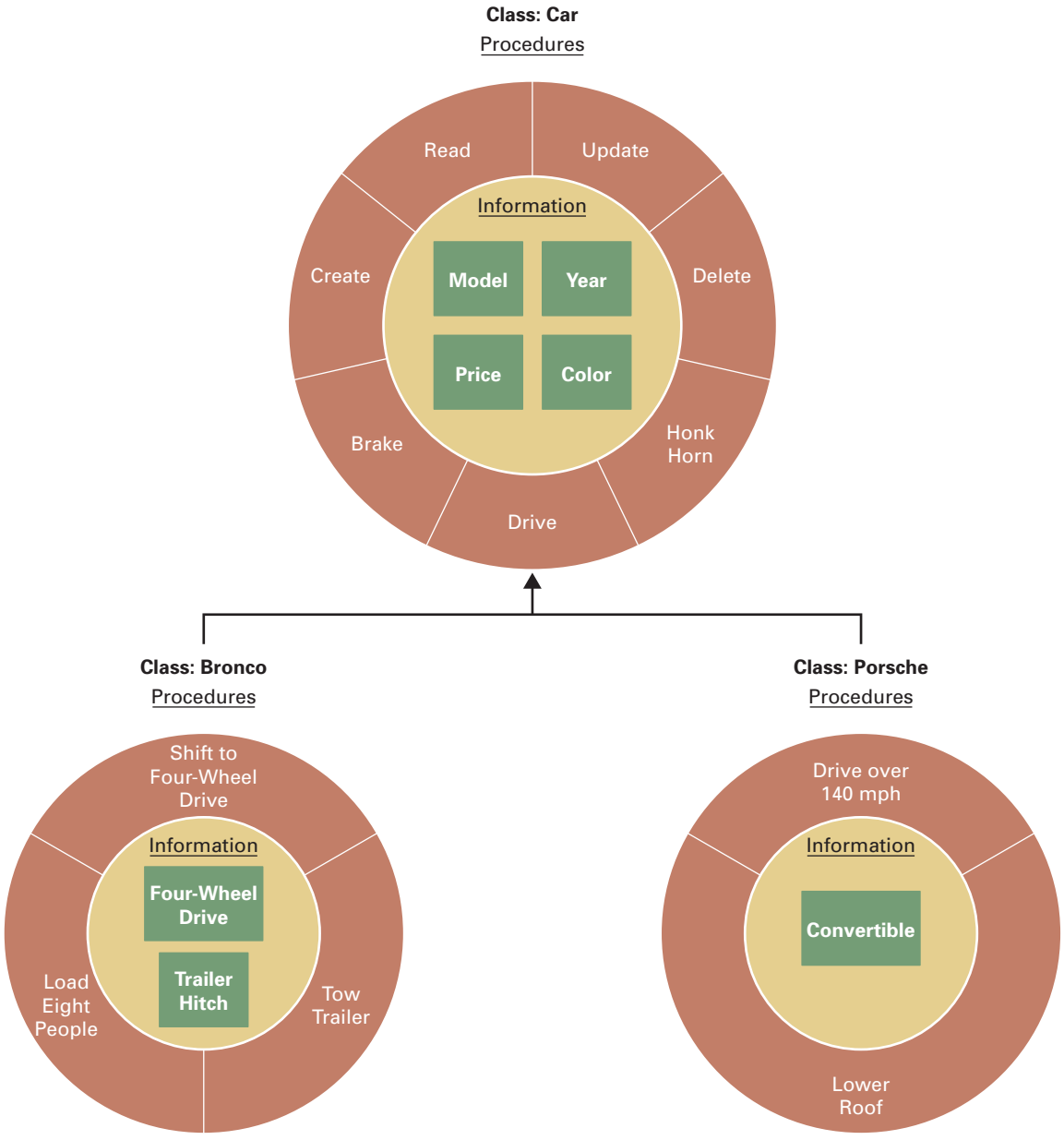


Figure G.6
Inheritance

the Porsche subclass contains a procedure to *Drive over 140 mph*. These unique features are stored in the subclass because they are not generic enough to store in the Car superclass. If you stored the procedure for *Drive over 140 mph* in the Car superclass, then every subclass of Car would inherit this procedure. This would be a mistake since there are many types of cars that can't drive over 140 mph. Hence, you must store this unique feature at the subclass level.

Inheritance, or the ability to define superclass and subclass relationships, is a unique feature of object-oriented technologies, and the primary benefit of inheritance is *reuse*. Consider our car example. If you decided to add a piece of information called *Mileage* (i.e., miles per gallon) to the mix, you would develop it in only one place, the Car superclass, and it would automatically be inherited by every subclass. The *Mileage* information

is *reused* by all the subclasses, along with any associated procedure(s). Thus, a business benefit from the inheritance principle is the ability to easily expand and maintain a system.

If you were using a traditional approach to developing and maintaining this system, the *Mileage* information and procedures would have to be developed in each of the Car, Bronco, and Porsche classes because the traditional approach does not support the concept of inheritance (and thus reuse). If you had 50 subclasses of the Car superclass, you would have to spend a great deal of time and energy (both scarce organizational resources) to build the *Mileage* information and procedures into all 50 subclasses. Using the object-oriented approach, you build them only once in the superclass and all 50 subclasses automatically inherit the information and procedures.

