

Java Fundamentals - CSE

Week 2 - Data Types

Dr. Thiago Braga Rodrigues

Thiago.Braga@tus.ie

TUS
2025

First Program

```
1 public class HelloPrinter
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, World!");
6     }
7 }
```

- 1: Declares a 'class' HelloPrinter
 - -- Every Java program has one or more classes.
- 3: Declares a method called 'main'
 - -- Every Java application has exactly one 'main' method
 - -- Entry point where the program starts
- 5: Method System.out.println outputs 'Hello, World!'
 - -- A statement must end with a semicolon (;)

Goals

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read and process inputs, and display the results

Contents

- Variables
- Arithmetic
- Input and Output
- Problem Solving:
 - First Do It By Hand
- Strings

Numbers and character strings (such as the ones in this display board) are important data types in any Java program. In this chapter, you will learn how to work with numbers and text, and how to write simple programs that perform useful tasks with them.

2.1 Variables

- Most computer programs hold temporary values in named storage locations
 - Programmers name them for easy access
- There are many different types (sizes) of storage to hold different things
- You 'declare' a variable by telling the compiler:
 - What type (size) of variable you need
 - What name you will use to refer to it



Syntax 2.1: Variable Declaration

- When declaring a variable, you often specify an initial value
- This is also where you tell the compiler the size (type) it will hold

```
int cansPerPack = 6;
```

Example Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
<code>int cans = 6;</code>	Declares an integer variable and initializes it with 6.
<code>int total = cans + bottles;</code>	The initial value need not be a fixed value. (Of course, <code>cans</code> and <code>bottles</code> must have been previously declared.)
 <code>bottles = 1;</code>	Error: The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.1.4.
 <code>int volume = "2";</code>	Error: You cannot initialize a number with a string.
<code>int cansPerPack;</code>	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
<code>int dollars, cents;</code>	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

Why different types?

- There are three different types of variables that we will use in this chapter:



- | | |
|--|---------------------|
| 1) A whole number (no fractional part) | <code>int</code> |
| 2) A number with a fraction part | <code>double</code> |
| 3) A word (a group of characters) | <code>String</code> |

- Specify the type before the name in the declaration

```
int cansPerPack = 6;  
double canVolume = 12.0;
```


Number Literals in Java

Table 2 Number Literals in Java

Number	Type	Comment
6	int	An integer has no fractional part.
-6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type double.
1.0	double	An integer with a fractional part .0 has type double.
1E6	double	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type double.
2.96E-2	double	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
 100,000		Error: Do not use a comma as a decimal separator.
 3 1/2		Error: Do not use fractions; use decimal notation: 3.5






Naming Variables

- Name should describe the purpose
 - 'canVolume' is better than 'cv'
- Use These Simple Rules
 - 1) Variable names must start with a letter or the underscore (_) character
 - Continue with letters (upper or lower case), digits or the underscore
 - 2) You cannot use other symbols (? or %...) and spaces are not permitted
 - 3) Separate words with 'camelHump' notation
 - Use upper case letters to signify word boundaries
 - 4) Don't use reserved 'Java' words

Variable Names in Java

- Legal and illegal variable names

Table 3 Variable Names in Java

Variable Name	Comment
canVolume1	Variable names consist of letters, numbers, and the underscore character.
x	In mathematics, you use short variable names such as x or y . This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 38).
 CanVolume	Caution: Variable names are case sensitive. This variable name is different from canVolume, and it violates the convention that variable names should start with a lowercase letter.
 6pack	Error: Variable names cannot start with a number.
 can volume	Error: Variable names cannot contain spaces.
 double	Error: You cannot use a reserved word as a variable name.
 1tr/fl.oz	Error: You cannot use symbols such as / or .

