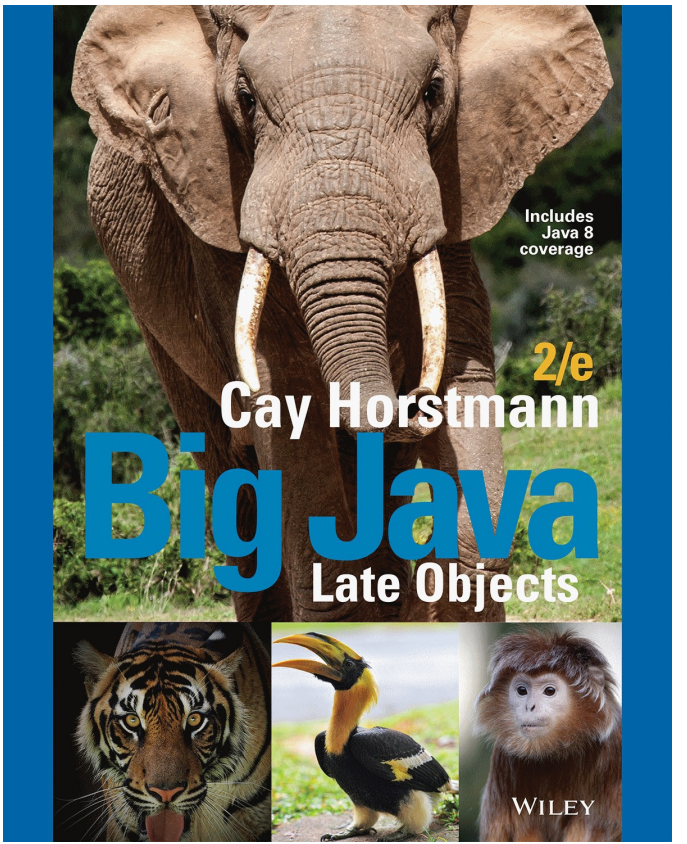
Chapter 2 - Fundamental Data Types



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

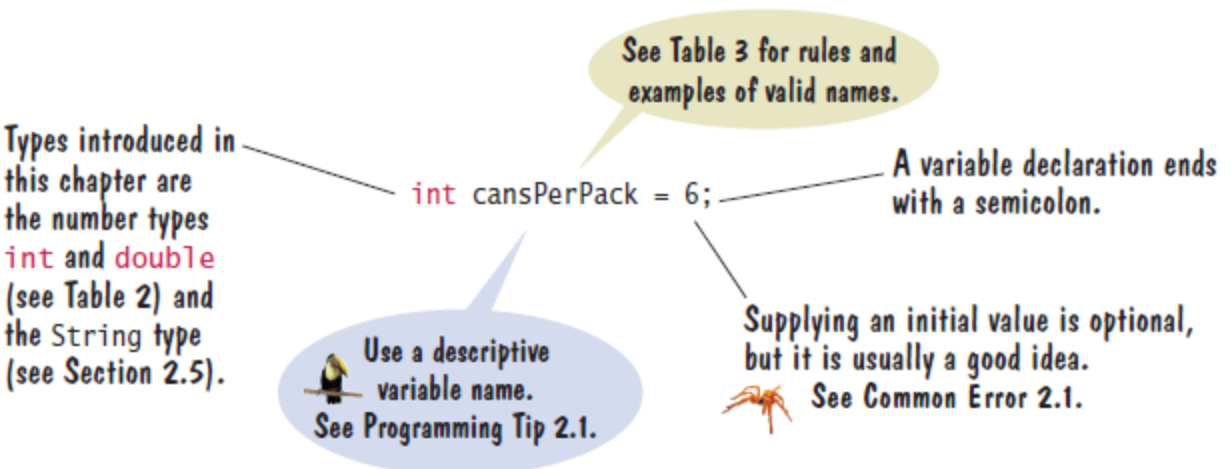
- Most computer programs hold temporary values in named storage locations
  - Programmers name them for easy access
- There are many different types (sizes) of storage to hold different things
- You 'declare' a variable by telling the compiler:
  - What type (size) of variable you need
  - What name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



