Programming Fundamentals

Hyesung Park
Na'el Abu-halaweh
Sonal S. Dekhane
Wei Jin
Robert Lutz
Richard W. Price
Tacksoo Im

Programming Fundamentals

Hyesung Park, Na'el Abu-halaweh, Sonal S. Dekhane, Wei Jin, Robert Lutz, Richard W. Price, Tacksoo Im

Table of Contents

1. Getting Started

1.1. Learning Outcomes

Students will be able to

- Understand computer basics.
- Understand programming basics.
- Understand binary number system.
- Begin using the Java programming language.
- Display output on the console.
- Explain the differences between syntax errors, runtime errors, and logic errors.

1.2. Key Terms

Review the important terms.

1.3. Resources

1.3.1. Text

- Think Java Computer Programming by Allen Downey and Chris Mayfield.
- Core Java : Core Java Complete by Cay S. Horstmann
- Essentials of the Java Programming: Essentials by Oracle.com
- Memory Bits and Bites
- Bits and Bytes
- Convert a decimal number to binary numbers
- How the Binary Number System Works
- Binary Addition
- Binary Subtraction
- Method of Complements

1.3.2. Video/Tutorial

- Core Java 11: Fundamentals by Cay S. Horstmann
- An Introduction to Java link
- Understand the Fundamental Concepts of Object-Oriented Programming Why OOP?
- Early Computing
- What is Algorithm?

- Kahn Academy: How Computers Work
- YouTube Video on Fetch Decode Execute Cycle

1.4. Overview

You might have heard about computers many times. You might have questions about how computers perform so many tasks. You might have questions about programming. What is programming? Welcome! If you start to ask all these questions, then you are in the right place to start to learn about programming. We use programming to develop software. A program is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, like solving a system of equations or finding the roots of a polynomial. It can be a symbolic computation, like searching and replacing text in a document or compiling a program.

Learning a programming language is similar to learning a foreign language, such as French. The ultimate goal of learning a foreign language is to be able to compose sentences to communicate (the communication problem). Similarly, the ultimate goal of learning to program is to write proper statements that solve a particular problem, such as calculating the volume of a sphere or calculating the projectile of a rocket.

A programming language offers its set of different types of statements for programmers to use. Before you can write code using a type of statements, you need to learn its

1. syntax
2. semantics

For French, syntax is the grammar of the language and semantics is what a sentence means. For programming, when we specify how the code should look like and structural rules, we are talking about syntax. Programming languages, such as Java, have very strict syntax rules, much stricter than a natural language (e.g. French). If your code does not comply with the syntax rules, the compiler (the translator) will not understand your code and therefore cannot translate your code into machine code that can be executed on CPU.

In order to learn how to compose the right sentences in French to convey certain meanings, you will normally first do some reading comprehension exercises to learn how to interpret what sentences mean. Similarly, in order to write code to solve a problem, you first need to read code written by others and understand what that code does. What a piece of code does is similar to the meaning of a sentence in a foreign language. We call this semantics of the code. Only if you understand the semantics (the behavior) of code can you compose the right code to have the desired behaviors.

Through syntax and semantics learning, you will build your tool set of a programming language's different types of statements. When the times come to write code to solve a problem, you need to pick up the right statements and use them in the right way and right order to achieve the desired behavior.

1.5. Basic Computer Architecture

The semantics of statement (an instruction written in high-level programming language, defined in the section above) is what it does on a computer. We need some very basic understanding and vocabulary in order to be able to state the behavior of a statement.

First, get some understanding of How Computer Works by watching the fun videos on Khan Academy: Khan Academy's How Computer Works. Please watch the last two videos "CPU, Memory, Input and Output" and "Hardware and Software" (staring Bill Gates and other real technical people).

Second, watch this video on YouTube for a more detailed explanation about how CPU execute instructions in stored in memory: YouTube video (It's less than 8 mins and please be patient and watch the whole video.)

We can see that memory holds instructions and data. Each memory location has a unique address. Instructions are fetched by CPU and executed by CPU. Besides the instructions, memory also holds data that need to be manipulated by the program.

1.6. Programming Basics

Computer programs, known as software, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine. Computers do not understand human languages, so you need to use computer languages to communicate with them.

Programs are written using programming languages.

1.6.1. What is Computer Science?

1.6.2. What is a Computer?

1.6.3. Early Computing

1.6.4. What is Programming?

High-level languages:

Java, Python, C, C++, PHP, Ruby, and JavaScript.

Low-level languages:

It is a machine language and only a computer is capable of reading and interpreting the low-level languages of a collection of binary digits or bits.

How do low-level languages and high-level language work (Compilation of Java Programs)?

> *Before programs can run, programs in "high-level languages" have to be translated into a "low-level language", also called "machine language". This translation takes some time, which is a small disadvantage of high-level languages.*
>
> *But high-level languages have two major advantages:*
>
> *It is much easier to program in a "high-level language". Programs take less time to write, they are shorter and easier to read, and they are more likely to be correct. "High-level languages" are portable, meaning they can run on different kinds of computers with few or no modifications. Low-level programs can only run on one kind of computer, and have to be rewritten to run on another.*
>
> *Two kinds of programs translate "high-level languages" into "low-level languages: "interpreters and compilers". An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.*

(Read more from the Compiling Java Program)

1.7. Binary Number System (Resources)

- Memory Bits and Bytes
- Encoding Information
- Decimal to Binary
- Binary & data
- The Binary Number System
- Binary Addition
- Binary Subtraction

1.7.1. Complements

In mathematics and computing, there are two types of complements, the radix-minus-one complement and the radix complements. The complement by itself is the radix complement.

These complements are used to make the arithmetic operation in the digit system easier.

While we are familiar with decimal system, we will discuss the decimal complements first. In the decimal system, there are the nines complement and tens complement.

1.7.2. Decimal Complements

If A is a decimal number, the nines complement of A is obtained by subtracting A from (10^n – 1); and the tens complement of A is that we add 1 with the nines complement of A.

For example

	Example 1	*Example 2*	*Example 3*
Decimal number	4206	9674	124134
Nines complement	5793	0325	875865
Tens complement	5794	0326	875866

We can illustrate the use of complements by taking 4206. The nine's complement of 4206 will be

$$9999 - 4206 = 5793$$

Then, ten's complement of 5793 is 5794 as below.

$$5793 + 1 = 5794$$

1.7.3. Binary Complements

The principles of complements in the decimal system can be translated into the binary digit system. If B is a binary number, the one's complement of B is obtained by subtracting each digit of B from 1, and the two's complement of B is obtained by adding 1.

For example

		Number
1	Binary number	1111 0000 1111
2	One's complement	0000 1111 0000
3	Two's complement	0000 1111 0001

We can illustrate the example of binary complements as below.

* Step 1: First, make the one's complement of 1111 0000 1111.

```
        1111 1111 1111
(-)     1111 0000 1111
_____________________
        0000 1111 0000
```

Simply you will change 0 to 1 and 1 to 0.