The Assignment Statement

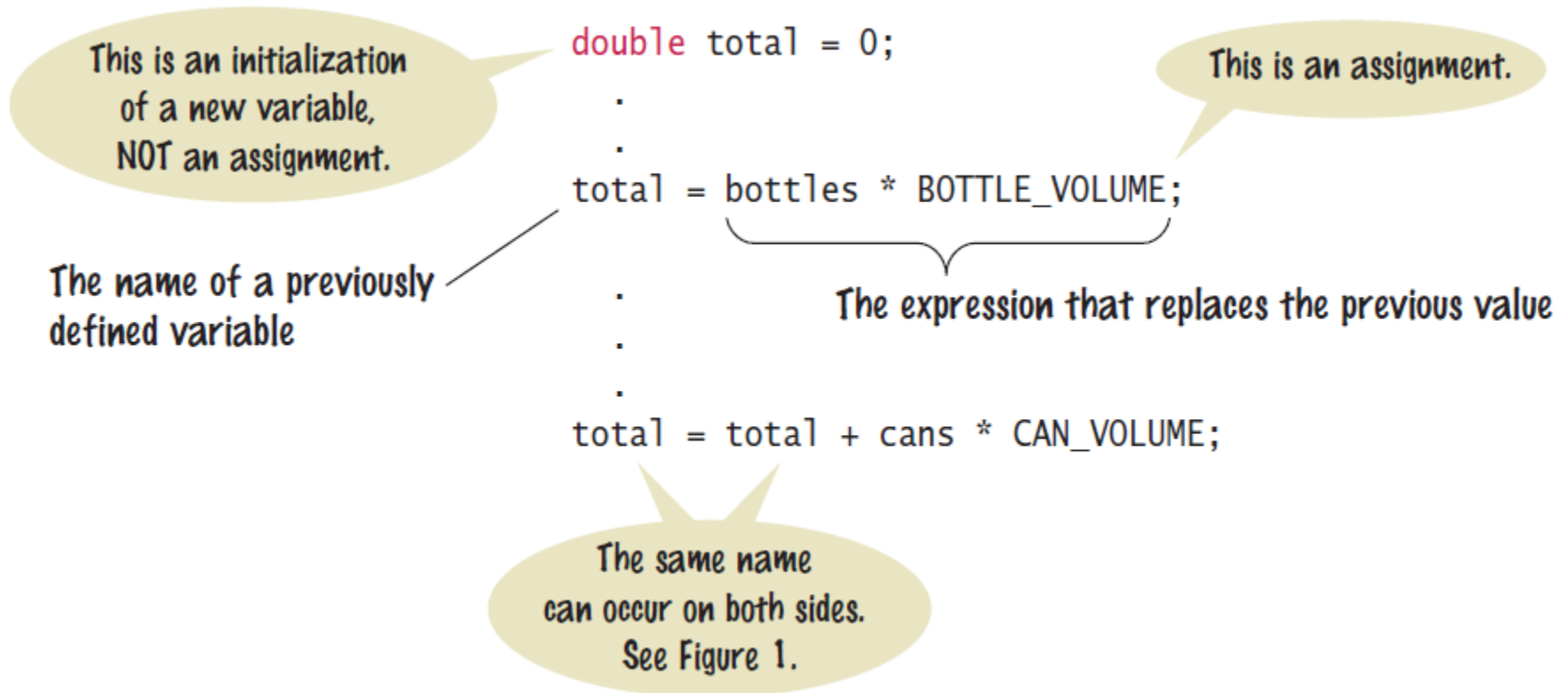
- Use the 'assignment statement' (with an '=') to place a new value into a variable

```
int cansPerPack = 6;    // declare & initialize  
cansPerPack = 8;       // assignment
```

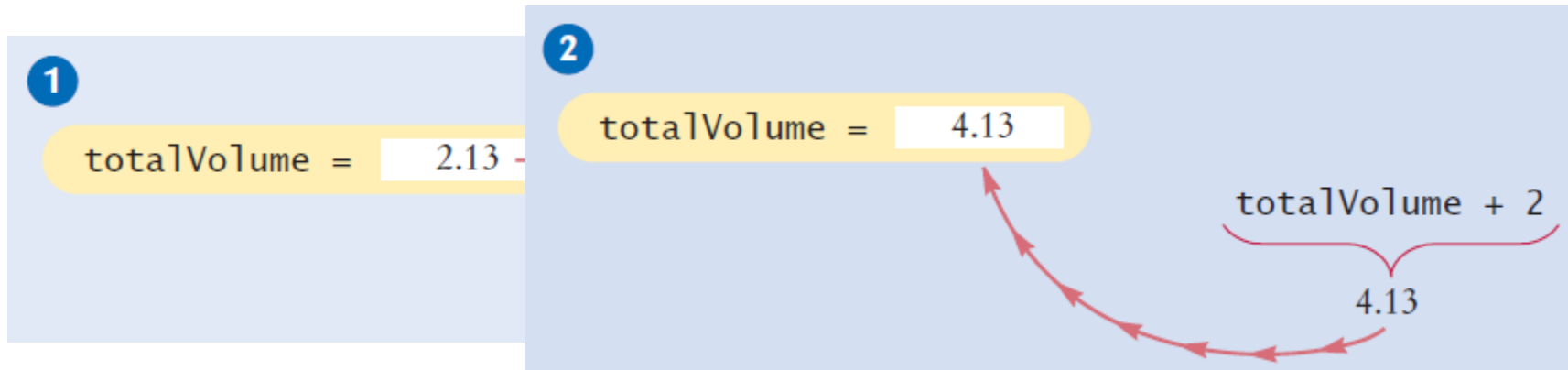
- Beware: The = sign is **NOT** used for comparison:
 - It copies the value on the right side into the variable on the left side
 - You will learn about the comparison operator soon.

