Using Data for Computation

2

2.1 Data

2.1.1 Data and Their Taxonomy

The previous chapter presented how to write programs using only `System.out.println` and `System.out.print` statements. This chapter introduces how to store, modify, and retrieve information. The medium in which information is stored is called **data**.[1]

In Java, every data has its **type**, which specifies how the data encodes its information, as well as what operations can be performed on the data. There are two broad classifications of data types in Java: **primitive data types** and **object data types**.

A primitive data type is one that uses only a predetermined number of **bits** in representation, where a bit is the fundamental unit of information and has two possible values (0 and 1). There are eight primitive data types: `boolean` for the logical values, `char` for the characters, `byte`, `short`, `int`, and `long` for whole numbers of various capacities, and `float` and `double` for real numbers of various capacities.

On the other hand, an object data type is a custom-designed data type. A programmer designs a new object data type by putting together existing data types and defining the permissible operations for acting on the information stored. Some object data types require an indefinite number of bits for representation.

In addition to the major difference between primitive and object data types, the following distinctions can be made among data.

1. There are data with names referencing and data without these names.
   (a) The former kind is a **variable**. A special variable is one whose value cannot be changed during the course of program. Such a variable is called a **constant**
   (b) The latter kind consists of **literals** and **return values**. A literal is a data whose value is exactly specified. A return value is a value that a method returns.

[1]The term data is used for both singular and plural forms. The original singular form of "data" is "datum", but this is rarely used nowadays.

M. Ogihara, *Fundamentals of Java Programming*,
https://doi.org/10.1007/978-3-319-89491-1_2

2. Some variables and constants are accessible only inside the method in which they appear; others are accessible from everywhere in the class. The former ones are **local**, and the latter ones are **global**.
   (a) Some global variables in an object class play the role of storing information for individual objects of the class. Those variables are **instance variables** (or **field variables**).
   (b) Each global constant, as well as each instance variable, has a specific visibility type (public, private, protected, and package).

2.1.2 Number Literals

The sequence `"Hello, World!"` is a literal of the data type `String`. Literals are available not only for `String` but also for all the primitive data types.

The following is an example of a code using literals:

```
1  public class Literals
2  {
3    public static void main( String[] args )
4    {
5      System.out.print( "Rank number " );
6      System.out.print( 1 );
7      System.out.print( " in my all-time favorite NFL QB list is " );
8      System.out.print( "Steve Young" );
9      System.out.println( "." );
10     System.out.print( "His overall quarterback rating is " );
11     System.out.print( 96.8 );
12     System.out.println( "." );
13   }
14 }
```

Listing 2.1 A program that produces a comment about an NFL quarterback

In Lines 5, 7–9, and 12, `String` literals appear inside the parentheses. In Lines 6 and 11, literals `1` and `96.8` appear inside the parentheses. These are number literals (an `int` literal and a `double` literal, to be precise).

When this program runs, `System.out.print` converts these two numbers to character sequences, then prints those sequences on the screen.

```
1  Rank number 1 in my all-time favorite NFL quarterbacks is Steve Young.
2  His overall quarterback rating is 96.8.
```

By adding the same sequence of statements for two more quarterbacks, the following program is obtained.

```
1  public class Literals01
2  {
3    public static void main( String[] args )
4    {
5      System.out.print( "Rank number " );
6      System.out.print( 1 );
7      System.out.print( " in my all-time favorite NFL QB list is " );
8      System.out.print( "Steve Young" );
9      System.out.println( "." );
10     System.out.print( "His overall QB rating is " );
11     System.out.print( 96.8 );
12     System.out.println( "." );
13
```

Listing 2.2 A program that produces comments about three NFL quarterbacks (part 1). The program header and the part that produces comments about the first player

For these additional two players, number literals appear in Lines 15, 20, 24 and 29.

```
14     System.out.print( "Rank number " );
15     System.out.print( 2 );
16     System.out.print( " in my all-time favorite NFL QB list is " );
17     System.out.print( "Peyton Manning" );
18     System.out.println( "." );
19     System.out.print( "His overall QB rating is " );
20     System.out.print( 96.5 );
21     System.out.println( "." );
22
23     System.out.print( "Rank number " );
24     System.out.print( 3 );
25     System.out.print( " in my all-time favorite NFL QB list is " );
26     System.out.print( "Tom Brady" );
27     System.out.println( "." );
28     System.out.print( "His overall QB rating is " );
29     System.out.print( 97.0 );
30     System.out.println( "." );
31   }
32 }
```

Listing 2.3 A program that produces comments about three NFL quarterbacks (part 2). The part that produces comments about the second and the third players

The program produces the following output[2]:

```
1 Rank number 1 in my all-time favorite NFL quarterbacks is Steve Young.
2 His overall quarterback rating is 96.8.
3 Rank number 2 in my all-time favorite NFL quarterbacks is Peyton Manning.
4 His overall quarterback rating is 96.5.
5 Rank number 3 in my all-time favorite NFL quarterbacks is Tom Brady.
6 His overall quarterback rating is 97.0.
```

[2]Jon Steven "Steve" Young (born October 11, 1961) is a former NFL quarterback and a sportscaster. He played 13 seasons with the San Francisco 49ers, and led the team to three Super Bowl wins. Peyton Williams Manning (born March 24, 1976) is a former NFL quarterback who played with the Indianapolis Colts and later with the Denver Broncos. He led each team to one Super Bowl win. Thomas Edward Patrick Brady Jr. (born August 3, 1977) is an NFL quarterback for the New England Patriots. He led the team to five Super Bowl wins.

2.1.3 Variable Declarations

A variable is a type of data with a reference name. Simply by putting its name in the code, the value of the data in that specific moment can be looked up.

Since each data has a unique type, a variable is specified with its name and its type. This is called a **declaration**. A declaration takes the following form:

```
DATA_TYPE VARIABLE_NAME;
```

Remember that the tab-stop can be used in place of the white space, and adding more tab-stops or white space after that is possible as well.

One example is the following:

```
1 public static ...
2 {
3   ...
4   int myInteger;
5   ...
6 }
```

Here, `int` is the data type and `myInteger` is the name of the data. Notice the semicolon `;` appearing at the end. Each line of local/global variable/constant declarations requires a semicolon at the end. The above is an instance in which the variable is a local variable. A local variable belongs to a method and its declaration appears inside the method that it belongs to. The "locality" of variables becomes important when multiple methods are written in a source code.

To declare a global variable, its declaration is placed outside the methods, at depth 1. For example, the following declares a global variable named `myInteger` of data type `int`.

```
1   static int myInteger;
2
3   public static ...
4   {
5     ...
6   }
```

The attachment of `static` is required for the declaration of a global variable.

It is possible to join declarations of multiple variables of the same type in one line via separating the names with commas, as shown next:

```
int oneInteger, anotherInteger, yetAnotherInteger;
```

Here, `oneInteger`, `anotherInteger`, and `yetAnotherInteger` are all of the `int` data type. This single-line declaration is equivalent to:

```
1 int oneInteger;
2 int anotherInteger;
3 int yetAnotherInteger;
```

To name a variable, a sequence of contiguous characters chosen from the alphabet, numerals, the **underscore** `_`, and the **dollar sign** `$` must be used. In addition, the first character of the sequence must be a letter of the alphabet.[3] Thus, according to this rule, 55 cannot be a variable name. The

[3]Technically, a variable name can start with the underscore or the dollar sign, but the convention is to start a variable name with a lowercase letter.

variable names, method names, and class names are all case-sensitive. The following conventions are generally used:

- The name of a variable must start with a lowercase letter.
- The name of a constant must contain no lowercase letter.
- The name of a class must start with an uppercase letter.

2.1.4 Assigning Values to Variables

As mentioned earlier, variables (or constants) are places where information is stored. The action of giving a value to a variable/constant is called an **assignment**.

The syntax for an assignment is:

```
VARIABLE = VALUE;
```

Here, the equal sign = appearing in the middle symbolizes the assignment. The left-hand side of the assignment, `VARIABLE`, is the name of the variable in which a value is to be stored. The right-hand side of an assignment, `VALUE`, is the value being stored. JVM can interpret this as a data having the same type as the variable. The value in an assignment can be:

- a literal of the same type;
- a variable of the same type;
- a call to a method that returns with data of the same type; or
- a formula of the same type.

Next is an example of assigning a value to a variable:

```
1 int myInteger;
2 myInteger = 55;
```

The first line here is a variable declaration. The type of the data is `int` and the name of the data is `myInteger`. The second line is an assignment. The right-hand side of the assignment is an `int` literal with the value of 55. This action stores the value of 55 into the variable `myInteger`.

It is possible to combine a declaration of a variable and an initial value assignment to the variable all in a single statement, like this one:

```
int myInteger = 55;
```

It is also possible to combine multiple declarations and assignments of multiple variables in one line, so long as they all have the same type. The following is an example of such combinations:

```
int oneInteger = 17, anotherInteger, yetAnotherInteger = 23;
```

This single line of code declares `int` variables, `oneInteger`, `anotherInteger`, and `yetAnotherInteger`, and assigns the value of 17 to `oneInteger` and the value of 23 to `yetAnotherInteger` (note that 17 and 23 are literals).