Another example of how inheritance makes it easy to expand a system would be if you decided to add a new subclass, Volkswagen Beetle, to the system. If you placed this subclass within the Car superclass, it would automatically inherit all the information and procedures already developed in the Car superclass. All you would need to do is develop any unique information and procedures that the Volkswagen Beetle class requires. If you added a Volkswagen Beetle class using a traditional approach, you'd have to build in every single piece of information and every procedure including *Model, Year, Price, CRUD, Honk Horn,* and so on. As you can see, inheritance is a valuable principle for businesses in terms of saving money, time, and effort when developing and maintaining information systems.

ENCAPSULATION

Encapsulation means information hiding. This concept has a simple definition and provides tremendous benefits when you're building an information system. Let's take a look at how object-oriented technologies encapsulate information.

Objects are sometimes referred to as black boxes, in that the information inside an object is hidden and all that can be viewed is the object. Imagine you're in a park and you see a dog. You can instantly determine the dog is an object of the class Dog. You don't know all of the information about the dog, however. You can't tell the dog's name, weight, height, or other characteristics just by looking at it. Hence, the actual information stored in the object is hidden, or encapsulated, but this doesn't prevent you from seeing the dog and being able to communicate with the dog.

Let's take a look at an example of encapsulation. If you order AT&T digital cable, you receive a remote control that is used as the interface to the digital cable box. An ***interface*** is any device that calls procedures, such as a keyboard, mouse, or touch screen. Not long ago, AT&T changed its entire digital cable system including the menu colors, item locations, and cable features. Suddenly, when a user turned on the television everything on the digital cable menu looked completely different. However, the remote control, or system interface, continued to work exactly the same as it did before the system changed. The same buttons on the remote control were used to turn the TV on and off, to adjust the volume up and down, and to select different menu items. Since the remote control didn't change, customers could use their system exactly the same way as they did before. The only change the users had to deal with was getting used to the new look and feel of the improved digital cable system.

AT&T used encapsulation to hide the digital cable system changes from its users so as not to bother them with them. The system changed significantly but, because the user's remote control, or interface, continued to work exactly the same, the users were unaffected. The users continued to receive the same high-quality service from AT&T. This is a great example of the benefits to a company and its customers from encapsulation.

POLYMORPHISM

Polymorphism, like so many technical words, though it looks and sounds intimidating, has a very simple meaning. ***Polymorphism*** means "to have many forms." What do you think of when you hear the word *bark?* A dog's bark? A tree's bark? A small ship? This is true polymorphism, having the ability to use the same word to mean different things.

Let's take a look at Figure G.7 for an example of polymorphism. Notice that the Rectangle, Square, and Circle classes all contain procedures called *Calculate Area.* Actually, the *Calculate Area* procedure is different for each class. The formula to calculate the area of a rectangle (*Length* × *Width*) is different from the formula to calculate the area of a square (*Side* × *Side*). This is an example of polymorphism because each class has an identically named procedure that performs different calculations.

If you build a procedure using a traditional technology, you must ensure that the procedure has a unique name. Consider our previous example of cars. If you have 50 different models of cars in the system, each must have a unique name for the *Honk Horn* procedure such as *Porsche Honk Horn, Bronco Honk Horn, Beetle Honk Horn,* and so on. Further, each model of car may have variations (e.g., Beetle XL and Beetle Sportster); these must also have different names for each procedure for honking a horn. Using an object-oriented approach removes the problem of defining complex naming structures and increases productivity.