Assignment Syntax

- The value on the right of the '=' sign is copied to the variable on the left



Updating a Variable



- Step by Step:

`totalVolume = totalVolume + 2;`

1. Calculate the right-hand side of the assignment Find the value of `totalVolume`, and add 2 to it
2. Store the result in the variable named on the left side of the assignment operator (`totalVolume` in this case)

Constants

- When a variable is defined with the reserved word **final**, its value can never be changed

```
final double BOTTLE_VOLUME = 2;
```

- It is good style to use named constants to explain numerical values to be used in calculations
 - Which is clearer?

```
double totalVolume = bottles * 2;  
double totalVolume = bottles * BOTTLE_VOLUME;
```
- A programmer reading the first statement may not understand the significance of the 2
- Also, if the constant is used in multiple places and needs to be changed, only the initialization changes

Constant Declaration

The `final` reserved word indicates that this value cannot be modified.

```
final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
```

Use uppercase letters for constants.

This comment explains how the value for the constant was determined.

- It is customary (not required) to use all UPPER_CASE letters for constants

Java Comments

- There are three forms of comments:

1: `//` single line (or rest of line to right)

2: `/*`

multi-line – all comment until matching

`*/`

3: `/**`

multi-line Javadoc comments

`*/`

Use comments to add explanations for humans who read your code. The compiler ignores comments.

- Use comments at the beginning of each program, and to clarify details of the code

Java Comment Example

```
1  /**
2   This program computes the volume (in liters) of a six-pack of soda
3   cans and the total volume of a six-pack and a two-liter bottle.
4   */
5  public class Volume1
6  {
7      public static void main(String[] args)
8      {
9          int cansPerPack = 6;
10         final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
11         double totalVolume = cansPerPack * CAN_VOLUME;
12
13         System.out.print("A six-pack of 12-ounce cans contains ");
14         System.out.print(totalVolume);
15         System.out.println(" liters.");
16
17         final double BOTTLE_VOLUME = 2; // Two-liter bottle
```

- Lines 1 - 4 are Javadoc comments for the class `Volume1`
- Lines 10 and 17 use single-line comment to clarify the unit of measurement

Common Error 2.1

- Undeclared Variables

- You must declare a variable before you use it: (i.e. above in the code)

```
double canVolume = 12 * literPerOunce; // ??  
double literPerOunce = 0.0296;
```

- Uninitialized Variables

- You must initialize (i.e. set) a variable's contents before you use it

```
int bottles;  
int bottleVolume = bottles * 2;    // ??
```

Common Error 2.2

- Overflow means that storage for a variable cannot hold the result

```
int fiftyMillion = 50000000;  
System.out.println(100 * fiftyMillion);  
// Expected: 5000000000
```

- Why?
 - The result (5 billion) overflowed int capacity
 - Maximum value for an int is **+2,147,483,647**
- Use a long instead of an int (or a double)

Value Ranges per Type

❑ Integer Types

- **byte:** A very small number (-128 to +127)
- **short:** A small number (-32768 to +32767)
- **int:** A large number (-2,147,483,648 to +2,147,483,647)
- **long:** A huge number

❑ Floating Point Types

- **float:** A huge number with decimal places
- **double:** Much more precise, for heavy math

❑ Other Types

- **boolean:** **true** or **false**
- **char:** One symbol in single quotes 'a'

2.2 Arithmetic



- Java supports all of the same basic math as a calculator:
 - Addition +
 - Subtraction -
 - Multiplication *
 - Division /
- You write your expressions a bit differently though..
Algebra Java

$$\frac{a + b}{2}$$

$$(a + b) / 2$$

- Precedence is similar to Algebra:
 - PEMDAS
 - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