Here, `anotherInteger` does not have an assignment. Does it have a value? Yes, the default value of a variable of a primitive number type immediately after declaration is 0.

This one-line code is equivalent to:

```
int oneInteger, anotherInteger, yetAnotherInteger;
oneInteger = 17;
yetAnotherInteger = 23;
```

as well as to:

```
int oneInteger, anotherInteger, yetAnotherInteger;
oneInteger = 17, yetAnotherInteger = 23;
```

For the code to compile, the value assigned to a variable must match the data type of the variable. In the above code fragment, an attempt to assign numbers with a decimal point,

```
int oneInteger, anotherInteger, yetAnotherInteger;
oneInteger = 17.5;
yetAnotherInteger = 23.0;
```

results in a syntax error, because neither `17.5` nor `23.0` are integers.

The Attribute `final`

By attaching `final` in front of the data type specification in a variable declaration, any future value assignments to the variable are prohibited. Thus, by the attachment of `final`, a variable becomes a constant.

```
  static final int YOUR_INTEGER = 20;

  public static void main( String[] args )
  {
    final int MY_NUMBER = 17;
    ...
  }
```

The first variable, `YOUR_INTEGER`, is a global constant and the second variable, `MY_NUMBER`, is a local constant. Java requires that a value must be assigned to a constant using a combined declaration and assignment. Therefore, in the above code example, splitting the declaration and assignment of either constant into a standalone declaration and a standalone assignment is rejected as a syntax error.

A global constant may have an explicit visibility specification. As mentioned in Chap. 1, there are three explicit visibility specifications: `public`, `private`, and `protected`. In the source code:

```
public class PubConstEx
{
  public static final int COMMONINTEGER = 20;

  public static void main( String[] args )
  {
    final int MY_NUMBER = 17;
    ...
  }
}
```

COMMONINTEGER is a global constant accessible from outside and MY_NUMBER is declared to be a local constant. By combining the class name and the global variable name, as in PubConstEx.COMMONINTEGER, the value 20 can be accessed from other source codes.

Reassignment

If a variable is not a constant, the value of the variable can be updated any number of times. Consider the next code:

```
1 int myInteger;
2 myInteger = 63;
3 ...
4 myInteger = 97;
5 ...
6 myInteger = 20;
```

The dotted parts represent some arbitrary code. Assuming that no assignments to myInteger appear in the dotted part, the value of myInteger changes from 63 to 97 in Line 4 and from 97 to 20 in Line 6.

2.2 The Primitive Data Types

Java has eight primitive data types. They are **boolean**, **byte**, **short**, **int**, **long**, **float**, **double**, and **char**. Table 2.1 shows the range of possible values for each type.

boolean

The boolean is a data type for storing a logical value. There are only two possible values for a boolean data type: true and false. Here is an example of declaring boolean variables and assigning values to them.

```
1 boolean logicA, logicB, logicC;
2 logicA = true;
3 logicB = false;
```

Table 2.1 The primitive data type

Type	No. bits	Data representation
boolean	1	Logical value, true or false
byte	8	Integer, [−128, 127]
short	16	Integer, [−32,768, 32,767]
int	32	Integer, [−2,147,483,648, 2,147,483,647]
long	64	Integer, [−9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
float	32	Floating point number, approximately from $1.4013 \cdot 10^{-45}$ to $3.4028 \cdot 10^{+38}$ (positive or negative)
double	64	Floating point number, approximately from $4.9407 \cdot 10^{-324}$ to $1.7977 \cdot 10^{+308}$ (positive or negative)
char	16 (unsigned)	UNICODE character, [0, 65,535]

There are three operators available for `boolean`: the **negation** `!`, the **conjunction** `&&`, and the **disjunction** `||`. The `boolean` data type will be discussed in Chap. 6.

`byte`
The data type `byte` consists of eight bits, so there are $2^8 = 256$ possible values. The value range of `byte` is between $-2^7 = -128$ and $2^7 - 1 = 127$. The **hexadecimal encoding** is an encoding that is often used for specifying a `byte` literal. This is the encoding that divides the bits into groups of four and uses a unique character to represent each of the 16 possible values for each group. The value range of four bits is between 0 and 15, so naturally the numerals `0` through `9` are used for representing numbers 0 through 9. For the numbers 10 through 15, the letters `a` through `f` (or their upper case letters `A` through `F`) are used. In hexadecimal encoding, a byte must be represented with two characters. The combination `YZ` represents

The number `Y` represents multiplied by 16, plus the number `Z` represents.

Thus, "5f" in hexadecimal is not 65 ($= 5 * 10 + 15$) but 95 ($= 16 * 5 + 5$) and "dc" in hexadecimal is 220 in decimal ($= 13 * 16 + 12$). Hexadecimal encoding makes it possible to represent bit sequences compactly: four characters for sixteen bits, eight characters for 32 bits, and sixteen characters for 64 bits. In hexadecimal encoding, to indicate that a literal is presented, the prefix `0x` must be attached. For example, `0x33` is 33 in hexadecimal, which is equal to 51 in decimal.

```
byte myByteValue = 0x3f;
```

`short`
The data type `short` consists of sixteen bits. It has 65,536 possible values. The value range is between $-2^{15} = -32{,}768$ and $2^{15} - 1 = 32{,}767$.

`int`
`int` is a data type that consists of 32 bits. It has 4,294,967,296 possible values. The value range is between $-2^{31} = -2{,}147{,}483{,}648$ and $2^{31} - 1 = 2{,}147{,}483{,}647$. The default data type of a whole number literal is `int`.

`long`
The data type `long` consists of 64 bits. It has 18,446,744,073,709,551,616 possible values. The value range is between $-2^{63} = -9{,}223{,}372{,}036{,}854{,}775{,}808$ and $2^{63} - 1 = 9{,}223{,}372{,}036{,}854{,}775{,}807$. When presenting a literal in `long`, if the value that the literal represents goes beyond the range of `int`, the letter `L` must be attached as a suffix at the end of the number sequence, like this one:

```
long myLongNumber = 1234567890987654321L;
```

`float` **and** `double`
`float` and `double` are data types for real numbers. They use floating point representations. Basically, a floating point representation divides the available bits into three parts: the **sign**, the **significand**, and the **exponent**. Let S be the sign ($+1$ or -1). The number of bits for the sign is one, and the bit is at the highest position of the bits allocated for the data type. The bits of the significand represent a number between 0 and 1. If that part has m bits and the bit sequence is $b_1 \cdots b_m$, that part represents the sum of all 2^{-i} such that $b_i = 1$. Let C be this fractional number. Each floating point

encoding system uses a fixed "base" for exponentiation, which is either 2 or 10. Let B be this base. If there are n bits in the exponent, that part encodes a number between -2^{n-1} and $2^{n-1}-1$ (for whole number types). Let E be this number. Now, the bits of the floating number altogether represent

$$S \cdot (1 + C) \cdot B^E.$$

To designate that a floating point number literal is a `float`, the letter `F` must be attached as a suffix.

Here is an example of declaring and assigning a literal value to one variable for each primitive number type:

```
1  byte myByte;
2  short myShort;
3  int myInt;
4  long myLong;
5  float myFloat;
6  double myDouble;
7  myByte = 0x3d;
8  myShort = 1345;
9  myInt = 90909;
10 myLong = 1234567890123456789L;
11 myFloat = -3.145F;
12 myDouble = 1.7320504;
```

The numbers appearing after `'='` in lines 7–12 are all literals.

`char`

The last primitive data type is `char`. The representation of `char` requires sixteen bits. By attaching the apostrophe before and after of a character, a `char` literal is specified, as in `'A'` and `'x'`.

```
1  char myChar1 = 'A';
2  char myChar2 = 'x';
```

The default value of a `char` type variable is `\0`. This is the character corresponding to the number 0. This must not be confused this with the numeral `0`. There is no direct arithmetic operation that can be applied to `char` data for producing another `char`, but if a `char` data appears in a mathematical formula, it acts as if it were an `int`. Since the single quotation mark is used for a `char` literal, to specify the single quotation mark itself as a `char` literal, attaching a backslash as is must be done to represent the double quotation mark in `String` literals; that is, `'\"` represents the single quotation mark. Because of this special use of backslash, two backslashes are used to mean the backslash itself as a literal, as in `'\\'`. The other escape sequences, `'\t'` and `'\n'`, are valid for `char` literals too.