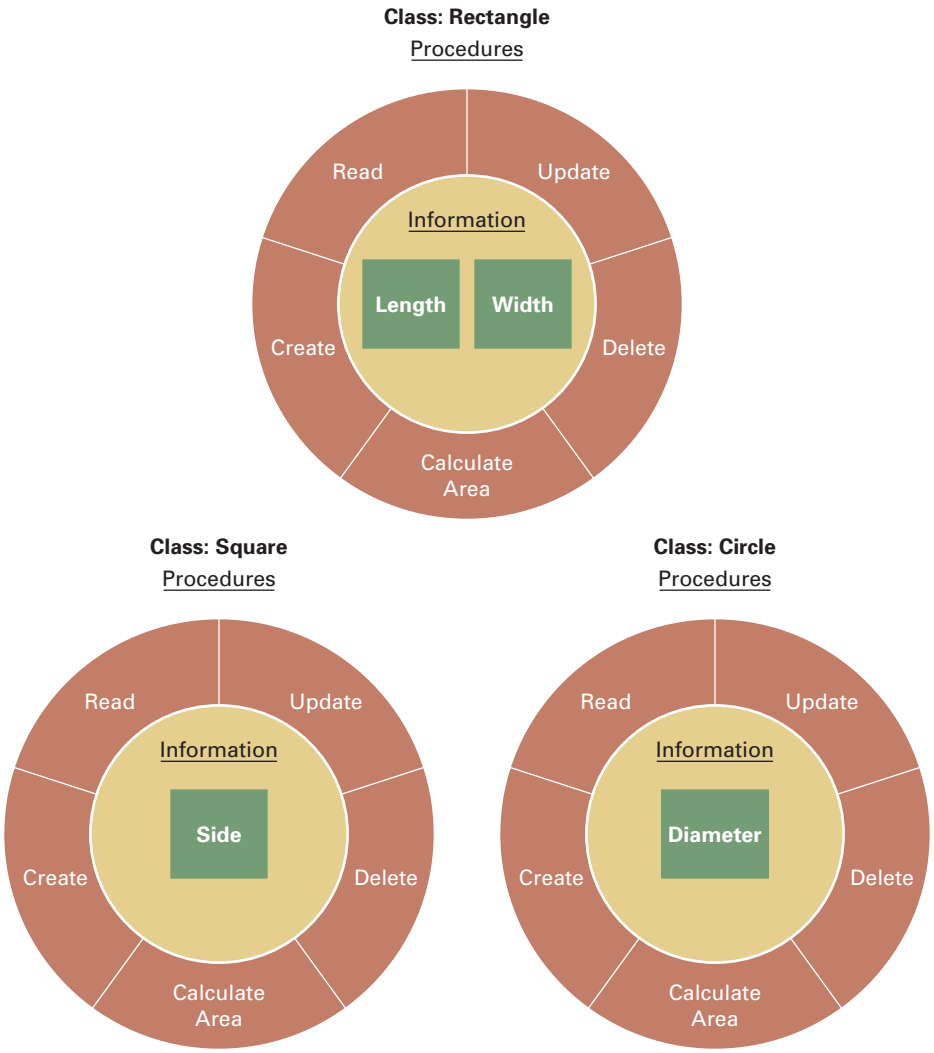


Figure G.7
Polymorphism

Putting It All Together: A Business Example

Let's assume you're starting your own business, Ice Blue Snowboards (see Figure G.8). Your business manufactures and sells snowboards, bindings, boots, and apparel. To prepare for the launch of the business, you researched similar businesses to discover any problems they've encountered so you can avoid making the same mistakes. The following is a list of common competitor problems:

- Eighteen months to get a new product to market
- Inventory control
- Scalability and expandability

Let's take a detailed look at each problem and discuss how using an object-oriented approach will help you minimize or eliminate these problems.

EIGHTEEN MONTHS TO GET A NEW PRODUCT TO MARKET

For your business to be competitive, you must be able to get your new products on the market. Having a fast time-to-market is critical for the success of your new business. Eighteen months is simply too long for you to wait to get a new product on the market. Using an object-oriented approach will help you reduce this critical time-to-market factor. The typical high-level processes for developing new products include the following:

1. Generating and accepting the idea
2. Manufacturing the product
3. Updating all current systems to support the new product
4. Implementing the new product

Each time they develop a new product, much of the 18 months it takes your competitors to get the product on the market is taken up with the first two activities—generating and accepting the idea and manufacturing the product. But most businesses, using the traditional approach, also spend a lot of time—several months—updating all current systems to support the new product. Using an object-oriented approach you can significantly reduce the amount of time it takes Ice Blue Snowboards to update all its current systems to support the new product. Let's see how.

First, you would create a superclass called Snowboard. The Snowboard superclass would be responsible for tracking all snowboard information including *Price, Model, Features, Discounts,* and so on. The Snowboard superclass would also be responsible

Figure G.8
Ice Blue Snowboards

ICE BLUE
SNOWBOARDS

for establishing all the procedures associated with marketing and selling snowboards including advertising and promoting in magazines and competitor pricing analysis.

Second, you would design an interface enabling all your employees to perform all these procedures. This interface would encapsulate the information and procedures in the Snowboard superclass and allow you the flexibility to change things without affecting the productivity of your employees.

Third, you would practice inheritance. You would create a subclass for each particular type of snowboard. Defining subclasses, or using inheritance, saves you a great deal of time and energy because each subclass inherits all the information and procedures from the superclass. The only work required to create a subclass will be defining the unique information or procedures associated with the subclass. After creating each class, once again you can reuse the classes across all your business applications. Finally, you would create objects that communicate with all of the other objects across every system in your business.

Figure G.9 is an example of your snowboard business case class diagram. This figure assumes your business offers an electronic catalog on the Internet listing all the snowboarding products you sell. By creating a Catalog object whose primary procedure is to

