1 Introduction

Java is the first major programming language to be shaped by the World Wide Web (the Web). Java allows you to do traditional programming. Java also has many special features and libraries that allow you conveniently to write programs that can use the Web's resources. These include extensive support for graphical user interfaces, the ability to embed a Java program in a Web document, easy communication with other computers around the world, and the ability to write programs that run in parallel or on several computers at the same time.

In this chapter we give an overview of how to solve a problem on a computer. In this process, you must first construct a recipe for solving the problem. Then you must convert the recipe into a detailed set of steps that the computer can follow. Finally, you must use a programming language, such as Java, to express the steps in a form "understandable" by the computer. The Java form of the solution is then translated by a program called a compiler into the low-level operations that the computer hardware can follow directly.

We then discuss why Java is creating such a fuss in the computing world. In general terms, we explain the importance of computing on the Web and the character of the graphical user interfaces, or GUIs, that are partly behind the switch to Java.

Throughout this text we feature carefully described examples, many of which are complete programs. We often dissect them, allowing you to see in detail how each Java programming construct works. Topics introduced in this chapter are presented again in later chapters, with more detailed explanations. The code and examples are meant to convey the flavor of how to program. You should not be concerned about underlstanding the details of the examples. They are given in the spirit of providing an overview. If you are already familiar with what a program is and want to start immediately on the nitty-gritty of Java programming, you may skip or scan this chapter.

1.1 Recipes

Computer programs are detailed lists of instructions for performing a specific task or for solving a particular type of problem. Programlike instruction lists, sometimes called *algorithms*, for problem solving and task performance are commonly found in everyday situations. Examples include detailed instructions for knitting a sweater, making a dress, cooking a favorite meal, registering for classes at a university, traveling from one destination to another, and using a vending machine. Examining one of these examples is instructive.

Consider this recipe for preparing a meat roast.

> Sprinkle the roast with salt and pepper. Insert a meat thermometer and place in oven preheated to 150 °C. Cook until the thermometer registers 80 °C–85 °C. Serve roast with gravy prepared from either meat stock or from pan drippings if there is a sufficient amount.

The recipe is typically imprecise—what does "sprinkle" mean, where is the thermometer to be inserted, and what is a "sufficient amount" of pan drippings?

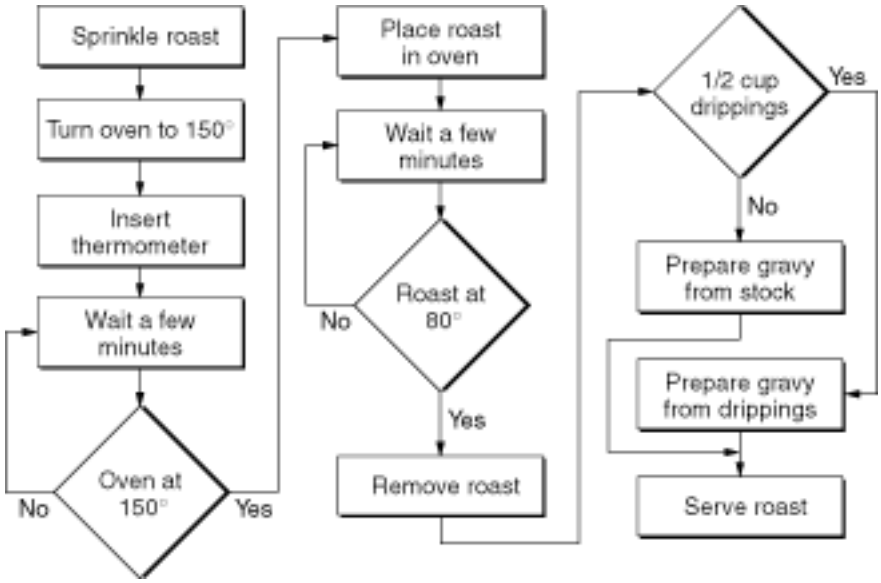
However, the recipe can be formulated more precisely as a list of instructions by taking some liberties and reading "between the lines."