* Step 2: Two's complement of One's complement of 1111 0000 1111 which is now 0000 1111 0000. You are adding 1 to 0000 1111 0000 to make it Two's complement.

```
        0000 1111 0000
(+)     1
_____________________
        0000 1111 0001
```

More example resources

Two’s Complement
Ones' complement

1.8. Algorithms

1.8.1. Algorithms are Everywhere!

- Amazon.com
- Bank Systems
- Cash Registers
- Hospitals
- Internet Browsers
- Search Engines

1.8.2. What are Algorithms?

An algorithm specifies a series of steps that perform a particular computation or task. Algorithms were originally born as part of mathematics - the word "algorithm" comes from the writer Muhammad ibn Musa-al-Khwarizmi - but currently the word is associated with computer science.

More background information about Algorithms

In programming, each step should be clear and defined precisely to solve a particular problem most effectively.

What is Algorithm in brief? - Video Clip Resource

1.8.3. Algorithm Examples

Write an algorithm to add two numbers entered by user:

Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum. sum←num1+num2
Step 5: Display sum
Step 6: Stop

Write an algorithm to calculate and record the interest on a home mortgage for a client.

(Assuming that balance of the mortgage and the interest rate appear in the client's record - a collection of related data items. e.g. data on a given client, and a file is a collection of similar records) - Flowchart is provided in Figure 1.

Step 1: Start

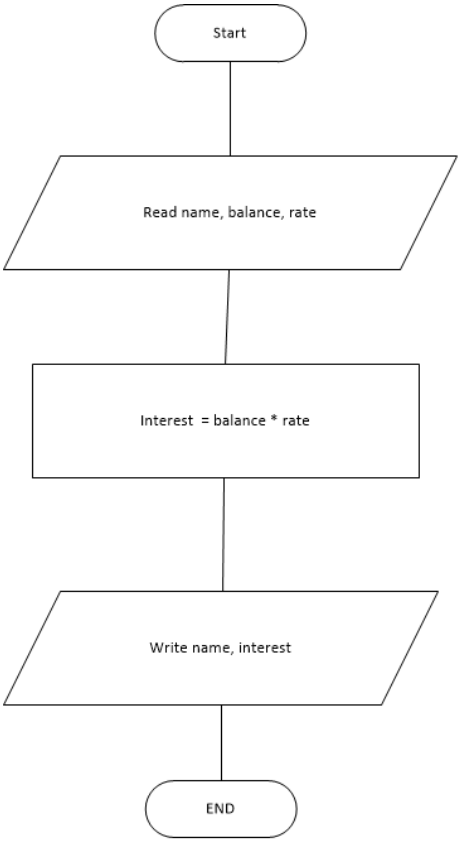
Step 2: Obtain name, balance, and rate from the client's record.

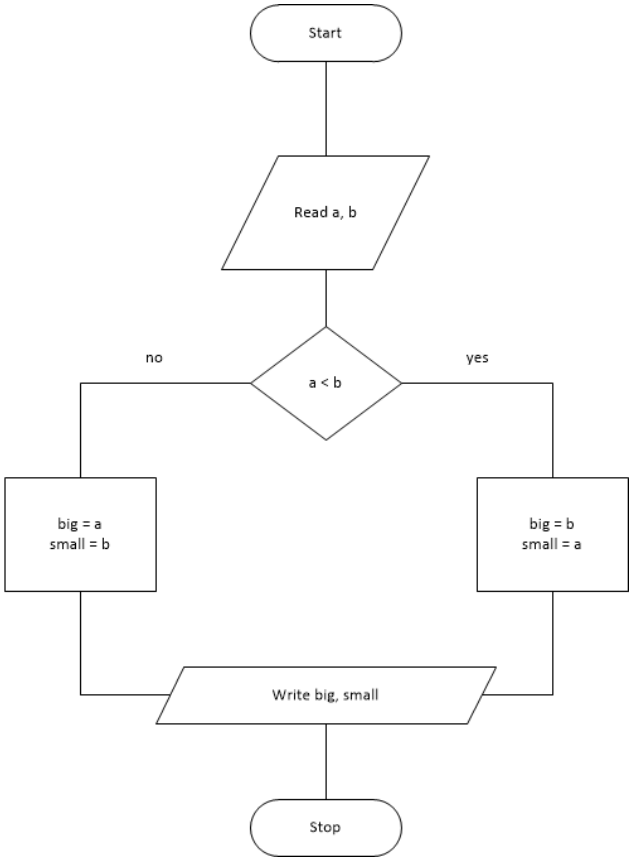
Step 3: Compute: interest = balance * rate.

Step 4: Record client's name and interest in the interest file.

More examples of Algorithm in Programming

There are two ways of presenting an algorithm. One way is by a flowchart and another way is to write it in pseudocode language. Figure 1 shows two flowcharts. First one is based on the mortgage interest algorithm and second flowchart reads two numbers, a and b. Then prints them in descending order, after assigning the larger number to big and the smaller number to small. Two arrows leaving the decision if a < b. If the answer is no, it is labeled "no" and the other labeled "yes".





1.9. Code Conventions for the Java Programming Language

Coding Standards
Code Conventions by Oracle

1.10. What is Java

Java is one of the most widely used computer programming languages. According to Hostmann (2016), Java is a well-designed programming language and is an efficient and secure environment with a huge library.
The White Paper Buzzwords

- Simple
- Object Oriented
- Distributed
- Robust
- Secure
- Architecture-Neutral

- Portable
- Interpreted
- High-Performance
- Multithreaded
- Dynamic

1.11. History of Java

1.11.1. History of Java by javapoint

1.11.2. A Short History of Java

1.12. Java Versions

Version	Release Date	Extended Support Until
JDK Beta	1995	
JDK 1.0	January 1996	
JDK 1.1	February 1997	
J2SE 1.2	December 1998	
J2SE 1.3	May 2000	
J2SE 1.4	February 2002	February 2013
J2SE 5.0	September 2004	April 2015
Java SE 6	December 2006	December 2018
Java SE 7	July 2011	July 2022
Java SE 8	March 2014	March 2025
Java SE 9	September 2017	N/A
Java SE 10	March 2018	N/A
Java SE 11	September 2018	September 2026

Version	Release Date	Extended Support Until
java SE 12	March 2019	N/A
java SE 13	September 2019	N/A
java SE 14	March 2020	N/A

1.13. Features of Java Programs

1.13.1. Basic Form of the first program "MyFirstJava"

```
public class MyFirstJava {
   public static void main(String[] args) {
      System.out.println(“My First Java!”);
   }
}
```

When MyFirstJava program runs it displays:

My First Java!

If you compare this output with the original code of **MyFirstJava**, you will find there is no double quotations. My First Java! is a String literal in the code and it is defined by using double quotation like System.out.println("My First Java!"). System.out.println is to display the String literal you want to print out on the console/screen (Read more)

> *"System.out.println displays results on the screen; the name println stands for “print line”. Confusingly, print can mean both “display on the screen” and “send to the printer”. In this book, we’ll try to say “display” when we mean output to the screen. Like most statements, the print statement ends with a semicolon (;).*
>
> *Java is “case-sensitive”, which means that uppercase and lowercase are not the same. In this example, System has to begin with an uppercase letter; system and SYSTEM won’t work."*

Following link shows how this MyFirstJava program works step by step by using the Java Visualizer program. MyFirstJava.java by Java Visualizer or use following embedded visualizer by scrolling down and to the right to expand the window.