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

Example Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
`int cans = 6;`	Declares an integer variable and initializes it with 6.
`int total = cans + bottles;`	The initial value need not be a fixed value. (Of course, `cans` and `bottles` must have been previously declared.)
`bottles = 1;`	**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.
`int volume = "2";`	**Error:** You cannot initialize a number with a string.
`int cansPerPack;`	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
`int dollars, cents;`	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) `int`

  2) A number with a fraction part `double`

  3) A word (a group of characters) `String`

- Specify the type before the name in the declaration

```
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

```
amt = 6 * 12.0;
PI = 3.14;
canVol = 0.335;
```

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 (see Appendix C)

Variable Names in Java

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.
⛔ `ltr/fl.oz`	**Error:** You cannot use symbols such as / or `.`

The Assignment Statement

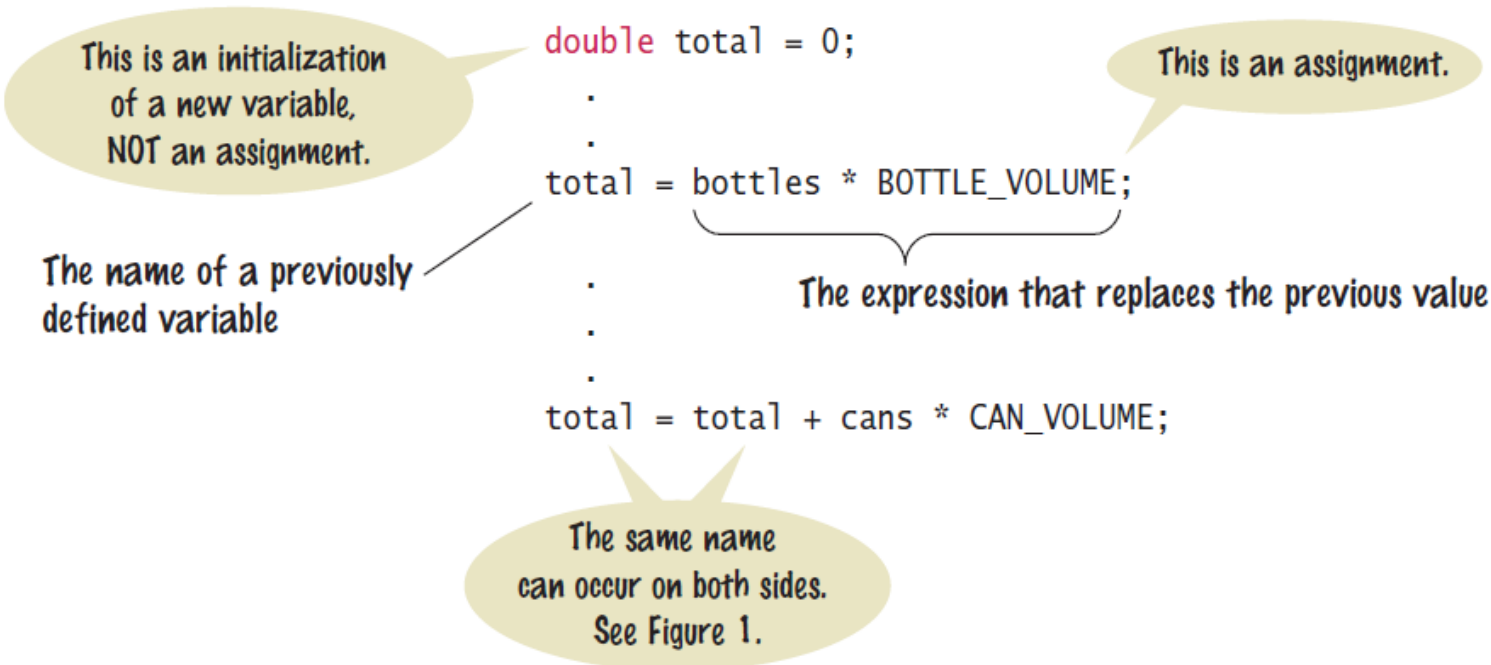
- Use the 'assignment statement' (with an '=') to place a new value into a variable

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

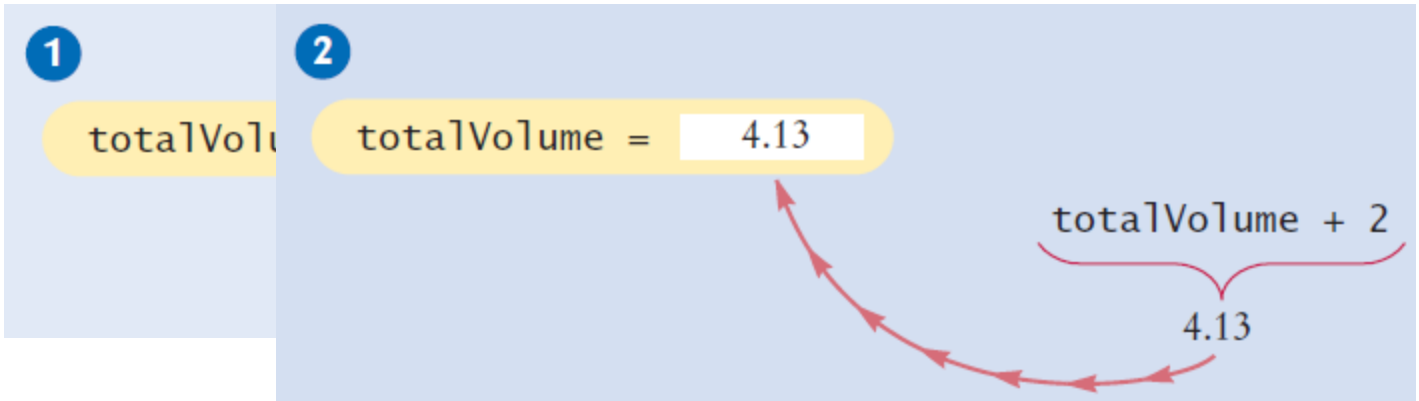
- Beware: The **=** sign is **NOT** used for comparison:
  - It copies the value on the right side into the variable on the left side
  - You will learn about the comparison operator in the next chapter

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)

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

