Chapter 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
▪ To learn how to use the Java String type
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

Syntax: `typeName variableName = value;`

or

`typeName variableName;`

Types introduced in this chapter are the number types `int` and `double` (see Table 2) and the String type (see Section 2.5).

See Table 3 for rules and examples of valid names.

A variable declaration ends with a semicolon.

Supplying an initial value is optional, but it is usually a good idea.

Use a descriptive variable name. See Programming Tip 2.1.

See Common Error 2.1.
An Example: Soda Deal

- Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (12 fluid ounces equal approximately 0.355 liters.)

- **List of variables:**
  - Type of number:
    - Number of cans per pack: Whole number
    - Ounces per can: Whole number
    - Ounces per bottle: Number with fraction
Variables and Contents

▪ Each variable has an identifier (name) and contents
▪ You can (optionally) set the contents of a variable when you declare it

    int cansPerPack = 6;

▪ Imagine a parking space in a parking garage
  ▪ Identifier: J053
  ▪ Contents: Bob’s Chevy
### Table 1  Variable Declarations in Java

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>int cans = 6;</td>
<td>Declares an integer variable and initializes it with 6.</td>
</tr>
<tr>
<td>int total = cans + bottles;</td>
<td>The initial value need not be a fixed value. (Of course, cans and bottles must have been previously declared.)</td>
</tr>
<tr>
<td>![error] int bottles = 1;</td>
<td><strong>Error:</strong> 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.</td>
</tr>
<tr>
<td>![error] int volume = &quot;2&quot;;</td>
<td><strong>Error:</strong> You cannot initialize a number with a string.</td>
</tr>
<tr>
<td>int cansPerPack;</td>
<td>Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.</td>
</tr>
<tr>
<td>int dollars, cents;</td>
<td>Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.</td>
</tr>
</tbody>
</table>
Why Different Types?

- There are three different types of variables that we will use in this chapter:
  1) A whole number (no fractional part) \( \text{int} \)
  2) A number with a fraction part \( \text{double} \)
  3) A word (a group of characters) \( \text{String} \)

- Specify the type before the name in the declaration

```java
int cansPerPack = 6;
double canVolume = 12.0;
```
Why Different Variables?

- Back to the garage analogy, parking spaces may be different sizes for different types of vehicles
  - Bicycle
  - Motorcycle
  - Full Size
  - Electric Vehicle
Number Literals in Java

- Sometimes when you just type a number, the compiler has to ‘guess’ what type it is

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

- Java stores numbers with fractional parts as ‘floating point’ numbers.
- They are stored in four parts
  - Sign
  - Mantissa
  - Radix
  - Exponent
- A ‘**double**’ is a double-precision floating point number: It takes twice the storage (52 bit mantissa) as the smaller ‘**float**’ (23 bit mantissa)

<table>
<thead>
<tr>
<th>Parts of a floating point number -5:</th>
<th>Sign</th>
<th>Mantissa</th>
<th>Radix $^{\text{exponent}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>5</td>
<td>10 $^0$</td>
</tr>
</tbody>
</table>
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 (see Appendix C)
### Table 3 Variable Names in Java

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>canVolume1</td>
<td>Variable names consist of letters, numbers, and the underscore character.</td>
</tr>
<tr>
<td>x</td>
<td>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).</td>
</tr>
<tr>
<td>! CanVolume</td>
<td><strong>Caution:</strong> Variable names are case sensitive. This variable name is different from <code>canVolume</code>, and it violates the convention that variable names should start with a lowercase letter.</td>
</tr>
<tr>
<td>❌ 6pack</td>
<td><strong>Error:</strong> Variable names cannot start with a number.</td>
</tr>
<tr>
<td>❌ can volume</td>
<td><strong>Error:</strong> Variable names cannot contain spaces.</td>
</tr>
<tr>
<td>❌ double</td>
<td><strong>Error:</strong> You cannot use a reserved word as a variable name.</td>
</tr>
<tr>
<td>❌ ltr/fl.oz</td>
<td><strong>Error:</strong> You cannot use symbols such as <code>/</code> or.</td>
</tr>
</tbody>
</table>
The Assignment Statement