Class and Methods in brief

A **method** is a named sequence of statements. This program defines one method named main:

public static void main(String[] args) The name and format of main is special: when the program runs, it starts at the first statement in main (All java programs begin here) and ends when it finishes the last statement. Later, we will see programs that define more than one method.

A **class** is a collection of methods. In this program defines a class named MyFirstJava. You can give a class any name you like, but it is conventional to start with a capital letter. The name of the class has to match the name of the file it is in, so this class has to be in a file named **MyFirstJava.java**.

Java uses **curly braces ({ and })** to group things together. In MyFirstJava.java, the outermost braces contain the class definition, and the inner braces contain the method definition.

1.13.2. JavaTutor Example

JavaTutor Example (click "Vizualize Execution")

System.out.println appends a newline. This newline starts from the beginning of the next line. If you want to make the String phrases appear in one line, then use **print** instead of **println** (see below example).

ClassName is written by a programmer and from the example above, MyFirstJava is the class name.

main is a special method which makes the program execute and **println** means display a message on the screen. The println statement terminates with a semi-colon(;).

1.13.3. Use of print vs. println MyPrintln

```
public class MyPrintln {
   public static void main(String [] args) {
      System.out.print("My First Java class");
      System.out.println(" is fun!");
   }
{
```

1.13.4. JavaTutor Example

Java Tutor Example (click "Vizualize Execution")

1.13.5. Use of println with ThreeMessages

```
public class ThreeMessages {
   public static void main(String [] args) {
      System.out.println("First, create a java program.");
      System.out.println("Second, compile a java program.");
      System.out.println("Third, execute a java program.");
   }
}
```

1.13.6. JavaTutor Example

Java Tutor Example (click "Vizualize Execution")

1.13.7. Simple Computation

```
public class SimpleComputation {
   public static void main(String[] args) {
      System.out.println("10.5 * 5 / 4 – 5.2 / 2.0 = ");
      System.out.println("10.5 * 5 / 4 – 5.2 / 2.0");
   }
}
```

1.13.8. JavaTutor Example

Java Tutor Example (click "Vizualize Execution")

1.13.9. Escape Sequences

- How to use escape sequences
- From Java Tutorials

1.13.10. Programming Errors in General (not only for Java)

- Syntax errors
- Runtime errors
- Logic errors

1.14. Creating, Compiling, and Executing a Java Program

You must first set up the environment to compile and execute java programs. To be able to compile and run programs, you must install the JDK and configure it. Java SE 12.0.1 is the latest release for the Java SE Platform (as of June 09, 2019). Orale JDK

1.14.1. Video: JDK 12 install in Windows 10

1.14.2. Video: JDK 12 install on Mac OS X

1.14.3. Compiling and Running a Simple Program

1.15. Exercises

1.15.1. Write a program named Banana.java that displays as below:

A banana is an edible fruit.
If you wait the correct amount of time for it to ripen,
it will be sweet and delicious.

1.15.2. Write a program named Fibonacci.java that displays the result of

$$1 + 1 + 2 + 3 + 5 + 8 + 13 + 21$$

1.15.3. Determine the nines complements of the decimal numbers:

4535
507606
78534019

1.15.4. Determine the one's and two's complements of the binary numbers:

110011
10001100
10111011101

1.15.5. Convert each binary number to its decimal equivalent:

101
110110
111011001
101010101

1.15.6. Convert each decimal number to its binary equivalent:

2857
4503
46098
694

1.15.7. Write a program named Formula.java that displays the result of

$$\frac{5.6 \times 5.6 - 4 \times 6.2 \times 5.1}{2 \times 7.8 - 3 \times 5.6}$$

1.15.8. Write a program named Molecular.java that displays the following table:

Atom	Weight (grams / molecule)
H	1.00794
C	12.0107
O	15.9994

1.15.9. SpeedLight.java

The speed of sound is approximately 340 meter per second. Assume that you just saw a lightning flash and you heard the sound of thunder 5 seconds later. Write a program named SpeedLight.java that calculate the distance to a lightning strike based on the time elapsed between the flash and the sound of thunder.

1.15.10. Chocolate.java

Assume there are 9 bags of chocolate bars. Each bag has two chocolate bars. The bag is big enough to have three chocolate bars. If you want to take all the chocolates out of each bag and add three chocolate bars to each bag, how many bags will you need? Write a program to compute the number of bags you will need to add three chocolates instead of two chocolates.

1.15.11. Stamps.java

Susan and Jean just started collecting stamps as a hobby. Susan has 8 endangered animal collection stamps and Jean has 40 racing car collection stamps. How many more does Jean have than Susan? Write a program named Stamps.java that compute the difference between Jean's and Susan's collections of the stamps.

1.15.12. Cycle.java

In the Cycle shop, there are 10 bicycles and X numbers of tricycles. Assume that you count the number of wheels of the bicycles and there are 47 wheels of the bicycles and tricycles. How many of tricycle does this Cycle shop have? Write a program named Cycle.java and compute the total number tricycles at the shop.

1.15.13. FindX.java

Write a program named FindX.java to compute the number X based on the following formula:

$$5 + 19 + X + 47 = 194$$

1.15.14. MaleStudent.java

Assume that there are 389 students in a small middle school. There are 175 female students. Write a program named MaleStudent.java to compute how many students are male in this middle school.

1.15.15. Circle.java

Write a program named Circle.java that displays the area and perimeter of a Circle that has a radius of 9.5 using the following formula:

$$area = radius \times radius \times Math.PI$$
$$perimeter = 2 \times radius \times Math.PI$$

1.16. Do You Have Any Questions about Chapter 1?

Comments

2. Datatype, Variables, and Expressions

2.1. Learning Outcomes

- Understand the concept of variables.
- Understand the concept of data types in Java.
- Understand the syntax and semantics of data declaration statements and assignment statements.
- Compose arithmetic and String expressions.
- Use Scanner to get information from user.
- Print values and texts.

2.2. Key Terms

Review the important terms on Think Java Chapter 2 and Chapter 3.

2.3. Resources

Text

- Think Java by Allen Downey and Chris Mayfield. You can use the online interactive version or download the PDF version.
- Oracle Java Tutorial

Video/Instruction

Converting a decimal number to floating-point format

2.4. Overview

Programming is to write code that can be executed by the computer to solve a particular problem. A program consists of instructions that a computer can execute. We call instructions written in a high-level programming language ***statements***. In this chapter, we will learn about statements that get input from user, perform computations, store data and computation results in memory and display messages.

The following Java program was first introduced in Chapter 1. In fact, it contains only one statement: ***System.out.println("My First Java!");***

MyFirstJava.java

```
public class MyFirstJava {
   public static void main(String[] args) {
      System.out.println("My First Java!");
   }
}
```

We can see that the program MUST contain other lines to conform to Java structure rules (i.e. syntax). The following is the class block or "wrap", containing the class header and class block (i.e. the pair of curly brackets).

```
public class MyFirstJava {

}
```

The following is the method block or "wrap", containing the method header and method block (i.e. the pair of curly brackets).

```
public static void main(String [] args) {

}
```

We can see that the method block is indented inside the class block. We can also see that the statement is further indented inside the method block. The indentation is for highlighting the structure of a program.

