Connecting with Computer Science, 2e

Chapter 14

Programming I

Objectives

- In this chapter you will:
  - Learn what a program is and how it's developed
  - Understand the difference between a low-level and high-level language
  - Be introduced to low-level languages, using assembly language as an example
  - Learn about program structure, including algorithms and pseudocode

Objectives (cont'd.)

- In this chapter you will (cont'd.):
  - Learn about variables and how they're used
  - Explore the control structures used in programming
  - Understand the terms used in object-oriented programming

Why You Need to Know About... Programming

- Examples of programs in everyday functions:
  - Cars, space shuttles, ATMs, and microwaves
- It is important to develop a quality programming product
  - People depend on it
- Programming is essential to future computing career

What Is a Program?

- Program:
  - Collection of statements or steps
    - Solves a problem
    - Converted into a language the computer understands to perform tasks
- Algorithm:
  - Logically ordered set of statements
    - Used to solve a problem
- Interpreter:
  - Translates program's statements into a language the computer understands

What Is a Program? (cont'd.)

- Compiler:
  - Application reading all the program's statements
    - Converts them into computer language
    - Produces an executable file running independently of an interpreter
- Programs are developed to help perform tasks
  - Communicate with users to meet their needs
- Causes of program failure:
  - Piece of logical functionality left out of the program
  - Contains logic errors in one or more statements

I Speak Computer

- First step in programming
  - Determine language to communicate with the computer
  - Computers only speak binary
- Many choices:
  - Ada, Assembly, C, C++, C#
  - COBOL, FORTRAN, Delphi (Pascal)
  - Java and JavaScript
  - Lisp, Perl, Smalltalk, Visual Basic
- Each has its own strengths and weaknesses

I Speak Computer (cont'd.)

- Low-level language:
  - Uses binary code for instructions
- Machine language:
  - Lowest-level programming language
    - Consists of binary bit patterns
- Assembly language:
  - One step up from machine language
    - Assigns letter codes to each machine-language instruction

I Speak Computer (cont'd.)

- Assembler:
  - Program that reads assembly-language code and converts it into machine language
- High-level language:
  - Written in a more natural language that humans can read and understand

I Speak Computer (cont'd.)