2.3 Using Variables for Computation

2.3.1 Quarterbacks Program (Reprise)

Using the fact that reassignments of values can be made to non-final variables, the previous three-favorite-quarterback program can be rewritten using three variables:

- an `int` variable, `rank`, for specifying the rank in the list
- a `String` variable, `name`, for the name of the quarterback, and
- a `double` variable, `qbr`, for the quarterback rating.

```
1  public class Literals02
2  {
3    public static void main( String[] args )
4    {
5      int rank;
6      String name;
7      double qbr;
8
9      rank = 1;
10     name = "Steve Young";
11     qbr = 96.8;
12     System.out.print( "Rank number " );
13     System.out.print( rank );
14     System.out.print( " in my all-time favorite NFL QB list is " );
15     System.out.print( name );
16     System.out.println( "." );
17     System.out.print( "His overall QB rating is " );
18     System.out.print( qbr );
19     System.out.println( "." );
20
```

Listing 2.4 A program that produces comments about three NFL quarterbacks using variables (part 1). The variable declarations and the part that produced the comments about the first player

Note that the variable declarations appear in Lines 4, 5, and 7. The variable declarations are followed by three blocks of the same format, each consisting of eleven lines. The first lines of each block assign values to the variables. For example, the first lines of the first block are:

```
1      rank = 1;
2      name = "Steve Young";
3      qbr = 96.8;
```

The next eight lines of code make the presentation, with the three literals for rank, name, and rating substituted with their respective names.

```
1      ...
2      System.out.print( rank );
3      ...
4      System.out.print( name );
5      ...
6      ...
7      System.out.print( qbr );
8      ...
```

In these three lines, the values of these variables substitute their respective locations into the print statements. Note that the action order is:

$$\text{declaration} \rightarrow \text{assignment} \rightarrow \text{reference}$$

Since assignments can be made multiple times to non-final variables, a variable declaration is effective until the end of the innermost pair of curly brackets that contains it. This means that two declarations of the same variables cannot intersect. In the above program, the innermost matching pair containing a variable declaration appears at Lines 4 and 44. Thus, the declarations are valid until Line 44. Formally, the range of lines in which a variable declaration is valid is called the **scope of the variable**.

```
21      rank = 2;
22      name = "Peyton Manning";
23      qbr = 96.5;
24      System.out.print( "Rank number " );
25      System.out.print( rank );
26      System.out.print( " in my all-time favorite NFL QB list is " );
27      System.out.print( name );
28      System.out.println( "." );
29      System.out.print( "His overall QB rating is " );
30      System.out.print( qbr );
31      System.out.println( "." );
32
33      rank = 3;
34      name = "Tom Brady";
35      qbr = 97.0;
36      System.out.print( "Rank number " );
37      System.out.print( rank );
38      System.out.print( " in my all-time favorite NFL QB list is " );
39      System.out.print( name );
40      System.out.println( "." );
41      System.out.print( "His overall QB rating is " );
42      System.out.print( qbr );
43      System.out.println( "." );
44    }
45  }
```

Listing 2.5 A program that produces comments about three NFL quarterbacks using variables (part 1). The part that produces the comments about the second and the third players

Reserved Names

The names appearing in Table 2.2 cannot be used as the name of a variable, a method, or a class. These are called the **reserved names**.

2.3.2 Number Arithmetics

2.3.2.1 Number Operations

In Java, the four standard arithmetic operations in mathematics (addition, subtraction, multiplication, and division) are represented with the standard mathematical symbols (`+`, `-`, `*`, and `/`, respectively). The negative sign `-` can be used for flipping the sign. The regular parentheses `( )` can be used for flipping the sign. There is no symbol for representing exponentiation.

The symbols that represent binary operations are called **operators**. The values that are evaluated with an operator are called **operands**. Since `-` acts on a single value, it is a **unary operator**. Since `+`, `-`, `*`, `/`, and `%` take two values, they are **binary operators**.

There are other additional operator types, such as unary bit shift (`<<`, `>>`, and `>>>`), unary bit complement (`~`), and unary bit-wise (`^`, `|`, and `&`). This textbook does not use these bit operators (see Table 1.2)

Table 2.2 The list of reserved words in Java

abstract	boolean	break	case	catch	char	class
const	continue	default	do	double	else	enum
extends	final	finally	float	for	goto	if
import	implements	import	instanceof	int	interface	long
native	new	package	private	protected	public	short
static	strictfp	super	return	switch	synchronized	this
throw	throws	transient	try	void	volatile	while

In addition to using actual values (as represented by literals), variables can be used in mathematical formulas. When evaluating a formula that contains a variable, the value of the variable at the moment of evaluation is used. Consider the following code fragment:

```
1 double x, y, z;
2 x = 3.5;
3 y = 4.5;
4 z = -x + y + 1.0;
5 System.out.println( z );
```

In the fourth line of the program,

```
4 z = -x + y + 1.0;
```

the values that `x` and `y` hold (that is, `3.5` and `4.5`, respectively) substitute their respective locations into the right-hand side of the formula. The result of the evaluation is `2.0`. Since there is `=`, this value is assigned to the variable `z`. So, when the program executes `System.out.println( z )`, this new value of `2.0` emerges in output:

```
2.0
```

It is possible to write more complicated formulas. For example, in

```
1 double x, w;
2 x = 2.0;
3 w = ( x + 1.0 ) * ( -1.0 + x );
4 System.out.println( w );
```

the value of `x` becomes `2.0` in Line 2. The value of the formula `( x + 1.0 ) * ( -1.0 + x )` then becomes `3.0`. This value is assigned to `w`. So, the output of the program is:

```
3.0
```

Alternatively, the code could be written as either

```
1 double x, w;
2 x = 2.0;
3 w = x * x - 1.0 * x + 1.0 * x - 1.0;
4 System.out.println( w );
```

or

```
1 double x, w;
2 x = 2.0;
3 w = ( x * x ) - 1.0;
4 System.out.println( w );
```

Both produce the same output as the original.

The Remainder Operator

The remainder operator `%` works as follows. Let `a` and `b` be two numbers. If the value of `b` is 0, `a % b` is undefined, an attempt to execute the operation produces a run-time error. Otherwise, if `a` and `b` have the same signs, the value of `a % b` is `a - d * b`, where `d` is the integer part of `a` divided by `b`, and if `a` and `b` have opposite signs, the value of `a % b` is `a + d * b`, where `d` is the integer part

of |a| divided by |b|. For example, `10 % 3` is equal to `1` and `-17.0 % 5.0` is `-2.0` since the integer part of `-17.0 / 5.0` is equal to `-3`.

2.3.2.2 Evaluation of Formulas

To evaluate formulas with more than two operations, Java prioritizes these operators in the same way we would in arithmetics.

- `*`, `/`, and `%` have the same level of priority.
- `+` and `-` have the same level of priority.
- The `-` for switching the sign has the highest priority. Next in priority is the { `*`, `/`, `%` } group. Last is the { `+`, `-` } group.
- The evaluation of a formula is from left to right using the following principles:
  - If there are parentheses in the formula, evaluate the leftmost and innermost parenthetical clause to reduce it to a single value.
  - If the formula does not have parentheses and has one of `*`, `/`, and `%`, process the leftmost one of the three kinds.
  - If the formula does not have parentheses and has no `*`, `/`, and `%`, process the leftmost operation.

In the code

```
1 double myDouble = 10.5;
2 int myInt = 11;
3 myDouble = -3 % 2 + (3 * 8 + myDouble * myInt) % 6;
```

the evaluation proceeds as follows:

```
1 -3 % 2 + (3 * 8 + 10.5 * 11) % 6
2 -1 + (3 * 8 + 10.5 * 11) % 6
3 -1 + (24 + 10.5 * 11) % 6
4 -1 + (24 + 115.5) % 6
5 -1 + 139.5 % 6
6 -1 + 1.5
7 0.5
```

`0.5` becomes the value of `myDouble`.

Here is how to use data (and some arithmetics on the data) to perform computation. Consider a program that evaluates several formulas involving a set of unknowns (which may appear in more than one formula). The user enters the values for the unknowns. It is possible to ask the user to enter the value of a variable whenever the calculation needs to use the value. However, since some variables are used more than once and there is no guarantee that the user enters a consistent value to a variable, the program instead stores the values of the unknowns into variables.

This first example is for computing various geometric values with respect to a radius R. A `double` variable, `radius`, is used to represent the value of the radius. Suppose the following four quantities are to be computed from R:

1. the perimeter of a circle having radius R,
2. the area of a circle having radius R,
3. the surface area of a sphere having radius R, and
4. the volume of a sphere having radius R.

The following mathematical formulas can be used in calculating the four quantities:

$$2\pi R,\ \pi R^2,\ 4\pi R^2, \text{ and } \tfrac{4}{3}\pi R^3.$$

The program uses four variables, `circlePerimeter`, `circleArea`, `ballArea`, and `ballVolume`, to record the quantities. Lines 5 and 6 of the code declare these variables.

In Line 8, the value of `radius` is set to 10.0. The program then successively computes the four quantities in Lines 10 through 13, and uses the literal `3.14159265` for π. Lines 15 through 18 print the four quantities.

```
1  public class RadiusPrimitive
2  {
3    public static void main( String[] args )
4    {
5      double radius;
6      double circlePerimeter, circleArea, ballArea, ballVolume;
7
8      radius = 10.0;
9
10     circlePerimeter = 2.0 * 3.14159265 * radius;
11     circleArea = 3.14159265 * radius * radius;
12     ballArea = 4.0 * 3.14159265 * radius * radius;
13     ballVolume = 4.0 * 3.14159265 * radius * radius * radius / 3.0;
14
15     System.out.println( circlePerimeter );
16     System.out.println( circleArea );
17     System.out.println( ballArea );
18     System.out.println( ballVolume );
19   }
20 }
```

Listing 2.6 A preliminary version of the program for computing values for a given radius

This code produces the following output:

```
1 62.831853
2 314.159265
3 1256.63706
4 4188.7902
```

Two changes will be made to the program to obtain the next code. First, noticing that the value 3.14159265 as π appears in multiple formulas, a variable can be used to store a value for π. Second, the four quantities that are calculated will be printed with their names.