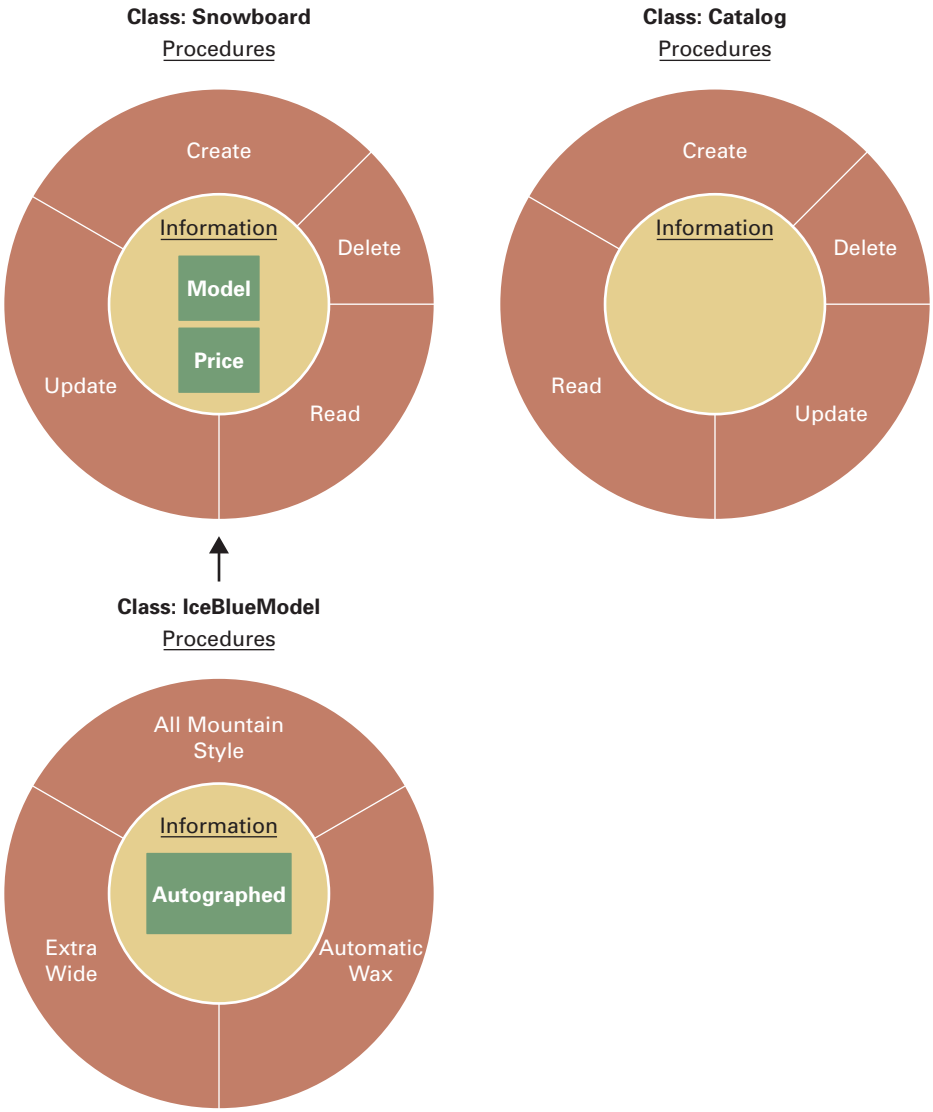


Figure G.9
Snowboard Business Case Class Diagram

update the electronic catalog, you won't be required to change anything when adding a new product to the catalog. The Catalog object already understands how products are added and listed in the catalog. The new product object simply sends a message to the Catalog object to execute the *Update* procedure in the Catalog object. The new product will be added to the catalog without any need for system modification.

The keys to reducing the time-to-market for your products, from an IT system point of view, are many. You want to create superclasses that contain generic information and procedures that all subclasses can inherit and reuse. You want to develop interfaces that support the concept of information hiding so your employees aren't constantly faced with the task of relearning how to use a system. Reducing your time-to-market is a critical competitive advantage.

INVENTORY CONTROL

Controlling inventory is always a challenge for business. Businesses need to have sufficient inventory on hand to meet current production needs while minimizing the associated expenses of storing the inventory, transporting the inventory, and maintaining the inventory. This is a supply chain management problem (see Chapter 2). Managing the supply chain is fundamental to the success of any business, small or large. If you neglect to manage your supply chain, then you'll find your business hit by high storage costs, by the inability to manufacture products because of low inventories, or by lost inventory.

To define and efficiently implement a supply chain management system, you would define such superclass objects as Shipping, Distribution, and Vehicle. Creating shipping, distribution, and vehicle objects allows you to control your inventory. Shipping objects know their origin, their destinations, and their primary goal of arriving at the destination on time with the lowest expense. Distribution objects designate which modes of transportation will be used to ship the inventory. Vehicle objects move the inventory. Using these three objects will help your business track and maintain correct inventory levels.

SCALABILITY AND EXPANDABILITY

Scalability refers to how well your system can adapt to the increased demands of additional users, more information to handle, and the need for faster processing speeds. When a system grows, the size of the database that stores the information grows. As the database grows, the system tends to perform procedures more slowly. ***Expandability*** refers to how easy it is to add features and functions to a system. If you design a system without thinking about its expandability, you'll run into major problems when your business starts to grow.

Take a look at Figure G.10. Can you explain why the ExpertSnowModel class was unable to be placed in a superclass/subclass relationship with the Snowboard class? As we discussed, it's important to take advantage of inheritance as it saves time and effort when developing a system. The reason the ExpertSnowModel class can't inherit the functionality from the Snowboard class is because of the procedure called *Deep Powder Specific*. *Deep Powder Specific* is a procedure that doesn't apply to all types of snowboards. Snowboards are designed for powder, ice, and all mountain terrain. For this reason, the ExpertSnowModel class can't be designated as a Snowboard subclass. If you did place the ExpertSnowModel class as a subclass of Snowboard, then an ExpertSnowModel would be able to perform the *Deep Powder Specific* procedure, a rather large system error. Defining superclasses that are too specific and thus not being able to take advantage of inheritance is a common mistake associated with object-oriented systems.