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

```java
int cansPerPack = 6;   // declare & initialize
```

```java
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 in the next chapter
Assignment Syntax

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

```java
double total = 0;
.
.
total = bottles * BOTTLE_VOLUME;
.
.
total = total + cans * CAN_VOLUME;
```
Updating a Variable

**Step by Step:**

```plaintext
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)
Declarations vs. Assignments

- Variable declarations and an assignment statements are different
  ```
  int cansPerPack = 6;  Declaration
  ...  
  cansPerPack = 8; Assignment statement
  ```
- Declarations define a new variable and can give it an initial value
- Assignments modify the value of an existing variable
Constants

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

```java
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?

```java
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.

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

- 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 at the beginning of each program, and to clarify details of the code
- Use comments to add explanations for humans who read your code
- The compiler ignores comments
Java Comment Example

```java
/**
 * This program computes the volume (in liters) of a six-pack of soda cans and the total volume of a six-pack and a two-liter bottle.
 */

public class Volume1 {

    public static void main(String[] args) {
        int cansPerPack = 6;
        final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
        double totalVolume = cansPerPack * CAN_VOLUME;

        System.out.print("A six-pack of 12-ounce cans contains ");
        System.out.print(totalVolume);
        System.out.println(" liters.");

        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
Self Check 2.1

Declare a variable suitable for holding the number of bottles in a case.

**Answer:** One possible answer is

```java
int bottlesPerCase = 8;
```

You may choose a different variable name or a different initialization value, but your variable should have type int.
What is wrong with the following variable declaration?

```c
int ounces per liter = 28.35
```

**Answer:** There are three errors:

- You cannot have spaces in variable names.
- The variable type should be double because it holds a fractional value.
- There is a semicolon missing at the end of the statement.
Declare and initialize two variables, `unitPrice` and `quantity`, to contain the unit price of a single bottle and the number of bottles purchased. Use reasonable initial values.

**Answer:**
```java
double unitPrice = 1.95;
int quantity = 2;
```
Self Check 2.4

Use the variables declared in Self Check 3 to display the total purchase price.

Answer:

```java
System.out.print("Total price: ");
System.out.println(unitPrice * quantity);
```
Self Check 2.5

Some drinks are sold in four-packs instead of six-packs. How would you change the Volume1.java program to compute the total volume?

**Answer:** Change the declaration of `cansPerPack` to

```java
int cansPerPack = 4;
```
Self Check 2.6

What is wrong with this comment?

double canVolume = 0.355; /* Liters in a 12-ounce can //

**Answer:** You need to use a */ delimiter to close a comment that begins with a /*:

double canVolume = 0.355;
/* Liters in a 12-ounce can */
Suppose the type of the `cansPerPack` variable in `Volume1.java` was changed from `int` to `double`. What would be the effect on the program?

**Answer:** The program would compile, and it would display the same result. However, a person reading the program might find it confusing that fractional cans are being considered.
Self Check 2.8

Why can’t the variable `totalVolume` in the `Volume1.java` program be declared as `final`?

**Answer:** Its value is modified by the assignment statement.
Self Check 2.9

How would you explain assignment using the parking space analogy?

**Answer:** Assignment would occur when one car is replaced by another in the parking space.
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;  // ??
Overflow means that storage for a variable cannot hold the result

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

- Will print out 705032704
- 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)
Common Error 2.3

- Roundoff Errors
  - Floating point values are not exact
    - This is a limitation of binary values (no fractions):
      ```java
double price = 4.35;
double quantity = 100;
double total = price * quantity;
// Should be 100 * 4.35 = 435.00
System.out.println(total); // Prints 434.99999999999999
```
- You can deal with roundoff errors by rounding to the nearest integer (see Section 2.2.5) or by displaying a fixed number of digits after the decimal separator (see Section 2.3.2).
# All of the Java Numeric Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>The integer type, with range –2,147,483,648 (Integer.MIN_VALUE) . . . 2,147,483,647 (Integer.MAX_VALUE, about 2.14 billion)</td>
</tr>
<tr>
<td>byte</td>
<td>The type describing a byte consisting of 8 bits, with range –128 . . . 127</td>
</tr>
<tr>
<td>short</td>
<td>The short integer type, with range –32,768 . . . 32,767</td>
</tr>
<tr>
<td>long</td>
<td>The long integer type, with about 19 decimal digits</td>
</tr>
<tr>
<td>double</td>
<td>The double-precision floating-point type, with about 15 decimal digits and a range of about ±10^{308}</td>
</tr>
<tr>
<td>float</td>
<td>The single-precision floating-point type, with about 7 decimal digits and a range of about ±10^{38}</td>
</tr>
<tr>
<td>char</td>
<td>The character type, representing code units in the Unicode encoding scheme (see Section 2.6.6)</td>
</tr>
</tbody>
</table>