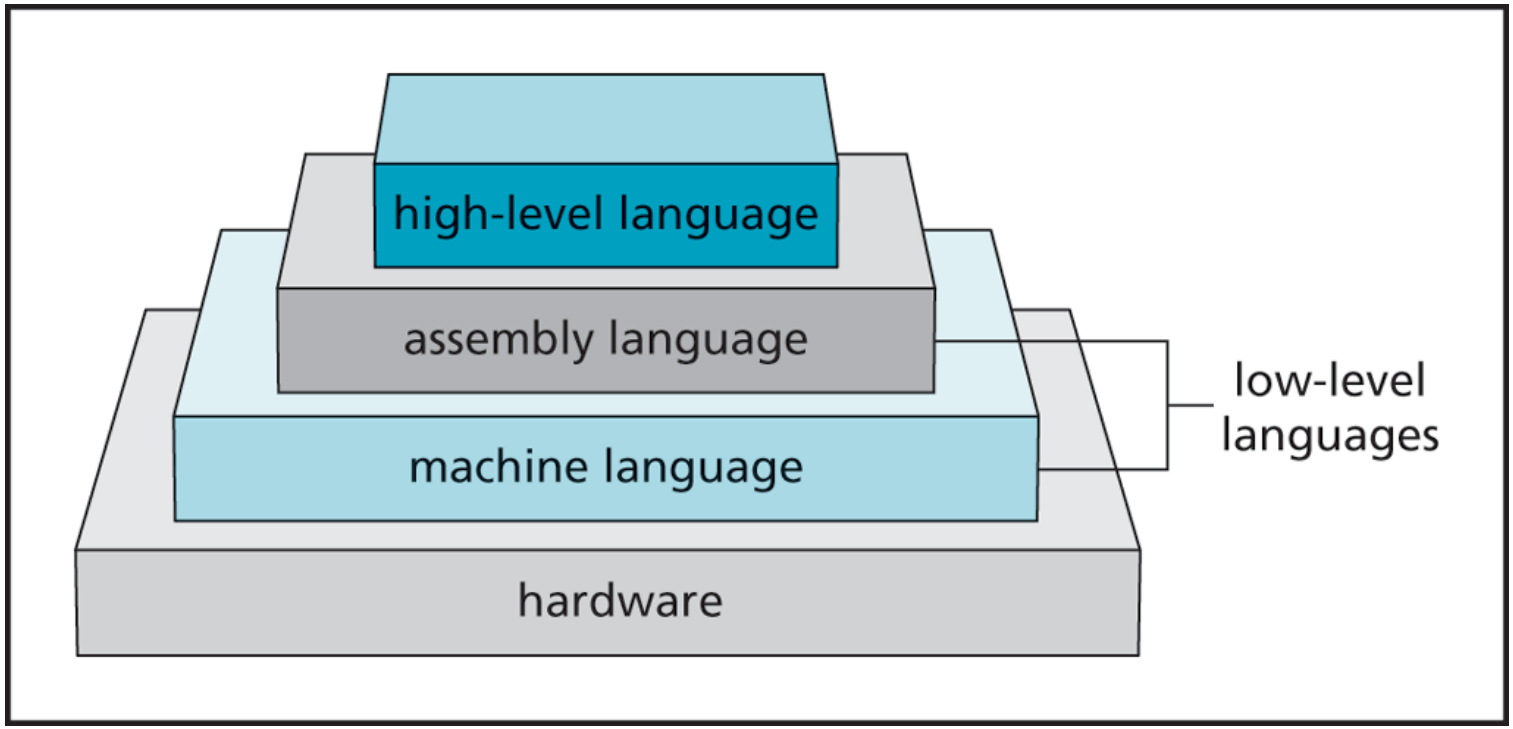

Figure 14-1, Different types of programming languages

Low-Level Languages

- Few people code in machine language
- Assembly language:
  - Simulates machine language
  - Written with more English-like statements
  - Advantages:
    - Corresponds to one machine instruction
    - Programs are usually smaller and run faster than programs in higher-level languages
    - Powerful
  - Closely tied to the CPU type
    - Assemblers written for every type of CPU

Assembly-Language Statements

- Registers in a CPU
  - Special memory locations for storing information programs can use
  - Registers: AX, BX, CX, and DX
    - General-purpose registers (GPRs)
    - Used mainly for arithmetic operations or accessing an element in an array
- Consists of text instructions
  - Converted one by one into machine (binary) instructions
- Disadvantage: hard to read and understand

Assembly-Language Statements (cont'd.)

- Syntax:
  - Rules for how a programming language's statements must be constructed
- `mov`: moves values around
  - Example: move the value of 8 into the CX register
    - `mov cx, 8`
  - Can move a value from memory to a register, from a register to memory, from register to register
    - `mov dx, cx`

Assembly-Language Statements (cont'd.)

- `add`: takes a value on the right and adds it to the value on the left
  - Example: storing value of 11 in DX register
    - `mov cx, 3`
    - `mov dx, 8`
    - `add dx, cx`
- `inc`: adds 1 to the register being used
  - Example: add 1 to DX register to get 12
    - `inc dx`

Assembly-Language Statements (cont'd.)

- `sub`: tells the assembler to subtract one number from another number
  - Example: DX = DX – CX
    - CX register still contains value 4
    - DX register contains value 3
    - `mov cx, 4`
    - `mov dx, 7`
    - `sub dx, cx`

Assembly-Language Statements (cont'd.)

- `cmp`: tells assembler to compare two values
  - Result sets flag bits in the flags (FL) register
    - If the result of the compare equals 0, zero (ZR) flag is set to a binary 1, and the sign (SF) flag is set to 0
    - If the result of the compare is a negative number, ZR flag bit is set to a binary 0, and the SF flag is set to 1
  - Example: DX – CX = 0, ZR flag is set to 1
    - `mov cx, 4`
    - `mov dx, 7`
    - `cmp dx, cx`

Assembly-Language Statements (cont'd.)

- `jnz`: tests value of ZR flag maintained by the system
  - If set to 1: jump somewhere else in the program
  - Not set: assembler continues to process code on the next line
  - Example:
    - `mov cx, 4`
    - `mov dx, 7`
    - `cmp dx, cx`
    - `jnz stop`

High-Level Languages

- Writes programs independent of computer or CPU
- Advantages:
  - Easier to write, read, and maintain
  - Can accomplish much more with a single statement
  - No one-to-one relationship between a statement and a binary instruction
- Disadvantages:
  - Programs generally run slower
    - Must be compiled or interpreted
- Examples: Java, C++, Delphi, and C#

High-Level Languages (cont'd.)

- Integrated development environment (IDE):
  - Interface provided with software development languages
    - Incorporates all tools needed to write, compile, and distribute programs
    - Tools often include editor, compiler, graphical designer, and more

High-Level Languages (cont'd.)