```
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 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

```
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 2.2

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

```
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`)

All of the Java Numeric Types

Type	Description	
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)	Whole Numbers (no fractions)
`byte`	The type describing a byte consisting of 8 bits, with range –128 . . . 127	
`short`	The short integer type, with range –32,768 . . . 32,767	
`long`	The long integer type, with about 19 decimal digits	
`double`	The double-precision floating-point type, with about 15 decimal digits and a range of about $\pm 10^{308}$	Floating point Numbers
`float`	The single-precision floating-point type, with about 7 decimal digits and a range of about $\pm 10^{38}$	
`char`	The character type, representing code units in the Unicode encoding scheme (see Section 2.6.6)	Characters (no math)

- Each type has a range of values that it can hold

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`

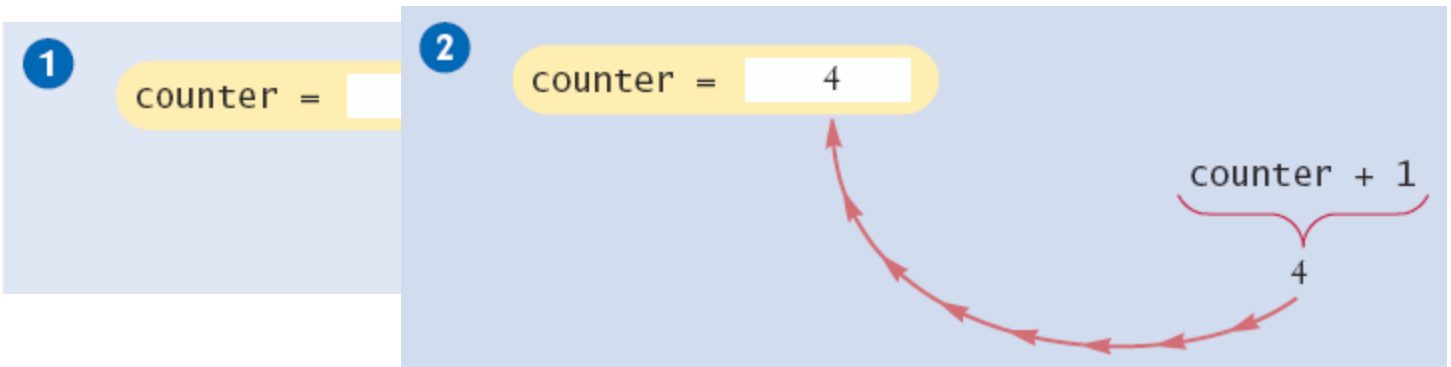
- Precedence is similar to Algebra:
  - PEMDAS
    - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

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;
```

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
`counter = counter + 1;`	`counter++ ;`
`counter = counter - 1;`	`counter-- ;`

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
- 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

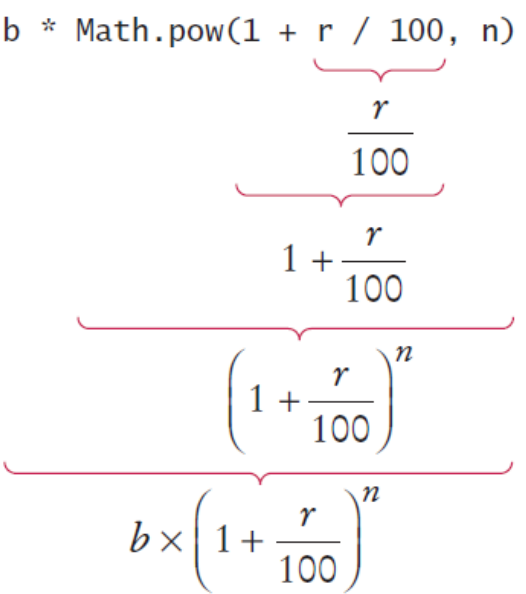
- In Java, there are no symbols for power and roots

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

Becomes:

```
b * 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 6 Mathematical Methods

Method	Returns
`Math.sqrt(x)`	Square root of x (≥ 0)
`Math.pow(x, y)`	x^y ($x > 0$, or $x = 0$ and $y > 0$, or $x < 0$ and y is an integer)
`Math.sin(x)`	Sine of x (x in radians)
`Math.cos(x)`	Cosine of x
`Math.tan(x)`	Tangent of x
`Math.toRadians(x)`	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
`Math.toDegrees(x)`	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
`Math.exp(x)`	e^x
`Math.log(x)`	Natural log ($\ln(x)$, $x > 0$)
`Math.log10(x)`	Decimal log ($\log_{10}(x)$, $x > 0$)
`Math.round(x)`	Closest integer to x (as a `long`)
`Math.abs(x)`	Absolute value $\lvert x \rvert$
`Math.max(x, y)`	The larger of x and y
`Math.min(x, y)`	The smaller of x and y

