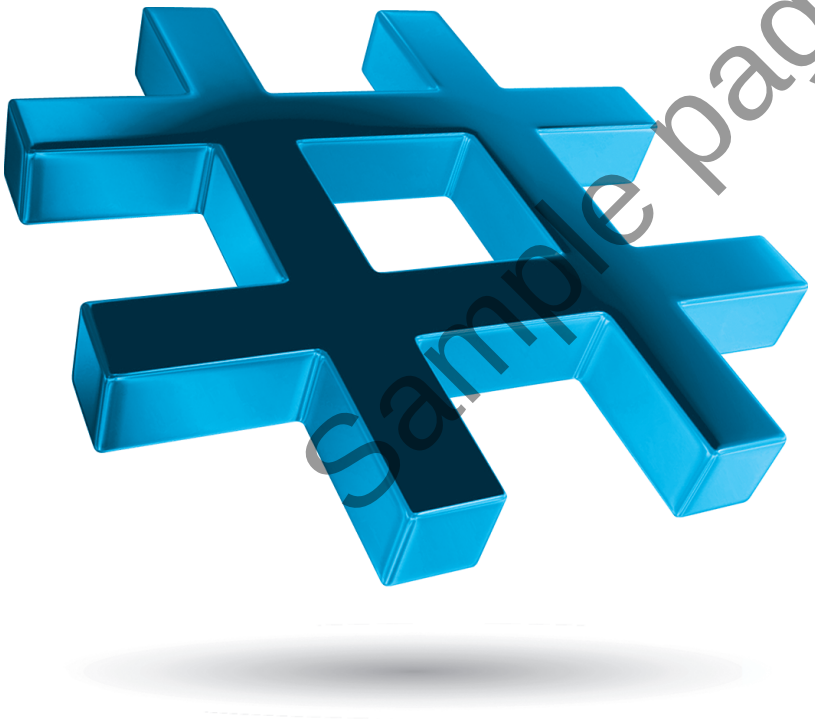
Introduction to Computers, the Internet and Visual C#

1



Objectives

In this chapter you'll:

- Learn basic computer hardware, software and data concepts.
- Be introduced to the different types of computer programming languages.
- Understand the history of the Visual C# programming language and the Windows operating system.
- Learn what cloud computing with Microsoft Azure is.
- Understand the basics of object technology.
- Be introduced to the history of the Internet and the World Wide Web.
- Understand the parts that Windows, .NET, Visual Studio and C# play in the C# ecosystem.
- Test-drive a Visual C# drawing app.

Outline

1.1 Introduction

Welcome to C#[1]—a powerful computer-programming language that's easy for novices to learn and that professionals use to build substantial computer applications. Using this book, you'll write instructions commanding computers to perform powerful tasks. *Software* (i.e., the instructions you write) controls *hardware* (i.e., computers and related devices).

There are billions of personal computers in use and an even larger number of mobile devices with computers at their core. Since it was released in 2001, C# has been used primarily to build applications for personal computers and systems that support them. The explosive growth of mobile phones, tablets and other devices also is creating significant opportunities for programming mobile apps. With this new sixth edition of *Visual C# How to Program*, you'll be able to use Microsoft's new Universal Windows Platform (UWP) with Windows 10 to build C# apps for both personal computers and Windows 10 Mobile devices. With Microsoft's purchase of Xamarin, you also can develop C# mobile apps for Android devices and for iOS devices, such as iPhones and iPads.

1.2 Computers and the Internet in Industry and Research

These are exciting times in the computer field! Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Hewlett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, eBay and many more. These companies are major employers of people who study computer science, computer engineering, information systems or related disciplines. At the time of this writing, Google's parent company, Alphabet, and Apple were the two most

1. The name C#, pronounced "C-sharp," is based on the musical # notation for "sharp" notes.

valuable companies in the world. Figure 1.1 provides a few examples of the ways in which computers are improving people's lives in research, industry and society.