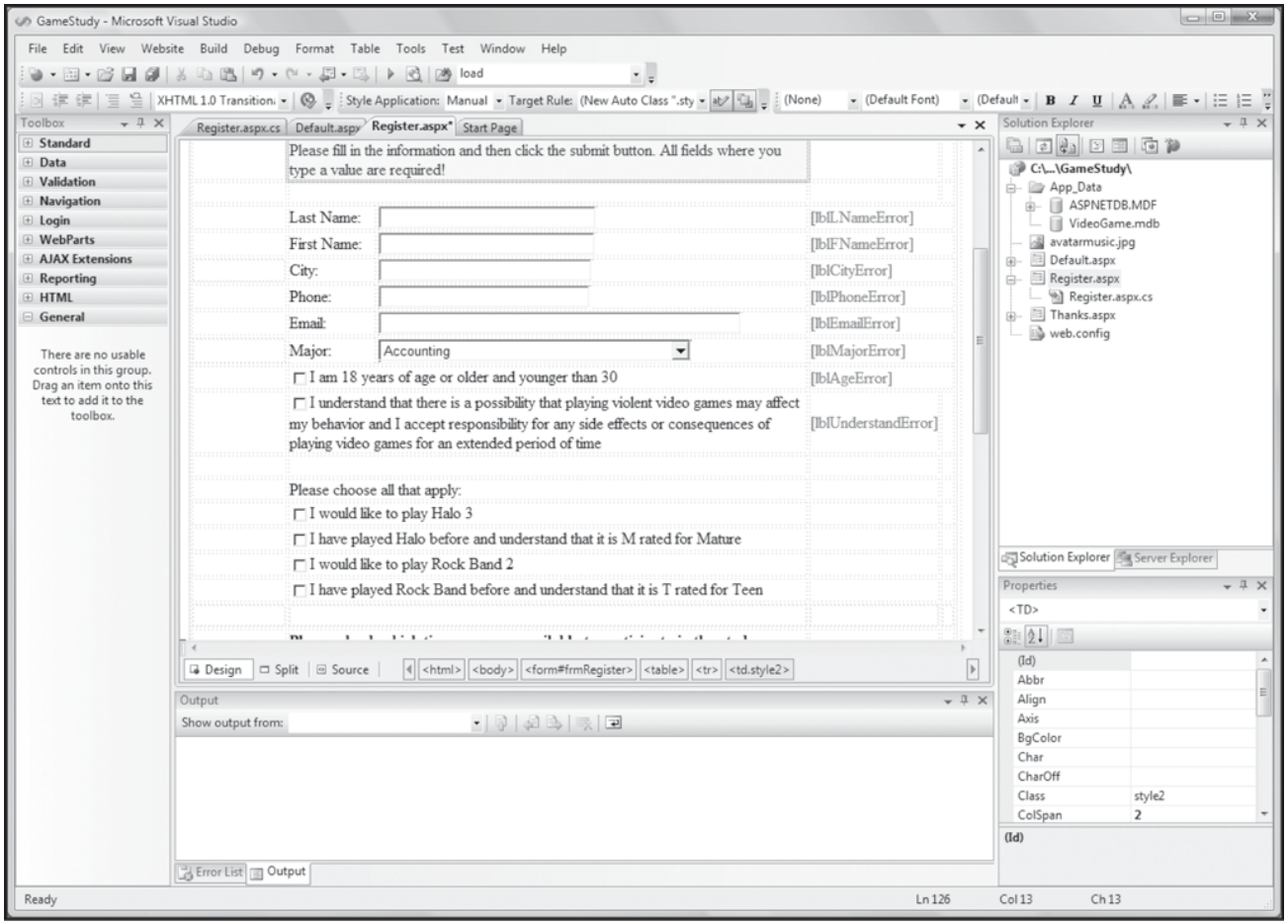

Figure 14-2, An IDE makes software development easier

Structure of a Program

- Before writing a program in any language:
  - Know how the program should work
  - Know the language syntax
    - Formal definition of how statements must be constructed in the programming language
- Learning a programming language is similar to learning a foreign language

Algorithms

- Help describe the method used to solve a problem
  - Break down each task in the plan into smaller subtasks
    - For many tasks, plan a series of logical steps to accomplish them
  - Provide a logical solution to a problem
  - Consist of steps to follow to solve the problem
  - Convert algorithm into programming statements by representing the steps in some format
    - Pseudocode is often used

Pseudocode

- Readable description of an algorithm written in human language
  - Template describing what needs converting into programming language syntax
  - No formal rules for writing pseudocode
    - Information should explain the process to someone with little experience in solving this type of problem
  - Practice provides necessary skill

Pseudocode (cont'd.)

Celsius	Fahrenheit
0 C	32.0 F
1 C	33.8 F
2 C	35.6 F
3 C	37.4 F
4 C	39.2 F
5 C	41.0 F
6 C	42.8 F
7 C	44.6 F
8 C	46.4 F
9 C	48.2 F
10 C	50.0 F
11 C	51.8 F
12 C	53.6 F
13 C	55.4 F
14 C	57.2 F
15 C	59.0 F
16 C	60.8 F
17 C	62.6 F
18 C	64.4 F
19 C	66.2 F
20 C	68.0 F
21 C	69.8 F
22 C	71.6 F
23 C	73.4 F
24 C	75.2 F
25 C	77.0 F
26 C	78.8 F
27 C	80.6 F
28 C	82.4 F
29 C	84.2 F
30 C	86.0 F

Figure 14-3, A temperature conversion chart

Pseudocode (cont'd.)