The following is another Java program that contains more statements. We will use it to illustrate some important Java concepts. We can see that the Java file name and the class name have to be the same. Java is case sensitive, which means that upper and lower cases have to match exactly.

MySecondJava.java

```
public class MySecondJava {
   public static void main(String[] args) {
      int studentsPerClass = 28;
      int totalStudents;
      double myWeight = 179.3;
      double myTargetWeight;
      String greeting = "Hello, ";
      String name;

      totalStudents = studentsPerClass * 10;
      myTargetWeight = myWeight - 8;
      name = "John";
   }
}
```

2.5. Variable

As introduced in Chapter 1, data are stored in memory. Instead of directly using memory addresses, Java uses variables. When the Java compiler translates a program, a variable is given a specific memory location (address).

In the above program ***MySecondJava.java***, ***studentsPerClass***, ***myWeight***, and ***greeting*** are variable names, each representing a memory location for storing data.

We want to choose meaningful names for variables, indicating the purpose of each variable. If the name you choose consists of only one word, spell that word in all lowercase letters, such as ***greeting***. If it consists of more than one word, capitalize the first letter of each subsequent word, such as ***studentsPerClass*** and ***myWeight***.

For more information, see this site: Java variable name rules and conventions

2.6. Data Type

A program can process different types of data. We will talk about three basic data types in Java: integer, real value, and text. Integers are whole numbers (without a decimal point). Real values are numbers that include decimal places. Texts are sequences of characters, including punctuation, symbols, and whitespace. Every value in Java has a corresponding data type. The table below shows examples of each of the three types.

Table 1. Basic Data Types

Data Type	Example Values	Java Data Types

integer	2, -2, 0, 834529	byte, short, int, long
real value	2.0, -2.235, 0.0, 8329.123782	float, double
text	"Hello World!", "Coconut", "0", "4 + 6"	String

Note that text data are always surrounded by quotes. Some text may look like numbers, but as long as they are surrounded by quotes, they are treated like text.

Java uses its special key words to represent these data types. There are four integer types.

- **byte**: 8-bit integer
- **short**: 16-bit integer
- **int**: 32-bite integer (most often used)
- **long**: 64-bite integer

Note: An integer value as in Table 1 is by default as the ***int*** type. To represent a ***long*** value, we need to append the value with an L or l (i.e. either upper or lowercase). For example, 2L and 23458907234556L are two long values.

The format to represent real values are called floating point. There are two floating-point data types:

- **float**: 32-bit floating point
- **double**: 64-bit floating point (most often used)

An example for converting a decimal number to float

If you are curious, you can watch this video for converting a decimal number to floating-point format. The example used in the video is 13.1875. You will see how it is converted to float.

Note that a real value as in Table 1 is by default the ***double*** type. To represent a float value, we need to append the value with an F or f (i.e. either upper or lowercase). For example, 2.0F and -2.235f are two values of the ***float*** type.

For text, the data type name is **String**. Note that, the type name is capitalized.

2.7. Declaration Statement: Declare Data Type for a Variable

In Java, each variable must be associated with a certain data type. We use declaration statement to specify the data type of a variable. For example, the following statement declares the variable ***studentsPerClass*** as an ***int*** variable.

int studentsPerClass;

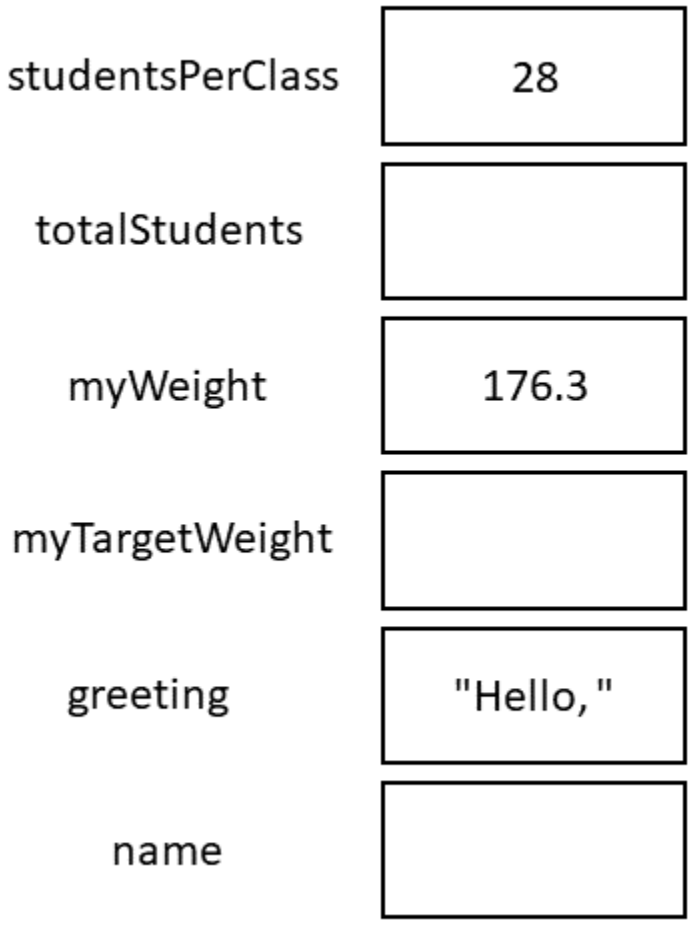
The following is also a declaration statement. Besides declaring the variable ***studentsPerClass*** as an ***int*** variable, it also initializes the variable with an int value 28.

int studentsPerClass = 28;

You can also declare multiple variables of the same type in one declaration statement.

int studentsPerClass = 28, totalStudents;

After all the declaration statements in ***MySecondJava.java*** are executed, the memory looks like below:



2.8. Assignment Statement: Change the Value of a Variable

The following statements are assignment statements. Note that they are similar to a declaration statement with an initial value, but there is no data type at the left. These variables have already been declared earlier, so no data types are needed on the left.