- Each type has a range of values that it can hold
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’
### Storage per Type (in bytes)

#### Integer Types
- **byte:**
- **short:**
- **int:**
- **long:**

#### Floating Point Types
- **float:**
- **double:**

#### Other Types
- **boolean:**
- **char:**
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

\[
\frac{a + b}{2} \quad (a + b) / 2
\]

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:

```c
double area, pi = 3.14;
int radius = 3;
area = radius * radius * pi;
```
Incrementing a Variable

Step by Step:

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)

```java
counter = counter + 1;
```
Shorthand for Incrementing

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

<table>
<thead>
<tr>
<th>Long Way</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter = counter + 1;</td>
<td>counter++ ;</td>
</tr>
<tr>
<td>counter = counter - 1;</td>
<td>counter--;</td>
</tr>
</tbody>
</table>
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

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

\[ b \times \left(1 + \frac{r}{100}\right)^n \]

Becomes:

\[ b \times Math.pow(1 + r / 100, n) \]

- Analyzing the expression:

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sqrt(x)</td>
<td>Square root of $x$ ($\geq 0$)</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td>$x^y$ ($x &gt; 0$, or $x = 0$ and $y &gt; 0$, or $x &lt; 0$ and $y$ is an integer)</td>
</tr>
<tr>
<td>Math.sin(x)</td>
<td>Sine of $x$ ($x$ in radians)</td>
</tr>
<tr>
<td>Math.cos(x)</td>
<td>Cosine of $x$</td>
</tr>
<tr>
<td>Math.tan(x)</td>
<td>Tangent of $x$</td>
</tr>
<tr>
<td>Math.toRadians(x)</td>
<td>Convert $x$ degrees to radians (i.e., returns $x \cdot \pi/180$)</td>
</tr>
<tr>
<td>Math.toDegrees(x)</td>
<td>Convert $x$ radians to degrees (i.e., returns $x \cdot 180/\pi$)</td>
</tr>
<tr>
<td>Math.exp(x)</td>
<td>$e^x$</td>
</tr>
<tr>
<td>Math.log(x)</td>
<td>Natural log ($\ln(x)$, $x &gt; 0$)</td>
</tr>
<tr>
<td>Math.log10(x)</td>
<td>Decimal log ($\log_{10}(x)$, $x &gt; 0$)</td>
</tr>
<tr>
<td>Math.round(x)</td>
<td>Closest integer to $x$ (as a long)</td>
</tr>
<tr>
<td>Math.abs(x)</td>
<td>Absolute value $</td>
</tr>
<tr>
<td>Math.max(x, y)</td>
<td>The larger of $x$ and $y$</td>
</tr>
<tr>
<td>Math.min(x, y)</td>
<td>The smaller of $x$ and $y$</td>
</tr>
</tbody>
</table>
Floating-Point to Integer Conversion