Line 8 declares a new variable, `pi`, in which the value of π is stored (Line 10). In the ensuing calculation, the variable `pi` is used in places where the value of π is needed. Also, the attribute of `final` is attached to the variable so as to make it a local constant and prevent value changes.

```
1  // compute values given a radius
2  public class Radius
3  {
4    public static void main( String[] args )
5    {
6      double radius;
7      double circlePerimeter, circleArea, ballArea, ballVolume;
8      double pi;
9      //--- set the values of pi and radius
10     pi = 3.14159265;
11     radius = 10.0;
```

Listing 2.7 The code for computing values for a given radius (part 1). Quantity calculation

The second part of the code is for reporting the results of the calculation.

```
12      // calculcate the values
13      circlePerimeter = 2.0 * pi * radius;
14      circleArea = pi * radius * radius;
15      ballArea = 4.0 * pi * radius * radius;
16      ballVolume = 4.0 * pi * radius * radius * radius / 3.0;
17      //-- output the values
18      System.out.print( "circlePerimeter = " );
19      System.out.println( circlePerimeter );
20      System.out.print( "circleArea = " );
21      System.out.println( circleArea );
22      System.out.print( "ballArea = " );
23      System.out.println( ballArea );
24      System.out.print( "ballVolume = " );
25      System.out.println( ballVolume );
26    }
27  }
```

Listing 2.8 The code for printing the values of the four quantities. The part for calculating the quantities and printing the results

To make clear which value represents which quantity, the program uses a `print` statement. The program prints the name of the quantity preceding the presentation of the value. The statement

```
System.out.println( circlePerimeter );
```

prints the value of the variable `circlePerimeter` and proceeds to the next line.

This code produces the following output:

```
1  circlePerimeter = 62.831853
2  circleArea = 314.159265
3  ballArea = 1256.63706
4  ballVolume = 4188.7902
```

With this arrangement, the correspondence between the value and the meaning will be clear to the user when reading the output.

Here is another, more obscure, way of calculating the four quantities in a row. The program uses the facts that, for a fixed radius value,

(a) the area of the circle is the perimeter times the radius divided by 2,
(b) the surface area of the ball is four times the area of the circle, and
(c) the volume of the ball is the area of the ball times the radius divided by 3.

Based upon these facts, the program obtains the value for the variable `circleArea` with a formula that contains `circlePerimeter`, obtains the value for the variable `ballArea` with a formula that contains `circleArea`, and obtains the value for the variable `ballVolume` with a formula that contains `ballArea`. Note that the variable `pi` is now a constant named `PI` with the attribute of `final` (with its name in all uppercase according to the naming convention).

```
// compute values given a radius
public class RadiusAlternative
{
  public static void main( String[] args )
  {
    double radius;
    double circlePerimeter, circleArea, ballArea, ballVolume;
    final double PI = 3.14159265;
    //--- set the values of PI and radius
    radius = 10.0;
    // calculcate the values
    circlePerimeter = 2.0 * PI * radius;
    circleArea = radius * circlePerimeter / 2.0;
    ballArea = 4.0 * circleArea;
    ballVolume = ballArea * radius / 3.0;
    //-- output the values
    System.out.print( "circlePerimeter = " );
    System.out.println( circlePerimeter );
    System.out.print( "circleArea = " );
    System.out.println( circleArea );
    System.out.print( "ballArea = " );
    System.out.println( ballArea );
    System.out.print( "ballVolume = " );
    System.out.println( ballVolume );
  }
}
```

Listing 2.9 An alternative for the calculation of values associated with a circle and a ball

Yet another modification will be made by moving the constant `PI` outside the method, thereby changing it from a local constant to a global constant.

Note that the `static` attribute must be attached to the declaration.

```
// compute values given a radius
public class RadiusAlternative2
{
  static final double PI = 3.14159265;
  public static void main( String[] args )
  {
    double radius;
    double circlePerimeter, circleArea, ballArea, ballVolume;
    //--- set the values of PI and radius
    radius = 10.0;
    // calculcate the values
    circlePerimeter = 2.0 * PI * radius;
    circleArea = radius * circlePerimeter / 2.0;
    ballArea = 4.0 * circleArea;
    ballVolume = ballArea * radius / 3.0;
    //-- output the values
    System.out.print( "circlePerimeter = " );
    System.out.println( circlePerimeter );
    System.out.print( "circleArea = " );
    System.out.println( circleArea );
    System.out.print( "ballArea = " );
    System.out.println( ballArea );
    System.out.print( "ballVolume = " );
    System.out.println( ballVolume );
  }
}
```

Listing 2.10 The radius code with the value of π as the global constant

2.3.2.3 Mixing Different Number Types in a Formula

There are two rules that Java applies when two different number types appear as operands.

- Either one of them is a floating point number type (i.e., `float` or `double`) if and only if the result is a floating point number type.
- The number of bits of the resulting data type is the maximum of the numbers of bits for the two number types.

For example, if the operands are `byte` and `float`, the result will be a `float`. Table 2.3 shows the primitive data type that results by mixing data types.

The next code attempts to compute the product and division between two numbers, 11 and 3. Each number is either an `int` or a `double`. In each of the statements in Lines 14, 15, 18, 19, 22, 23, 26, and 27, there appears an unfamiliar use of `System.out.println`. For example, Line 14 goes:

```
14      System.out.println( aInt / bInt );
```

Appearing inside the parentheses is a formula. Conveniently, if a formula appears inside the parentheses, both `System.out.print` and `System.out.println` evaluate the formula, convert the result to a character sequence, and print the sequence on the screen.

The program first declares the variables it is going to use:

```
1  public class NumberTest
2  {
3      //--- try four possible cases of "11 divided by 3"
4    public static void main( String[] args )
5    {
6      int aInt, bInt;
7      double aDouble, bDouble;
8      aInt = 11;
9      bInt = 3;
10     aDouble = 11.0;
11     bDouble = 3.0;
```

Listing 2.11 A program that demonstrates the use of operators on `double` and/or `int` variables (part 1)

The program then executes a number of operations on the variables.

Table 2.3 The number types chosen when two primitive data types are processed in binary arithmetic operations

Type	byte	short	int	long	float	double
byte	byte	short	int	long	float	double
short	short	short	int	long	float	double
int	int	int	int	long	float	double
long	long	long	long	long	double	double
float	float	float	float	float	double	double
double	double	double	double	double	double	double

```
12      // case 1
13      System.out.println( "aInt vs bInt. / and %" );
14      System.out.println( aInt / bInt );
15      System.out.println( aInt % bInt );
16      // case 2
17      System.out.println( "aInt vs bDouble. / and %" );
18      System.out.println( aInt / bDouble );
19      System.out.println( aInt % bDouble );
20      // case 3
21      System.out.println( "aDouble vs bInt. / and %" );
22      System.out.println( aDouble / bInt );
23      System.out.println( aDouble % bInt );
24      // case 4
25      System.out.println( "aDouble vs bDouble. / and %" );
26      System.out.println( aDouble / bDouble );
27      System.out.println( aDouble % bDouble );
28    }
29  }
```

Listing 2.12 A program that demonstrates the use of operators on `double` and/or `int` variables (part 1)

The result of executing the code is as follows:

```
1  aInt vs bInt. / and %
2  3
3  2
4  aInt vs bDouble. / and %
5  3.6666666666666665
6  2.0
7  aDouble vs bInt. / and %
8  3.6666666666666665
9  2.0
10 aDouble vs bDouble. / and %
11 3.6666666666666665
12 2.0
```

When a literal of a primitive number type appears, its default type is `int` for a whole number and `double` for a floating point number. To treat a data as an alternate type, attach, in front of it, the alternate type enclosed in a matching pair of parentheses. For example, `(byte)12` instructs to treat the `int` type value of `12` as a `byte` type. The action of attaching a data type to treat a data as a different type is called **casting**.

Using casting, a floating point number can be truncated to an integer; that is, for a floating point number `x`, `(int)x` is a 32-bit whole number that is equal to the integer part of `x`.

2.3.3 Computing the Body-Mass Index

The next example is a program for computing the **Body-Mass Index** for multiple combinations of weight and height. The Body-Mass Index measures the balance between the height and weight of a human body. The lower the index is, the lighter the person is. The following formula defines the Body-Mass Index:

$$\text{BMI} = 703 * \text{ weight (in pounds)} / (\text{height } \times \text{ height}) \text{ (in inches)}$$

In the program, the computation is carried out in the following steps:

- declare variables for storing weight, height, and the BMI value;
- assign a value to weight and a value to height;
- compute the BMI value;
- print the result;
- reassign a value to weight and a value to height;
- compute the BMI value with respect to the reassigned weight and height;
- print the result.