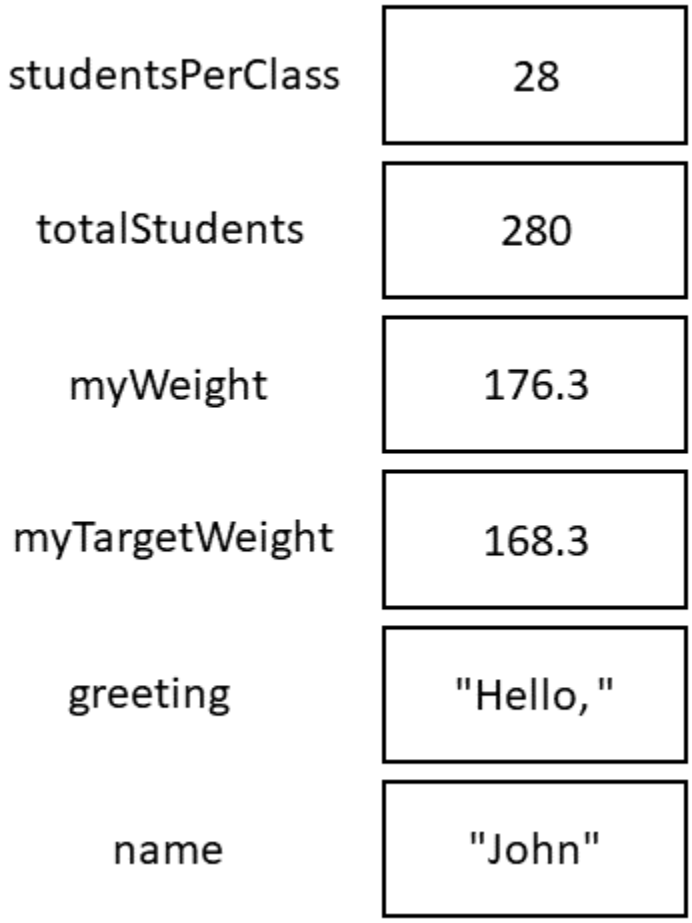
```
totalStudents = studentsPerClass * 10;
myTargetWeight = myWeight - 8;
name = "John";
```

An assignment statement puts a new value into a variable. The right side of an assignment statement is first evaluated, and the result will be put into the variable on the left.

The following describes what happens after each of the statements above is executed.

- Since ***studentsPerClass*** contains a value 28, ***studentsPerClass * 10*** will be 280. After the first assignment, the variable ***totalStudents*** is 280.
- Since ***myWeight*** contains a value 176.3, ***myWeight - 8*** will be 168.3. After the second assignment, the variable ***myTargetWeight*** is 168.3.
- The last assignment gives the variable ***name*** a value "John".

After these assignment statements are executed, the memory looks like below.



2.8.1. JavaTutor Example

JavaTutor Example (click “Vizualize Execution”)

Note that if a variable is declared for a certain data type, trying to put a value of another incompatible type will cause a violation of syntax. For example, the compiler will report a syntax error for the following statement, since "168.3" is a String (text) and is not compatible with a double data type.

```
myTargetWeight = "168.3"; //This will cause a syntax error!
```

2.9. Expressions

An arithmetic expression (an expression for numeric values) is often used in an assignment statement. For example, ***studentsPerClass * 10*** and ***myWeight - 8*** are both arithmetic expressions. Please refer to Section 2.5 of Think Java for more information on this topic.

Java also has an operator, **%**, to get remainder of a division. Please refer to Section 3.8 of Think Java for more information on the remainder operator.

Further, Java also has String operations. Please refer to Section 2.8 Think Java for more information on String expressions. This section also talk about how String values and numeric values are "added" together.

For a more detailed introduction to floating-point numbers and the associated rounding errors, please read Section 2.6 and 2.7 of Think Java.

Finally, Java can convert a value of one data type to a value of another data type. For example, ***(int) 123.56*** converts a double value 123.56 to an int value 123. ***(int)*** is called a type cast operator. Please refer to Section 3.7 of Think Java for more information on this topic.

2.10. Output Statement

The current version of ***MySecondJava.java*** does not interact with the user who runs the program at all. Variable declarations and assignments all happen in memory. We will extend the program by adding output statements at the end. They communicate with the user by printing information on the screen.

MySecondJava.java

```
public class MySecondJava {
   public static void main(String [] args) {
      int studentsPerClass = 28;
      int totalStudents;
      double myWeight = 176.3;
      double myTargetWeight;
      String greeting = "Hello, ";
      String name;


      totalStudents = studentsPerClass * 10;
      myTargetWeight = myWeight - 8;
      name = "John";

      System.out.println(totalStudents);
      System.out.println(myTargetWeight);
      System.out.println(greeting + name + "!");
   }
}
```

The first output statement will print the value of variable ***totalStudents***, which is 280, on a line. The second output statement will print the value of the variable ***myTargetWeight***, which is 168.3, on the second line. The third output statement will print the result of the String expression ***greeting*** + ***name*** + ***"!"***, which is "Hello, John!" on the third line. Note that the quotation marks are not printed. The following is what the program will print:

280
168.3
Hello, John!

2.10.1. Formatting outputs using printf

In addition to print and println, you can use ***System.out.printf*** to format the output. It's a very useful method and will be used in many of the exercises at the end of the chapter. Please read Section 3.5 of Think Java on how to have more control on output format.

2.11. Comments and Order of Execution

The following program contains comments. Comments are not instructions for the computer to follow, but instead notes for programmers to read. There are two types of comments in Java.

Line comments start with //. Anything following //, on the same line, will not be executed. Often, at the very beginning of a program, comments are used to indicate the author and other information. Even though line comments can be used, but for comments that span several line, we usually use block comments, which start with /* and end with **/. Anything in between /** and */ will not be executed. Comments are also used to explain tricky sections of code or disable code from executing.

MyThirdJava.java

```
/* MyThirdJava.java
  Author: ___ ___
  The program is for illustrating the order of execution.
 */

public class MyThirdJava {
   public static void main(String [] args) {
      int a = 10;  //variable declaration with an initial value
      int b = 5;   //variable declaration with an initial value
      int sum;     //variable declaration

      sum = a + b; //assignment
      a = 100;     //assignment

      System.out.println(sum); //output
   }
}
```

Now take a look at the program, can we give a guess as to what the program will print? Is it 15 or 105?

Some of you will guess 105. The reasoning might be the following: Even though a starts with 10, but it changes to 100. Since variable sum contains the result of a + b, so sum should be 105 at the end. However, this is INCORRECT.

The statements are executed in the order that they show up in the program. When line ***sum = a + b;*** is executed, sum gets the value 15. When line ***a = 100;*** is executed, a gets a new value 100. Note that sum is not changed and retains the value 15. The output statement thus prints 15.

2.12. Declare Constants and Get User Input with Scanner

Read the following program.

Miles100ToKMs.java

```
/* Miles100ToKMs.java
  Author: ___ ___
  The program is for illustrating the order of execution.
 */

public class Miles100ToKMs {
   public static void main(String [] args) {
      final double KMS_PER_MILE = 1.609; //declare a constant variable
      double miles = 100.0;


      double kilometers = miles * KMS_PER_MILE;


      System.out.println(miles + " miles = " + kilometers + "  kilometers");
   }
}
```

The first statement ***final double KMS_PER_MILE = 1.609;*** declares a constant variable ***KMS_PER_MILE***. Note that the keyword ***final*** is prepend to the declaration. For a constant variable, an initial value must be assigned to the variable at the declaration. Also note that, by convention, the name for a constant variable is all capital letters with underscores separating words. However, if a variable's name is all captial letters, it is not automatically a constant. The keyword ***final*** is needed to make a variable constant. Please refer to Section 3.4 of Think Java for more information on constants.

We can see that the program converts 100.0 miles into kilometers. First, variable ***miles*** gets a value 100.0. Then the variable ***kilometers*** gets a value 160.9. Last the output statement prints the value of the expression ***miles + " miles = " + kilometers + " kilometers"***.