- The Java compiler does not allow direct assignment of a floating-point value to an integer variable:
  ```java
  double balance = total + tax;
  int dollars = balance; // Error
  ```
- You can use the ‘cast’ operator: `(int)` to force the conversion:
  ```java
  double balance = total + tax;
  int dollars = (int) balance; // no Error
  ```
- You lose the fractional part of the floating-point value (no rounding occurs)
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`.

```java
long rounded = Math.round(balance);
```
## Arithmetic Expressions

<table>
<thead>
<tr>
<th>Mathematical Expression</th>
<th>Java Expression</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{x + y}{2} )</td>
<td>((x + y) / 2)</td>
<td>The parentheses are required; ( x + y / 2 ) computes ( x + \frac{y}{2} ).</td>
</tr>
<tr>
<td>( \frac{xy}{2} )</td>
<td>(x * y / 2)</td>
<td>Parentheses are not required; operators with the same precedence are evaluated left to right.</td>
</tr>
<tr>
<td>( \left(1 + \frac{r}{100}\right)^n )</td>
<td>(\text{Math.pow}(1 + r / 100, n))</td>
<td>Use (\text{Math.pow}(x, n)) to compute (x^n).</td>
</tr>
<tr>
<td>( \sqrt{a^2 + b^2} )</td>
<td>(\text{Math.sqrt}(a * a + b * b))</td>
<td>(a * a) is simpler than (\text{Math.pow}(a, 2)).</td>
</tr>
<tr>
<td>( \frac{i + j + k}{3} )</td>
<td>((i + j + k) / 3.0)</td>
<td>If (i, j,) and (k) are integers, using a denominator of 3.0 forces floating-point division.</td>
</tr>
<tr>
<td>( \pi )</td>
<td>(\text{Math.PI})</td>
<td>(\text{Math.PI}) is a constant declared in the \text{Math} class.</td>
</tr>
</tbody>
</table>
Self Check 2.10

A bank account earns interest once per year. In Java, how do you compute the interest earned in the first year? Assume variables `percent` and `balance` of type `double` have already been declared.

**Answer:**
```java
double interest = balance * percent / 100;
```
Self Check 2.11

In Java, how do you compute the side length of a square whose area is stored in the variable area?

**Answer**: `double sideLength = Math.sqrt(area);`
Self Check 2.12

The volume of a sphere is given by

\[ V = \frac{4}{3} \pi r^3 \]

If the radius is given by a variable `radius` of type `double`, write a Java expression for the volume.

**Answer:** `4 * Math.PI * Math.pow(radius, 3) / 3` or `(4.0 / 3) * Math.PI * Math.pow(radius, 3)`, but not `(4 / 3) * Math.PI * Math.pow(radius, 3)`
Self Check 2.13

What is the value of \( \frac{1729}{10} \) and \( 1729 \mod 10 \)?

**Answer:** 172 and 9
Self Check 2.14

If \( n \) is a positive integer, what is \( \frac{n}{10} \mod 10 \)?

**Answer:** It is the second-to-last digit of \( n \). For example, if \( n \) is 1729, then \( \frac{n}{10} \) is 172, and \( \frac{n}{10} \mod 10 \) is 2.
Common Error 2.4

- 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
Common Error 2.5

- Unbalanced Parenthesis

  - Which is correct?
    \[-(b \times b - 4 \times a \times c) / (2 \times a) \]

    \[-(b \times b - (4 \times a \times c)) / (2 \times a) \]

- The count of (and) must match

- Unfortunately, it is hard for humans to keep track

  - Here's a handy trick

  - Count (as +1, and) as -1: Goal: 0

    \[-(b \times b - (4 \times a \times c) ) \) / (2 \times a) \]