Next is the program `BMI.java` that does this.

```
1  public class BMI
2  {
3    public static void main( String[] args )
4    {
5      // first time
6      double weight = 140.0;    // weight
7      double height = 67.0;     // height
8      double bmi = 703.0 * weight / (height * height);
9      System.out.print( "BMI = " );
10     System.out.println( bmi );
11     // second time
12     weight = 150.0;       // weight
13     height = 70.0;        // height
14     bmi = 703.0 * weight / ( height * height );
15     System.out.print( "BMI = " );
16     System.out.println( bmi );
17   }
18 }
```

Listing 2.13 The code for computing the BMI for predetermined combinations of height and weight

Here is what happens in the code:

- What appears after the first comment is the declaration of a `double` variable `weight` (Line 6). Here, the program assigns the value of 140.0 immediately to the variable. The next line (Line 7) does the same for `height` with the value of 67.0. Both these lines use the idea of combining a variable declaration and a value assignment in one line.
- The next line (Line 8) declares a `double` variable `bmi` and assigns to it a value using the formula `703.0 * weight / (height * height)`. This line also uses the idea of combining a variable declaration and a value assignment in one line. Furthermore, the line takes advantage of the fact that by the time the code execution reaches this third declaration and assignment, both `weight` and `height` have acquired new values.
- The parentheses surrounding the second multiplication line designate that the multiplication must take place before the division (again, Java follows our common sense evaluation of mathematical formulas). If the parentheses are removed, the last multiplication symbol must be replaced with the division symbol; that is,

```
double bmi = 703.0 * weight / height / height;
```

Otherwise, the code

```
double bmi = 703.0 * weight / height * height;
```

will divide the product of `703.0` and the value of `weight` by the value of `height` and then multiply it by the value of `height`.

- The ensuing two lines are for producing the result on the screen. The first of the two is for printing the `String` literal `"BMI = "`, and the second is for printing the value and proceeding to the next line.
- Then the program assigns new values to `weight` and `height` and then recomputes the BMI value. Since these lines are in the scope of the three variables, the type declaration `double` must not appear again.

The execution of the code results in the following:

```
1 BMI = 21.92470483403876
2 BMI = 21.520408163265305
```

Note the difference, between the two lines, in the number of digits after the decimal point. The first one has fourteen digits and the second one has fifteen. In both cases the actual BMI value has infinitely many digits and thus the floating point expression cannot correctly represent the value. The value that appears on the screen is only an approximation. Chapter 8 describes how to control the number of digits appearing after the decimal point.

2.3.4 Sum of Integers from 1 to 100 à la Gauss

Johann Carl Friedrich Gauss (April 30, 1777 to February 23, 1855) is a German mathematician who made important contributions to many fields of mathematics. There is a famous story stating that he was a genius even as a school kid. The story goes like this.

> One day a teacher asked the class to calculate, on a sheet of paper, the sum of integers from 1 to 100. While all the other classmates were adding the numbers one after another, Gauss raised his hand to tell his teacher he had completed the task. Miraculously, the answer he gave was correct. Stunned, the teacher asked Gauss to explain how he had gotten to his answer. Gauss explained: If you add the first number, 1, and the last number, 100, the result is 101. If you add the second number, 2, and the second to last number, 99, the result is 101. If you keep going this way, the smaller number increases by 1 and the larger number decreases by 1, and so the sum is always 101. Since there are 100 numbers to add, there are 50 such pairs. Thus, the total is 101 * 50 = 5050.

Based upon his observation, the sum of all integers between 1 and $n \geq 1$ can be quickly computed. If n is an even number, the total is given as the following formula

$$(n + 1) * n/2$$

If n is an odd number, the middle number $(n + 1)/2$ does not pair with other numbers, and since there are $(n - 1)/2$ pairs, the total is

$$(n + 1) * (n - 1)/2 + (n + 1)/2 = (n + 1) * ((n - 1)/2 + 1/2) = (n + 1) * n/2$$

Thus, regardless of whether n is even or odd, the total is $(n + 1) * n/2$.

The following program demonstrates the use of integer variables, which computes the sum of integer sequences à la Gauss.

```
1  public class Gauss
2  {
3    public static void main( String[] args )
4    {
5      int n, sum;
6      n = 100;
7      sum = ( n + 1 ) * n / 2;
8      System.out.print( "The sum of integers from 1 to " );
9      System.out.print( n );
10     System.out.print( " = " );
11     System.out.println( sum );
12     n = 1000;
13     sum = ( n + 1 ) * n / 2;
14     System.out.print( "The sum of integers from 1 to " );
15     System.out.print( n );
16     System.out.print( " = " );
17     System.out.println( sum );
18   }
19 }
```

Listing 2.14 The code for computing $1 + \cdots + n$ for $n = 100$ and $n = 1000$

The code uses two `int` variables, `n` and `sum`. The program assigns the value of 100 and then the value of 1000 to the variable `n`. The five lines that appear after the first assignment are identical to the five lines that appear after the second assignment. With these identical five lines, the program does the following:

(a) it computes the sum using the formula,
(b) it prints a `String` literal `"The sum of integers from 1 to "`,
(c) it prints the value of `n`,
(d) it prints another `String` literal `" = "`, and
(e) it prints the value of `sum`.

The output of the program is as follows:

```
1  The sum of integers from 1 to 100 = 5050
2  The sum of integers from 1 to 1000 = 500500
```

2.3.4.1 Truncation of Real Numbers

Each of the floating point data types, `float` and `double`, has a finite number of bits for representation. This limitation sometimes results in odd output. The next program shows such an example. It works with two variables, `v` and `a`. The program assigns the initial value of `17.0` to `v`, fixes the value of `a` to 3.42567824 (many digits!) and then updates the value `v` four times by: dividing by `a`, subtracting 1.0, multiplying by `a`, and adding `a`. Before starting the series of modifications, as well as after each of the four modifications, the program prints the value of `v` with additional information regarding what value `v` represents. Since $(((v/a)-1)*a)+a = v$, the value is anticipated to return, at the end, to the original value, `17.0`.

```
1  public class RepresentationTest
2  {
3    public static void main( String[] args )
4    {
5      double v = 17.0;
6      double a = 3.42567824;
7
8      System.out.print( "Start: " );
9      System.out.println( v );
10
11     v = v / a;
12     System.out.print( "/ a :  " );
13     System.out.println( v );
14
15     v = v - 1.0;
16     System.out.print( "- 1.0: " );
17     System.out.println( v );
18
19     v = v * a;
20     System.out.print( "* a :  " );
21     System.out.println( v );
22
23     v = v + a;
24     System.out.print( "+ a :  " );
25     System.out.println( v );
26   }
27 }
```

Listing 2.15 A program that shows the limitation of using a finite number of bits for representating real numbers

The result is the following:

```
1 Start: 17.0
2 / a :  4.96252094008689
3 - 1.0: 3.9625209400868897
4 * a :  13.574321760000002
5 + a :  17.0
```

The first line of the output is the original value of `17.0`. The second is the value immediately after `v = v / a`. The third is the value immediately after `v = v - 1.0`. The fourth is the value immediately after `v = v * a`. The last is the value immediately after `v = v + a`.

Notice that there is a difference between the length of the second output line (noted as `/ a`) and the length of the third output line (noted as `- 1.0`). Again, the difference is due to the fact that the correct value of `v / a` requires an infinite number of digits. Also notice the `0000002` at the end the fourth line. With a pencil calculation, the value is `13.57432176`, but the representation does not capture this correctly, hence the extra seven digits appearing at the end. Even though there are some discrepancies, when the output moves to the last line, the tiny quantity of `0.000000000000002` at the end vanishes, and so the output is `17.0` instead of `17.000000000000002`.

2.3.5 Simplified Expressions

There is a way to simplify a mathematical formula that updates a variable using its current value. For a formula of the type

```
a = a ∘ b;
```

where ∘ is one of the five operations $\{+, -, /, *, \%\}$, the expression:

```
a ∘= b;
```

can be used instead. For example,

```
1 int a, b;
2 a = 20;
3 b = 13;
4 a = a + b;
5 b = b * 3;
```

can be simplified as

```
1 int a, b;
2 a = 20;
3 b = 13;
4 a += b;
5 b *= 3;
```

A special case of this short-hand is when the intended operation is either adding 1 to `a` or subtracting 1 from `a`. In this situation, the short-hand of either `a++` or `++a` can be used for adding 1 and the short-hand of either `a-` or `-a` can be used for subtracting 1. The difference between having the `++` or `-` before or after the variable name is based on whether the change (i.e., adding 1 or subtracting 1) takes place *before* or *after* the evaluation of the entire formula takes place. For example, in the next code, adding 1 to `b` occurs before setting the value of `a` to the product of the value of `b` and the value of `c`, and adding 1 to `e` occurs after setting the value of `d` to the product of the value of `e` and the value of `f`.

```
1 a = ++b * c
2 d = e++ * f
```

The following program demonstrates the use of the simplified operations. The program uses two `int` variables, `myInt` and `other`, and initializes the two variables with the values 10 and 13. The program updates `myInt` five times using