Floating-Point to Integer Conversion

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

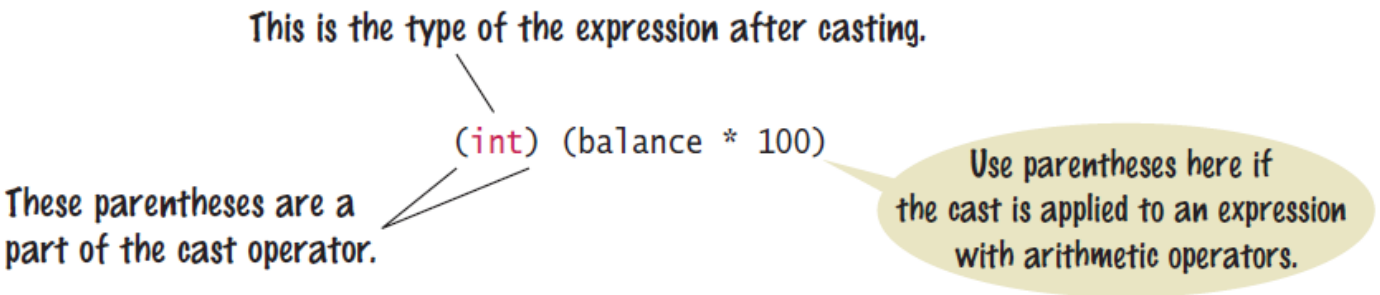
```
double balance = total + tax;
int dollars = balance; // Error
```

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

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

  2. Setup an object of the `Scanner` class

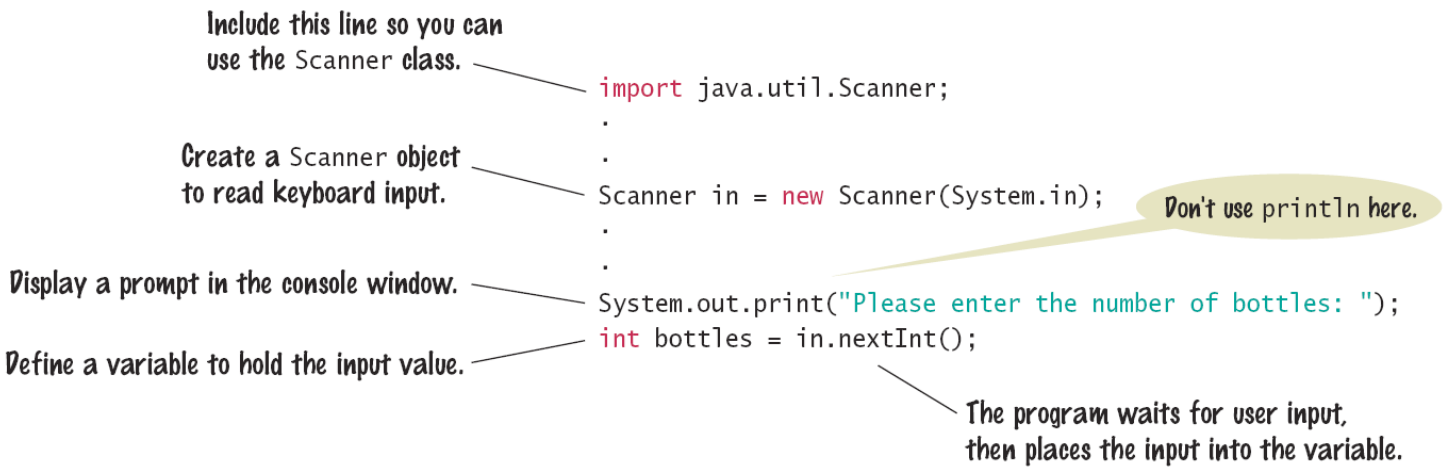
     ```
     Scanner in = new Scanner(System.in);
     ```