    1 2 1 0 -1 -2
Input and Output

- **Reading Input**
  - 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
    - For now, don’t worry about the details
  - This is a three step process in Java
    1. **Import the Scanner class from its ‘package’**
       ```java
       java.util import java.util.Scanner;
       ```
    2. **Setup an object of the Scanner class**
       ```java
       Scanner in = new Scanner(System.in);
       ```
    3. **Use methods of the new Scanner object to get input**
       ```java
       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

```
import java.util.Scanner;

Scanner in = new Scanner(System.in);

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

Include this line so you can use the `Scanner` class.
Create a `Scanner` object to read keyboard input.
Display a prompt in the console window.
Define a variable to hold the input value.

The program waits for user input, then places the input into the variable.
Outputting floating point values can look strange:

Price per liter: 1.21997

To control the output appearance of numeric variables, use formatted output tools such as:

System.out.printf("%.2f", price);
Price per liter: 1.22
System.out.printf("%10.2f", price);
Price per liter: 1.22

The %10.2f is called a format specifier
### Format Types

- Formatting is handy to align columns of output

#### Table 8 Format Specifier Examples

<table>
<thead>
<tr>
<th>Format String</th>
<th>Sample Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%d&quot;</td>
<td>24</td>
<td>Use d with an integer.</td>
</tr>
<tr>
<td>&quot;%5d&quot;</td>
<td>24</td>
<td>Spaces are added so that the field width is 5.</td>
</tr>
<tr>
<td>&quot;Quantity:%5d&quot;</td>
<td>Quantity: 24</td>
<td>Characters inside a format string but outside a format specifier appear in the output.</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>1.21997</td>
<td>Use f with a floating-point number.</td>
</tr>
<tr>
<td>&quot;%.2f&quot;</td>
<td>1.22</td>
<td>Prints two digits after the decimal point.</td>
</tr>
<tr>
<td>&quot;%7.2f&quot;</td>
<td>1.22</td>
<td>Spaces are added so that the field width is 7.</td>
</tr>
<tr>
<td>&quot;%s&quot;</td>
<td>Hello</td>
<td>Use s with a string.</td>
</tr>
<tr>
<td>&quot;%d %.2f&quot;</td>
<td>24 1.22</td>
<td>You can format multiple values at once.</td>
</tr>
<tr>
<td>&quot;Hello%nWorld%n&quot;</td>
<td>Hello World</td>
<td>Each %n causes subsequent output to continue on a new line.</td>
</tr>
</tbody>
</table>

- You can also include text inside the quotes:

```
System.out.printf("Price per liter: %10.2f", price);
```
Formatted Output Examples

▪ Left Justify a String:
  System.out.printf("%-10s", "Total:");

▪ Right justify a number with two decimal places
  System.out.printf("%10.2f", price);

▪ And you can print multiple values
  System.out.printf("%-10s%10.2f", "Total:", price);
import java.util.Scanner;

/**
 * This program prints the price per ounce for a six-pack of cans.
 */
public class Volume2 {
    public static void main(String[] args) {
        // Read price per pack
        final double CANS_PER_PACK = 6;
        double packVolume = canVolume * CANS_PER_PACK;

        // Compute and print price per ounce
        double pricePerOunce = packPrice / packVolume;
        System.out.printf("Price per ounce: %8.2f", pricePerOunce);
        System.out.println();
    }
}
Self Check 2.15

Write statements to prompt for and read the user's age using a Scanner variable named `in`.

Answer:
```java
System.out.print("How old are you? ");
int age = in.nextInt();
```
What is wrong with the following statement sequence?

```java
System.out.print("Please enter the unit price: ");
double unitPrice = in.nextDouble();
int quantity = in.nextInt();
```

**Answer:** There is no prompt that alerts the program user to enter the quantity.
What is problematic about the following statement sequence?

```
System.out.print("Please enter the unit price: ");
double unitPrice = in.nextInt();
```

**Answer:** The second statement calls `nextInt`, not `nextDouble`. If the user were to enter a price such as 1.95, the program would be terminated with an “input mismatch exception”.
What is problematic about the following statement sequence?

```java
    System.out.print("Please enter the number of cans");
    int cans = in.nextInt();
```

**Answer:** There is no colon and space at the end of the prompt. A dialog would look like this:

Please enter the number of cans
What is the output of the following statement sequence?