- `+= other`,
- `*= other`,
- `-= other`,
- `/= other`, and
- `%= other`

in this order, and reports the action it has performed, as well as the outcome.

```
1  public class ShortHandExperiment
2  {
3    public static void main( String[] args )
4    {
5      int myInt, other;
6      myInt  = 10;
7      other = 13;
8      System.out.print( "myInt is " );
9      System.out.print( myInt );
10     System.out.print( ", other is " );
11     System.out.println( other );
12
13     myInt += other;
14     System.out.print( "Executed myInt += other\tmyInt is " );
15     System.out.println( myInt );
16
17     myInt *= other;
18     System.out.print( "Executed myInt *= other\tmyInt is " );
19     System.out.println( myInt );
20
21     myInt -= other;
22     System.out.print( "Executed myInt -= other\tmyInt is " );
23     System.out.println( myInt );
24
25     myInt /= other;
26     System.out.print( "Executed myInt /= other\tmyInt is " );
27     System.out.println( myInt );
28
29     myInt %= other;
30     System.out.print( "Executed myInt %= other\tmyInt is " );
31     System.out.println( myInt );
32
```

Listing 2.16 A program that demonstrates the use of mathematical short-hand expressions (part 1)

In the second part, the program executes `myInt += other` four times while modifying the value of `other` with

- `++other`,
- `other++`,
- `-other`, and
- `other-`

in this order, and reports the action it has performed and the outcome.

```
33      myInt += ++other;
34      System.out.print( "Executed myInt += ++other\tmyInt is " );
35      System.out.print( myInt );
36      System.out.print( ", other is now " );
37      System.out.println( other );
38
39      myInt += other++;
40      System.out.print( "Executed myInt += other++\tmyInt is " );
41      System.out.print( myInt );
42      System.out.print( ", other is now " );
43      System.out.println( other );
44
45      myInt += --other;
46      System.out.print( "Executed myInt += --other\tmyInt is " );
47      System.out.print( myInt );
48      System.out.print( ", other is now " );
49      System.out.println( other );
50
51      myInt += other--;
52      System.out.print( "Executed myInt += other--\tmyInt is " );
53      System.out.print( myInt );
54      System.out.print( ", other is now " );
55      System.out.println( other );
56    }
57  }
```

Listing 2.17 A program that demonstrates the use of mathematical short-hand expressions (part 2)

The program produces the following output:

```
1  myInt is 10, other is 13
2  Executed myInt += other myInt is 23
3  Executed myInt *= other myInt is 299
4  Executed myInt -= other myInt is 286
5  Executed myInt /= other myInt is 22
6  Executed myInt %= other myInt is 9
7  Executed myInt += ++other       myInt is 23, other is now 14
8  Executed myInt += other++       myInt is 37, other is now 15
9  Executed myInt += --other       myInt is 51, other is now 14
10 Executed myInt += other--       myInt is 65, other is now 13
```

The next code uses two variables, `int myInt` and `double myReal`, to store an integer and a floating point number, respectively. The program computes the product of the two variables and stores the value of the product in a `double` variable, `result`. In addition, in the assignment of the product, the program executes one of `++` or `-` either before or after the two variables:

```
1  result = myReal++ * myInt++;
2  result = ++myReal * ++myInt;
3  result = myReal-- * myInt--;
4  result = --myReal * --myInt;
```

These actions appear in Lines 19, 35, 51, and 66.

Before each operation, the program prints the values of `myInt` and `myReal`, using four statements:

```
1     System.out.print( "myReal = " );
2     System.out.print( myReal );
3     System.out.print( " and myInt = " );
4     System.out.println( myInt );
```

The first and third lines announce the variables whose values are to be printed and the second and the fourth lines print their values.

After the operation, the program reports the action it has performed and the values of the three variables, and then draws a bunch of dashes.

Here is the code, presented in multiple parts. ++ or - appears in Lines 19, 35, 51, and 67.

```
1  public class ShortHandNew
2  {
3    public static void main( String[] args )
4    {
5      ////////////////////////////////////////////////////////////
6      // declaration and initialization
7      ////////////////////////////////////////////////////////////
8      int myInt;
9      double myReal, result;
10     myReal = 89.5;
11     myInt = 17;
12     ////////////////////////////////////////////////////////////
13     // first round
14     ////////////////////////////////////////////////////////////
15     System.out.print( "myReal = " );
16     System.out.print( myReal );
17     System.out.print( " and myInt = " );
18     System.out.println( myInt );
19     result = myReal++ * myInt++;
20     System.out.println( "Execute myReal++ * myInt++" );
21     System.out.print( "The result is " );
22     System.out.println( result );
23     System.out.print( "Now myReal = " );
24     System.out.print( myReal );
25     System.out.print( " and myInt = " );
26     System.out.println( myInt );
27     System.out.println( "---------------------------------------");
```

Listing 2.18 A program that demonstrate the use of ++ and −− (part 1)

```
28      ////////////////////////////////////////////////////////
29      // second round
30      ////////////////////////////////////////////////////////
31      System.out.print( "myReal = " );
32      System.out.print( myReal );
33      System.out.print( " and myInt = " );
34      System.out.println( myInt );
35      result = ++myReal * ++myInt;
36      System.out.println( "Execute ++myReal * ++myInt" );
37      System.out.print( "The result is " );
38      System.out.println( result );
39      System.out.print( "Now myReal = " );
40      System.out.print( myReal );
41      System.out.print( " and myInt = " );
42      System.out.println( myInt );
43      System.out.println( "---------------------------------------");
44      ////////////////////////////////////////////////////////
45      // third round
46      ////////////////////////////////////////////////////////
47      System.out.print( "myReal = " );
48      System.out.print( myReal );
49      System.out.print( " and myInt = " );
50      System.out.println( myInt );
51      result = myReal-- * myInt--;
52      System.out.println( "Execute myReal-- * myInt--" );
53      System.out.print( "The result is " );
54      System.out.println( result );
55      System.out.print( "Now myReal = " );
56      System.out.print( myReal );
57      System.out.print( " and myInt = " );
58      System.out.println( myInt );
59      System.out.println( "---------------------------------------");
60      ////////////////////////////////////////////////////////
61      // fourth round
62      ////////////////////////////////////////////////////////
63      System.out.print( "myReal = " );
64      System.out.print( myReal );
65      System.out.print( "and myInt = " );
66      System.out.println( myInt );
67      result = --myReal * --myInt;
68      System.out.println( "Execute --myReal * --myInt" );
69      System.out.print( "The result is " );
70      System.out.println( result );
71      System.out.print( "Now myReal = " );
72      System.out.print( myReal );
73      System.out.print( " and myInt = " );
74      System.out.println( myInt );
75      System.out.println( "---------------------------------------");
76    }
77  }
```

Listing 2.19 A program that demonstrate the use of ++ and −− (part 2)

Executing the program produces the following result:

```
myReal = 89.5 and myInt = 17
Execute myReal++ * myInt++
The result is 1521.5
Now myReal = 90.5 and myInt = 18
-------------------------------------
myReal = 90.5 and myInt = 18
Execute ++myReal * ++myInt
The result is 1738.5
Now myReal = 91.5 and myInt = 19
-------------------------------------
myReal = 91.5 and myInt = 19
Execute myReal-- * myInt--
The result is 1738.5
Now myReal = 90.5 and myInt = 18
-------------------------------------
myReal = 90.5and myInt = 18
Execute --myReal * --myInt
The result is 1521.5
Now myReal = 89.5 and myInt = 17
-------------------------------------
```

2.4 An Introduction to the `String` Data Type

2.4.1 The `String` Data Type

Recall that `"Hello, World!"` is a `String` literal. `String` is not a primitive data type. Rather, `String` is an object type that encompasses a series of `char` data along with an `int` value representing the length of the series. To create data of an object type, a special method called **constructor** must be used. The use of any constructor requires a special keyword `new`. However, since `String` is such an important data type, the Java language designers have made it possible to create a `String` literal by specifying the character sequence that it encapsulates (using the double quotation mark at the beginning and end of the sequence). The way to declare a `String` variable and assign a value to it is the same as the other data types. The following source code is a new version of the previous quarterback program, with the use of `String` variables.

The program decomposes the messages into the variable parts and the common parts. The first line of the message takes the format of:

`Rank No.` 1 `in my all-time favorite NFL QB list is` `Steve Young` `.`

where the texts in a box are the unchanged parts. Thus, the first line is split into five parts total: the three parts in a box, the rank number, and the name of the quarterback. The boxed parts are character sequences. So is the name of the quarterback. The rank number can be thought of as an integer. The variable names, `rankText`, `favText`, and `period`, are given to the three unchanged text parts respectively, and the variable names, `rank` and `name`, are assigned to the remaining two. The output of the line is:

```
rankText rank favText name period
```

The texts appearing in the boxes, including all the white space, are stored in the variables `rankText`, `favText`, and `period`. The integer literal `1` is stored in the `int` variable `rank` and the `String` literal `"Steve Young"` is assigned to the variable `name`. By printing the first four of the five components (`rankText`, `rank`, `favText`, and `name`) successively with `System.in.print`

and then the last of the five components, period, with System.out.println, the same output can be generated.