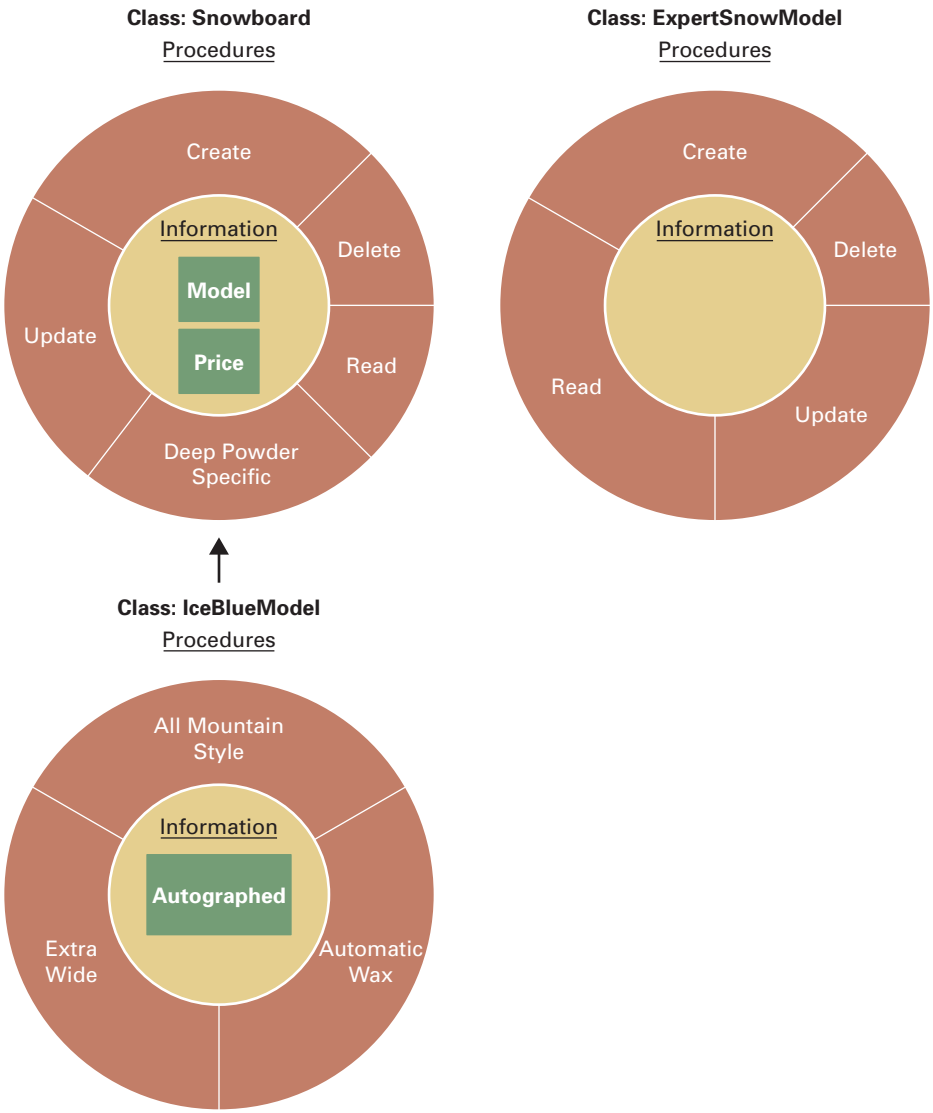


Figure G.10
Scalable Systems

Did you notice anything else wrong with the class diagram? The ExpertSnowModel class doesn't contain any information. This indicates a disconnect between the information and procedures. The ExpertSnowModel class contains the *Create, Read, Update,* and *Delete* procedures, but what are these procedures going to manipulate if there isn't any information? This would be a typical error using the traditional technology approach because information and procedures are viewed separately. Using the object-oriented approach, you notice this disconnect right away.

Types of Object-Oriented Technologies

LEARNING OUTCOME 5

Many technologies in use today support object-oriented concepts and techniques. Many more of these technologies are being developed every day as the business world continues to rush toward the use of objects. As a knowledge worker, it's important that you have some general knowledge of the different types of object-oriented technologies available and used throughout IT departments.

OBJECT-ORIENTED PROGRAMMING LANGUAGES

In general, programs are what make computers work. A ***program*** is a set of instructions that, when executed, cause a computer to behave in a specific manner. A program is almost like a recipe. It contains ingredients and directions (information and procedures) that tell the computer how to perform different tasks. Microsoft Word and Excel are both examples of programs.

A ***programming language*** is the tool developers use to write a program. Just as English, French, and Italian are different languages you can use to write a term paper, Java and C++ are two different types of languages you can use to write a program. An ***object-oriented programming language*** is a programming language used to develop object-oriented systems. Programming languages that are not object-oriented cannot handle classes, objects, messages, inheritance, or encapsulation. Currently, there are close to 100 different object-oriented programming languages available. The most popular of them today are Java, C++, Smalltalk, C#, VB.NET, and ASP.NET.