```java
int volume = 10;
System.out.printf("The volume is %5d", volume);
```

**Answer:**

The total volume is 10

There are four spaces between `is` and 10. One space originates from the format string (the space between `s` and `%`), and three spaces are added before 10 to achieve a field width of 5.
Using the printf method, print the values of the integer variables bottles and cans so that the output looks like this:

    Bottles: 8
    Cans: 24

The numbers to the right should line up. (You may assume that the numbers have at most 8 digits.)

**Answer:** Here is a simple solution:

    System.out.printf("Bottles: %8d\n", bottles);
    System.out.printf("Cans: %8d\n", cans);

Note the spaces after Cans:. Alternatively, you can use format specifiers for the strings. You can even combine all output into a single statement:

    System.out.printf("%-9s%8d\n%-9s%8d", "Bottles: ", bottles, "Cans:", cans);
Tip 2.2 Java API Documentation

- Lists the classes and methods of the Java API
  - On the web at: http://download.oracle.com/javase/6/docs/api
A very important step for developing an algorithm is to first carry out the computations *by hand*. Example Problem:

- A row of black and white tiles needs to be placed along a wall. For aesthetic reasons, the architect has specified that the first and last tile shall be black.
- Your task is to compute the number of tiles needed and the gap at each end, given the space available and the width of each tile.
Start with Example Values

- **Givens:**
  - Total width: 100 inches
  - Tile width: 5 inches

- **Test your values**
  - Let’s see…100/5 = 20, perfect! 20 tiles. No gap.
  - But wait… BW…BW “…first and last tile shall be black.”

- **Look more carefully at the problem…**
  - Start with one black, then some number of WB pairs

  ![Diagram](image)

  - **Observation:** each pair is 2x width of 1 tile
    - In our example, 2 x 5 = 10 inches
Start with Example Values

Total width: 100 inches
Tile width: 5 inches

- Calculate total width of all tiles
  - One black tile: 5”
  - 9 pairs of BWs: 90”
  - Total tile width: 95”

- Calculate gaps (one on each end)
  - 100 – 95 = 5” total gap
  - 5” gap / 2 = 2.5” at each end
Now Devise an Algorithm

▪ Use your example to see how you calculated values
▪ How many pairs?
  ▪ Note: must be a whole number
    \[
    \text{Integer part of:} \quad \frac{(\text{total width} - \text{tile width})}{2} \times \text{tile width}
    \]
▪ How many tiles?
  \[\text{1} + 2 \times \text{the number of pairs}\]
▪ Gap at each end
  \[
  \frac{(\text{total width} - \text{number of tiles} \times \text{tile width})}{2}
  \]
Translate the pseudocode for computing the number of tiles and the gap width into Java.

**Answer:**

```java
int pairs = (totalWidth - tileWidth) / (2 * tileWidth);
int tiles = 1 + 2 * pairs;
double gap = (totalWidth - tiles * tileWidth) / 2.0;
```

Be sure that `pairs` is declared as an `int`. 
Self Check 2.22

Suppose the architect specifies a pattern with black, gray, and white tiles, like this:

```
[Black] [Gray] [White] [Gray] [Black]
```

Again, the first and last tile should be black. How do you need to modify the algorithm?

**Answer:** Now there are groups of four tiles (gray/white/gray/black) following the initial black tile. Therefore, the algorithm is now

```
number of groups = integer part of (total width - tile width) / (4 x tile width)
```