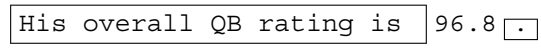
To be more precise, the following code will be used:

```
1     String rankText, favText, period;
2     int rank;
3     String name;
4
5     rankText = "Rank number ";
6     favText = " in my all-time favorite NFL QB's is ";
7     period = ".";
8
9     rank = 1;
10    name = "Steve Young";
11    System.out.print( rankText );
12    System.out.print( rank );
13    System.out.print( favText );
14    System.out.print( name );
15    System.out.println( period );
```

This series of actions produces the desired output line:

```
Rank number 1 in my all-time favorite NFL QB's is Steve Young.
```

The second line can be decomposed in the same manner. The line is



Again, the texts in a box are the unchanged parts. Since we already have declared the variable period and assigned a literal consisting of a period to the variable, we will introduce only one new variable, overText, which holds the other unchanged part His overall QB rating is.

The variable part in this line is a floating point number, so a double variable is used. The name of the variable is qbr. By assigning the value 96.8 to qbr and printing the three components successively, the second line of the output is reproduced.

The following is the code that represents this decomposition:

```
1     String overText;
2     double qbr;
3     overText = "His overall QB rating is ";
4
5     qbr = 96.8;
6     System.out.print( overText );
7     System.out.print( qbr );
8     System.out.println( period );
```

the code will produce the output:

```
His overall QB rating is 96.8.
```

By grouping the same type of variables together, rearranging the assignments and the declarations, and making an assignment to each unchanged part immediately after declaring it, the next code is obtained. In the next code, Lines 5–7 are the variable declarations, Lines 8–11 are the assignments to the unchanged parts, Lines 13–15 are the assignments to the variable parts, and Lines 16–23 are the print statements.

```
1  public class Literals03
2  {
3    public static void main( String[] args )
4    {
5      int rank;
6      String name, rankText, favText, overText, period;
7      double qbr;
8      rankText = "Rank number ";
9      favText = " in my all-time favorite NFL QB list is ";
10     overText = "His overall QB rating is ";
11     period = ".";
12
13     rank = 1;
14     name = "Steve Young";
15     qbr = 96.8;
16     System.out.print( rankText );
17     System.out.print( rank );
18     System.out.print( favText );
19     System.out.print( name );
20     System.out.println( period );
21     System.out.print( overText );
22     System.out.print( qbr );
23     System.out.println( period );
24
25     rank = 2;
26     name = "Peyton Manning";
27     qbr = 96.5;
28     System.out.print( rankText );
29     System.out.print( rank );
30     System.out.print( favText );
31     System.out.print( name );
32     System.out.println( period );
33     System.out.print( overText );
34     System.out.print( qbr );
35     System.out.println( period );
36
37     rank = 3;
38     name = "Tom Brady";
39     qbr = 97.0;
40     System.out.print( rankText );
41     System.out.print( rank );
42     System.out.print( favText );
43     System.out.print( name );
44     System.out.println( period );
45     System.out.print( overText );
46     System.out.print( qbr );
47     System.out.println( period );
48   }
49 }
```

Listing 2.20 A program that produces comments about some NFL quarterbacks using variables and literals

Here is the output of the code:

```
Rank number 1 in my all-time favorite NFL QB's is Steve Young.
His overall QB rating is 96.8.
Rank number 2 in my all-time favorite NFL QB's is Peyton Manning.
His overall QB rating is 96.5.
Rank number 3 in my all-time favorite NFL QB's is Tom Brady.
His overall QB rating is 97.0.
```

2.4.2 String Concatenation

2.4.2.1 Concatenating Two String Data

A String object can represent a very, very long sequence.[4] To specify a String, the beginning and ending double quotation marks must appear in the same line. Therefore, to define a long (say, 900 characters) String literal, the width of the terminal screen is too small; viewing it on a screen results in wraparound, i.e., the character sequence flows into the next line. For example, if a command line interface window has the width of 64 characters (this quantity may change as the window is resized) and a String has 900 characters in a single line, the line will be divided into much smaller segments on the screen.

Fortunately, in Java, it is possible to **concatenate** String literals and variables using the + sign to mean concatenation. It is also possible to concatenate between a String data and data of other data types. Using concatenation, the process of generating output can be simplified.

The following code is a new version of the "favorite quarterbacks" program that uses the concatenation operator:

```
public class Literals04
{
  public static void main( String[] args )
  {
    int rank;
    String name;
    double qbr;
    String rankText = "Rank number ";
    String favText = " in my all-time favorite NFL QB list is ";
    String overallText = "His overall QB rating is ";
    String period = ".";

    rank = 1;
    name = "Steve Young";
    qbr = 96.8;
    System.out.println( rankText + rank + favText + name + period );
    System.out.println( overallText + qbr + period );

```

Listing 2.21 A program that produces comments about three NFL quarterbacks using String concatenation (part 1)

[4] Since int is the data type for specifying the position of a character letter in a character sequence, the limit on the length is $2^{31} - 1$.

```
19      rank = 2;
20      name = "Peyton Manning";
21      qbr = 96.5;
22      System.out.println( rankText + rank + favText + name + period );
23      System.out.println( overallText + qbr + period );
24
25      rank = 3;
26      name = "Tom Brady";
27      qbr = 97.0;
28      System.out.println( rankText + rank + favText + name + period );
29      System.out.println( overallText + qbr + period );
30    }
31  }
```

Listing 2.22 A program that produces comments about three NFL quarterbacks using `String` concatenation (part 2)

Here is another example of using the concatenation operator. The program prints a quote from *Gettysburg Address* by Abraham Lincoln (February 12, 1809 to April 15, 1865):

> *"Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal."*

It is possible to declare a `String` variable and store this sentence in one line. If the width of the screen is 64, the declaration and assignment will appear as:

```
1 String address1 = "Four score and seven years ago our fathers b
2 rought forth on this continent, a new nation, conceived in Libe
3 equal.";
```

with two mid-word breaks. Using the connector +, the code can be made easier to recognize:

```
1   String address1 = "Four score and seven years ago our"
2                   + " fathers brought forth on this continent,"
3                   + " a new nation, conceived in Liberty, and"
4                   + " dedicated to the proposition that all men"
5                   + " are created equal.";
```

Because of the white space appearing at the start of the second, the third, the fourth, and the fifth literals, `System.out.println( address1 )` has the same effect as before:

```
1 Four score and seven years ago our fathers brought forth
2  on this continent, a new nation, conceived in Liberty,
3 and dedicated to the proposition that all men are create
4 d equal.
```

To avoid wraparound, the newline character `\n` to force line breaks can be inserted, e.g.,

```
1 String address1 =
2   "Four score and seven years ago our fathers brought forth\n"
3 + "on this continent, a new nation, conceived in Liberty,\n"
4 + "and dedicated to the proposition that all men are\n"
5 + "created equal.\n";
```

The output of System.out.println(address1) then changes to:

```
Four score and seven years ago our fathers brought forth
on this continent, a new nation, conceived in Liberty,
and dedicated to the proposition that all men are
created equal.
```

Here is another example. The example uses a String variable named row. The variable row has the four lines of the song "Row, row, row your boat". The literal has the newline at the end of each line, so printing it produces the four lines, as shown next:

```
Row, row, row your boat,
Gently down the stream.
Merrily, merrily, merrily, merrily,
Life is but a dream.
```

The code for defining the String is as follows:

```
String row = "Row, row, row your boat,\n"
           + "Gently down the stream.\n"
           + "Merrily, merrily, merrily, merrily,\n"
           + "Life is but a dream.\n"
```

When printing many short lines, the lines can be connected into a single line with a "\n in between, thereby reducing the number of lines in the program. For example,

```
String count = "One\nTwo\nThree\nFour\nFive\n"
         + "Six\nSeven\nEight\nNine\nTen\n";
System.out.print( count );
```

produces the output

```
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
```

2.4.2.2 Concatenating String Data with Other Types of Data

When multiple number data are connected with a String object using the plus sign, the concatenation shows some peculiar behavior. This is because the plus sign has two roles as both the addition operator of numbers and the String connector. Suppose the concatenation has more than two terms and is free of parentheses. The java compiler interprets this from left to right, and in each concatenation, if both terms are numbers, then the compiler treats it as the number addition.

In the following code fragment:

```
String word1 = "csc120";
String word2 = 4 + 5 + 6 + "ab" + word1;
```

The value of `word2` becomes the literal `"15abcsc120"`, but not `"456abcsc120"`. This is because the first and the second concatenations are additions.

Consider the following example:

```
//-- examples of string manipulation
public class StringVariables {
  public static void main( String[] args ) {
    String myString = "abcde";
    int myInteger = 10;
    double myDouble = 9.5;
    System.out.print( "myString = " + myString );
    System.out.print( ", myInteger = " + myInteger );
    System.out.println( ", myDouble = " + myDouble );

    System.out.print( "myString + myInteger + myDouble = " );
    System.out.println( myString + myInteger + myDouble );

    System.out.print( "myString + (myInteger + myDouble) = " );
    System.out.println( myString + ( myInteger + myDouble ) );

    System.out.print( "myInteger + myString + myDouble = " );
    System.out.println( myInteger + myString + myDouble );

    System.out.print( "myInteger + myDouble + myString = " );
    System.out.println( myInteger + myDouble + myString );
  }
}
```

Listing 2.23 A program that contains `print` statements that print values of `String` data generated by concatenation

In this code, both `System.out.print` and `System.out.println` have one or two concatenations appearing in their parentheses. The execution of the code produces the following result:

```
myString = abcde, myInteger = 10, myDouble = 9.5
myString + myInteger + myDouble = abcde109.5
myString + (myInteger + myDouble) = abcde19.5
myInteger + myString + myDouble = 10abcde9.5
myInteger + myDouble + myString = 19.5abcde
```

For `String` concatenation, the short-hand expression of += is available. In other words, `w += x` can be used in place of `w = w + x`.

Summary

- Data carries information.
- A data with a name is a variable. An example of data without a name is a literal.
- A variable declaration requires the type and the name of the variable.
- It is possible to declare multiple variables of the same type with just one type specification.
- A variable declaration with `final` is a declaration of a constant.
- Each variable declaration is valid until the end of the innermost pair of matching curly brackets that contain it. This is the scope of the variable.
- Two declarations having the same variable name must not have overlapping scopes.
- The declaration of a variable must appear before an assignment to the variable.
- A variable name must consist of letters and numerals, and must start with a letter.
- The reserved words of Java cannot be used for names.
- All the methods in a class have access to global variables.
- Global variable declarations appear at depth 1 with the attribute of `static` while local variable declarations appear at depth 2.
- There are eight primitive data types. Other data types are object data types.
- `+`, `-`, `*`, `/`, and `%` are five elementary mathematical operations.
- Five shorthand expressions are available for self-updating. They are `+=`, `-=`, `*=`, `/=`, `%=`, `++`, and `--`.
- It is possible to attach `++` or `--` to a variable, and the position of the attachment can be either before or after the variable.
- When a binary operator operates on two numbers of different types, the program chooses to represent both with a floating number if and only if one of them is of a floating number. Furthermore, for two numbers of different types, the program will choose the type with the larger number of bits to represent both numbers.
- It is possible to concatenate `String` literals and `String` variables with the `+` sign.
- When a number concatenates with a `String`, the result is a `String`.

Exercises

1. **Memory size** The following are the four primitive data types in Java that represent whole numbers. State how many bits are required to represent each of them.
   (a) `byte`
   (b) `short`
   (c) `int`
   (d) `long`
2. **Memory size** The following are the two primitive data types in Java that represent floating point numbers. State how many bits are required to represent each of them.
   (a) `float`
   (b) `double`
3. **Casting** State whether or not the following statements are correct.
   (a) To represent a whole number literal as a `short`, attach `S` at the end, e.g., as in `120S`.
   (b) To represent a whole number literal as a `byte`, attach `B` at the end, e.g., as in `120B`.
   (c) To represent a floating point number literal as a `float`, attach `F` at the end, e.g., as in `120.5F`.

4. **Data type** Suppose `a` is a `long` variable, `b` is an `int` variable, and `c` is a `double` variable. State the data types of the following formulas:
   (a) `a / b`
   (b) `a / c`
   (c) `b / c`
5. **Declaring variables** Write a program, `MyFavorites`, that declares two variables, `String word` and `int lucky`, assigns some literals to them, and then prints the values as follows:

```
My favorite word is "XXX".
My lucky number is YYY.
```

   where XXX is for the value of `word` and YYY is for the value of `lucky`.
6. **Value assignments to variables** Consider the following series of statements:

```
int a = 11;
int b = 3;
int c = a / b + a % b;
a = a + b;
b = a - b;
```

   What are the values of `a`, `b` and `c` after the very last statement?
7. **Cyclic value assignments**
   Consider the following series of statements:

```
int a, b, c, d;
a = 11;
b = 12;
c = 13;
d = 14;
a = b;
b = c;
c = d;
d = a;
```

   What are the values of the variables `a`, `b`, `c`, and `d` after each of the eight assignments?
8. **What's wrong with the code?** Assume that the code below is part of the method `main` of a class. State what is wrong with the code:

```
int numberX = 0, numberY = 2;
int numberX += numberY, numberZ;
numberZ = 0.5;
realW = 71.5;
double realW *= realW;
```

9. **What's wrong with the code?** Assume that the code below is part of the method `main` of a class. State what is wrong with the code:

```
int a = 11, b = 10;
int b = a * b;
final int c = a;
c = a - 7;
```

10. **Variable evaluation, ++ and −−** After executing the code below, what are the values of `x`, `y`, `z` and `w`?

```
int x, y, z, w;
x = 10;
y = 3;
z = x * y--;
w = --x * --y;
```

11. **Variable evaluation, ++ and −− with division** After executing the code below, what are the values of `z` and `w`?

```
int x, y, z, w;
x = -11;
y = 4;
z = x / ++y;
w = ++x % y--;
```

12. **Variable evaluation, String and number** After executing the code below, what are the values of `z` and `w`?

```
String x, y, z, w;
x = "emerson";
y = "palmer";
z = x + "lake" + y;
w = 1 + 2 + z + 3 + 4;
```

13. **Value exchanging** Let `a` and `b` be `int` variables. Write a code for exchanging the values of `a` and `b`. For example, when `a` has the value of 10 and `b` has the value of 7, the exchange will make the value of `a` equal to 7 and `b` equal to 10. Assume that `a` and `b` have been already declared in the code and have been assigned values, so the task at hand is to swap the values of the two.
14. **Value exchanging, again** Let `a`, `b`, and `c` be `int` variables. Write a code for exchanging the values of the three (the original value of `a` goes to `b`, the original value of `b` goes to `c`, and the original value of `c` goes to `a`).
15. **Short-hand** Suppose `a` and `b` are `int` variables. What are the values of `a` and `b` after executing the following?

```
a = 3;
b = 2;
a *= b;
b *= a;
a *= b;
b *= a;
```

Programming Projects

16. **Gravity again** Recall that if an object is released so that it falls, the speed of the object at t seconds after its release is gt and the distance the object has travelled in the t seconds after release is $\frac{1}{2}gt^2$. Here, g is the gravity constant. Its value is approximately 9.8. Write a program, `Gravity`, in which the method `main` performs the following actions:
    (a) The program declares variables `t` for the travel time, `speed` for the speed, and `distance` for the distance traveled;
    (b) The program assigns some value to `t`;
    (c) The program calculates the speed and the distance;
    (d) The program prints the calculated values.
    (e) The program assigns a different value to `t` and repeats Steps 1–4.
17. **Computing the tax from a subtotal and then the total** Write a program, `ComputeTaxAndTotal`, that computes the tax and the total in the following manner:
    - The program uses an `int` variable `subtotal` to store the subtotal in cents. Since we do not know (yet) how to receive an input from the user, assign some int literal to this variable in the code.
    - The program use a `double` variable `taxPercent` to store the tax rate in percent. Again, since we do not know (yet) how to receive an input from the user, assign some int literal to this variable in the code (for example, the tax rate is 5.5% in the state of Massachusetts).
    - The program then computes the tax amount as a whole number in cents, in an `int` variable `tax`. Using the casting of `(int)`, a floating point number can be converted to a whole number by rounding it down.
    - The program then computes the total, and stores it in an `int` variable `total`. (Again, this quantity is in cents.)
    - The program reports the result of the calculation in dollars and cents for the subtotal, the tax, and the total, and then reports the tax rate.

    The output the code may look like:

```
1 The subtotal = 110 dollars and 50 cents.
2 The tax rate = 5.5 percent.
3 The tax = 6 dollars and 7 cents.
4 The total = 116 dollars and 57 cents.
```

18. **Speeding fine** In the town of Treehead, the speeding fines are $20 times the mileage beyond the speed limit. For example, if a driver was driving at 36 mph on a 30 mph road, his fine is $120. Write a program, `SpeedingFine`, that produces the speeding fines for the following combinations of speed and speed limit:
    - (50 mph, 35 mph)
    - (30 mph, 25 mph)
    - (60 mph, 45 mph)

    In the program, declare `int` variables, `speed`, `limit`, and `fine`. Compute the fine by assigning values to the first two variables and multiplying it by the rate of `20`. To report the results, write a series of `print/println` statements in which the speed, the limit, and the fine appear as variables. The code should execute this series three times and, before each series, the assignment to the three variables must appear.

    Here is an execution example of the program.

```
1 The fine for driving at 50 mph on a 35 mph road is 300 dollars.
2 The fine for driving at 30 mph on a 25 mph road is 100 dollars.
3 The fine for driving at 60 mph on a 45 mph road is 300 dollars.
```

19. **The area of a trapezoid** Write a program, `Trapezoid`, that computes the area of a trapezoid when given the top and bottom lengths and the height. The program specifies the top, the bottom, and the height of the trapezoid in individual `System.out.println` statements. The program computes the area by directly putting the formula (bottom + top) * height / 2 inside a System.out.println statement. Freely choose the variables for the top, the bottom, and the height.

    For example, the output of the program can be:

```
1 Top: 10.0
2 Bottom: 20.5
3 Height: 24.4
4 Area: 372.09999999999997
```

    Split each line of the output into two parts: the first part prints the text, including the whitespace, using a `System.out.print` statement, and the second part prints the quantity using a `System.out.println` statement. For example, the first line should use the following two statements:

```
1   System.out.print( "Top: " );
2   System.out.println( 10.0 );
```