COOKING A ROAST

1. Sprinkle roast with 1/8 teaspoon salt and pepper.
2. Turn oven on to 150° C.
3. Insert meat thermometer into center of roast.
4. Wait a few minutes.
5. If oven does not yet register 150° C, go back to step 4.
6. Place roast in oven.
7. Wait a few minutes.
8. Check meat thermometer. If temperature is less than 80° C, go back to step 7.
9. Remove roast from oven.
10. If there is at least 1/2 cup of pan drippings, go to step 12.
11. Prepare gravy from meat stock and go to step 13.
12. Prepare gravy from pan drippings.
13. Serve roast with gravy.

These steps comprise three categories of instructions and activities—those that involve manipulating or changing the ingredients or equipment, those that just examine or test the "state" of the system, and those that transfer to the next step. Steps 1 and 6 are examples of the first category; the temperature test in step 8 and the pan dripping test in step 10 are instances of the second category; and the transfers in steps 5 and 8 ("go to step *x*") are examples of the last category.

By using suitable graphical symbols for each of these categories, a simple two-dimensional representation of our cooking algorithm can be obtained, as shown in the following illustration.



Such a figure is called a *flowchart*. To perform the program (prepare the roast), just follow the arrows and the instructions in each box. The manipulation activities are contained in rectangles, the tests are shown in diamonds, and the transfer or flow of control is determined by the arrows. Because of their visual appeal and clarity, flowcharts are often used instead of lists of instructions for informally describing programs. Some cookbook authors even employ flowcharts extensively. In this book we use flowcharts in Chapter 3, Statements and Control Flow, when describing the behavior of some Java language constructs.

1.2 Algorithms—Being Precise

Our recipe for preparing a roast can't be executed by a computer because the individual instructions are too loosely specified. Let's consider another example—one that manipulates numbers instead of food. You need to pay for some purchase with a dollar bill and get change in dimes and pennies. The problem is to determine the correct change with the fewest pennies. Most people do this simple everyday transaction unthinkingly. But how do we precisely describe this algorithm?

In solving such a problem, trying a specific case can be useful. Let's say that you need to pay 77 cents and need change for a dollar. You can easily see that one dollar

minus the 77 cents leaves you with 23 cents in change. The correct change having the fewest coins in dimes and pennies would be two dimes and three pennies. The number of dimes is the integer result of dividing 23 by 10 and discarding any fraction or remainder. The number of pennies is the remainder of dividing the 23 cents by 10. An algorithm for performing this change for a dollar is given by the following steps.

CHANGE-MAKING ALGORITHM

1. Assume that the price is written in a box labeled `price`.
2. Subtract the value of `price` from 100 and place it in a box labeled `change`.
3. Divide the value in `change` by 10, discard the remainder, and place the result in a box labeled `dimes`.
4. Take the integer remainder of `change` divided by 10 and place it in a box labeled `pennies`.
5. Print out the values in the boxes `dimes` and `pennies` with appropriate labels.
6. Halt.

This algorithm has four boxes, namely, `price`, `change`, `dimes`, and `pennies`. Let's execute this algorithm with the values given. Suppose that the price is 77 cents. Always start with the first instruction. The contents of the four boxes at various stages of execution are shown in the following table.

Box	Step 1	Step 2	Step 3	Step 4	Step 5
`price`	77	77	77	77	77
`change`		23	23	23	23
`dimes`			2	2	2
`pennies`				3	3

To execute step 1, place the first number, 77, in the box `price`. At the end of instruction 2, the result of subtracting 77 from 100 is 23, which is placed in the box `change`. Each step of the algorithm performs a small part of the computation. By step 5, the correct values are all in their respective boxes and are printed out. Study the example until you're convinced that this algorithm will work correctly for any price under $1.00. A good way to do so is to act the part of a computer following the recipe. Following a set of instructions in this way, formulated as a computer program, is called *hand simulation* or *bench testing*. It is a good way to find errors in an algorithm or program. In computer parlance these errors are called *bugs,* and finding and removing them is called *debugging*.