  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



Formatted Output

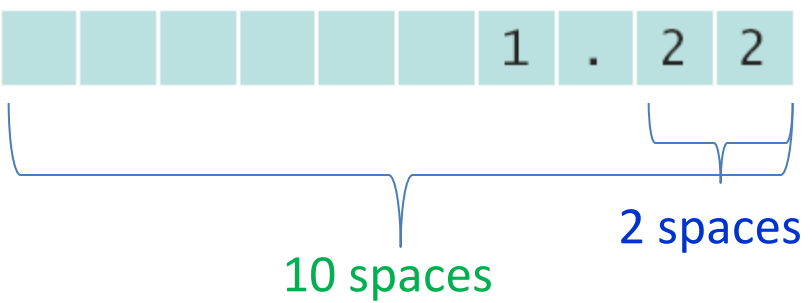
- Outputting floating point values can look strange:
  Price per liter: 1.21997
- To control the output appearance of numeric variables, use formatted output tools such as:

```
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

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

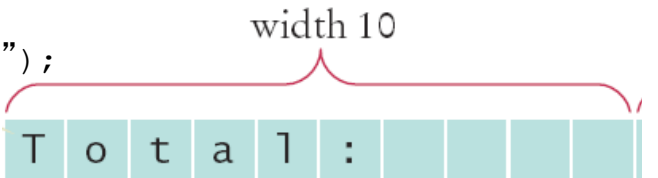
- You can also include text inside the quotes:

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

Formatted Output Examples

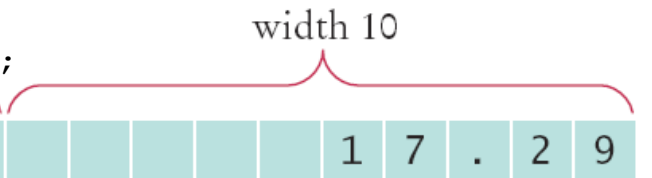
- Left Justify a String:

```
System.out.printf("%-10s", "Total:");
```



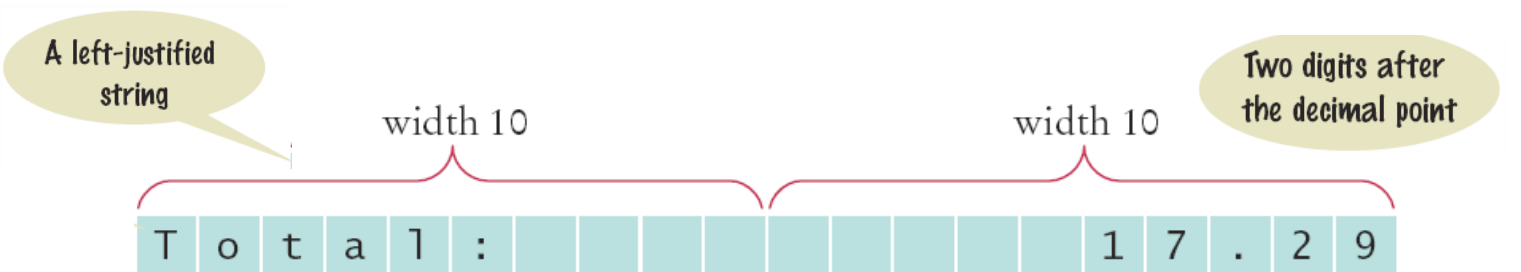
- Right justify a number with two decimal places

```
System.out.printf("%10.2f", price);
```



- And you can print multiple values