Name	Description
Electronic health records	These might include a patient's medical history, prescriptions, immunizations, lab results, allergies, insurance information and more. Making these available to health-care providers across a secure network improves patient care, reduces the probability of error and increases the health-care system's overall efficiency, helping control costs.
Human Genome Project	The Human Genome Project was founded to identify and analyze the 20,000+ genes in human DNA. The project used computer programs to analyze complex genetic data, determine the sequences of the billions of chemical base pairs that make up human DNA and store the information in databases, which have been made available over the Internet to researchers in many fields.
AMBER™ Alert	The AMBER (America's Missing: Broadcast Emergency Response) Alert System helps find abducted children. Law enforcement notifies TV and radio broadcasters and state transportation officials, who then broadcast alerts on TV, radio, computerized highway signs, the Internet and wireless devices. AMBER Alert partners with Facebook, whose users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.
World Community Grid	People worldwide can donate their unused computer processing power by installing a free secure software program that allows the World Community Grid (`http://www.worldcommunitygrid.org`) to harness unused capacity. This computing power, accessed over the Internet, is used in place of expensive supercomputers to conduct scientific research projects that are making a difference—providing clean water to third-world countries, fighting cancer, growing more nutritious rice for regions fighting hunger and more.
Cloud computing	**Cloud computing** allows you to use software, hardware and information stored in the "cloud"—i.e., accessed on remote computers via the Internet and available on demand—popular examples are Dropbox, Google Drive and Microsoft OneDrive. You can increase or decrease resources incrementally to meet your needs at any given time, so cloud services can be more cost effective than purchasing expensive hardware to ensure that you have enough storage and processing power to meet peak-level needs. Using cloud-computing services shifts the burden of managing these applications from the business to the service provider, saving businesses time, effort and money. In an online chapter, you'll use **Microsoft Azure**—a cloud-computing platform that allows you to develop, manage and distribute your apps in the cloud. With Microsoft Azure, your apps can store their data in the cloud so that it's available at all times from any of your desktop computers and mobile devices. For information on Microsoft Azure's free and paid services visit `https://azure.microsoft.com`.
Medical imaging	X-ray computed tomography (CT) scans, also called CAT (computerized axial tomography) scans, take X-rays of the body from hundreds of different angles. Computers are used to adjust the intensity of the X-rays, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image. MRI scanners use a technique called magnetic resonance imaging to produce internal images noninvasively.

Fig. 1.1 | Improving people's lives with computers. (Part 1 of 2.)

Name	Description
GPS	Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time-stamped signals to the GPS device, which calculates the distance to each satellite, based on the time the signal left the satellite and the time the signal arrived. This information helps determine the device's exact location. GPS devices can provide step-by-step directions and help you locate nearby businesses (restaurants, gas stations, etc.) and points of interest. GPS is used in numerous location-based Internet services such as check-in apps to help you find your friends (e.g., Foursquare and Facebook), exercise apps such as Map My Ride+, Couch to 5K and RunKeeper that track the time, distance and average speed of your outdoor ride or jog, dating apps that help you find a match nearby and apps that dynamically update changing traffic conditions.
Robots	Robots can be used for day-to-day tasks (e.g., iRobot's Roomba vacuuming robot), entertainment (e.g., robotic pets), military combat, deep sea and space exploration (e.g., NASA's Mars rover Curiosity) and more. Researchers, such as those at RoboHow (`http://robohow.eu`), are working to create autonomous robots that perform complex human manipulation tasks (such as cooking) and that can learn additional tasks both from the robots' own experiences and from observing humans performing other tasks.
E-mail, Instant Messaging and Video Chat	Internet-based servers support all of your online messaging. E-mail messages go through a mail server that also stores the messages. Instant Messaging (IM) and Video Chat apps, such as Facebook Messenger, WhatsApp, AIM, Skype, Yahoo! Messenger, Google Hangouts, Trillian and others, allow you to communicate with others in real time by sending your messages and live video through servers.
E-commerce	This technology has exploded with companies like Amazon, eBay, Alibaba, Walmart and many others, causing a major shift away from brick-and-mortar retailers.
Internet TV	Internet TV set-top boxes (such as Apple TV, Android TV, Roku, Chromecast and TiVo) allow you to access an enormous amount of content on demand, such as games, news, movies, television shows and more, and they help ensure that the content is streamed to your TV smoothly.
Streaming music services	Streaming music services (such as Apple Music, Pandora, Spotify and more) allow you to listen to large catalogues of music over the web, create customized "radio stations" and discover new music based on your feedback.
Self-driving cars and smart homes	These are two enormous markets. Self-driving cars are under development by many technology companies and car manufacturers—they already have an impressive safety record and soon could be widely used saving lives and reducing injuries. Smart homes use computers for security, climate control, minimizing energy costs, automated lighting systems, fire detection, window control and more.
Game programming	Global video-game revenues are expected to reach $107 billion by 2017 (`http://www.polygon.com/2015/4/22/8471789/worldwide-video-games-market-value-2015`). The most sophisticated games can cost over $100 million to develop, with the most expensive costing half a billion dollars (`http://www.gamespot.com/gallery/20-of-the-most-expensive-games-ever-made/2900-104/`). Bethesda's *Fallout 4* earned $750 million in its first day of sales (`http://fortune.com/2015/11/16/fallout4-is-quiet-best-seller/`)!