We executed the change-making algorithm by acting as an agent, mechanically following a list of instructions. The execution of a set of instructions by an agent is called a *computation*. Usually the agent is a computer; in that case, the set of instruc-

tions is a computer program. In the remainder of this book, unless explicitly stated otherwise, we use *program* to mean *computer program*.

The algorithm for making change has several important features that are characteristic of all algorithms.

ALGORITHMS

- The sequence of instructions will terminate.
- The instructions are precise. Each instruction is unambiguous and subject to only one interpretation.
- The instructions are simple to perform. Each instruction is within the capabilities of the executing agent and can be carried out exactly in a finite amount of time; such instructions are called *effective*.
- There are inputs and outputs. An algorithm has one or more outputs ("answers") that depend on the particular input data.
- The input to the change-making algorithm is the price of the item purchased. The output is the number of dimes and pennies.

Our description of the change-making algorithm, although relatively precise, is not written in any formal programming language. Such informal notations for algorithms are called *pseudocode*, whereas real code is something suitable for a computer. Where appropriate we use pseudocode to describe algorithms. Doing so allows us to explain an algorithm or computation to you without all the necessary detail needed by a computer.

The term *algorithm* has a long, involved history, originally stemming from the name of a well-known Arabic mathematician of the ninth century, Abu Jafar Muhammed Musa Al-Khwarizmi. It later became associated with arithmetic processes and then, more particularly, with Euclid's algorithm for computing the greatest common divisor of two integers. Since the development of computers, the word has taken on a more precise meaning that defines a real or abstract computer as the ultimate executing agent—any terminating computation by a computer is an algorithm, and any algorithm can be programmed for a computer.

1.3 Implementing Our Algorithm in Java

In this section we implement our change-making algorithm in the Java programming language. You need not worry about following the Java details at this point; we cover all of them fully in the next two chapters. We specifically revisit this example in . For now, simply note the similarity between the following Java program and the informal algorithm presented earlier. You not only have to be able to formulate a recipe and make it algorithmic, but you finally have to express it in a computer language.

```
// MakeChange.java - change in dimes and pennies
import tio.*; // use the package tio

class MakeChange {
  public static void main (String[] args) {
    int price, change, dimes, pennies;

    System.out.println("type price (0:100):");
    price = Console.in.readInt();
    change = 100 - price;           //how much change
    dimes = change / 10;            //number of dimes
    pennies = change % 10;          //number of pennies
    System.out.print("The change is :");
    System.out.print(dimes);
    System.out.print(" dimes ");
    System.out.print(pennies);
    System.out.print(" pennies.\n");
  }
}
```

Dissection of the MakeChange Program

- `import tio.*; // use the package tio`

  A package is a library or collection of previously written program parts that you can use. This line tells the Java compiler that the program `MakeChange` uses information from the package `tio`. We developed this package especially for this book to simplify keyboard input for you. It allows you to write `Console.in.readInt()`, which we explain shortly.

- `int price, change, dimes, pennies;`

  This program declares four integer variables. These hold the values to be manipulated.

- `System.out.println("type price(0 to 100):");`

  This line is used to *prompt* you to type the price. Whenever a program is expecting a user to do something, it should print out a prompt telling the user what to do. The part in quotes appears on the user's screen when the program is run.

- `price = Console.in.readInt();`

  The `Console.in.readInt()` is used to obtain the input from the keyboard. The value read is stored in the variable `price`. The symbol `=` is called the *assignment operator.* Read the first line as "`price` is *assigned the value* obtained from `Console.in.readInt`." At this point you must type in an integer price. For example, you would type 77 and then hit Enter.

❑
```
change = 100 - price;       //how much change
```

This line computes the amount of change.

❑
```
dimes = change / 10;         //number of dimes
pennies = change % 10;       //number of pennies
```

The number of dimes is the integer or whole part of the result of dividing `change` by 10. The symbol `/`, when used with two integers, computes the whole (nonfraction) part of the division. The number of pennies is the integer remainder of `change` divided by 10. The symbol `%` is the integer remainder or modulo operator in Java. So if `change` is 23, the integer divide of 23/10 is 2 and the integer remainder or modulo of 23 % 10 is 3.

❑
```
System.out.print("The change is : ");
System.out.print(dimes);
System.out.print(" dimes ");
System.out.print(pennies);
System.out.print(" pennies.\n");
```

In this example the `System.out.print()` statements cause the values between the parentheses to be printed on the computer console. The first one just prints out the characters between the quotation marks. The second one converts the value in `dimes` to the sequence of digits and prints those digits. The other print statements are similar. For an input value of 77, the output would be

```
The change is : 2 dimes 3 pennies
```

The `\n` in the last print statement indicates that a newline should be sent to the console, ending the line of output.

1.4 Why Java?

A variety of programming languages are available for expressing programs, but the most useful ones are suitable for both machine and human consumption. In this book we cover programming by using one such language—the programming language Java.

Java is a relatively new programming language, first introduced in 1995. In this book a language is needed that allows algorithms to be easily understood, designed, analyzed, and implemented as computer programs. The following is an excerpt from the original paper introducing Java to the world.

In his science-fiction novel, *The Rolling Stones*, Robert A. Heinlein comments:

> *Every technology goes through three stages: first a crudely simple and quite unsatisfactory gadget; second, an enormously complicated group of gadgets designed to overcome the shortcomings of the original and achieving thereby somewhat satisfactory performance through extremely complex compromise; third, a final proper design therefrom.*

Heinlein's comment could well describe the evolution of many programming languages. Java presents a new viewpoint in the evolution of programming languages--creation of a small and simple language that's still sufficiently comprehensive to address a wide variety of software application development. Although Java is superficially similar to C and C++, Java gained its simplicity from the systematic removal of features from its predecessors.

We agree with its creators that Java has, to a large extent, lived up to the promise of a small, simple, yet comprehensive programming language. As such, Java is both an excellent programming language for real program development and a good first programming language.

Possibly even more important for Java's success are the features that make it appropriate for development of programs distributed over the Internet. These programs, called *applets*, execute inside Internet browsers such as Netscape and Internet Explorer.

1.5 Network Computing and the Web

Much of modern computing is done in a connected environment. That is, computers are connected to other computers in a network. This connection allows the computers to exchange information or to "talk to each other." Java was developed with network computing as the normal environment. It has many features and libraries of parts of programs that promote networking. Earlier languages and systems viewed a computer as a lone instrument doing significant computations, the results of which would largely be output to a screen or printer. By promoting networking whereby computers are connected and pass results to each other dynamically—and cooperate on large-scale computations—Java became the principal language for computing on networks.

The largest network is the global network called the *Internet*. Using the Internet, researchers at the European Particle Physics Laboratory (CERN) developed a way to share information via a formatting language called *hyper-text markup-language*, or *HTML*. The computers exchanged HTML documents by means of a protocol called *hyper-text transfer protocol*, or *HTTP*. A *protocol* is like a language that computers use to "talk" to each other. HTML allows one electronic document to link to another electronic document on a different computer. The program used to view HTML documents and

follow the links is called a *browser*. With the click of a button, a user could move from one document on one computer to another, related document, on another computer. Because of these links, the resulting web of documents was dubbed the *World Wide Web* or simply *the Web.* Today, many people equate the Web with the Internet, but the Web is only one application of the technology known as the Internet.

Java programs, called *applets*, can be written to automatically load and execute by following a link in the Web. This ability to embed Java programs in HTML documents is a primary factor in the success of Java. With Java applets, Web documents are no longer static text, images, or video segments. Web documents can provide all the interaction of any program. We discuss graphical user interfaces and applets in Chapter 8, Graphical User Interfaces: Part I.

If you're interested in an early exposure to applets, we also provide templates for simple applets in the exercises at the end of Chapters 2 through 6.

From the World Wide Web (www) and HTTP, you can begin to make some sense of the ubiquitous Internet addresses such as *http://www.company.com*. This type of address is called a *universal resource locator*, or *URL*. These are addresses embedded in HTML documents, linking one document to another.

Client-server computing is an essential new element in computers that manage communications and computations. The server, typically a fast computer with a very large amount of hard disk storage, gives out information to clients upon request. An example would be a server giving out stock market quotes to clients that request it. The network support in Java makes implementing client-server programming solutions easy. In Chapter 13, Concurrent Programming with Java Threads we show you how to implement a simple Java server program that can connect and respond to requests from multiple clients on other computers.

1.6 Human-Computer Interaction and the GUI

In the early days of computing, most interaction between a computer and a human being was in the form of typed input to the computer and printed output to the person. Most human-computer interaction today is done with a *graphical user interface* (*GUI*—pronounced gooey). The user has a keyboard, usually similar to a standard typewriter keyboard, a pointing device, usually a mouse, and a display device capable of displaying both text and graphics.

The user's display is usually divided into viewing areas called *windows*. Associated with the windows are menus of commands relating to manipulation of the data displayed in the window. The display (screen) may also contain icons, or small graphical images, whose visual appearance is designed to signify their functions. For example, a picture of a trash can represents deleting a file, and an arrowhead is used to scroll a viewing screen. The following screen shot of a typical desktop shows several windows, icons, and a row of menus across the top of each window.

For many years, the task of creating a graphical user interface for a new application was extremely time-consuming and difficult. The programming languages and operating systems in use did not make building such interfaces easy. These interfaces were usually built from libraries of program parts that were suitable only for one particular operating system. If a company building a new program wanted to have its program run on computers with different operating systems, it would have to create multiple versions of the programs.

Java provides a solution. Software companies now sell programs that make building graphical user interfaces relatively easy. Java includes a library of program parts called *Swing* for building graphical user interfaces. The program parts in Swing can be used on any computer that can run Java programs. Thus a programmer can now write one version of a program and have it run on many different kinds of computers. Although in-depth coverage of Swing is beyond the scope of this book, Swing is simple enough to use that even beginning programmers can begin to build programs with graphical user interfaces. We discuss Swing further in Chapter 8, Graphical User Interfaces: Part I and Chapter 9, Graphical User Interfaces: Part II.

By the end of this book, you will be writing your own sophisticated applets and regular Java programs with GUIs. If you've used a Java-enabled browser, such as Netscape or Explorer, you've probably unknowingly made use of an applet. If not, you can try out a simple applet that lets you draw on the screen right now by going to *http://www.cse.ucsc.edu/~pohl/JBD/chap8/MiniCalcApplet.html*. This is the same applet presented in Section 8.8 Applets.

SUMMARY

- Informally, an algorithm is a list of instructions for performing a specific task or for solving a particular type of problem. Algorithmlike specifications for problem solving and task performance are commonly found in everyday situations. A recipe is a kind of algorithm.
- One graphical representation of an algorithm is a flow chart. To perform the algorithm, the user just follows the arrows and the instructions in each box. The manipulation activities are contained in rectangles, the tests are shown in diamonds, and the transfer or flow of control is determined by the arrows. Because of their visual appeal and clarity, flow charts are often used instead of lists of instructions for informally describing algorithms.
- Algorithms that can be executed on computers are called programs. Programs are written in special languages called programming languages, such as Java, which we use in this book.
- Modern computing is done in a connected environment. A computer is connected to other computers in a network. This connection allows the computers to exchange information or to "talk to each other." Java was developed with network computing as the normal environment. The largest network is the global network called the Internet. This network is used to exchange documents throughout the world with a common format called HTML. This format allows links to be followed across the Internet. Because of these links, the resulting web of documents was named the World Wide Web or simply the Web.

7. It is possible to write Java programs called applets, that are automatically loaded and executed as the result of following a link in the Web. This ability to embed Java programs in HTML documents is a primary factor in the success of Java. With Java applets, Web documents are no longer static text, images, or video segments. Web documents can provide all the interactions of any program.

REVIEW QUESTIONS

1. What is an algorithm?
2. A graphical description of a computation using boxes and arrows is called a _____.
3. Informal, but relatively precise descriptions of algorithms are called _____.
4. What is bench testing? Why is it important?
5. What did `tio` refer to in the `MakeChange.java` program?
6. What was `System.out.print()` used for in the Java program?
7. What is HTML? What is HTTP? What is URL?

8. What is a browser? Name a commonly used browser.
9. What are the primary components of a typical human–computer interface today?
10. The information provided by the user to a program when it executes is called the _____. The answers provided by the program are called the _____.
11. What is Swing?

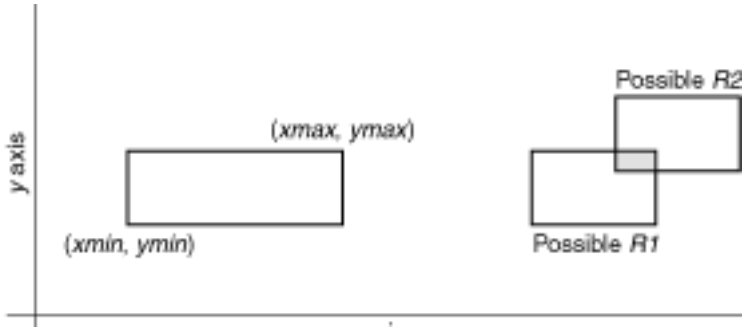
EXERCISES

1. Write a flow chart for the following recipe for cooking baked beans (taken from *The Natural Foods Cookbook*, Nitty Gritty Productions, Concord, California, 1972).

> Bring apple juice and water to a boil and add beans so slowly that boiling doesn't stop. Reduce heat after beans are in water and simmer 2 to 2½ hours or until beans are almost tender. Drain beans, reserving liquid, and add other ingredients to beans. Place in oiled baking dish and bake covered 2 to 3 hours in 2508 oven. Uncover for the last hour of cooking. If beans become dry, add a little of the reserved bean water. About 15 minutes before removing from the oven, add the fresh diced tomato.

2. Let m and n be positive integers. Draw a flowchart for lthe following algorithm for computing the quotient q and remainder r of dividing m by n.
    a. Set q to 0.
    b. If $m < n$, then stop. q is the quotient and m is the remainder.
    c. Replace m by $m - n$.
    d. Increment q by *1*.
    e. Go to step b.
3. Bench test the algorithm in the previous exercise. Use $m = 4$ and $n = 2$ as values.
4. Let a rectangle that is aligned with the coordinate axes be represented by the coordinates of its lower left and upper right corners, (*xmin*, *ymin*) and (*xmax*, *ymax*), respectively, as shown in the following illustration. Given two such rectangles, *R1* and *R2*, devise an algorithm that finds the rectangle, if any, that is common to both *R1* and *R2*. The input data are the eight real numbers representing the coordinates

of the rectangles' corners. The illustration shows two rectangles that have a common rectangle, shown in grey.



5. Suppose that a line segment is represented by its two endpoints, $PE = (xe, ye)$ and $PP = (xp, yp)$. For two line segments *L1* and *L2*, devise an algorithm that computes the coordinates of their point of intersection, if any. (Two line segments intersect if they have only one point in common.)
6. Make a list of all the automatic interactions that you have with computers—for example, utility bills, automatic mailing lists, class scheduling, and so on.
7. Give a detailed step-by-step procedure, also called an algorithm, for
    a. traveling from your place of residence to school or work,
    b. manually dividing one number by another (long division),
    c. taking an unordered set of numbers and putting them in ascending sequence (i.e., "sorting" the numbers in ascending sequence), or
    d. playing, and never losing, the game of Tic-Tac-Toe.