OBJECT-ORIENTED DATABASE SYSTEMS

Relational databases (the most popular, which you learned about in Chapter 3) organize information into fields, records, and files (or relations). ***Object-oriented databases*** work with traditional database information and also with complex data types such as diagrams, schematic drawings, video, and sound and text documents.

The relational database model, although it may allow you to store and view such data types, does not include good mechanisms for allowing you to manipulate and change information within those data types. For example, you can include a CAD drawing of a part as a field in a relational database, but it's literally impossible to work with any specific information in the drawing (such as cuts, specific components, and the ordering of assembly) without having that information also stored in other fields.

Another feature of object-oriented databases is that you are not restricted to two-dimensional tables. This gives you greater flexibility in storing and defining procedures that work with complex data types. In fact, most of today's multimedia applications rely on the use of objects and object-oriented databases.

Most other database models also restrict you to working with specific data types: alphabetic, numeric, decimal, currency, date, and so on. In an object-oriented database environment, you can create and work with data types that may be unique to a specific business process. For example, if you had an object that included an address you could easily design this field to include a street address and a unit number. You could then define a procedure that requires the entry of both pieces of information. This is an example of a unique data type that requires not only a street address but also a unit number.

OBJECT-ORIENTED TECHNOLOGIES AND CLIENT/SERVER ENVIRONMENTS

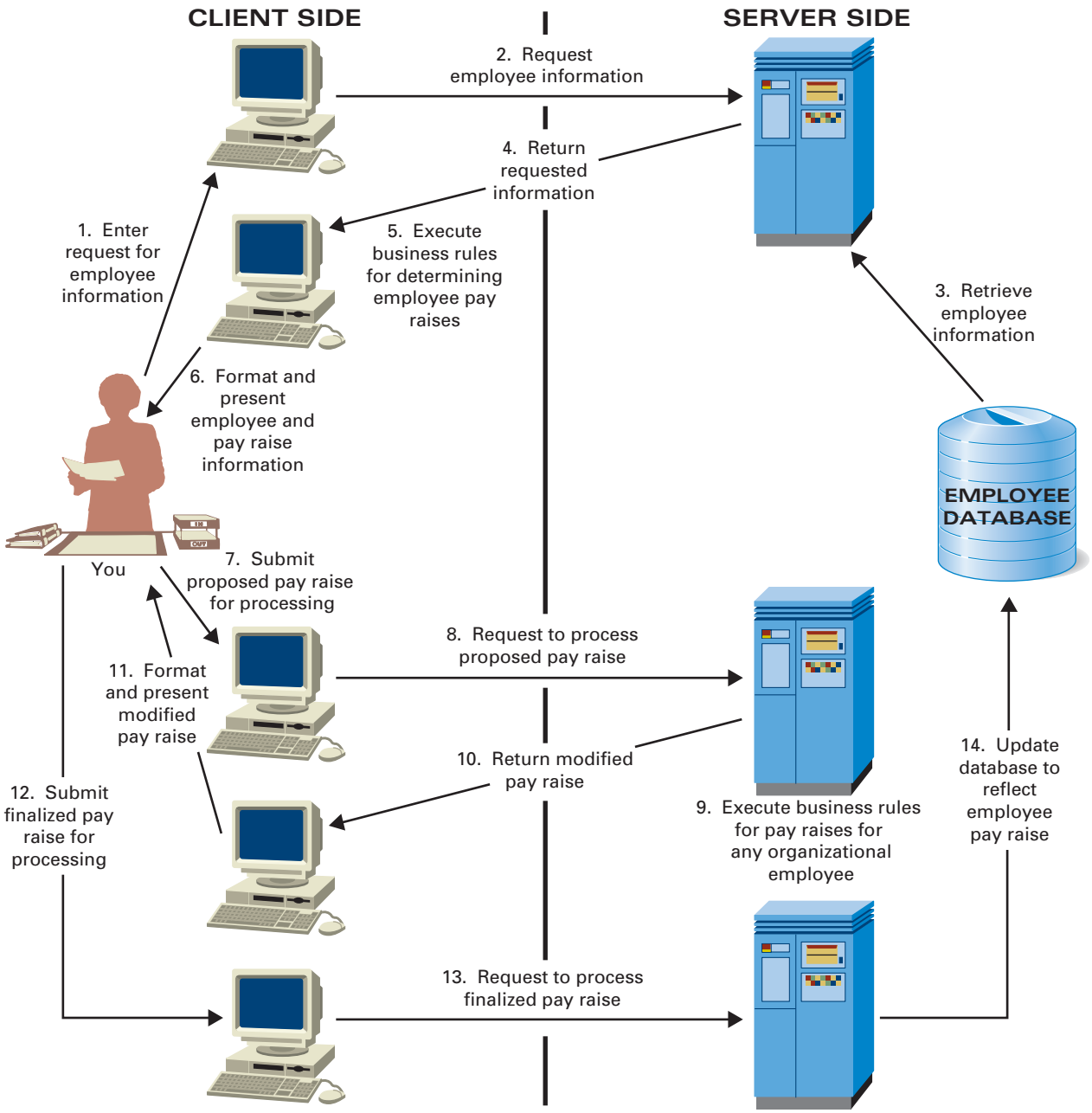
Client/server is the emerging blueprint for organizational networks, and most organizations are choosing to develop client/server networks through object-oriented technologies. A ***client/server network*** is a network in which one or more computers are servers and provide services to the other computers, which are called clients. Spreading objects across a client/server network makes logical sense: Client workstations contain objects with local procedures for working with local information, and servers contain objects with global procedures for working with global information.

