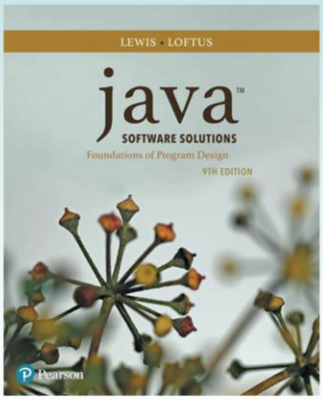
Chapter 2
Data and Expressions



Java Software Solutions
Foundations of Program Design
9th Edition

John Lewis
William Loftus

Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs

Expressions

- An *expression* is a combination of one or more operators and operands
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands are floating point values, then the result is a floating point value

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3   equals   4
8 / 12   equals   0
```

- The remainder operator (%) returns the remainder after dividing the first operand by the second

```
14 % 3   equals   2
8 % 12   equals   8
```


-Care must be taken when using the division operator

-Results depend on the data types of the operands used in the operation:

Division

-If both are integers, **integer division** is applied

-Integer division: fractional part of result is **discarded** (not rounded)!

-If one or both are float or double, **floating point division** is applied

-Floating point division: fractional part of result is **kept** (result is float,double)

Remainder (also called modulus)

-Returns remainder after division (not fractional part, whole remainder)

Quick Check

What are the results of the following expressions?

```
    12 / 2
12.0 / 2.0
    10 / 4
  10 / 4.0
    4 / 10
  4.0 / 10
    12 % 3
    10 % 3
    3 % 10
```

Quick Check

What are the results of the following expressions?

```
   12 / 2  =  6
12.0 / 2.0 =  6.0
   10 / 4  =  2
 10 / 4.0  =  2.5
   4 / 10  =  0
 4.0 / 10  =  0.4
   12 % 3  =  0
   10 % 3  =  1
   3 % 10  =  0
```

Operator Precedence

- Operators can be combined into larger expressions

```
result = total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated before addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

-Operator precedence enforced using actual values (literals) or using variables, e.g.

literals:3 * 4 / 5 + 1

variables: x * y / j + k

-Operator precedence follows same order as those in mathematical expressions

-Expressions in parentheses evaluated first

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
```

```
a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

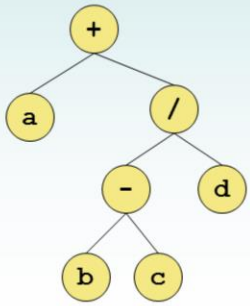
```
a / (b + c) - d % e
  2     1     4   3
```

```
a / (b * (c + (d - e)))
  4     3     2     1
```

Expression Trees

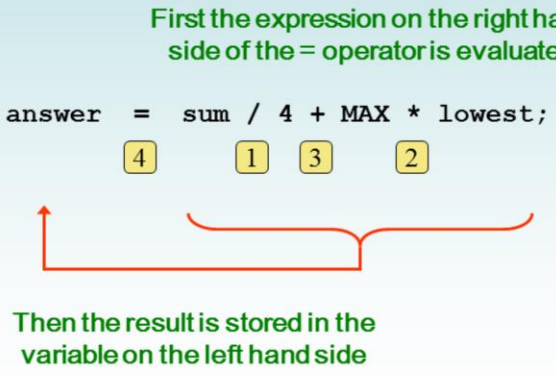
- The evaluation of a particular expression can be shown using an *expression tree*
- The operators lower in the tree have higher precedence for that expression

```
a + (b - c) / d
```



Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators



-In other words, the expression is solved **before** the assignment

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count = count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

-Other terms we use for adding/subtracting by one is **increment, decrement**

-Let's look at Listing 2.7 closely, note use of a constant to represent 9.0/5.0

-If we needed to convert another temperature, we can reuse this constant

-This is more efficient than repeating 9.0/5.0 multiple times

-We avoid using actual values in code (also referred to as "hard-coding")

Increment and Decrement

- The increment (++) and decrement (--) operators use only one operand
- The statement

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```


-Increment and decrement operators are **unary** operators

-Unary operators operate on a single operand (the values around the operator)

-Note these modify the actual value of the operand

Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

```
count++
```

- or *prefix form*:

```
++count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

-Postfix and prefix affect expressions differently

-Example in book:

```
int count = 15;
int total;
total = count++;    // first, total assigned 15, then
count is incremented to 16 (postfix)
```

```
int count = 15;
int total;
total = ++count;    // first, count is incremented to 16
(prefix), then total assigned 16
```

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the += operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)

-Note string concatenation with += operator

String test = “Hello”;

test += “ World”; // result is “Hello World”

Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs

Data Conversion

- Sometimes it is convenient to convert data from one type to another
- For example, in a particular situation we may want to treat an integer as a floating point value
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversion

- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - promotion
  - casting

Data Conversion

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- Example:

```
int dollars = 20;
double money = dollars;
```

- Only widening conversions can happen via assignment
- Note that the value or type of `dollars` did not change

Promotion

- *Promotion* happens automatically when operators in expressions convert their operands
- Example:

```
int count = 12;
double sum = 490.27;
result = sum / count;
```

- The value of `count` is converted to a floating point value to perform the division calculation

Casting

- *Casting* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;
float result = (float) total / 6;
```

- Without the cast, the fractional part of the answer would be lost

-A **cast** is actually an operator () with the data type inside the parentheses

-Note an integer cast on a float/double will **truncate** the fractional part

```
double money = 84.69;
int dollars = (int)money;          // stores 84
```

Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

-Note once again the use of the term **class** and **object**, be sure you know the difference

-Similar to a blueprint, the class is the **definition** of a data model

-Similar to a house, the object is the **realization** or **instantiation** of a data model

-A class is actually a data type in addition to the primitive data types (int,float,…)

-To create objects of a class, we declare them in our programs

-So far, we are using classes others have written, later we'll write our own!

-The Scanner class is an example of a class written by someone else

-Note what this class can "do" in Fig. 2.7 that lists some of its **methods**

-It is provided in a library that can we access called the **Java standard class library**

-We'll look at classes and objects in depth in the next chapter

Reading Input

- The following line creates a `Scanner` object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```


-To create an object from a class definition, we use the **new** operator

-We also use a special type of method in the class called a **constructor**

-This method, along with the new operator, creates an object in memory for us to use

-You might be wondering why we didn't have to use this syntax (as below) for a String object

String hello = "Hello";

-As we'll see in the next chapter, this statement is actually a shortcut to

String hello = new String("Hello");

-The new operator and constructor are automatically called

-Note the System.in is an object from a class that describes input (e.g. keyboard)

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used
- The `nextLine` method reads all of the input until the end of the line is found
- See `Echo.java`
- The details of object creation and class libraries are discussed further in Chapter 3

-The import statement specifies the libraries that classes are contained within

-We import a library so we can use the names of classes contained within them

-Some classes (e.g. String class) are part of libraries that are automatically imported

-The Scanner class is part of a class library not automatically imported

-When we declare a Scanner object, note how Eclipse flags this as an error

-It mentions how “Scanner cannot be resolved to a type”

-This is because the Java compiler doesn’t know what the type is until we import the library

-If you click on the Eclipse error icon (to the left of the editor), it offers solutions

-One solution is to Import ‘Scanner’ which is part of the a library named java.util

-Obviously, you can also type the import statement yourself

-Another shortcut is Ctrl-Shift-O (or Source->Organize imports)

-We’ll study the import statement in depth in the next chapter

```
//********************************************************************
//  Echo.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a string from the user.
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

Sample Run

```
Enter a line of text:
You want fries with that?
You entered: "You want fries with that?"
```

```
//********************************************************************
//  Echo.java
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a character string from the user.
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
- See `GasMileage.java`

-A **token** refers to a separate element contained within an input

-A **delimiter** is the character(s) that separates different tokens

-Strings are typically made up of tokens and delimiters

-For example:

"Hello World", a white space is the delimiter between tokens "Hello" and "World"

"Doe,John", a comma is the delimiter between tokens "Doe" and "John"

-The Scanner class uses the default white space as delimiters although it can be changed

```
//********************************************************************
//  GasMileage.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the Scanner class to read numeric data.
//********************************************************************

import java.util.Scanner;

public class GasMileage
{
   //-----------------------------------------------------------------
   //  Calculates fuel efficiency based on values entered by the
   //  user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int miles;
      double gallons, mpg;

      Scanner scan = new Scanner (System.in);
```

continue

continue

```
      System.out.print ("Enter the number of miles: ");
      miles = scan.nextInt();

      System.out.print ("Enter the gallons of fuel used: ");
      gallons = scan.nextDouble();

      mpg = miles / gallons;

      System.out.println ("Miles Per Gallon: " + mpg);
   }
}
```

continue

```
        Sy
        mi

        Sy
        gallons = scan.nextDouble();

        mpg = miles / gallons;

        System.out.println ("Miles Per Gallon: " + mpg);
    }
}
```

Sample Run

```
Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
```