**CASE and OOP**

You hear about efficiency and productivity everywhere. They are particularly important for software development. Two resources that promise to help are *CASE tools* and *object-oriented software development.*

**CASE Tools** Professional programmers are constantly looking for ways to make their work easier, faster, and more reliable. One tool we mentioned in Chapter 13, CASE, is meeting this need. **Computer-aided software engineering (CASE) tools** provide some automation and assistance in program design, coding, and testing.

**Object-Oriented Software Development**

Traditional systems development is a careful, step-by-step approach focusing on the procedures needed to complete a certain objective. **Object-oriented software development** focuses less on the procedures and more on defining the relationships between previously defined procedures or “objects.” **Object-oriented programming (OOP)** is a process by which a program is organized into objects. Each **object** contains both the data and processing operations necessary to perform a task. Let’s explain what this means.

In the past, programs were developed as giant entities, from the first line of code to the last. This has been compared to building a car from scratch. Object-oriented programming is like building a car from prefabricated parts – carburetor, alternator, fenders, and so on. Object-oriented programs use objects that are reusable, self-contained components. Programs built with these objects assume that certain functions are the same. For example, many programs, from spreadsheets to database managers, have an instruction that will sort lists of names in alphabetical order. A programmer might use this object for alphabetizing in many other programs. There is no need to invent this activity anew every time. C++ is one of the most widely used object-oriented programming languages.

**CONCEPT CHECK**

What are CASE tools?

What is object-oriented software development?

What is object-oriented programming?

**Object-Oriented Programming Concepts**

If you've never used an object-oriented language before, you need to understand the underlying concepts before you begin writing code. You need to understand what an object is, what a class is, how objects and classes are related, and how objects communicate by using messages.

**What Is an Object?**

Objects are keys to understanding *object-oriented* technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

These real-world objects share two characteristics: They all have state and behavior. For example, dogs have state (name, color and breed, hungry) and behavior (barking, fetching, and wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels and number of gears) and behavior (braking, accelerating, slowing down and changing gears).

A software object maintains its state in one or more *variables*. A variable is an item of data named by an identifier. A software object implements its behavior with *methods*. A method is a function (subroutine) associated with an object.

***Definition:*** An object is a software bundle of variables and related methods.

You can represent real-world objects by using software objects. You can also use software objects to model abstract concepts. For example, an event is a common object used in GUI window systems to represent the action of a user pressing a mouse button or a key on the keyboard.

*Instance variables* contain the state for a particular bicycle object (speed, pedal cadence, and current gear), and in object-oriented terminology, a particular object is called an instance. In addition to its variables, the software bicycle would also have methods to brake, change the pedal cadence, and change gears. These methods are formally known as *instance methods* because they inspect or change the state of a particular bicycle instance.

**What Is a Message?**

A single object alone is generally not very useful. Instead, an object usually appears as a component of a larger program or application that contains many other objects. Your bicycle hanging from a hook in the garage is just a bunch of titanium alloy and rubber; by itself, the bicycle is incapable of any activity. The bicycle is useful only when another object (you) interacts with it (pedal).

Software objects interact and communicate with each other by sending messages to each other. When object A wants object B to perform one of B's methods, object A sends a message to object B

These are the three components that comprise a message:

1. The object to which the message is addressed (YourBicycle)
2. The name of the method to perform (changeGears)
3. Any parameters needed by the method (lowerGear)

**What Is a Class?**

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an *instance* of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle’s state is independent of and can be different from that of other bicycles.

When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same *blueprint*.

In object-oriented software, it’s also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. A software blueprint for objects is called a *class*.

**Definition:**A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

The class for our bicycle example would declare the instance variables necessary to contain the current gear, the current cadence, and so on, for each bicycle object. The class would also declare and provide implementations for the instance methods that allow the rider to change gears, brake, and change the pedaling cadence.