In Figure G.11, the server handles the entire data management function, the client handles the entire presentation function, and both share in processing the logic or

business rules. So the server object contains procedures for retrieving and storing information (data management) and for processing some of the logic or business rules. Likewise, the client object contains procedures for some of the logic or business rules and for presenting information (the presentation function). To demonstrate, assume you are the manager of the manufacturing division in an organization and need to give pay raises to each of your employees. For assigning pay raises, you also have to follow several rules—some for just your division and some for all organizational employees.

In an object-oriented environment, your client workstation contains an object for assigning pay raises to manufacturing division employees according to manufacturing division rules. The server contains an object for assigning pay raises according to organizational rules and for retrieving information from and saving information to the database.

Figure G.11
Object-Oriented Technologies and Client/Server—The Perfect Match



Let's review how this would actually work. First, your client object, or Manufacturing Employee object, asks you for an employee and sends a message to the server object, or Organizational Employee object, to retrieve that employee's information from the database. Your client object then executes the rules for determining a pay raise for a manufacturing employee, displays that information to you, and sends the proposed pay raise to the server object. The server object then executes the organizational rules for assigning a pay raise and returns the modified pay raises to your client object. Your client object then displays that information to you and allows you to submit the finalized pay raises for processing to the server object. Finally, the server object updates the Employee database to reflect the employee's pay raise.

It's possible that while you are assigning pay raises to employees of your manufacturing division, various managers of other departments could be assigning pay raises for their employees. In this instance, you and any other managers essentially share the server object—another example of reuse.

THE FUTURE OF OBJECT-ORIENTED TECHNOLOGIES

The number of object-oriented development tools out there increases daily and thus understanding object-oriented concepts becomes ever more critical. In the future, object-oriented tools will perform tasks and provide functionality that we haven't even thought of yet. We hope this module has provided you with a solid understanding of object-oriented technologies and concepts behind them to prepare you for your job as a knowledge worker.

SUMMARY: STUDENT LEARNING OUTCOMES REVISITED

1. **Explain the primary difference between the traditional technology approach and the object-oriented technology approach.** The primary difference between the traditional technology approach and the object-oriented technology approach is the way information and procedures are viewed and developed. The ***traditional technology approach*** has two primary views of any system—information and procedures—and it keeps these two views separate and distinct at all times. The primary problem with this approach is that the separate views allow for information disconnects. For example, you might have all of the required information but not have the correct procedures to manipulate the information. The ***object-oriented approach*** combines information and procedures into a single view.
2. **List and describe the five primary object-oriented concepts.** The five primary object-oriented concepts are
   1. Information is any key characteristic stored within a system.
   2. ***Procedure*** manipulates or changes information.
   3. ***Class*** contains information and procedures and acts as a template to create objects (instances of a class).
   4. ***Object*** is an instance of a class.
   5. ***Messages*** are how objects communicate with each other. One object can send a message to another object asking it to perform a certain procedure.

   In general, the combination of information and procedures creates a class. A class creates objects, and objects communicate with other objects via messages. A system usually contains many classes, and many objects can be created from a single class.
3. **Explain how classes and objects are related.** Classes contain information and procedures and

are used as a template to create objects. The class is basically a blank template, which defines all of the different types of information the system can store about an object. An object can only be created from a class. Once you create an object from a class, you can fill in the template with the actual object information.

4. **Discuss the three fundamental principles of object-oriented technologies.** The three fundamental principles of object-oriented technologies are the following:
   - ***Inheritance*** is the ability to define superclass and subclass relationships among classes. The subclass inherits all the information and procedures from its superclass. Other names for superclass/subclass relationships include parent/child relationships and generalization/specification relationships.
   - ***Encapsulation*** means information hiding. A remote control is used to control AT&T's digital cable. The remote control hides the system information from the knowledge workers.
   - ***Polymorphism*** simply means to have many forms. Basically, polymorphism provides you with the ability to use the same word and have it mean different things. The word *bark* is an example of polymorphism. Bark can refer to a dog's bark or tree bark. The same word for a procedure can be used to mean different things in different classes.
5. **Describe two types of object-oriented technologies.** One type of object-oriented technology is an object-oriented programming language. An ***object-oriented programming language*** is a programming language used to develop object-oriented systems. For example, just as English, French, and Italian are different languages you can use to write a paper, Java and C++ are two different languages you can use to write a program. An object-oriented programming language must be used in order to develop an object-oriented system.

   A second type of object-oriented technology is an object-oriented database system. *Object-oriented databases* work with traditional database information and also with complex data types such as diagrams, schematic drawings, video, sound, and text documents.