Fig. 1.1 | Improving people's lives with computers. (Part 2 of 2.)

1.3 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second! China's National University of Defense Technology's Tianhe-2 supercomputer can perform over 33 quadrillion calculations per second (33.86 *petaflops*)![2] To put that in perspective, *the Tianhe-2 supercomputer can perform in one second about 3 million calculations for every person on the planet!* And supercomputing upper limits are growing quickly.

Computers (i.e., **hardware**) process *data* under the control of sequences of instructions called **computer programs**. These programs guide the computer through *actions* specified by people called **computer programmers**. The programs that run on a computer are referred to as **software**. In this book, you'll learn several key programming methodologies that are enhancing programmer productivity, thereby reducing software development costs—*object-oriented programming*, *generic programming*, *functional programming* and *structured programming*. You'll build C# apps (short for applications) for a variety of environments including the *desktop*, mobile devices like *smartphones* and *tablets*, and even "the *cloud*."

Computers consist of devices referred to as hardware (e.g., the keyboard, screen, mouse, hard disks, memory, DVD drives and processing units). Computing costs are *dropping dramatically*, due to rapid developments in hardware and software technologies. Computers that filled large rooms and cost millions of dollars decades ago are now inscribed on silicon chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

1.3.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the hardware supporting these technologies. For many decades, hardware costs have fallen rapidly.

Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel—a leading manufacturer of the processors in today's computers and embedded systems. Moore's Law and related observations apply especially to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which they *execute* their programs (i.e., do their work). These increases make computers more capable, which puts greater demands on programming-language designers to innovate.

Similar growth has occurred in the communications field—costs have plummeted as enormous demand for communications *bandwidth* (i.e., information-carrying capacity) has attracted intense competition. We know of no other fields in which technology improves so quickly and costs fall so rapidly. Such phenomenal improvement is truly fostering the *Information Revolution*.

2. `http://www.top500.org`.

1.3.2 Computer Organization

Regardless of differences in *physical* appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.2).