```
System.out.printf("%-10s%10.2f", "Total:", price);
```



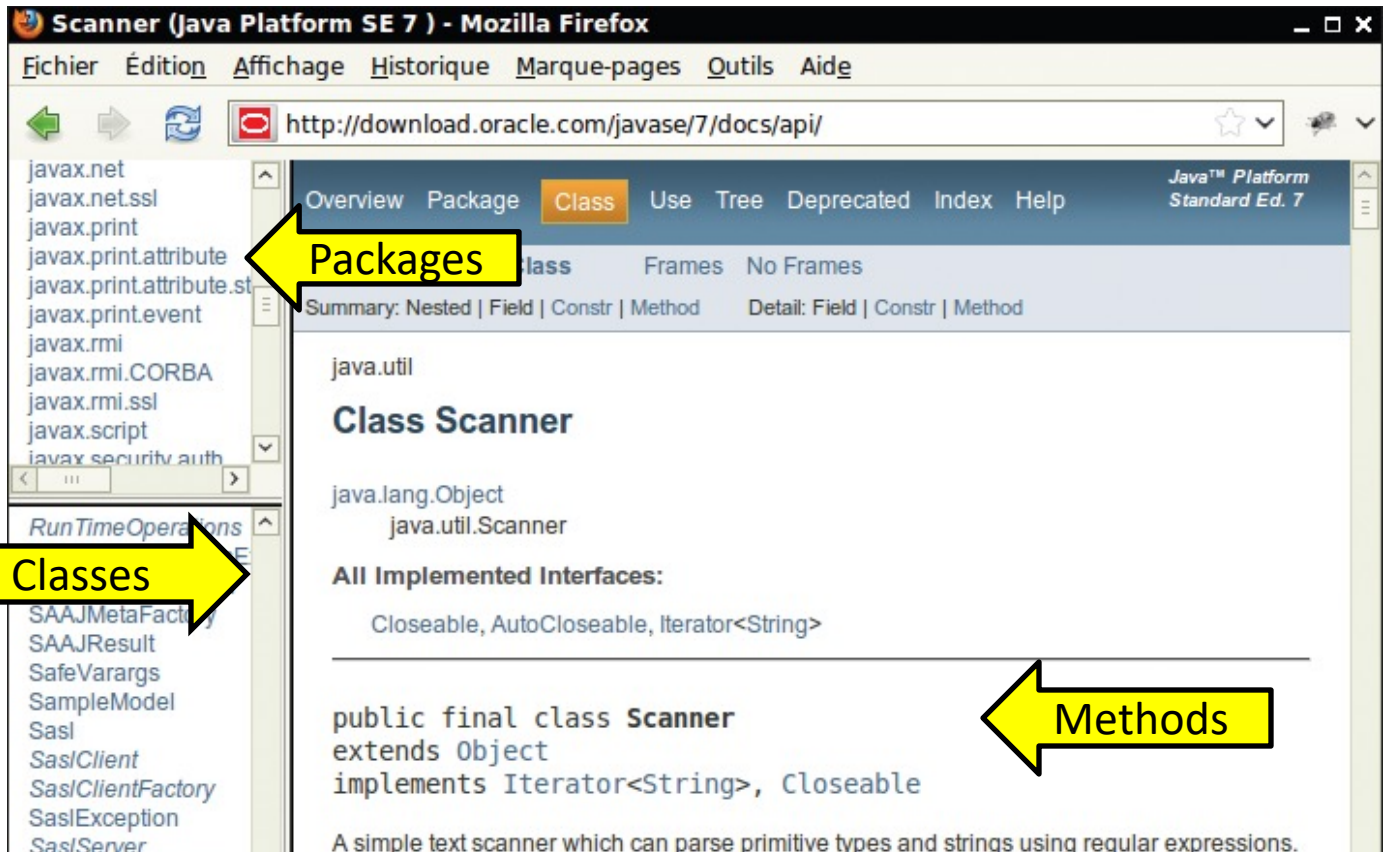
Volume2.java

section_3/Volume2.java

```
1  import java.util.Scanner;
2
3  /**
4     This program prints the price per ounce for a six-pack of cans.
5  */
6  public class Volume2
7  {
8     public static void main(String[] args)
9     {
10       // Read price per pack
11
12 24       final double CANS_PER_PACK = 6;
13 25       double packVolume = canVolume * CANS_PER_PACK;
14 26
15 27       // Compute and print price per ounce
16 28
17 29       double pricePerOunce = packPrice / packVolume;
18 30
19 31       System.out.printf("Price per ounce: %8.2f", pricePerOunce);
20 32       System.out.println();
21 33    }
   34 }
```

Tip 2.2 Java API Documentation

- Lists the classes and methods of the Java AP
  - On the web at: http://download.oracle.com/javase/6/docs/api



Strings

- The `String` Type:
  - `Type  Variable  Literal`
  - `String name = "Harry"`
- Once you have a `String` variable, you can use methods such as:

```
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

```
String fName = "Harry"
String lName = "Morgan"
String name = fname + lname;  // HarryMorgan
```

- You wanted a space in between?

```
String name = fname + " " + lname;  // Harry Morgan
```

- To concatenate a numeric variable to a `String`:

```
String a = "Agent";
int n = 7;
String bond = a + n;    // Agent7
```

- Concatenate `String`s and numerics inside `println`:

```
System.out.println("The total is " + total);
```

String Input

- You can read a `String` from the console with:

```
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:

```
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:

```
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`

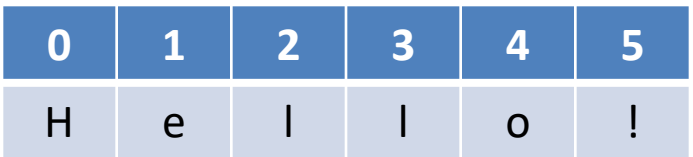
```
char initial = 'B';
```

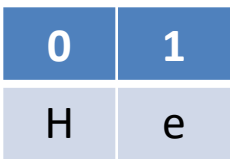
- Use double quotes around a `String`

```
String initials = "BRL";
```

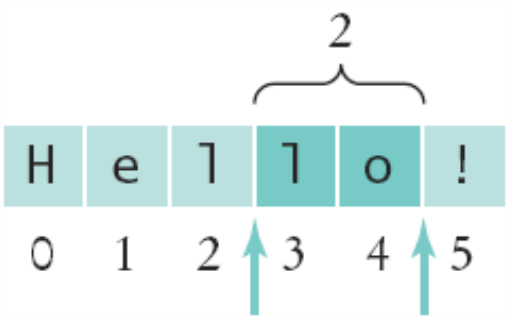
Copying a char from a String

- 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);
```

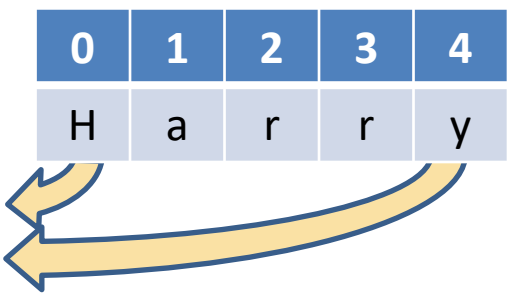
Copying a Portion of a `String`

- Each `char` inside a `String` has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- 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);
```



String Operations (1)

Table 9 String Operations

Statement	Result	Comment
`string str = "Ja";` `str = str + "va";`	`str` is set to `"Java"`	When applied to strings, `+` denotes concatenation.
`System.out.println("Please"` `  + " enter your name: ");`	Prints `Please enter your name:`	Use concatenation to break up strings that don't fit into one line.
`team = 49 + "ers"`	`team` is set to `"49ers"`	Because `"ers"` is a string, 49 is converted to a string.

`String first = in.next();`
`String last = in.next();`
`(User input: Harry Morgan)`
	`first` contains `"Harry"`
	`last` contains `"Morgan"`
		The `next` method places the next word into the string variable.

`String greeting = "H & S";` `int n = greeting.length();`	`n` is set to 5	Each space counts as one character.
`String str = "Sally";` `char ch = str.charAt(1);`	`ch` is set to `'a'`	This is a `char` value, not a `String`. Note that the initial position is 0.

String Operations (2)

Statement	Result	Comment
`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`.