Let's talk about ***miles + " miles = " + kilometers + " kilometers"***. It is a mixed-type expression and evaluated to ***"100 miles = 160.9 kilometers".*** Please refer to the note below and Section 2.8 Think Java for more information.

String and Numeric Values Mixed Operations

Variables ***miles*** and ***kilometers*** are double values, while ***" miles = "*** and ***" kilometers"*** are String values. Since the addition operator + is left-associative, the expression is evaluated from left to right. The following shows how the expression is evaluated:

1. First, ***miles* + *" miles = "*** is evaluated. When a numeric value is added to a String value, the Java compiler will insert code to automatically convert the numeric value to the corresponding String value. In this case, ***miles***'s value 100.0 is converted to ***"100.0"***. Then ***"100.0" + " miles = "*** will be performed and results in a longer String ***"100 miles = "***.
2. Next, ***"100 miles = " + kilometers*** will be evaluated. It is a mixed-type operation and ***kilometers***'s value 160.9 is converted to "160.9". ***"100 miles = " + "160.9"*** will be carried out and results in ***"100 miles = 160.9"***.

Last, ***"100 miles = 160.9" + " kilometers"*** will be evaluated. Both values besides the + operator are String values and they are concatenated, resulting in ***"100 miles = 160.9 kilometers"***.

Now we understand the program: It converts 100 miles to the corresponding kilograms. However, this program has a big drawback. What is it?

This program can ONLY convert 100.0 miles. If there is a need to convert 200.0 miles, the programmer will need to modify 100.0 to 200.0. This basically changes the program itself and the program will need to be recompiled. A user will need to get this new program for converting 200.0 miles.

The following is a program that can get user input for distance in miles and then convert it into the corresponding kilometers. There is no need to modify and recompile the program for converting different values. The program can be run as many times as needed to convert whatever the values to be converted.

MilesToKMs.java

```
/* MilesToKMs.java
  Author: ___ ___
  The program is for illustrating the order of execution.
 */

public class MilesToKMs {
   public static void main(String [] args) {
      final double KMS_PER_MILE = 1.609; //declare a constant variable

      java.util.Scanner  input = new java.util.Scanner(System.in);
      System.out.print("Enter the distance in miles: ");
      double miles = input.nextDouble();


      double kilometers = miles * KMS_PER_MILE;


      System.out.println(miles + " miles = " + kilometers + " kilometers");
   }
}
```

The following are three new lines in the above program:

```
java.util.Scanner  input = new java.util.Scanner(System.in);  //new line 1
System.out.print("Enter the distance in miles: ");          //new line 2
double miles = input.nextInt();                            //new line 3
```

The second line is an output statement that prints a prompt for a user. It tells the user that the program expects a distance in miles to be entered. We will focus on the other two lines. We will talk about the first and third line below.

2.12.1. Create a Scanner Object

Line 1 ***java.util.Scanner input = new java.util.Scanner(System.in);*** looks complicated, but a closer look tells us that it's a declaration statement with an initial value assigned to the declared variable. The data type is ***java.util.Scanner***, the variable is ***input***, and the initial value is ***new java.util.Scanner(System.in);***.

The data type ***java.util.Scanner*** looks very long, but it really is ***Scanner.*** We will rewrite the program above as follows:

MilesToKMs.java

```
/* MilesToKMs.java
   Author: ___ ___
   The program is for illustrating the order of execution.
 */
import java.util.Scanner;

public class MilesToKMs {
    public static void main(String [] args) {
        final double KMS_PER_MILE = 1.609; //declare a constant variable

        Scanner input = new Scanner(System.in);
        System.out.print("Enter the distance in miles: ");
        double miles = input.nextDouble();


        double kilometers = miles * KMS_PER_MILE;


        System.out.println(miles + " miles = " + kilometers + " kilometers");
    }
}
```

We used the import statement ***import java.util.Scanner;*** to tell the Java compiler that data type Scanner is defined in the package ***java.util*** of the Java installation. A package contains multiple related classes. ***java.util.Scanner;*** is the full pathname of the data type Scanner. If we use the import statement, there is no need to use the full name in the remainder of the program. As a result,

java.util.Scanner input = new java.util.Scanner(System.in);

is simplified to

Scanner input = new Scanner(System.in);

Word ***new*** is a keyword for creating a new object. The expression ***new Scanner(System.in)*** creates a Scanner object that will extract information from ***System.in*** (see the note below). This object is assigned to the variable ***input***. Note that when an object is assigned to a variable, the reference (or location) of the object is stored in the variable.

Scanner, System, System.out, and System.in

Scanner *is a data type defined in class Scanner (in file Scanner.java). Similarly, the data type* **String** *is also a class defined* **String.java__** file. The reason that we don't need an import statement for String is because String belongs to the **java.lang** package, which is imported automatically.

System is another class in the ***java.lang*** package. It is a class that provides methods related to the "system" or environment where programs run.

System.out is an object contained in the ***System*** class. It represents the standard output device, normally the monitor.

System.in is an object representing the standard input device, normally the keyboard.

2.12.2. Invoke Methods on the Scanner Object to Retrieve Information

Remember that we can invoke the ***print*** and ***println*** methods on ***System.out*** by ***System.out.print(…)*** or ***System.out.println(…)*** to print information.

Similarly, we can also invoke different methods on a Scanner object to retrieve different types of information. For example, ***input.nextInt()*** will invoke the nextInt method on the Scanner object ***input*** and retrieves an integer. See Table 2 for more Scanner methods.

Table 2: Scanner Methods

Method	Description
nextInt()	retrieve an int value
nextDouble()	retrieve a double value
next()	retrieve a word (a word is a sequence of characters with no space or tab or newline)
nextLine()	retrieve a line of characters

When the statement ***double miles = input.nextDouble();*** is executed, the program will wait and allow the user to enter data. Whatever the user enters will be retrieved and returned by the ***nextDouble*** method and then assigned to the variable ***miles***.

See the step-by-step walkthrough of this program at Java Visualizer.

Note that when using ***nextLine*** and the other next methods (e.g. ***next***, ***nextDouble***, ***nextInt***) together, there is an unexpected behavior. Please read Section 3.9 of Think Java to understand this phenomenon and how to deal with this issue in your program.

2.13. Augmented Assignment Operators

Java designers want to make life easier for Java developers and they designed some short-cut assignment operators to save typing time.

Let's look at the following three statements. You might say, "Hi, x cannot be equal to x + 5!"

$$x = x + 5$$

We need to remember, however, that this is not a math equation, but an assignment statement. The right side of an assignment statement, which is always an expression, is first evaluated. The value of the expression will be assigned to the variable on the left.

Assume ***x***'s original value is 10, ***x + 5*** is evaluated to 15 and then 15 will be put into ***x***. After the assignment statement is executed, ***x***'s value becomes 15.