KEY TERMS AND CONCEPTS

Class, G.6
Client/server network, G.18
CRUD, G.3
Encapsulation, G.12
Expandability, G.16
Information decomposition, G.7
Information view, G.2
Inheritance, G.10
Interface, G.12
Message, G.8
Object, G.7
Object-oriented approach, G.4
Object-oriented database, G.18
Object-oriented programming language, G.18
Polymorphism, G.13
Procedure, G.3
Procedure view, G.3
Program, G.18
Programming language, G.18
Scalability, G.16
Traditional technology approach, G.2

SHORT-ANSWER QUESTIONS

1. What is an example of a real-world object-oriented system?
2. What are classes?
3. Why would you use information decomposition?
4. What is an instance of a class?
5. How do objects and classes relate?
6. How do superclasses and subclasses relate?
7. What is polymorphism?
8. Why is scalability important when building a system?
9. What is a benefit of using the object-oriented approach?

10. What is another term for a superclass/subclass relationship?
11. Why is expandability important when building a system?
12. What is another term used to describe information hiding?

ASSIGNMENTS AND EXERCISES

1. **CLASSES IN THE CLASSROOM** Take a look around your classroom and on a piece of paper list 20 different objects located in the classroom. Objects might include desks, chairs, lights, students, and so forth. If you were going to build a classroom inventory tracking system, how many classes would you need to define in order to track all the objects? What would be the name of each class? What information and procedures would be stored in each class? On a separate sheet of paper, draw a class diagram displaying all the classes along with the different types of information and procedures they contain. Be sure to try to take advantage of inheritance. Take both sheets of paper and match each object to an appropriate class. If all of the objects match to a class, you created a successful system. Chances are every student's class diagram is probably going to be different because there is no right or wrong answer to this exercise, so be creative and have fun defining your classes. Be sure to look at some of the other students' diagrams to see how they defined their classes.
2. **TREES-R-US** You've been hired to build an inventory tracking system for the Trees-R-Us landscaping company. Trees-R-Us is excited about the use of inheritance in object-oriented systems and wants to see how you're going to use it in the system. Trees-R-Us has already defined the Tree, Grass, Flowers, Fence, Equipment, and Plant classes. Your job is to define all of the information and procedures for each class and the inheritance structure of the classes, or the superclass and subclass relationships. Please provide a class diagram that displays all the classes for the Trees-R-Us inventory tracking system and be sure to include inheritance. Again, there is no right or wrong answer for this exercise, so be creative and look at some of the other students' class diagrams to see how they defined their classes.
3. **OBJECT-ORIENTED CONCEPTS AND A REAL-WORLD SYSTEM** Create a list of a computer and all its parts. Be sure to include the monitor, keyboard, mouse, hard drive, disk drive, memory, CD/DVD drive, software, and printer. Write a brief explanation answering each of the following:
   - What types of information are associated with the computer?
   - What types of procedures are associated with the computer?
   - What parts of the computer, if any, represent the classes?
   - What parts of the computer, if any, represent the objects?
   - How many classes are there?
   - How many objects are there?
   - How are messages used?
   - How do all the components work together to create a complete system?
4. **UNDERSTANDING OBJECT-ORIENTED CONCEPTS AND TERMINOLOGY** Create a brief presentation explaining the primary differences between each of the following:
   - Traditional technology approach and object-oriented technology approach
   - Information and procedures
   - Classes and objects
   - Messages and interfaces
   - Encapsulation and inheritance
   - Generalization and specialization

   Feel free to use any of the figures located in this module. You can find them on the Web site that supports this text at www.mhhe.com/haag.
5. **EXPLAINING OBJECT-ORIENTED TECHNOLOGIES TO YOUR MANAGER** Assume you're working for a large oil and gas company. Your current manager has very little experience with object-oriented technologies and has asked you to write a paper describing, in generic terms, each of the following object-oriented concepts.

Be sure to include explanations of how using these object-oriented concepts will contribute to building and implementing successful information systems.

- Encapsulation
- Polymorphism
- Inheritance

6. **CLASSES AT THE VIDEO STORE** Consider your local video rental store. What would be three important classes? How many different objects do you think there are for each class? On a separate sheet of paper, draw a class diagram displaying all the classes along with the different types of information and procedures they contain. Do you think video rental stores in general use object-oriented systems? Why or why not?

7. **RESEARCHING JAVA** Java is a popular object-oriented programming language. It was developed by Sun Microsystems in the early to mid-1990s. Connect to *Information Week* at www.informationweek.com and search on the term "Java." Find a case study that interests you about a company that has used Java to implement an object-oriented system. Prepare a short report for your class detailing that case study. Now, do some more research on the Web. Although Java is currently a popular object-oriented programming language, there are some new ones on the horizon that will compete against it. Find one such new and emerging object-oriented programming language. What is it? How is it designed to compete with Java? Who provides it?

PHOTO CREDITS

Fig. G.5a, p. 10, PRNewsFoto/Cambridge SoundWorks/AP/Wide World Photos
Fig. G.5b, p. 10, Weinberg/Clark/Getty Images
Fig. G.5c, p. 10, © Nance Trueworthy
Fig. G.8a, p. 16, © David Madison Sports Images, 2005
Fig. G.8c, p. 16, © David Madison Sports Images, 2005