Logical unit	Description
Input unit	This "receiving" section obtains information (data and computer programs) from **input devices** and places it at the disposal of the other units for processing. Most user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, motion and orientation information from an *accelerometer* (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or game controller (such as Microsoft® Kinect® for Xbox®, Wii™ Remote and Sony® PlayStation® Move) and voice input from devices like Amazon Echo and the forthcoming Google Home.
Output unit	This "shipping" section takes information the computer has processed and places it on various **output devices** to make it available for use outside the computer. Most information that's output from computers today is displayed on screens (including touch screens), printed on paper ("going green" discourages this), played as audio or video on PCs and media players (such as Apple's iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and "intelligent" appliances. Information is also commonly output to secondary storage devices, such as solid-state drives (SSDs), hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game-controller vibration, virtual reality devices like Oculus Rift and Google Cardboard and mixed reality devices like Microsoft's HoloLens.
Memory unit	This rapid-access, relatively low-capacity "warehouse" section retains information that has been entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is *volatile*—it's typically lost when the computer's power is turned off. The memory unit is often called either **memory**, **primary memory** or **RAM** (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 2 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A **byte** is eight bits. A bit is either a `0` or a `1`.
Arithmetic and logic unit (ALU)	This "manufacturing" section performs *calculations*, such as addition, subtraction, multiplication and division. It also contains the *decision* mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they're equal. In today's systems, the ALU is implemented as part of the next logical unit, the CPU.

Fig. 1.2 | Logical units of a computer. (Part 1 of 2.)

Logical unit	Description
Central processing unit (CPU)	This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today's computers have multiple CPUs and, hence, can perform many operations simultaneously. A multicore processor implements multiple processors on a single integrated-circuit chip—a *dual-core processor* has two CPUs, a *quad-core processor* has four and an *octa-core processor* has eight. Today's desktop computers have processors that can execute billions of instructions per second. Chapter 23 explores how to write apps that can take full advantage of multicore architecture.
Secondary storage unit	This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your *hard drive*) until they're again needed, possibly hours, days, months or even years later. Information on secondary storage devices is *persistent*—it's preserved even when the computer's power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per unit is much less. Examples of secondary storage devices include solid-state drives (SSDs), hard drives, DVD drives and USB flash drives, some of which can hold over 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 2 TB, and some desktop hard drives can hold up to 6 TB.

Fig. 1.2 | Logical units of a computer. (Part 2 of 2.)

1.4 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from the simplest data items (called "bits") to richer data items, such as characters, fields, and so on. Figure 1.3 illustrates a portion of the data hierarchy.

Bits

The smallest data item in a computer can assume the value `0` or the value `1`. It's called a **bit** (short for "binary digit"—a digit that can assume one of *two* values). Remarkably, the impressive functions performed by computers involve only the simplest manipulations of `0`s and `1`s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from `1` to `0` or from `0` to `1`).

Characters

It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., $, @, %, &, *, (,), –, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only `1`s and `0`s, so a computer's character set represents every character as a pattern of `1`s and `0`s. C# supports various character sets (including **Unicode**®), with

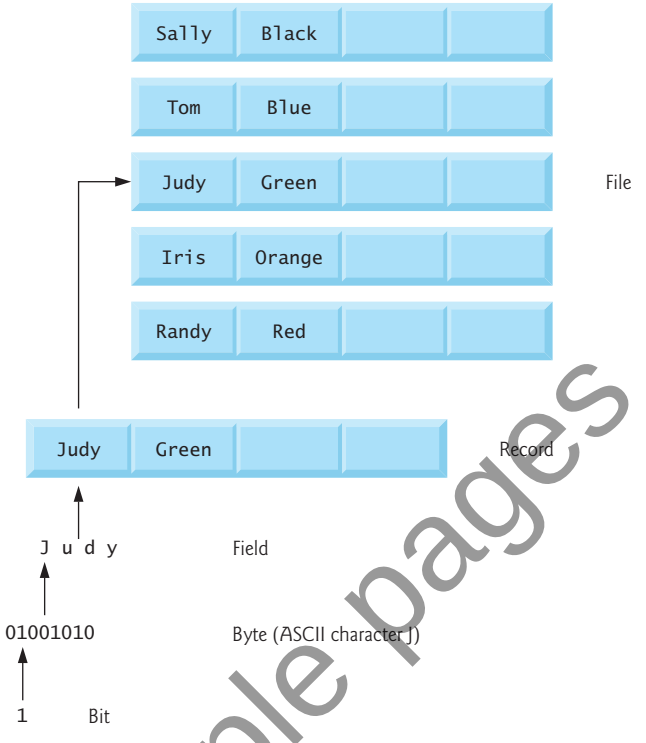


Fig. 1.3 | Data hierarchy.

some requiring more than one byte per character. Unicode supports many of the world's languages, as well as emojis. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters of the English alphabet, digits and some common special characters. We also provide an online appendix describing Unicode.

Fields

Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters can be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.

Records

Several related fields can be used to compose a **record**. In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses):

- Employee or student identification number (a whole number).
- Name (a string of characters).
- Address (a string of characters).

- Hourly pay rate (a number with a decimal point).
- Year-to-date earnings (a number with a decimal point).
- Amount of taxes withheld (a number with a decimal point).

Thus, a record is a group of related fields. In the preceding example, all the fields belong to the *same* employee. A company might have many employees and a payroll record for each.

To facilitate the retrieval of specific records from a file, at least one field in each record is chosen as a **record key**, which identifies a record as belonging to a particular person or entity and distinguishes that record from all others. For example, in a payroll record, the employee identification number normally would be the record key.

Files

A **file** is a group of related records. More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer. It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Database

A **database** is a collection of data organized for easy access and manipulation. The most popular model is the *relational database*, in which data is stored in simple *tables*. A table includes *records* and *fields*. For example, a table of students might include first name, last name, major, year, student ID number and grade-point-average fields. The data for each student is a record, and the individual pieces of information in each record are the fields. You can *search*, *sort* and otherwise manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with data from databases of courses, on-campus housing, meal plans, etc.

Big Data

The amount of data being produced worldwide is enormous and growing quickly. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily,[3] and according to Salesforce.com, as of October 2015 90% of the world's data was created in just the prior 12 months![4] According to an IDC study, the global data supply will reach 40 *zettabytes* (equal to 40 trillion gigabytes) annually by 2020.[5] Figure 1.4 shows some common byte measurements. **Big data** applications deal with massive amounts of data and this field is growing quickly, creating lots of opportunity for software developers. According to a study by Gartner Group, over four million IT jobs globally were expected to support big data in 2015.[6]

3. http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.
4. https://www.salesforce.com/blog/2015/10/salesforce-channel-ifttt.html.
5. http://recode.net/2014/01/10/stuffed-why-data-storage-is-hot-again-really/.
6. http://fortune.com/2013/09/04/the-big-data-employment-boom/.

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 (1024) bytes exactly
1 megabyte (MB)	1024 kilobytes	10^6 (1,000,000) bytes
1 gigabyte (GB)	1024 megabytes	10^9 (1,000,000,000) bytes
1 terabyte (TB)	1024 gigabytes	10^{12} (1,000,000,000,000) bytes
1 petabyte (PB)	1024 terabytes	10^{15} (1,000,000,000,000,000) bytes
1 exabyte (EB)	1024 petabytes	10^{18} (1,000,000,000,000,000,000) bytes
1 zettabyte (ZB)	1024 exabytes	10^{21} (1,000,000,000,000,000,000,000) bytes

Fig. 1.4 | Byte measurements.

1.5 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps.

Machine Languages

Any computer can directly understand only its own **machine language** (also called *machine code*), defined by its hardware architecture. Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.

Assembly Languages

Programming in machine language was simply too slow and tedious for most programmers. Instead, they began using English-like *abbreviations* to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert assembly-language programs to machine language. Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language. Assembly languages are still popular today in applications where minimizing memory use and maximizing execution efficiency is crucial.

High-Level Languages

To speed up the programming process further, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. High-level languages, such as C#, Visual Basic, C, C++, Java and Swift, allow you to write instructions that look more like everyday English and contain commonly used mathematical notations. Translator programs called **compilers** convert high-level language programs into machine language.

The process of compiling a large high-level-language program into machine language can take a considerable amount of computer time. **Interpreter** programs were developed to execute high-level language programs directly (without the need for compilation), although more slowly than compiled programs. **Scripting languages** such as the popular web languages JavaScript and PHP are processed by interpreters.

Performance Tip 1.1

Interpreters have an advantage over compilers in Internet scripting. An interpreted program can begin executing as soon as it's downloaded to the client's machine, without needing to be compiled before it can execute. On the downside, interpreted scripts generally run slower and consume more memory than compiled code. With a technique called JIT (just-in-time) compilation, interpreted languages can often run almost as fast as compiled ones.

1.6 Object Technology

C# is an object-oriented programming language. In this section we'll introduce the basics of object technology.

Building software quickly, correctly and economically remains an elusive goal at a time when demands for new and more powerful software are soaring. **Objects**, or more precisely—as we'll see in Chapter 4—the **classes** objects come from, are essentially *reusable* software components. There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc. Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating). Software developers have discovered that using a modular, object-oriented design-and-implementation approach can make software-development groups much more productive than was possible with earlier techniques—object-oriented programs are often easier to understand, correct and modify.

The Automobile as an Object

Let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*. What must happen before you can do this? Well, before you can drive a car, someone has to *design* it. A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house. These drawings include the design for an accelerator pedal. The pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel *hides* the mechanisms that turn the car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Before you can drive a car, it must be *built* from the engineering drawings that describe it. A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Methods and Classes

Let's use our car example to introduce some key object-oriented programming concepts. Performing a task in a program requires a **method**. The method houses the program statements that actually perform the task. It *hides* these statements from its user, just as a car's accelerator pedal hides from the driver the mechanisms of making the car go faster. In C#, we create a program unit called a class to house the set of methods that perform the class's tasks. For example, a class that represents a bank account might contain one method to *deposit* money to an account and another to *withdraw* money from an account. A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Making Objects from Classes

Just as someone has to *build a car* from its engineering drawings before you can actually drive a car, you must *build an object* from a class before a program can perform the tasks that the class's methods define. The process of doing this is called *instantiation*. An object is then referred to as an **instance** of its class.

Reuse

Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects. Reuse of existing classes when building new classes and programs saves time and effort. Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive *testing* (to locate problems), *debugging* (to correct those problems) and *performance tuning*. Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that's been spurred by object technology.

Messages and Method Calls

When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster. Similarly, you *send messages to an object*. Each message is implemented as a **method call** that tells a method of the object to perform its task. For example, a program might call a particular bank-account object's *deposit* method to increase the account's balance.

Attributes and Instance Variables

A car, besides having capabilities to accomplish tasks, also has *attributes*, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading). Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams (which, for example, include an odometer and a fuel gauge). As you drive an actual car, these attributes are carried along with the car. Every car maintains its *own* attributes. For example, each car knows how much gas is in its own gas tank, but not how much is in the tanks of *other* cars.

An object, similarly, has attributes that it carries along as it's used in a program. These attributes are specified as part of the object's class. For example, a bank-account object has a *balance attribute* that represents the amount of money in the account. Each bank-account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank. Attributes are specified by the class's **instance variables**.

Properties, `get` *Accessors and* `set` *Accessors*

Attributes are not necessarily accessible directly. The car manufacturer does not want drivers to take apart the car's engine to observe the amount of gas in its tank. Instead, the driver can check the fuel gauge on the dashboard. The bank does not want its customers to walk into the vault to count the amount of money in an account. Instead, the customers talk to a bank teller or check personalized online bank accounts. Similarly, you do not need to have access to an object's instance variables in order to use them. You should use the **properties** of an object. Properties contain **`get` accessors** for reading the values of variables, and **`set` accessors** for storing values into them.

Encapsulation

Classes **encapsulate** (i.e., wrap) attributes and methods into objects created from those classes—an object's attributes and methods are intimately related. Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves. This **information hiding**, as we'll see, is crucial to good software engineering.

Inheritance

A new class of objects can be created quickly and conveniently by **inheritance**—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own. In our car analogy, an object of class "convertible" certainly *is an* object of the more *general* class "automobile," but more *specifically*, the roof can be raised or lowered.

Object-Oriented Analysis and Design (OOAD)

Soon you'll be writing programs in C#. How will you create the **code** (i.e., the program instructions) for your programs? Perhaps, like many programmers, you'll simply turn on your computer and start typing. This approach may work for small programs (like the ones we present in the early chapters of the book), but what if you were asked to create a software system to control thousands of automated teller machines for a major bank? Or suppose you were asked to work on a team of thousands of software developers building the next generation of the U.S. air traffic control system? For projects so large and complex, you should not simply sit down and start writing programs.

To create the best solutions, you should follow a detailed **analysis** process for determining your project's **requirements** (i.e., defining *what* the system is supposed to do) and developing a **design** that satisfies them (i.e., deciding *how* the system should do it). Ideally, you'd go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it's called an **object-oriented analysis and design (OOAD) process**. Languages like C# are object oriented—programming in such a language, called **object-oriented programming (OOP)**, allows you to implement an object-oriented design as a working system.

The UML (Unified Modeling Language)

Although many different OOAD processes exist, a single graphical language for communicating the results of *any* OOAD process has come into wide use. This language, known as the Unified Modeling Language (UML), is now the most widely used graphical scheme for modeling object-oriented systems. We present our first UML diagrams in Chapters 4 and 5, then use them in our deeper treatment of object-oriented programming through Chapter 12. In our *optional* ATM Software Engineering Case Study in the online chapters, we present a simple subset of the UML's features as we guide you through an object-oriented design and implementation experience.

1.7 Internet and World Wide Web

In the late 1960s, ARPA—the Advanced Research Projects Agency of the United States Department of Defense—rolled out plans for networking the main computer systems of

approximately a dozen ARPA-funded universities and research institutions. The computers were to be connected with communications lines operating at speeds on the order of 50,000 bits per second, a stunning rate at a time when most people (of the few who even had networking access) were connecting over telephone lines to computers at a rate of 110 bits per second. Academic research was about to take a giant leap forward. ARPA proceeded to implement what quickly became known as the ARPANET, the precursor to today's **Internet**. Today's fastest Internet speeds are on the order of billions of bits per second with trillion-bits-per-second speeds on the horizon!

Things worked out differently from the original plan. Although the ARPANET enabled researchers to network their computers, its main benefit proved to be the capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on today's Internet, with e-mail, instant messaging, file transfer and social media such as Facebook and Twitter enabling billions of people worldwide to communicate quickly and easily.

The protocol (set of rules) for communicating over the ARPANET became known as the **Transmission Control Protocol (TCP)**. TCP ensured that messages, consisting of sequentially numbered pieces called *packets*, were properly routed from sender to receiver, arrived intact and were assembled in the correct order.

The Internet: A Network of Networks

In parallel with the early evolution of the Internet, organizations worldwide were implementing their own networks for both intraorganization (that is, within an organization) and interorganization (that is, between organizations) communication. A huge variety of networking hardware and software appeared. One challenge was to enable these different networks to communicate with each other. ARPA accomplished this by developing the **Internet Protocol (IP)**, which created a true "network of networks," the current architecture of the Internet. The combined set of protocols is now called **TCP/IP**.

Businesses rapidly realized that by using the Internet, they could improve their operations and offer new and better services to their clients. Companies started spending large amounts of money to develop and enhance their Internet presence. This generated fierce competition among communications carriers and hardware and software suppliers to meet the increased infrastructure demand. As a result, **bandwidth**—the information-carrying capacity of communications lines—on the Internet has increased tremendously, while hardware costs have plummeted.

The World Wide Web: Making the Internet User-Friendly

The **World Wide Web** (simply called "the web") is a collection of hardware and software associated with the Internet that allows computer users to locate and view multimedia-based documents (documents with various combinations of text, graphics, animations, audios and videos) on almost any subject. In 1989, Tim Berners-Lee of CERN (the European Organization for Nuclear Research) began developing **HyperText Markup Language (HTML)**—the technology for sharing information via "hyperlinked" text documents. He also wrote communication protocols such as **HyperText Transfer Protocol (HTTP)** to form the backbone of his new hypertext information system, which he referred to as the World Wide Web.