For assignment statements that have similar format, Java has a set of augmented assignment operators. For example, the statement on the left of each line below is equivalent to the statement on the right. The operators used in the statements on the right side are the ***augmented assignment operators*** **+=, -=, *=, /=, %=**. Note that there is no space inside the operators.

```
x = x + 5; <==> x += 5;
x = x - 5; <==> x -= 5;
x = x * 5; <==> x *= 5;
x = x / 5; <==> x /= 5;
x = x % 5; <==> x %= 5;
```

Note that the format of the original assignment statement has to be the following, where both <var>'s refer to the same variable.

```
<var> = <var> <operator> <expression>
```

Some Tricky Points

It's tricky when <expression> is more complicated than a single item. For example,

```
x = x - 5 + 4;
```

is not equivalent to

```
x -= 5 + 4; //WRONG
```

Instead, we need to convert the original assignment statement to the following before converting it using the augment assignment operator +=.

```
x = x - (5 - 4);
```

and then it can be safely converted to the following equivalent statement:

```
x -= 5 - 4; //CORRECT
```

If a variable is increased by 1 or reduced by 1, there are ways to further save typing. We can use ***increment operator*** ++ and ***decrement operator --***. On each line, all statements are equivalent.

```
x = x + 1; <==> x += 1; <==> x++; <==> ++x;
x = x - 1; <==> x -= 1; <==> x--; <==> --x;
```

Post/Pre Incrementation/Decrementation

There are some differences between the following pairs, if you used increment or decrement operators inside another statement.

```
x++ (post-incrementation) v.s. ++x (pre-incrementation)
x-- (post-decrementation) v.s. --x (pre-decrementation)
```

For x++, the value of x is used before x is incremented. For example, for the following two statements, x's value 10 first assigned to y and then x is incremented to 11, so after both statements are executed, x is 11 and y is 10. .

```
x = 10;
y = x++;
```

For ++x, x is incremented first before the value of x is used. For example, after for the following two statements, x is first incremented to 11 and then its value 11 is assigned to y, so after both statements are executed, x is 11 and y is 11.

```
x = 10;
y = ++x;
```

We will not dive too deep into this topic here. We believe that it makes code difficult to read and, therefore, we discourage this practice.

2.14. Exercises

2.14.1 Exercise 1

Write a program to convert 100 Fahrenheit to the corresponding temperature in Celsius. User proper variable names for the temperature in Fahrenheit and thee temperature in Celsius. Include a proper arithmetic expression to do the conversion.

2.14.2 Exercise 2

Write a program that does the following:

a. creates variables named day, date, month, and year. The variable day will contain the day of the week (like Friday), and date will contain the day of the month (like the 13th).
b. Assign values to those variables that represent 2019 Labor Day's date (9/2/2019, Monday).
c. Display the value of each variable on a line by itself. This is an intermediate step that is useful for checking that everything is working so far. Compile and run your program before moving on.
d. Finally, modify the program so that it displays the date in standard American format, for example: Thursday, July 16, 2015. 5. Modify the program so it also displays the date in European format.

American format:
Thursday, July 16, 2015
European format:
Thursday 16 July 2015

The final output should be in the following format. Note that the actual date should be for 2019 Labor Day.

2.14.3 Exercise 3

The point of this exercise is to (1) use some of the arithmetic operators, and (2) start thinking about compound entities (like time of day) that are represented with multiple values.

a. Create a new program called Time.java. From now on, we won't remind you to start with a small, working program, but you should.
b. Create variables named hour, minute, and second. Assign values to these variables that represent the time 30 seconds after 5:15pm. Use a 24-hour clock. For example, for 2pm, the value of hour is 14. Make the program calculate and display the number of seconds since midnight.
c. Calculate and display the number of seconds remaining in the day.
d. Calculate and display the percentage of the day that has passed. You might run into problems when computing percentages with integers, so consider using floating-point.

Hint: You might want to use additional variables to hold values during the computation. Variables that are used in a computation but never displayed are sometimes called "intermediate" or "temporary" variables.

2.14.4 Exercise 4

Write a program that converts a temperature from Celsius to Fahrenheit. It should

a. prompt the user for input,
b. read a double value from the keyboard,
c. calculate the result,
d. format the output to one decimal place using printf.

Here is the formula. Be careful not to use integer division! F = 9/5 C + 32

2.14.5 Exercise 5

Write a program that calculates the base area and volume of a cylinder, given the radius of the base circle and the length of the cylinder. Please use 3.1416 as the PI value.

2.14.6 Exercise 6

Write a program that convert a given total number of seconds into hours, minutes and seconds. It should (1) prompt the user for input, (2) read an integer from the keyboard, (3) calculate the result, and (4) display the output.

The following are two examples:

- if the total number of seconds is 125, it is 0 hours, 2 minutes and 5 seconds.
- If the total seconds is 7450, it is 2 hours, 4 minutes and 10 seconds.

2.14.7 Exercise 7

Given the length, width and height of the room, number of windows and number doors, the program will calculate the gallons and quarts of paint are needed to paint the room. For gallons, print an integer value. For the quarts, print a real number, no need to round down to integer.

Assume that (1) ceiling is painted, (2) 1 gallon of paint covers about 350 square feet, (3) each window is 15 square feet, and (4) each door is 21 square feet.

2.14.8 Exercise 8

Write a program that prompts the user to enter a number between 0 and 1000 (not including 1000) and print the digits in reverse order. The following are some examples:

- For a given value 256, the program should print 652.
- For 23, since the digit at hundreds is 0, the program should print 320.
- For 10, since the digit at ones and hundreds are 0, the program should print 010.
- For 0, the digits at ones, tens and hundreds are all 0, so the program should print 000.

2.14.9 Exercise 9

Write a program that prompts the user to enter a monthly saving amount and displays the account value after each of the first six months. When displaying the account values, please use printf to print only two decimal places.

We will assume that the yearly interest is 5%. Please declare a constant variable for the monthly interest rate as follows in your program:

```
final double MONTHLY_RATE = 0.05/12;
```

The following are how to calculate the account value at the end of each month:

```
month1Value: monthlySaving * (1+MONTHLY_RATE)

month2Value: (monthly1Value + monthlySaving) *  (1+MONTHLY_RATE)

month3Value: (monthly2Value + monthlySaving) *  (1+MONTHLY_RATE)
```

and so on.

2.14.10 Exercise 10

Write a program to calculate how many mile, feet, and inches a person walks a day, given the stride length in inches (the stride length is measured from heel to heel and determines how far you walk with each step), the number of steps per minute, and the minutes that person walks a day. Note that 1 mile is 5280 feet and 1 foot is 12 inches. In your program, there should be the following constants:

```
final int FEET_PER_MILE = 5280;

final int INCHES_PER_FEET = 12;

final int INCHES_PER_MILE = FEET_PER_MILE * INCHES_PER_FEET;
```

Hints: In the computation step, your program should first calculate the total inches that person walks a day. Then convert it to miles, remaining feet, and remaining inches. All variables should be of int type.

2.14.11 Exercise 11

Write a program that prompts user to enter the weight in pounds (real value) and height in feet and inches (both integers) and then calculates the BMI of the person. Please limit the print out of the BMI value to two decimal digits using printf.