- Start with the formulas needed in the algorithm:
  - Fahrenheit to Celsius: Celsius temp = (5/9) * (Fahrenheit temp – 32)
  - Celsius to Fahrenheit: Fahrenheit temp = ((9/5) * Celsius temp) + 32
- After formulas are proved correct, begin outlining steps to write a program
  - Input from the user
  - Calculates the conversions
  - Displays results to the user

Pseudocode (cont'd.)

```
Menu:
        Do you want to perform a conversion?
I       f Yes then
        Which conversion do you want to perform?
                If Celsius to Fahrenheit then
                Go to the Fahrenheit section
                If Fahrenheit to Celsius then
                Go to the Celsius section
        Else If No then
        Exit the program
Celsius:
        Ask the user for a temperature in Fahrenheit
        Apply the formula Celsius temp = (5/9) * (Fahrenheit
temp - 32) to the entered temperature
        Display the result, saying Fahrenheit temp ## converted to Celsius is XX
        Return to the Menu section
Fahrenheit:
        Ask the user for a temperature in Celsius
        Apply the formula Fahrenheit temp = ((9/5) * Celsius temp) + 32 to the
entered temperature
        Display the result, saying Celsius temp ## converted to Fahrenheit is XX
        Return to the Menu section
```

Choosing the Algorithm

- Determine best algorithm for the project
  - Example: many ways to get to Disney World
    - Fly
    - Drive
    - Hitchhike
    - Walk
  - Each has advantages and disadvantages

Testing the Algorithm

- Test before typing program code
  - Pretending to be an end user who is not knowledgeable about the program
  - Putting yourself in the user's shoes helps predict possible mistakes that users might make

Syntax of a Programming Language

- After defining an algorithm and testing the logic thoroughly, begin translating the algorithm
  - May have many different ingredients:
    - Variables
    - Operators
    - Control structures
    - Objects

Variables

- A name used to identify a certain location and value in the computer's memory
  - Program type determines variable types needed
  - When a variable is defined, the data type is specified
- Advantages:
  - Access memory location's content
    - Use its value in a program
  - Easy way to access computer memory
    - No need to know actual hardware address
- Identifier: name of a variable

Identifiers and Naming Conventions

- Identifier used to access memory contents associated with a variable
- Items to consider when deciding on an identifier:
  - Name should describe data being stored
    - Use variable-naming standards
  - Can use more than one word for a variable's identifier
    - Example: Sun standard
  - Use meaningful names

Operators

- Symbols used to indicate data-manipulation operations
  - Manipulate data stored in variables
  - Classified by data type
  - One may work on numbers, and another on characters (depending on definition)

Math Operators

- Mathematical operators:
  - Addition (+)
  - Subtraction (–)
  - Multiplication (*)
  - Division (/)
  - Modulus (%)
    - Returns the remainder when performing division

Math Operators (cont'd.)

operator	description
+	addition
-	subtraction
/	division
%	modulus or remainder
*	multiplication
+=	addition and then assignment
-=	subtraction and then assignment
*=	multiplication and then assignment
/=	division and then assignment
%=	modulus and then assignment

Table 14-1, Standard mathematical operators

Increment and Decrement Operators

- Most common programming instructions
  - Examples: `++` and `--`
  - Example: increment operator takes value stored in the `iCount` variable (5), adds 1 to it, stores the value 6 in the `iResult` variable
    - `iCount = 5`
    - `iResult = ++iCount`
  - Example: decrement operator takes value stored in the `iCount` variable (5), subtracts 1 from it, stores the value 4 in the `iResult` variable
    - `iCount = 5`
    - `iResult = --iCount`

Increment and Decrement Operators (cont'd.)

- Two types of increment and decrement operators
  - Pre operator places `++` or `--` symbol before the variable name
    - Preincrement: `++variable`
    - Predecrement: `--variable`
  - Post operator places `++` or `--` symbol after the variable name
    - Postincrement: `variable++`
    - Postdecrement: `variable--`

Relational Operators

- Main purpose is to compare values

operator	meaning
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equals

Table 14-2, Standard relational operators

Logical Operators

- Main function is to build a truth table when comparing expressions
  - Expression: programming statement returning a value when it's executed
    - Usually use relational operators to compare variables

operator	meaning
!	not
&&	and
||	or

Table 14-3, Standard logical operators

Logical Operators (cont'd.)

expression	value	explanation
`(iFirstNum >= iSecondNum) && (iThirdNum >= iFourthNum)`	T and T equals T	(15 >= 10) and (20 >= 15)
`(iFirstNum <= iSecondNum) && (iThirdNum >= iFourthNum)`	F and T equals F	(15 <= 10) and (20 >= 15)
`(iFirstNum == iSecondNum) && (iThirdNum == iFourthNum)`	F and F equals F	(15 == 10) and (20 == 15)
`(iFirstNum != iSecondNum) && (iThirdNum != iFourthNum)`	T and T equals T	(15 != 10) and (20 != 15)
`(iFirstNum >= iSecondNum) || (iThirdNum >= iFourthNum)`	T or T equals T	(15 >= 10) or (20 >= 15)
`(iFirstNum <= iSecondNum) || (iThirdNum >= iFourthNum)`	F or T equals T	(15 <= 10) or (20 >= 15)

Table 14-4, Boolean expressions

Precedence and Operators

- Precedence: order in which something is executed
- Symbols with a higher precedence executed before those with a lower precedence
  - Have a level of hierarchy
- Example: 2 + 3 * 4
  - Output = 14 (not 20)

Precedence and Operators (cont'd.)

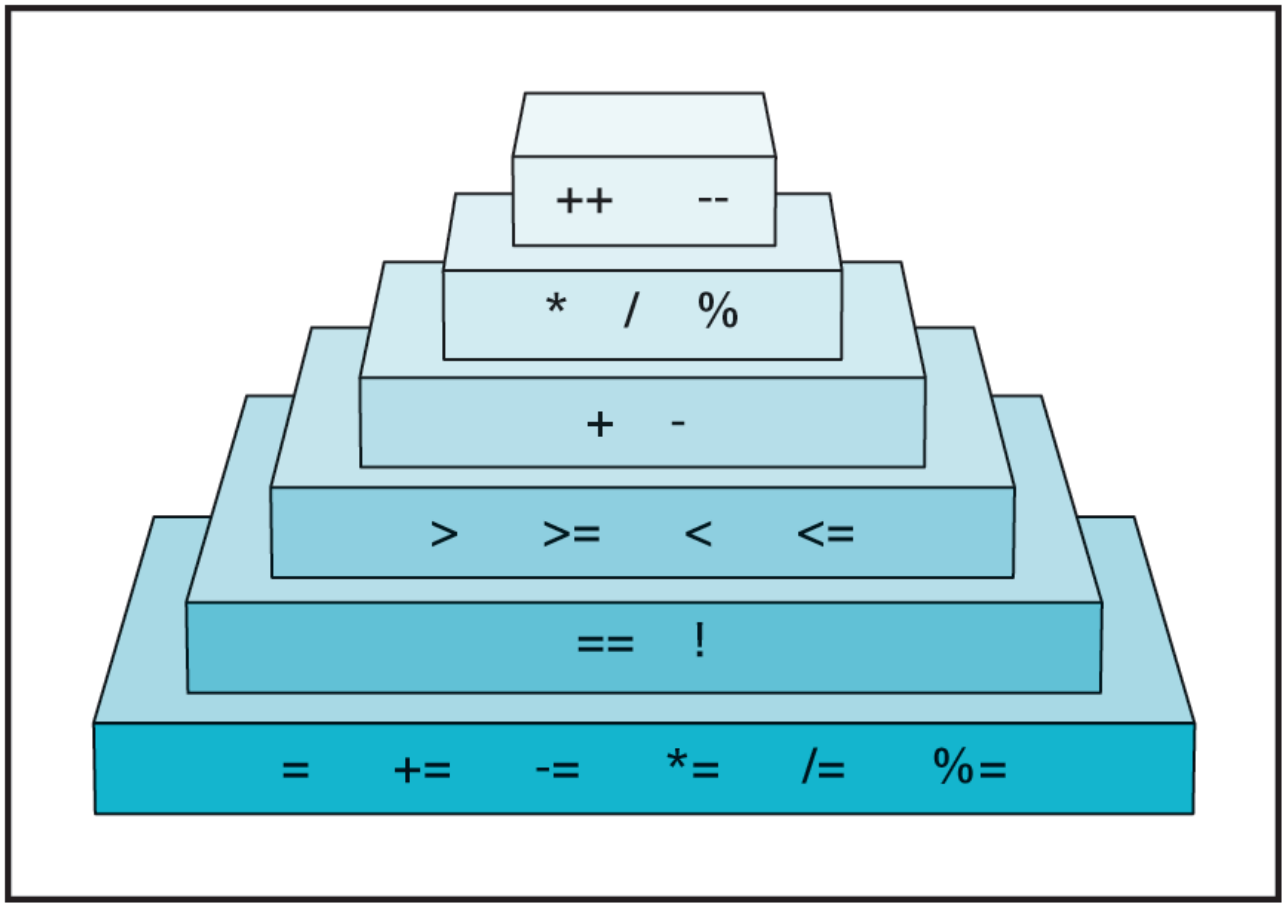

Figure 14-4, Order of relational and mathematical precedence

Control Structures and Program Flow

- Control structure: instruction that dictates the order in which statements in a program are executed
  - "Spaghetti code" results if not followed
- Four control structure types:
  - Invocation
  - Top down
  - Selection
  - Repetition
- Control structure performs a specific task

Invocation

- Act of calling something
  - Copy code for a specific task (called “functionality”) to a file and name it descriptively
  - Write a new program
    - “Call” (invoke) this piece of code without having to rewrite it
    - Saves time and money in program development
  - After piece of code used:
    - Control is passed back to the original program location to continue

Top Down (Also Called Sequence)

- Used when program statements are executed in a series
  - From top line to the bottom line one at a time
  - First statement executed is the first line in the program
  - Each statement executed in sequential order
    - Start with first line and continue until last line processed
- Most common structure
- Implemented by entering statements that do not call other pieces of code

Selection

- Make a choice (selection) depending on a value or situation
  - A standard part of most programs

Repetition (Looping)

- Used when source code is to be repeated
- Referred to as “looping”
  - Commonly used with databases or when you want an action to be performed one or many times
- Standard repetition constructs
  - `for`
  - `while`
  - `do-while`

Ready, Set, Go!

- Building blocks:
  - Variables, operators, and control structures
- Use Java to show examples of programming code:
  - Download Java
  - Choose an editor
  - Enter the program in a text file
  - Compile it from the command prompt
  - Run the program

Object-Oriented Programming

- Style of programming
- Involves representing items, things, and people as objects instead of basing program logic on actions
  - Object: includes qualities, what it does, and how it responds or interacts with other objects
  - Distinct features:
    - Characteristics
    - Work
    - Responses

Object-Oriented Programming (cont'd.)

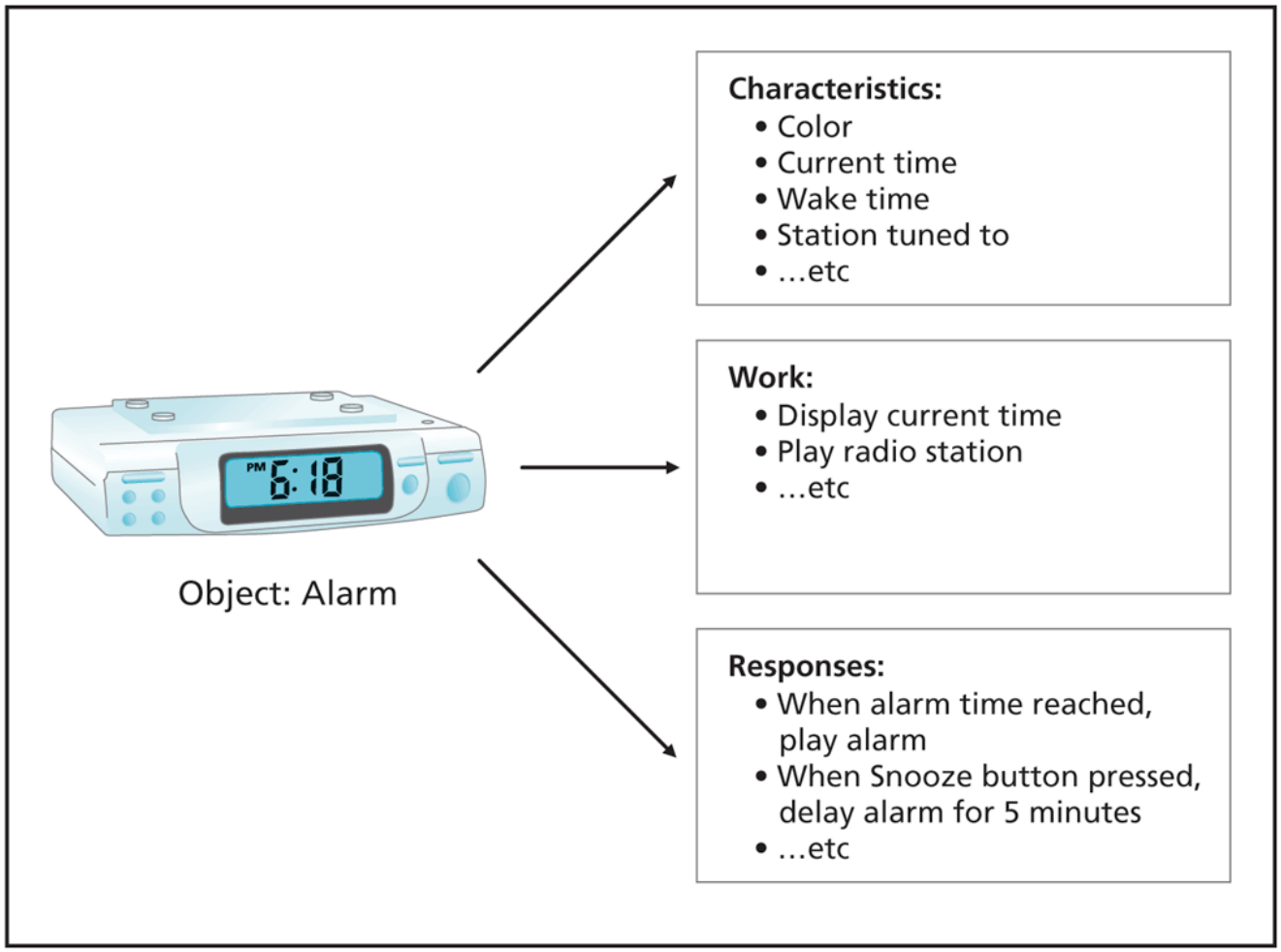

Figure 14-6, An object has characteristics, work, and responses

Object-Oriented Programming (cont'd.)

- Alarm object features:
  - Characteristics
  - Work
  - Responses
- High-level languages support OOP
- OOP can represent part of the program as a self-contained object
- Advantages: reusability and maintainability

How OOP Works

- Toy company division responsible for creating kung-fu action figure
  - Method one:
    - Give every division employee piece of plastic
    - Everyone carves the figure
  - Method two:
    - Create a mold (class or template in object-oriented terminology)
    - Figure can be mass-produced economically and efficiently

How OOP Works (cont'd.)

- Making the mold
  - Skeleton or the basic outline of a finished product
    - Defines figure's attributes
- Creating the figure
  - Pour plastic into the mold
    - Different colors of plastic in different parts of the mold create attributes
    - Mold defines what the plastic will be
- Putting the figure to work
  - Figure can perform some work or action

How OOP Works (cont'd.)

- Putting the figure to work (cont'd.)
  - Class: mold or template for creating the figure
  - Object: the figure
  - Instantiation: creation process
  - Constructor: method used to instantiate an object in a class
  - Property or an attribute: characteristic of the figure
  - Method: work performed by an object
  - Event or event handler: object's response to some action taken by the end user or system

How OOP Works (cont'd.)

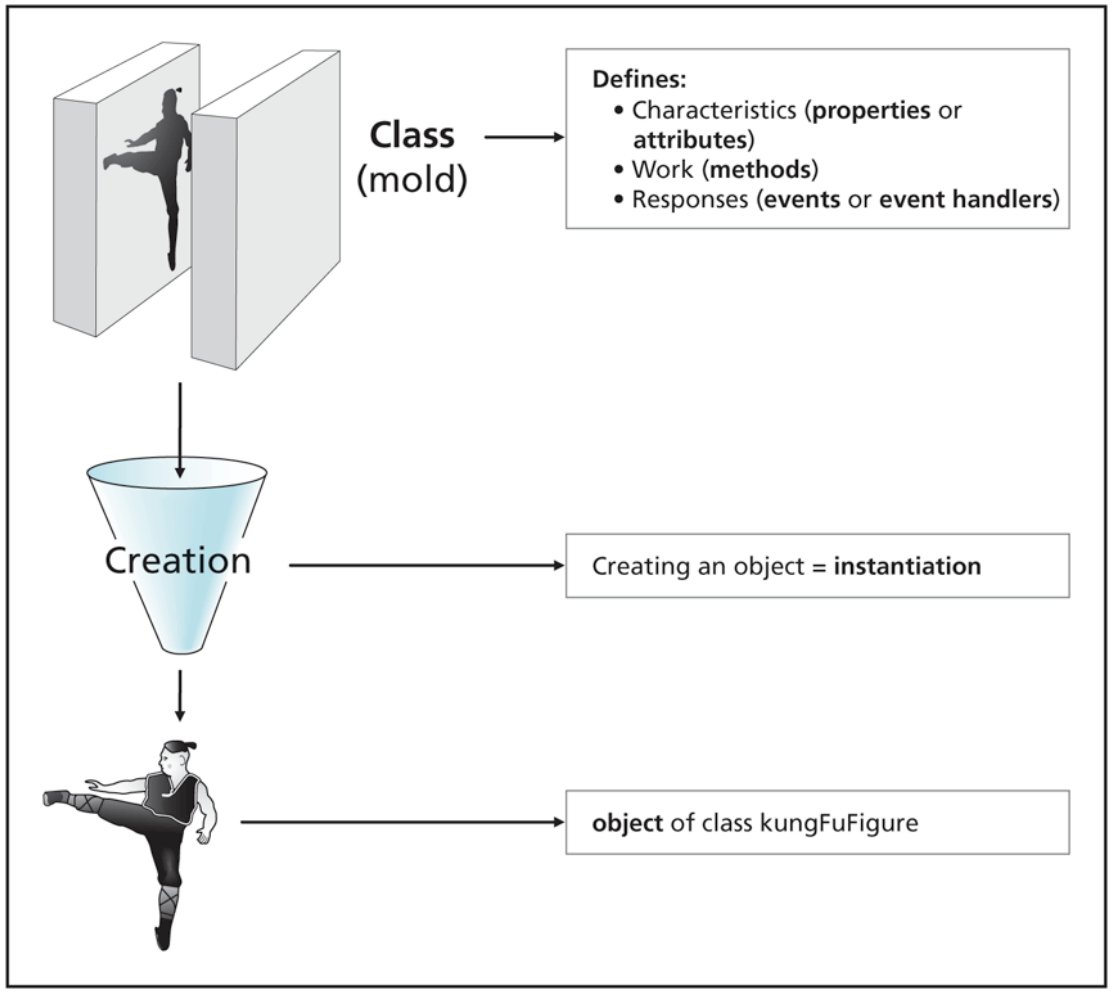

Figure 14-7, Making a plastic figure shows OOP concepts in action

How OOP Works (cont'd.)

- Inheritance:
  - Process of creating more specific classes based on generic classes
- Base (or parent) class:
  - General class from which other classes can be created via inheritance
- Subclass:
  - A more specific class, based on a parent class and created via inheritance

How OOP Works (cont'd.)

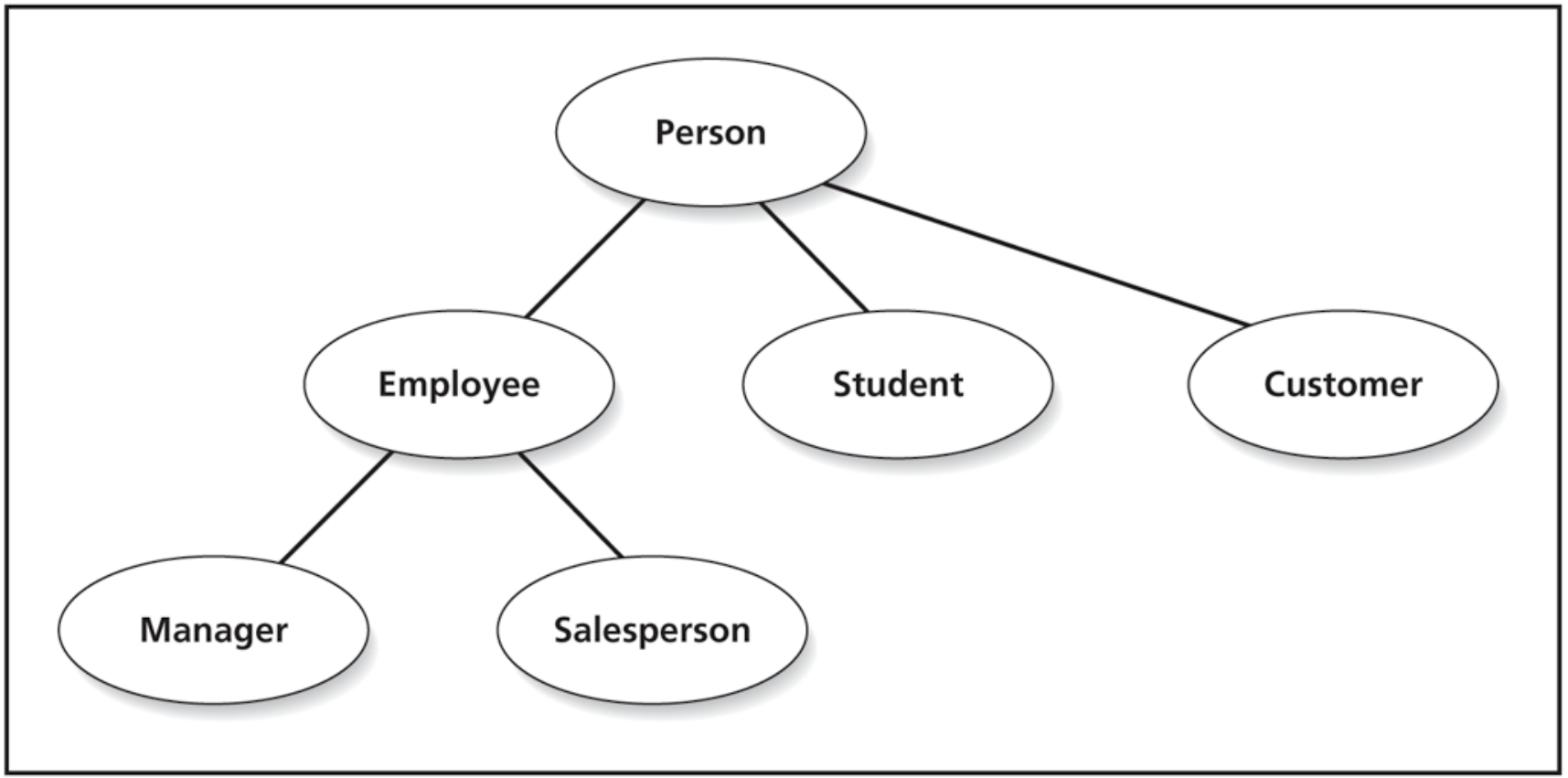

Figure 14-8, Inheritance promotes code reusability

How OOP Works (cont'd.)

- Encapsulation:
  - Process of hiding an object's operations from other objects
- Polymorphism:
  - An object's capability to use the same expression to denote different operations

Choosing a Programming Language

- Functions to consider:
  - Functionality
  - Vendor stability
  - Popularity
  - Job market
  - Price
  - Ease of learning
  - Performance
- Download trial versions and try for yourself

One Last Thought

- A program does whatever the programmer tells it to do
  - Blame program failure on the programmer, not the computer
- Key word: responsibility
  - Programs can help society or produce serious ramifications

Summary

- A program is a collection of statements or steps that solve a problem
  - There are many language choices available
    - Machine languages
    - Low-level languages
    - Assembly languages
    - High-level languages
- Integrated development environment (IDE)
  - Provides programming tools

Summary (cont'd.)

- Program structure
  - Based on algorithms
  - Represented with pseudocode
- Program language syntax
  - Variables, operators, control structures, and objects
  - Be aware of operator precedence
  - Avoid spaghetti code
  - Control structures
    - Innovation, sequence, selection, and looping

Summary (cont'd.)

- Start programming:
  - Obtain software package
  - Choose an editor
  - Write the code
  - Compile and fix errors
  - Run program
- Distinct features of object-oriented programming:
  - Characteristics, work, and responses
  - Inheritance, encapsulation, and polymorphism