```
number of tiles = 1 + 4 x number of groups
```

The formula for the gap is not changed.
A robot needs to tile a floor with alternating black and white tiles. Develop an algorithm that yields the color (0 for black, 1 for white), given the row and column number. Start with specific values for the row and column, and then generalize.

**Answer:** Clearly, the answer depends only on whether the row and column numbers are even or odd, so let’s first take the remainder after dividing by 2. Then we can enumerate all expected answers:

<table>
<thead>
<tr>
<th>Row %2</th>
<th>Column %2</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In the first three entries of the table, the color is simply the sum of the remainders. In the fourth entry, the sum would be 2, but we want a zero. We can achieve that by taking another remainder operation:

\[ \text{color} = ((\text{row} \mod 2) + (\text{column} \mod 2)) \mod 2 \]
Self Check 2.24

For a particular car, repair and maintenance costs in year 1 are estimated at $100; in year 10, at $1,500. Assuming that the repair cost increases by the same amount every year, develop pseudocode to compute the repair cost in year 3 and then generalize to year $n$.

**Answer:** In nine years, the repair costs increased by $1,400. Therefore, the increase per year is $1,400 / 9 ≈ $156. The repair cost in year 3 would be $100 + 2 \times $156 = $412. The repair cost in year $n$ is $100 + n \times $156. To avoid accumulation of roundoff errors, it is actually a good idea to use the original expression that yielded $156, that is,

\[
\text{Repair cost in year } n = 100 + n \times 1400 / 9
\]
The shape of a bottle is approximated by two cylinders of radius $r_1$ and $r_2$ and heights $h_1$ and $h_2$, joined by a cone section of height $h_3$. Using the formulas for the volume of a cylinder, $V = \pi r^2 h$, and a cone section,

$$V = \pi \frac{\left(r_1^2 + r_1 r_2 + r_2^2\right) h}{3},$$

develop pseudocode to compute the volume of the bottle. Using an actual bottle with known volume as a sample, make a hand calculation of your pseudocode.

**Answer:** The pseudocode follows easily from the equations:

- bottom volume = $\pi \times r_1^2 \times h_1$
- top volume = $\pi \times r_2^2 \times h_2$
- middle volume = $\pi \times (r_1^2 + r_1 \times r_2 + r_2^2) \times h_3 / 3$
- total volume = bottom volume + top volume + middle volume

Measuring a typical wine bottle yields $r_1 = 3.6$, $r_2 = 1.2$, $h_1 = 15$, $h_2 = 7$, $h_3 = 6$ (all in centimeters). Therefore,

- bottom volume = 610.73
- top volume = 31.67
- middle volume = 135.72
- total volume = 778.12

The actual volume is 750 ml, which is close enough to our computation to give confidence that it is correct.
**Strings**

- **The String Type:**
  - Type Variable Literal
  - String name = “Harry”

- Once you have a String variable, you can use methods such as:
  ```java
  int n = name.length(); // n will be assigned 5
  ```

- A String’s length is the number of characters inside:
  - An empty String (length 0) is shown as “”
  - The maximum length is quite large (an int)
### String Concatenation (+)

- You can ‘add’ one `String` onto the end of another
  ```java
  String fName = "Harry"
  String lName = "Morgan"
  String name = fname + lname; // HarryMorgan
  ```

- You wanted a space in between?
  ```java
  String name = fname + " " + lname; // Harry Morgan
  ```

- To concatenate a numeric variable to a `String`:
  ```java
  String a = "Agent";
  int n = 7;
  String bond = a + n; // Agent7
  ```

- **Concatenate** `Strings` and `numerics` inside `println`:
  ```java
  System.out.println("The total is " + total);
  ```
String Input

- You can read a String from the console with:
  ```java
  System.out.print("Please enter your name: ");
  String name = in.next();
  ```
  - The next method reads one word at a time
  - It looks for ‘white space’ delimiters

- You can read an entire line from the console with:
  ```java
  System.out.print("Please enter your address: ");
  String address = in.nextLine();
  ```
  - The nextLine method reads until the user hits ‘Enter’

- Converting a String variable to a number:
  ```java
  System.out.print("Please enter your age: ");
  String input = in.nextLine();
  int age = Integer.parseInt(input);  // only digits!
  ```
String Escape Sequences

▪ How would you print a double quote?
  ▪ Preface the " with a \ inside the double quoted String
    System.out.print("He said "Hello"");

▪ OK, then how do you print a backslash?
  ▪ Preface the \ with another \!
    System.out.print(""C:\Temp\Secret.txt");

▪ Special characters inside Strings
  ▪ Output a newline with a ‘\n’
    System.out.print("*\n**\n***\n");
Strings and Characters

- Strings are sequences of characters
  - Unicode characters to be exact
  - Characters have their own type: `char`
  - Characters have numeric values
    - See the ASCII code chart in Appendix B
    - For example, the letter ‘H’ has a value of 72 if it were a number

- Use single quotes around a `char`
  ```java
  char initial = ‘B’;
  ```

- Use double quotes around a `String`
  ```java
  String initials = "BRL";
  ```
Copying a char from a String

- A substring is a portion of a String.
- The `substring` method returns a portion of a String at a given index for a number of chars, starting at an index:

```
String greeting = "Hello!";
String sub = greeting.substring(0, 2);
String sub2 = greeting.substring(3, 5);
```
Copying a Portion of a String

- Each char inside a String has an index number:
- The first char is index zero (0)
- The charAt method returns a char at a given index inside a String:

```
String greeting = "Harry";
char start = greeting.charAt(0);
char last = greeting.charAt(4);
```
### Table 9 String Operations

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>string str = &quot;Ja&quot;; str = str + &quot;va&quot;;</td>
<td>str is set to &quot;Java&quot;</td>
<td>When applied to strings, + denotes concatenation.</td>
</tr>
<tr>
<td>System.out.println(&quot;Please&quot; + &quot; enter your name: &quot;);</td>
<td>Prints Please enter your name:</td>
<td>Use concatenation to break up strings that don’t fit into one line.</td>
</tr>
<tr>
<td>team = 49 + &quot;ers&quot;</td>
<td>team is set to &quot;49ers&quot;</td>
<td>Because &quot;ers&quot; is a string, 49 is converted to a string.</td>
</tr>
<tr>
<td>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</td>
<td>first contains &quot;Harry&quot; last contains &quot;Morgan&quot;</td>
<td>The next method places the next word into the string variable.</td>
</tr>
<tr>
<td>String greeting = &quot;H &amp; S&quot;; int n = greeting.length();</td>
<td>n is set to 5</td>
<td>Each space counts as one character.</td>
</tr>
<tr>
<td>String str = &quot;Sally&quot;; char ch = str.charAt(1);</td>
<td>ch is set to 'a'</td>
<td>This is a char value, not a String. Note that the initial position is 0.</td>
</tr>
<tr>
<td>Statement</td>
<td>Result</td>
<td>Comment</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| String str = "Sally";  
String str2 = str.substring(1, 4); | str2 is set to "all" | Extracts the substring starting at position 1 and ending before position 4. |
| String str = "Sally";  
String str2 = str.substring(1); | str2 is set to "ally" | If you omit the end position, all characters from the position until the end of the string are included. |
| String str = "Sally";  
String str2 = str.substring(1, 2); | str2 is set to "a" | Extracts a String of length 1; contrast with str.charAt(1). |
| String last = str.substring( str.length() - 1); | last is set to the string containing the last character in str | The last character has position str.length() - 1. |
What is the length of the string "Java Program"?

**Answer:** The length is 12. The space counts as a character.
Consider this string variable.
String str = "Java Program";
Give a call to the substring method that returns the substring "gram".

**Answer:** str.substring(8, 12) or str.substring(8)
Self Check 2.28

Use string concatenation to turn the string variable \texttt{str} from Self Check 27 into "Java Programming".

\textbf{Answer:} \texttt{str = str + "ming";}
What does the following statement sequence print?

```java
String str = "Harry";
int n = str.length();
String mystery = str.substring(0, 1) + str.substring(n - 1, n);
System.out.println(mystery);
```

**Answer:** Hy
Self Check 2.30

Give an input statement sequence to read a name of the form “John Q. Public”.

**Answer:**

```java
String first = in.next();
String middle = in.next();
String last = in.next();
```