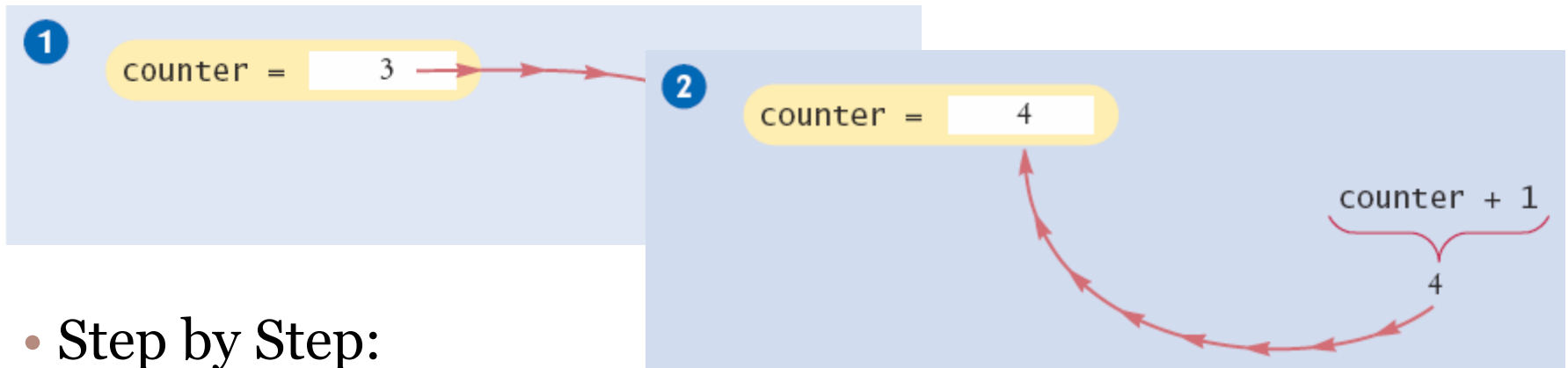
Mixing Numeric Types

- It is safe to convert a value from an integer type to a floating-point type
 - No 'precision' is lost
- But going the other way can be dangerous
 - All fractional information is lost
 - The fractional part is discarded (not rounded)
- If you mix types integer and floating-point types in an expression, no precision is lost:

```
double area, pi = 3.14;  
int radius = 3;  
area = radius * radius * pi;
```

Mixing integers and floating-point values in an arithmetic expression yields a floating-point value.

Incrementing a Variable



- Step by Step:

`counter = counter + 1;`

1. Do the right-hand side of the assignment first:

Find the value stored in `counter`, and add 1 to it

2. Store the result in the variable named on the left side of the assignment operator (`counter` in this case)

Shorthand for Incrementing

- Incrementing (+1) and decrementing (-1) integer types is so common that there are shorthand version for each

Long Way	Shortcut
<code>counter = counter + 1;</code>	<code>counter++ ;</code>
<code>counter = counter - 1;</code>	<code>counter-- ;</code>

Integer Division and Remainder

- When both parts of division are integers, the result is an integer.

- All fractional information is lost (no rounding)

- ```
int result = 7 / 4;
```

- The value of result will be 1

Integer division loses all fractional parts of the result and does not round

- If you are interested in the remainder of dividing two integers, use the % operator (called modulus):

- ```
int remainder = 7 % 4;
```

- The value of remainder will be 3

- Sometimes called modulo divide

Powers and Roots

- In Java, there are no symbols for power and roots

$$b \times \left(1 + \frac{r}{100}\right)^n \quad \square \quad \text{Becomes:}$$

$$\square \quad b * \text{Math.pow}(1 + r / 100, n)$$

- Analyzing the expression:

The Java library declares many Mathematical functions, such as `Math.sqrt` (square root) and `Math.pow` (raising to a power).

$$\begin{array}{c} b * \text{Math.pow}(1 + \underbrace{r / 100}_{1 + \frac{r}{100}}, n) \\ \underbrace{\left(1 + \frac{r}{100}\right)^n}_{b \times \left(1 + \frac{r}{100}\right)^n} \end{array}$$

Mathematical Methods

Method	Returns
<code>Math.sqrt(x)</code>	Square root of x (≥ 0)
<code>Math.pow(x, y)</code>	x^y ($x > 0$, or $x = 0$ and $y > 0$, or $x < 0$ and y is an integer)
<code>Math.sin(x)</code>	Sine of x (x in radians)
<code>Math.cos(x)</code>	Cosine of x
<code>Math.tan(x)</code>	Tangent of x
<code>Math.toRadians(x)</code>	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
<code>Math.toDegrees(x)</code>	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
<code>Math.exp(x)</code>	e^x
<code>Math.log(x)</code>	Natural log ($\ln(x)$, $x > 0$)

Floating-Point to Integer Conversion

- The Java compiler does not allow direct assignment of a floating-point value to an integer variable

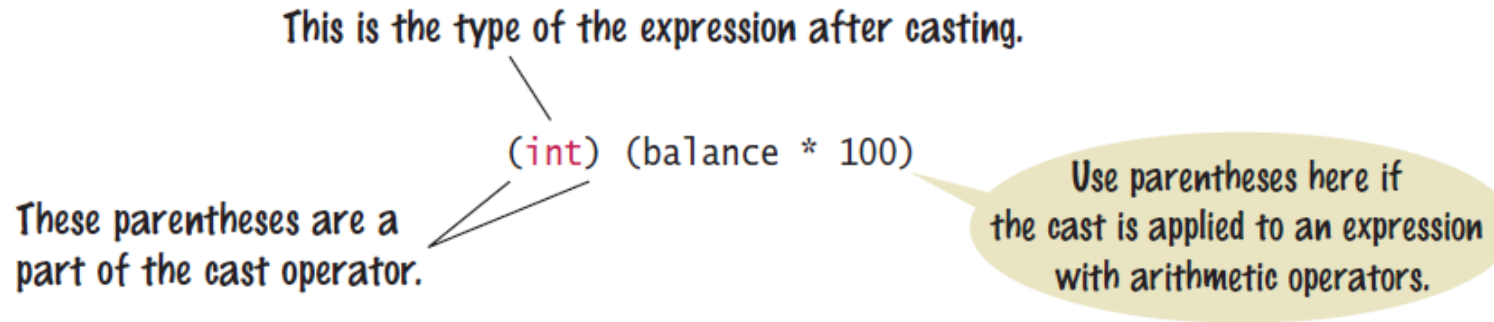
```
double balance = total + tax;  
int dollars = balance; // Error
```

- You can use the 'cast' operator: `(int)` to force the conversion:

```
double balance = total + tax;  
int dollars = (int) balance; // no Error
```

- You lose the fractional part of the floating-point value (no rounding occurs)

Cast Syntax



- Casting is a very powerful tool and should be used carefully
- To round a floating-point number to the nearest whole number, use the `Math.round` method
- This method returns a long integer, because large floating-point numbers cannot be stored in an `int`

```
long rounded = Math.round(balance);
```

Arithmetic Expressions

Mathematical Expression	Java Expression	Comments
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	The parentheses are required; <code>x + y / 2</code> computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	<code>x * y / 2</code>	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	<code>Math.pow(1 + r / 100, n)</code>	Use <code>Math.pow(x, n)</code> to compute x^n .
$\sqrt{a^2 + b^2}$	<code>Math.sqrt(a * a + b * b)</code>	<code>a * a</code> is simpler than <code>Math.pow(a, 2)</code> .
$\frac{i + j + k}{3}$	<code>(i + j + k) / 3.0</code>	If <i>i</i> , <i>j</i> , and <i>k</i> are integers, using a denominator of 3.0 forces floating-point division.
π	<code>Math.PI</code>	<code>Math.PI</code> is a constant declared in the <code>Math</code> class.

Common Error

- Unintended Integer Division

```
System.out.print("Please enter your last three test  
scores: ");  
int s1 = in.nextInt();  
int s2 = in.nextInt();  
int s3 = in.nextInt();  
double average = (s1 + s2 + s3) / 3; // Error
```

- Why?

- All of the calculation on the right happens first
 - Since all are ints, the compiler uses integer division
- Then the result (an int) is assigned to the double
 - There is no fractional part of the int result, so zero (.0) is assigned to the fractional part of the double

2.3 Input and Output

You might need to ask for input (aka prompt for input) and then save what was entered.

- We will be reading input from the keyboard
- This is a three step process in Java
 - 1) Import the Scanner class from its 'package' `java.util`
`import java.util.Scanner;`
 - 2) Setup an object of the Scanner class
`Scanner in = new Scanner(System.in);`
 - 3) Use methods of the new Scanner object to get input
`int bottles = in.nextInt();`
`double price = in.nextDouble();`

Syntax 2.3: Input Statement

- The **Scanner** class allows you to read keyboard input from the user
 - It is part of the Java API `util` package

Java classes are grouped into packages. Use the **import** statement to use classes from packages.

Include this line so you can use the Scanner class.

```
import java.util.Scanner;
```

Create a Scanner object to read keyboard input.

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

Don't use `println` here.

Display a prompt in the console window.

```
. System.out.print("Please enter the number of bottles: ");
```

Define a variable to hold the input value.

```
int bottles = in.nextInt();
```

The program waits for user input, then places the input into the variable.

Lecture 3

How to work with data

Objectives

Applied

- Given the specifications for an application that uses any of the eight primitive data types presented in this chapter, write the application.
- Use the NumberFormat, Math, Integer, Double, and BigDecimal classes to work with data.
- Given the Java code for an application that uses any of the language elements presented in this chapter, explain

Objectives (cont.)

Knowledge

- Describe any one of the eight primitive types.
- Distinguish between a variable and a constant.
- Given a list of names, identify the ones that follow the naming recommendations for constants presented in this chapter.
- Explain the difference between a binary operator and a unary operator and give an example of each.
- Explain the difference between prefixing and postfixing an increment or decrement operator.
- Explain what a shortcut assignment operator is and how you use one in an assignment statement.
- List the order of precedence for arithmetic operations and explain how you can change the order in which operations are performed.

Objectives (cont.)

- Explain what casting is, when it's performed implicitly, and when you must perform it explicitly.
- Describe how casting between int and double types can affect the decimal value in a result.
- Describe the primary uses of these classes: NumberFormat, Math, Integer, and Double.
- List two reasons for using the BigDecimal class.

The eight primitive data types

Type	Bytes	Use
byte	1	Very short integers from -128 to 127.
short	2	Short integers from -32,768 to 32,767.
int	4	Integers from -2,147,483,648 to 2,147,483,647.
long	8	Long integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
float	4	Single-precision, floating-point numbers from -3.4E38 to 3.4E38 with up to 7 significant digits.
double	8	Double-precision, floating-point numbers from -1.7E308 to 1.7E308 with up to 16 significant digits.
char	2	A single Unicode character that's stored in two bytes.
boolean	1	A <i>true</i> or <i>false</i> value.

Technical notes

- To express the value of a floating-point number, you can use *scientific notation* like 2.382E+5, which means 2.382 times 10^5 (a value of 238,200), or 3.25E-8, which means 3.25 times 10^{-8} (a value of .0000000325). Java will sometimes use this notation to display the value of a float or double data type.
- Because of the way floating-point numbers are stored internally, they can't represent the exact value of the decimal places in some numbers. This can cause a rounding problem in some business applications.

How to declare and initialize a variable in two statements

Syntax

```
type variableName;  
variableName = value;
```

Example

```
int counter;           // declaration statement  
counter = 1;           // assignment statement
```

How to declare and initialize a variable in one statement

Syntax

```
type variableName = value;
```

Examples

```
int counter = 1;
           // initialize an int variable
double price = 14.95;
           // initialize a double variable
float interestRate = 8.125F;
           // F indicates a floating-point value
long numberOfBytes = 20000L;
           // L indicates a long integer
int population = 1734323;
           // initialize an int variable
int population = 1_734_323;
           // underscores improve readability
double distance = 3.65e+9;
           // scientific notation
```

How to declare and initialize a variable in one statement (cont.)

Examples

```
char letter = 'A';  
           // stored as a two-digit Unicode character  
char letter = 65;  
           // integer value for a Unicode character  
boolean valid = false;  
           // where false is a keyword  
int x = 0, y = 0;  
           // initialize 2 variables with 1 statement
```

Naming conventions

- Start variable names with a lowercase letter and capitalize the first letter in all words after the first word.
- Try to use meaningful names that are easy to remember as you code.

Note

- The use of underscores in numeric literals was introduced in version 1.7 of the JDK.

How to declare and initialize a constant

Syntax

```
final type CONSTANT_NAME = value;
```

Examples

```
final int DAYS_IN_NOVEMBER = 30;  
final float SALES_TAX = .075F;  
final double LIGHT_YEAR_MILES = 5.879e+12
```

Naming conventions

- Capitalize all of the letters in constants and separate words with underscores.
- Try to use meaningful names that are easy to remember.

Arithmetic operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
+	Positive sign
-	Negative sign

Examples of simple assignment statements

```
int x = 14;
int y = 8;
int result1 = x + y;           // result1 = 22
int result2 = x - y;           // result2 = 6
int result3 = x * y;           // result3 = 112
int result4 = x / y;           // result4 = 1
int result5 = x % y;           // result5 = 6
int result6 = -y + x;          // result6 = 6
int result7 = --y;             // result7 = 7, y = 7
int result8 = ++x;             // result8 = 15, x = 15
```

```
double a = 8.5;
double b = 3.4;
double result9 = a + b;        // result9 = 11.9
double result10 = a - b;       // result10 = 5.1
double result11 = a * b;       // result11 = 28.90
double result12 = a / b;       // result12 = 2.5
double result13 = a % b;       // result13 = 1.7
double result14 = -a + b;      // result14 = -5.1
double result15 = --a;         // result15 = 7.5
double result16 = ++b;         // result16 = 4.4
```

Examples of simple assignment statements (cont.)

```
// character arithmetic
char letter1 = 'C';
           // letter1 = 'C'   Unicode integer is 67
char letter2 = ++letter1;
           // letter2 = 'D'   Unicode integer is 68
```


Assignment operators

Operator	Name
=	Assignment
+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
%=	Modulus

Statements that use the same variable on both sides of the equals sign

```
count = count + 1;      // count is increased by 1
count = count - 1;      // count is decreased by 1
total = total + 100.0;   // total is increased by 100.0
total = total - 100.0;   // total is decreased by 100.0
price = price * .8;      // price is multiplied by .8
sum = sum + nextNumber;  // sum is increased by value
                        // of nextNumber
```

Statements that use the shortcut operators to get the same results

```
count += 1;           // count is increased by 1
count -= 1;           // count is decreased by 1
total += 100.0;        // total is increased by 100.0
total -= 100.0;        // total is decreased by 100.0
price *= .8;           // price is multiplied by .8
sum += nextNumber;     // sum is increased by the value
                       // of nextNumber
```

The order of precedence for arithmetic operations

1. Increment and decrement
2. Positive and negative
3. Multiplication, division, and remainder
4. Addition and subtraction

Code that calculates a discounted price

Using the default order of precedence

```
double discountPercent = .2;           // 20% discount
double price = 100;                     // $100 price
price = price * 1 - discountPercent;    // price = $99.8
```

Using parentheses that specify the order of precedence

```
price = price * (1 - discountPercent);  // price = $80
```

Code that calculates the current value of a monthly investment

Using parentheses that specify the order of precedence

```
double currentValue = 5000;  
    // current value of investment account  
double monthlyInvestment = 100;  
    // amount added each month  
double yearlyInterestRate = .12;  
    // yearly interest rate  
  
currentValue = (currentValue + monthlyInvestment) *  
    (1 + (yearlyInterestRate/12));
```

Without using parentheses

```
currentValue += monthlyInvestment;    // add investment  
double monthlyInterestRate = yearlyInterestRate / 12;  
double monthlyInterest =  
    currentValue * monthlyInterestRate;  
currentValue += monthlyInterest;    // add interest
```

Prefix and postfix increment and decrement operators

```
int a = 5;  
int b = 5;  
int y = ++a;      // a = 6, y = 6  
int z = b++;      // b = 6, z = 5
```

How implicit casting works

Casting from less precise to more precise data types

Byte → short → int → long → float → double

Examples

```
double grade = 93;    // convert int to double
```

```
double d = 95.0;
```

```
int i = 86, j = 91;
```

```
double average = (d+i+j)/3;
```

```
                // convert i and j to double values
```

```
                // average = 90.666666...
```


How to code an explicit cast

Syntax

`(type) expression`

Examples

```
int grade = (int) 93.75;  
           // convert double to int (grade = 93)  
  
double d = 95.0;  
int i = 86, j = 91;  
double average = ((int)d+i+j)/3;  
                // convert d to int value (average = 90)  
  
double result = (double) i / (double) j;  
                // result has decimal places
```

How to cast between char and int types

```
char letterChar = 65;  
                // convert int to char (letterChar = 'A')  
char letterChar2 = (char) 65;  
                // this works too  
  
int letterInt = 'A';  
              // convert char to int (letterInt = 65)  
int letterInt2 = (int) 'A';  
              // this works too
```

The NumberFormat class

`java.text.NumberFormat`

Three static methods of the NumberFormat class

- `getCurrencyInstance()`
- `getPercentInstance()`
- `getNumberInstance()`

Three methods of a NumberFormat object

- `format(anyNumberType)`
- `setMinimumFractionDigits(int)`
- `setMaximumFractionDigits(int)`

The currency format

```
double price = 11.575;  
NumberFormat currency =  
NumberFormat.getCurrencyInstance();  
String priceString = currency.format(price);  
// returns $11.58
```

The percent format

```
double majority = .505;  
NumberFormat percent = NumberFormat.getPercentInstance();  
String majorityString = percent.format(majority);  
// returns 50%
```

The number format with one decimal place

```
double miles = 15341.253;  
NumberFormat number = NumberFormat.getInstance();  
number.setMaximumFractionDigits(1);  
String milesString = number.format(miles);  
// returns 15,341.3
```

Two NumberFormat methods coded in one statement

```
String majorityString =  
    NumberFormat.getPercentInstance().format(majority);
```

The Math class

`java.lang.Math`

Common static methods of the Math class

- `round(floatOrDouble)`
- `pow(number, power)`
- `sqrt(number)`
- `max(a, b)`
- `min(a, b)`
- `random()`

The round method

```
long result = Math.round(1.667);    // result is 2
int result = Math.round(1.49F);     // result is 1
```

The pow method

```
double result = Math.pow(2, 2);
                        // result is 4.0 (2*2)
double result = Math.pow(2, 3);
                        // result is 8.0 (2*2*2)
double result = Math.pow(5, 2);
                        // result is 25.0 (5 squared)
int result = (int) Math.pow(5, 2);
                        // result is 25 (5 squared)
```

The sqrt method

```
double result = Math.sqrt(20.25);    // result is 4.5
```

The max and min methods

```
int x = 67;  
int y = 23;  
int max = Math.max(x, y);           // max is 67  
int min = Math.min(x, y);           // min is 23
```

The random method

```
double x = Math.random() * 100;  
           // result is a value >= 0.0 and < 100.0  
long result = (long) x;  
           // converts the result from double to long
```


Constructors for the Integer and Double classes

- `Integer(int)`
- `Double(double)`

How to create Integer and Double objects

```
Integer quantityIntegerObject = new Integer(quantity);  
Double priceDoubleObject = new Double(price);
```

Two static methods of the Integer class

- `parseInt(stringName)`
- `toString(intName)`

Two static methods of the Double class

- `parseDouble(stringName)`
- `toString(doubleName)`

How to use static methods to convert primitive types to String objects

```
String counterString = Integer.toString(counter);  
String priceString = Double.toString(price);
```

How to use static methods to convert String objects to primitive types

```
int quantity = Integer.parseInt(quantityString);  
double price = Double.parseDouble(priceString);
```

The console for the formatted Invoice application

```
Enter subtotal:    150.50
Discount percent: 10%
Discount amount:   $15.05
Total before tax:  $135.45
Sales tax:         $6.77
Invoice total:     $142.22

Continue? (y/n):
```

The code for the formatted Invoice application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class InvoiceApp {
    public static void main(String[] args) {

        final double SALES_TAX_PCT = .05;

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get the input from the user
            System.out.print("Enter subtotal:    ");
            double subtotal = sc.nextDouble();

            // calculate the results
            double discountPercent = 0.0;
            if (subtotal >= 100)
                discountPercent = .1;
            else
                discountPercent = 0.0;
        }
    }
}
```

The code for the formatted Invoice application (cont.)

```
double discountAmount =
    subtotal * discountPercent;
double totalBeforeTax =
    subtotal - discountAmount;
double salesTax =
    totalBeforeTax * SALES_TAX_PCT;
double total = totalBeforeTax + salesTax;

// format and display the results
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
NumberFormat percent =
    NumberFormat.getPercentInstance();
```

The code for the formatted Invoice application (cont.)

```
String message =
    "Discount percent: "
    + percent.format(discountPercent) + "\n"
    + "Discount amount: "
    + currency.format(discountAmount) + "\n"
    + "Total before tax: "
    + currency.format(totalBeforeTax) + "\n"
    + "Sales tax: "
    + currency.format(salesTax) + "\n"
    + "Invoice total: "
    + currency.format(total) + "\n";
System.out.println(message);

// see if the user wants to continue
System.out.print("Continue? (y/n): ");
choice = sc.next();
System.out.println();
}
}
}
```

The console with an arithmetic bug

```
Enter subtotal:    100.05
Discount percent: 10%
Discount amount:   $10.00
Total before tax:  $90.04
Sales tax:         $4.50
Invoice total:     $94.55

Continue? (y/n) :
```

Debugging statements that can be added to the code

```
String debugMessage = "\nUNFORMATTED RESULTS\n"
    + "Discount percent: "
    + discountPercent + "\n"
    + "Discount amount:  "
    + discountAmount + "\n"
    + "Total before tax: "
    + totalBeforeTax + "\n"
    + "Sales tax:         "
    + salesTax + "\n"
    + "Invoice total:      " + total + "\n"
    + "\nFORMATTED RESULTS";

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


The console with debugging information

```
Enter subtotal:    100.05
```

UNFORMATTED RESULTS

```
Discount percent: 0.1
```

```
Discount amount:  10.005
```

```
Total before tax: 90.045
```

```
Sales tax:        4.50225
```

```
Invoice total:    94.54725
```

FORMATTED RESULTS

```
Discount percent: 10%
```

```
Discount amount:  $10.00
```

```
Total before tax: $90.04
```

```
Sales tax:        $4.50
```

```
Invoice total:    $94.55
```

```
Continue? (y/n) :
```

The BigDecimal class

`java.math.BigDecimal`

Constructors of the BigDecimal class

- `BigDecimal(int)`
- `BigDecimal(double)`
- `BigDecimal(long)`
- `BigDecimal(String)`

Methods of the `BigDecimal` class

- `add(value)`
- `compareTo(value)`
- `divide(value, scale, rounding-mode)`
- `multiply(value)`
- `setScale(scale, rounding-mode)`
- `subtract(value)`
- `toString()`

The RoundingMode enumeration

`java.math.RoundingMode`

Two of the values in the RoundingMode enumeration

- `HALF_UP`
- `HALF_EVEN`

The Invoice application output when BigDecimal arithmetic is used

```
Enter subtotal:    100.05
Subtotal:         $100.05
Discount percent: 10%
Discount amount:  $10.01
Total before tax: $90.04
Sales tax:        $4.50
Invoice total:    $94.54
```

```
Continue? (y/n) :
```

The import statement that's required for BigDecimal arithmetic

```
import java.math.*; // imports all classes and
                    // enumerations in java.math
```

The code for using BigDecimal arithmetic in the Invoice application

```
// convert subtotal and discount percent to BigDecimal
BigDecimal decimalSubtotal =
    new BigDecimal(Double.toString(subtotal));
decimalSubtotal =
    decimalSubtotal.setScale(2, RoundingMode.HALF_UP);
BigDecimal decimalDiscountPercent =
    new BigDecimal(Double.toString(discountPercent));

// calculate discount amount
BigDecimal discountAmount =
    decimalSubtotal.multiply(decimalDiscountPercent);
discountAmount = discountAmount.setScale(
    2, RoundingMode.HALF_UP);
```

The code for using BigDecimal arithmetic in the Invoice application (cont.)

```
// calculate total before tax, sales tax, and total
BigDecimal totalBeforeTax =
    decimalSubtotal.subtract(discountAmount);
BigDecimal salesTaxPercent =
    new BigDecimal(SALES_TAX_PCT);
BigDecimal salesTax =
    salesTaxPercent.multiply(totalBeforeTax);
salesTax = salesTax.setScale(2, RoundingMode.HALF_UP);
BigDecimal total = totalBeforeTax.add(salesTax);
```

How to create a BigDecimal object from another BigDecimal object

```
BigDecimal total2 = new BigDecimal(total.toString());
```