In 1994, Berners-Lee founded the **World Wide Web Consortium** (**W3C**, `http://www.w3.org`), devoted to developing web technologies. One of the W3C's primary goals

is to make the web universally accessible to everyone regardless of disabilities, language or culture.

Web Services

Web services are software components stored on one computer that can be accessed by an app (or other software component) on another computer over the Internet. With web services, you can create *mashups*, which enable you to rapidly develop apps by combining complementary web services, often from multiple organizations, and possibly other forms of information feeds. For example, 100 Destinations (`http://www.100destinations.co.uk`) combines the photos and tweets from Twitter with the mapping capabilities of Google Maps to allow you to explore countries around the world through the photos of others.

ProgrammableWeb (`http://www.programmableweb.com/`) provides a directory of over 15,000 APIs and 6,200 mashups, plus how-to guides and sample code for creating your own mashups. According to Programmableweb, the three most widely used APIs for mashups are Google Maps, Twitter and YouTube.

Ajax

Ajax technology helps Internet-based applications perform like desktop applications—a difficult task, given that such applications suffer transmission delays as data is shuttled back and forth between your computer and server computers on the Internet. Using Ajax, applications like Google Maps have achieved excellent performance, approaching the look-and-feel of desktop applications.

The Internet of Things

The Internet is no longer just a network of computers—it's an Internet of Things. A *thing* is any object with an IP address—a unique identifier that helps locate that *thing* on the Internet—and the ability to send data automatically over the Internet. Such things include:

- a car with a transponder for paying tolls,
- monitors for parking-space availability in a garage,
- a heart monitor implanted in a human,
- monitors for drinkable water quality,
- a smart meter that reports energy usage,
- radiation detectors,
- item trackers in a warehouse,
- mobile apps that can track your movement and location,
- smart thermostats that adjust room temperatures based on weather forecasts and activity in the home
- and many more.

1.8 C#

In 2000, Microsoft announced the C# programming language. C# has roots in the C, C++ and Java programming languages. It has similar capabilities to Java and is appropriate for

the most demanding app-development tasks, especially for building today's desktop apps, large-scale enterprise apps, and web-based, mobile and cloud-based apps.

1.8.1 Object-Oriented Programming

C# is *object oriented*—we've discussed the basics of object technology and we present a rich treatment of object-oriented programming throughout the book. C# has access to the powerful **.NET Framework Class Library**—a vast collection of prebuilt classes that enable you to develop apps quickly (Fig. 1.5). We'll say more about .NET in Section 1.9.

Some key capabilities in the .NET Framework Class Library	
Database	Debugging
Building web apps	Multithreading
Graphics	File processing
Input/output	Security
Computer networking	Web communication
Permissions	Graphical user interface
Mobile	Data structures
String processing	Universal Windows Platform GUI

Fig. 1.5 | Some key capabilities in the .NET Framework Class Library.

1.8.2 Event-Driven Programming

C# graphical user interfaces (GUIs) are **event driven**. You can write programs that respond to user-initiated **events** such as mouse clicks, keystrokes, timer expirations and *touches* and *finger swipes*—gestures that are widely used on smartphones and tablets.

1.8.3 Visual Programming

Microsoft's Visual Studio enables you to use C# as a *visual programming language*—in addition to writing program statements to build portions of your apps, you'll also use Visual Studio to conveniently drag and drop predefined GUI objects like *buttons* and *textboxes* into place on your screen, and label and resize them. Visual Studio will write much of the GUI code for you.

1.8.4 Generic and Functional Programming

Generic Programming

It's common to write a program that processes a collection of things—e.g., a collection of numbers, a collection of contacts, a collection of videos, etc. Historically, you had to program separately to handle each type of collection. With *generic programming*, you write code that handles a collection "in the general" and C# handles the specifics for each different type of collection, saving you a great deal of work. We'll study generics and generic collections in Chapters 20 and 21.