Note that

> *1 kilogram is 2.2 pounds.*
> *1 inch is 0.0254 meters.*

Your program first need to convert pounds to kilograms (i.e. pounds/2.2) and inches to meters (i.e. inches * 0.0254) and then use the following formula to calculate BMI:

> *kilogram / (meter)^2*

2.14.12 Exercise 12

Write a program to calculate the cost of a trip. The program prompts user to enter the distance in miles of the trip, the car efficiency in miles per gallon, and the cost of gas in dollars per gallon. Please limit the print out of the cost to two decimal digits using printf.

2.15. Do You Have Any Questions about Chapter 2?

Comments

3. Conditions

3.1. Learning Outcomes

- Learn about boolean Datatype.
- Understand relational operators and logical operators.
- Understand how to evaluate boolean expressions (conditions) and compound boolean expressions.
- Use conditional statement (if-statement), multibranch (ladder) if statement and nested if statement to solve common problems.
- Write switch statement and use it to write programs.

3.2. Key Terms

Review important terms.

3.3. Resources

- Java Documentation, the java tutorial, The if-then and if-then-else Statements
- Think Java, Chapters 5: How to Think Like a Computer Scientist by Allen Downey and Chris Mayfield Skip Recursion Sections(5.8 – 5.10)
- Safari, Flow Control Structure Video
- Lynda.com, "Workig with boolean Values and Expressions"
- Lynda.com, "Programming Conditional Logic"
- Lynda.com, "Using the switch Statement"
- Lynda.com, "Java 8 Essential Training"

3.4. Overview

In real life, we often encounter situations where we need to make decisions based on certain conditions. For example, based on the student numerical grade we want to determine whether the student is passing or failing the class.

So far, the programs we have been writing do not support different outcomes such as determining whether a student is passing or failing. They do not support computations that depend on the conditions, and they cannot validate user input. Our programs need to produce different results based on conditions encountered in the program.

In this chapter we introduce the boolean data type, a primitive type that can hold true or false. We discuss relational and logical operators and build an understanding of boolean expressions (conditions), and introduce conditional statements in Java (if, switch and the ternary expression).

By the end of this chapter, you should be able to use boolean expression and conditional statement effectively to solve common programming problems. It is important to note that the terms boolean expression and condition are used interchangeably throughout this chapter.

3.5. Relational Operators, Simple Boolean Expressions and the boolean Data type:

We will start this section by introducing the boolean data type. A variable of boolean data type can have only one of two values, either true or false. The example below shows how to declare and initialize a boolean data type:

```
boolean x;
x = true;
```

Or alternatively, the two steps can be combined in one statement as below:

```
boolean x = true;
```

Relational operators are used to check conditions. They can be used to check whether two values are equal, not equal, greater than or less than, the table below list the java different relational operators along with their mathematical equivalent operators, the result of an operational operator is either true or false (boolean):

Table 3. Relational Operators

Java	Mathematics
==	=
!=	≠
>	>
<	>
>=	≥
<=	≤

Note: == is used to check for equality

Boolean expressions use relational operators to check conditions and to compare variables against certain values. For example, in order for a student to pass a class his grade must be greater than or equal to 70; that is, we need to compare the student grade with 70. The boolean expression (condition) that represents this comparison is **grade >= 70**. A boolean expression will evaluate to either true or false. The figure below shows some examples of boolean expressions:

```
int i = 5;
int j = 10;
boolean b;

i == j; //Evaluates to false, because i is not equal to j
i < j;  //Evaluates to true, because i is less than j
i <= j; //Evaluates to true, because i is less than or equal to j

b = i > j; //the value assigned to b is false
b = i != j; //the value assigned to b is true
b = j >= i; //the value assigned to b is true
```

3.6. Logical Operators and Compound Boolean Expressions

Sometimes we need to check for two or more conditions in order to decide or pick an execution path. Logical operators allow us to combine two more conditions. A compound boolean expression consists of two or more boolean expressions joined with logical operators. For example, assume we have a class x that requires two prerequisites class A and class B, in order to decide whether a student can register for class x or not, we need to verify that the student has passed both class A and class B. In this section we will cover the logical and operator (&&), the logical *or* operator (||), the logical *negation* (*not*) operator (!) and *exclusive or* operator "XOR" (^). the four logical operators with their Java symbols are shown in the table below, ***and***, ***or*** and ***exclusive or*** are binary operators, that is they take two operands while ***not*** is a unary operator, that is, it has only one operand:

Table 4. Logical Operators

Logical Operator	Java Symbol
and	&&
or	||
not	!
XOR	^

The table below shows the truth table for &&, ||, ^ and !, where b1 and b2 stands for boolean expression 1 (condition1) and boolean expression 2 (condition2), respectively:

Table 5: Truth Table

b1	b2	b1 && b2	b1 || b2	b1 ^ b2	!b1
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Referring to the table above we notice the following:

- && result is true only if both boolean expression b1 and boolean expression b2 are true, otherwise, && result is false.
- || result is false only if both boolean expression b1 and boolean expression b2 are false, otherwise, || result is true.
- ^ result is false if both conditions are true or both conditions are false, otherwise ^ result is true. That is ^ result is true if both boolean expressions have different values.

Below are examples of compound boolean expressions:

```
int i = 5;
int j = 10;
int k = 15;

//&&
i < j && j < k; //true: both i < j and j < k are true
i > j && j < k; //false: i > j is false
i < j && j > k; //false: j > k is false
i > j && j > k; //false: both i > j and j > k are false

//||
i < j || j < k; //true: both i < j and j < k are true
i > j || j < k; //true: j < k is true
i < j || j > k; //true: i < j is true
i > j || j > k; //false: both i > j and j > k are false

//^
i < j ^ j < k; //false: both i < j and j < k are true
i > j ^ j < k; //true: i >j is false and j < k is true
i < j ^ j > k; //true: i < j is true and j > k is false
i > j ^ j > k; //false: both i > j and j > k are false

//!
!(i < j); //false: i < j is true
!(i > j); //true: i > j is false
```

Java use short circuit evaluation for both && and || operators. In the case of &&, if the first boolean expression (condition) evaluates to false, the result of && is false regardless of whether the second boolean expression evaluates to true or false. In the case of ||, if the first boolean expression (condition) evaluates to true, the result of || is true regardless of whether the second boolean expression evaluates to true or false. To better understand short circuit analysis, assume we have two boolean expressions b1 and b2. In the case of && (b1 && b2), assume b1 is false,

java will not evaluate b2 since, since b1 && b2 is false regardless the value of b2. In the case of || (b1 || b2), assume b1 is true, java will not evaluate b2 since, since b1 || b2 is true regardless the value of b2.

3.7. Operator Precedence

The table below show the operator Precedence from highest to lowest:

Table 6: Operator Precedence

Precedence
var++, var--
Unary +, Unary -, ++var, --var !
*, / , % + (addition), - (subtraction)
<, <=, >, >= ^
&& || =, +=, -=, *=, /=, %=

3.8. if Statement

In the introduction section, we presented the problem of determining whether a student is passing a class or not based on the student numerical grade; assuming a passing grade of 70, a student is passing the class if his/her grade is greater than or equal to 70. The java if statement is needed to solve this problem. In this section, we will introduce if, if-else, nested if, and multibranch if-else statements.

3.8.1. if Statement

The general syntax for an if statement is java is: