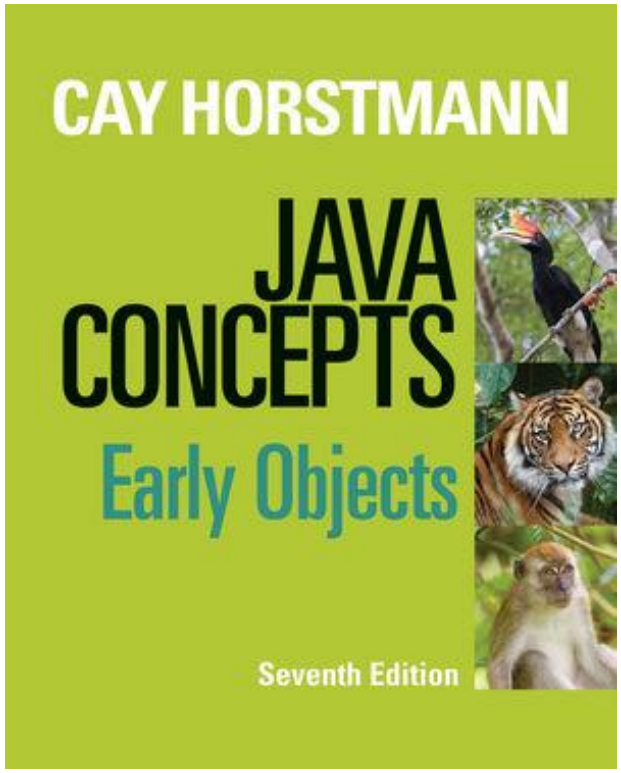


Chapter 4 – Fundamental Data Types

Chapter Goals

© Eyeidea/iStockphoto.

- To understand integer and floating-point numbers
- To recognize the limitations of the numeric types
- To become aware of causes for overflow and roundoff errors
- To understand the proper use of constants
- To write arithmetic expressions in Java
- To use the `String` type to manipulate character strings
- To write programs that read input and produce formatted output

Number Types

- Every value in Java is either:
  - a reference to an object
  - one of the eight primitive types
- Java has eight primitive types:
  - four integer types
  - two floating-point types
  - two other

Primitive Types

Type	Description	Size
`int`	The integer type, with range -2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`)	4 bytes
`byte`	The type describing a single byte, with range -128 . . . 127	1 byte
`short`	The short integer type, with range -32768 . . . 32767	2 bytes
`long`	The long integer type, with range -9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807	8 bytes
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 bit

Number Literals

- A number that appears in your code
- If it has a decimal, it is floating point
- If not, it is an integer

Number Literals

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

Overflow


© Douglas Allen/iStockphoto.

- Generally use an `int` for integers
- Overflow occurs when
  - The result of a computation exceeds the range for the number type
- Example

```
int n = 1000000;
System.out.println(n * n); // Prints -727379968, which is clearly wrong
```

  - 10^{12} is larger that the largest `int`
  - The result is truncated to fit in an `int`
  - No warning is given
- Solution: use `long` instead
- Generally do not have overflow with the `double` data type

Rounding Errors

© caracterdesign/iStockphoto.

- Rounding errors occur when an exact representation of a floating-point number is not possible.
- Floating-point numbers have limited precision. Not every value can be represented precisely, and roundoff errors can occur.
- Example:

```
double f = 4.35;
System.out.println(100 * f); // Prints 434.99999999999994
```

- Use `double` type in most cases

Constants: `final`

- Use symbolic names for all values, even those that appear obvious.
- A `final` variable is a constant
  - Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain.
- Convention: use all-uppercase names for constants:

```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;
payment = dollars + quarters * QUARTER_VALUE +
   dimes * DIME_VALUE + nickels * NICKEL_VALUE +
   pennies * PENNY_VALUE;
```

Constants: `static final`

- If constant values are needed in several methods,
  - Declare them together with the instance variables of a class
  - Tag them as `static` and `final`
  - The `static` reserved word means that the constant belongs to the class
- Give `static final` constants public access to enable other classes to use them.

Constants: `static final`

- Declaration of constants in the `Math` class

```
public class Math
{
   . . .
   public static final double E = 2.7182818284590452354;
   public static final double PI = 3.14159265358979323846;
}
```

- Using a constant

```
double circumference = Math.PI * diameter;
```

Syntax 4.1 Constant Declaration

Syntax

Declared in a method: `final` *typeName variableName* = *expression*;

Declared in a class: *accessSpecifier* `static final` *typeName variableName* = *expression*;

Declared in a method

```
final double NICKEL_VALUE = 0.05;
```

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

Use uppercase letters for constants.

```
public static final double LITERS_PER_GALLON = 3.785;
```

Declared in a class

Section_1/CashRegister.java

```
/**
   A cash register totals up sales and computes change due.
*/
public class CashRegister
{
   public static final double QUARTER_VALUE = 0.25;
   public static final double DIME_VALUE = 0.1;
   public static final double NICKEL_VALUE = 0.05;
   public static final double PENNY_VALUE = 0.01;

   private double purchase;
   private double payment;

   /**
      Constructs a cash register with no money in it.
   */
   public CashRegister()
   {
      purchase = 0;
      payment = 0;
   }

```

Continued

Section_1/CashRegister.java

```
   /**
      Records the purchase price of an item.
      @param amount the price of the purchased item
   */
   public void recordPurchase(double amount)
   {
      purchase = purchase + amount;
   }

   /**
      Processes the payment received from the customer.
      @param dollars the number of dollars in the payment
      @param quarters the number of quarters in the payment
      @param dimes the number of dimes in the payment
      @param nickels the number of nickels in the payment
      @param pennies the number of pennies in the payment
   */
   public void receivePayment(int dollars, int quarters,
         int dimes, int nickels, int pennies)
   {
      payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
            + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
   }

```

Continued

Section_1/CashRegister.java

```
47      /**
48         Computes the change due and resets the machine for the next customer.
49         @return the change due to the customer
50      */
51      public double giveChange()
52      {
53         double change = payment - purchase;
54         purchase = 0;
55         payment = 0;
56         return change;
57      }
58   }
```

Section_1/CashRegisterTester.java

```
/**
   This class tests the CashRegister class.
*/
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      CashRegister register = new CashRegister();

      register.recordPurchase(0.75);
      register.recordPurchase(1.50);
      register.receivePayment(2, 0, 5, 0, 0);
      System.out.print("Change: ");
      System.out.println(register.giveChange());
      System.out.println("Expected: 0.25");

      register.recordPurchase(2.25);
      register.recordPurchase(19.25);
      register.receivePayment(23, 2, 0, 0, 0);
      System.out.print("Change: ");
      System.out.println(register.giveChange());
      System.out.println("Expected: 2.0");
   }
}
```

Program Run:

```
Change: 0.25
Expected: 0.25
Change: 2.0
Expected: 2.0
```

Self Check 4.1

Which are the most commonly used number types in Java?

Answer: `int` and `double`

Self Check 4.2

Suppose you want to write a program that works with population data from various countries. Which Java data type should you use?

Answer: The world's most populous country, China, has about 1.2×10^9 inhabitants. Therefore, individual population counts could be held in an `int`. However, the world population is over 6×10^9. If you compute totals or averages of multiple countries, you can exceed the largest `int` value. Therefore, `double` is a better choice. You could also use `long`, but there is no benefit because the exact population of a country is not known at any point in time.

Self Check 4.3

Which of the following initializations are incorrect, and why?

```
1. int dollars = 100.0;
2. double balance = 100;
```

Answer: The first initialization is incorrect. The right hand side is a value of type `double`, and it is not legal to initialize an `int` variable with a `double` value. The second initialization is correct — an `int` value can always be converted to a `double`.

Self Check 4.4

What is the difference between the following two statements?

```
final double CM_PER_INCH = 2.54;
```

and

```
public static final double CM_PER_INCH = 2.54;
```

Answer: The first declaration is used inside a method, the second inside a class.

Self Check 4.5

What is wrong with the following statement sequence?

```
double diameter = . . .;
double circumference = 3.14 * diameter;
```

Answer: Two things

- You should use a named constant, not the "magic number" 3.14
- 3.14 is not an accurate representation of π.

Arithmetic Operators

- Four basic operators:
  - addition: `+`
  - subtraction: `-`
  - multiplication: `*`
  - division: `/`
- Expression: combination of variables, literals, operators, and/or method calls

```
(a + b) / 2
```

- Parentheses control the order of the computation

```
(a + b) / 2
```

- Multiplication and division have a higher precedence than addition and subtraction

```
a + b / 2
```

Arithmetic Operators

- Mixing integers and floating-point values in an arithmetic expression yields a floating-point value
  - `7 + 4.0` is the floating-point value `11.0`

Increment and Decrement

- The `++` operator adds 1 to a variable (increments)

```
counter++; // Adds 1 to the variable counter
```

- The `--` operator subtracts 1 from the variable (decrements)

```
counter--; // Subtracts 1 from counter
```

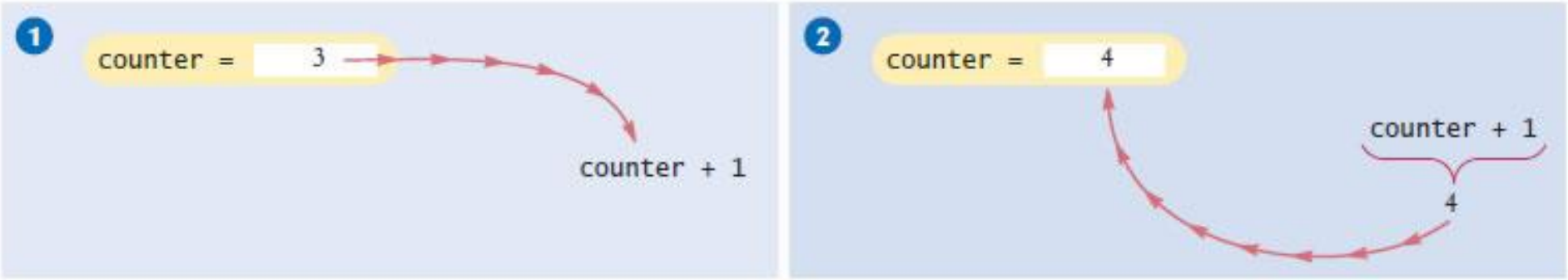


Figure 1 Incrementing a Variable

Integer Division and Remainder

- Division works as you would expect, as long as at least one of the numbers is a floating-point number.
- Example: all of the following evaluate to `1.75`

```
7.0 / 4.0
7 / 4.0
7.0 / 4
```

- If both numbers are integers, the result is an integer. The remainder is discarded
  - `7 / 4` evaluates to `1`
- Use `%` operator to get the remainder with (pronounced “modulus”, “modulo”, or “mod”)
  - `7 % 4` is `3`

Integer Division and Remainder

- To determine the value in dollars and cents of 1729 pennies
  - Obtain the dollars through an integer division by 100

```
int dollars = pennies / 100; // Sets dollars to 17
```

  - To obtain the remainder, use the % operator

```
int cents = pennies % 100; // Sets cents to 29
```

- Integer division and the % operator yield the dollar and cent values of a piggybank full of pennies.

© Michael Flippo/iStockphoto.

Integer Division and Remainder

Table 3 Integer Division and Remainder

Expression (where n = 1729)	Value	Comment
`n % 10`	9	`n % 10` is always the last digit of `n`.
`n / 10`	172	This is always `n` without the last digit.
`n % 100`	29	The last two digits of `n`.
`n / 10.0`	172.9	Because 10.0 is a floating-point number, the fractional part is not discarded.
`-n % 10`	-9	Because the first argument is negative, the remainder is also negative.
`n % 2`	1	`n % 2` is 0 if `n` is even, 1 or –1 if `n` is odd.

Powers and Roots

- Math class contains methods `sqrt` and `pow` to compute square roots and powers
- To take the square root of a number, use `Math.sqrt`; for example, `Math.sqrt(x)`
- To compute x^n, you write `Math.pow(x, n)`
  - To compute x^2 it is significantly more efficient simply to compute `x * x`
- In Java,

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

can be represented as

```
b * Math.pow(1 + r / 100, n)
```

Analyzing an Expression

```
b * Math.pow(1 + r / 100, n)
```

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

Figure 2
Analyzing an Expression

Mathematical Methods

Table 4 Mathematical Methods

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

Converting Floating-Point Numbers to Integers - Cast

- The compiler disallows the assignment of a `double` to an `int` because it is potentially dangerous
  - The fractional part is lost
  - The magnitude may be too large
  - This is an error

```
double balance = total + tax;
int dollars = balance; // Error: Cannot assign double to int
```

- Use the cast operator `(int)` to convert a convert floating-point value to an integer.

```
double balance = total + tax;
int dollars = (int) balance;
```

- Cast discards fractional part
- You use a cast (*typeName*) to convert a value to a different type.

Converting Floating-Point Numbers to Integers - Rounding

- `Math.round` converts a floating-point number to nearest integer:

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

- If `balance` is `13.75`, then `rounded` is set to `14`.

Syntax 4.2 Cast

Syntax (*typeName*) *expression*

This is the type of the expression after casting.

```
(int) (balance * 100)
```

These parentheses are a part of the cast operator.

Use parentheses here if the cast is applied to an expression with arithmetic operators.

Arithmetic Expressions

Table 5 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.

Self Check 4.6

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:

```
double interest = balance * percent / 100;
```

Self Check 4.7

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 4.8

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 * PI * Math.pow(radius, 3) / 3`

or `(4.0 / 3) * PI * Math.pow(radius, 3)`,

but not `(4 / 3) * PI * Math.pow(radius, 3)`

Self Check 4.9

What are the values of `1729 / 100` and `1729 % 100`?

Answer: `17` and `29`

Self Check 4.10

If `n` is a positive number, what is `(n / 10) % 10`?

Answer: It is the second-to-last digit of `n`. For example, if `n` is `1729`, then `n / 10` is `172`, and `(n / 10) % 10` is `2`.

Calling Static Methods

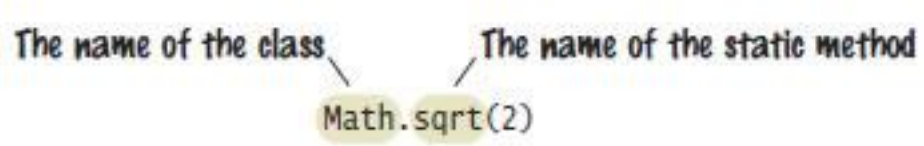
- Can not call a method on a number type `double`

```
root = 2.sqrt(); // Error
```

- Use a `static` method instead.
- A `static` method does not operate on an object:

```
double root = Math.sqrt(2); // Correct
```

- `static` methods are declared inside classes
- Calling a `static` method:



Reading Input

- When a program asks for user input
  - It should first print a message that tells the user which input is expected

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

- This message is called a **prompt**
  - Use the `print` method, not `println`, to display the prompt
  - Leave a space after the colon
- `System.in` has minimal set of features
  - Must be combined with other classes to be useful
- Use a class called `Scanner` to read keyboard input.

Reading Input - Scanner

- To obtain a `Scanner` object:

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

- Use the `Scanner`'s `nextInt` method to read an integer value:

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

- When the `nextInt` method is called,
  - The program waits until the user types a number and presses the Enter key;
  - After the user supplies the input, the number is placed into the bottles variable;
  - The program continues.

Reading Input - Scanner

- Use the `nextDouble` method to read a floating-point number:

```
System.out.print("Enter price: ");
double price = in.nextDouble();
```

- To use the `Scanner` class, import it by placing the following at the top of your program file:

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

Reading Input

© Media Bakery.

A supermarket scanner reads bar codes. The Java Scanner reads numbers and text.

Syntax 4.3 Input Statement

```
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.

Don't use println here.

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

Formatted Output

- Use the `printf` method to specify how values should be formatted.
- `printf` lets you print this

```
Price per liter: 1.22
```

- Instead of this

```
Price per liter: 1.215962441314554
```

- This command displays the price with two digits after the decimal point:

```
System.out.printf("%.2f", price);
```

Formatted Output

- You can also specify a *field width*:

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

- This prints 10 characters
  - Six spaces followed by the four characters `1.22`

- This command

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

- Prints

```
Price per liter: 1.22
```

Formatted Output

© Koele/iStockphoto.

You use the `printf` method to line up `your` output in neat columns.

Formatted Output

Table 6 Format Specifier Examples

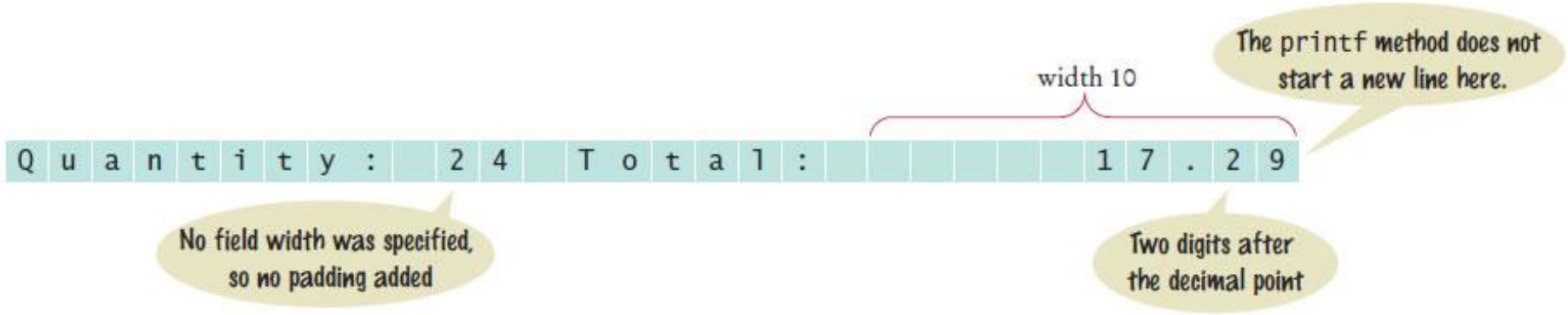
Format String	Sample Output	Comments
"%d"	24	Use d with an integer.
"%5d"	24	Spaces are added so that the field width is 5.
"Quantity:%5d"	Quantity: 24	Characters inside a format string but outside a format specifier appear in the output.
"%f"	1.21997	Use f with a floating-point number.
"%.2f"	1.22	Prints 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.

Formatted Output

- You can print multiple values with a single call to the `printf` method.
- Example

```
System.out.printf("Quantity: %d Total: %10.2f",
   quantity, total);
```

- Output explained:



section_3/Volume.java

```
import java.util.Scanner;

/**
    This program prints the price per liter for a six-pack of cans and
    a two-liter bottle.
*/
public class Volume
{
   public static void main(String[] args)
   {
      // Read price per pack

      Scanner in = new Scanner(System.in);

      System.out.print("Please enter the price for a six-pack: ");
      double packPrice = in.nextDouble();

      // Read price per bottle

      System.out.print("Please enter the price for a two-liter bottle: ");
      double bottlePrice = in.nextDouble();

```

Continued

section_3/Volume.java

```
23          final double CANS_PER_PACK = 6;
24          final double CAN_VOLUME = 0.355; // 12 oz. = 0.355 l
25          final double BOTTLE_VOLUME = 2;
26
27          // Compute and print price per liter
28
29          double packPricePerLiter = packPrice / (CANS_PER_PACK * CAN_VOLUME);
30          double bottlePricePerLiter = bottlePrice / BOTTLE_VOLUME;
31
32          System.out.printf("Pack price per liter:   %8.2f", packPricePerLiter);
33          System.out.println();
34
35          System.out.printf("Bottle price per liter: %8.2f", bottlePricePerLiter);
36          System.out.println();
37      }
38  }
```

Program Run

```
Please enter the price for a six-pack: 2.95
Please enter the price for a two-liter bottle: 2.85
Pack price per liter: 1.38
Bottle price per liter: 1.43
```

Self Check 4.11

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

Answer:

```
System.out.print("How old are you? ");
int age = in.nextInt();
```

Self Check 4.12

What is wrong with the following statement sequence?

```
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.

Self Check 4.13

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".

Self Check 4.14

What is problematic about the following statement sequence?

```
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 cans6
```

Self Check 4.15

What is the output of the following statement sequence?

```
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.

Self Check 4.16

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:

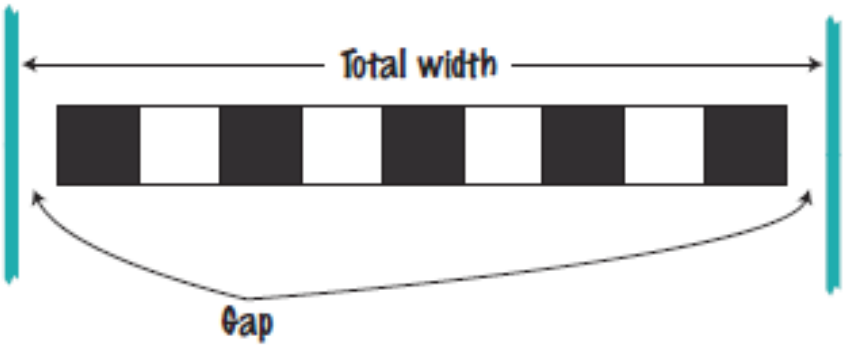
```
System.out.printf("Bottles: %8d\n", bottles);
System.out.printf("Cans: %8d\n", cans);
```

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\n”,
   "Bottles: ", bottles, "Cans:", cans);
```

Problem Solving: First Do It By Hand

- Very important step for developing an algorithm
  - Carry out the computations by hand first
- Pick concrete values for a typical situation to use in a hand calculation.
- Problem: A row of black and white tiles needs to be placed along a wall. First and last are black.
- Compute the number of tiles needed and the gap at each end, given the space available and the width of each tile.



Problem Solving: First Do It By Hand

- Use numbers
  - Total width: 100 inches
  - Tile width: 5 inches
- The first tile must always be black,
  - and then we add some number of white/black pairs:

Problem Solving: First Do It By Hand

- The first tile takes up 5 inches, leaving 95 inches to be covered by pairs.
  - Each pair is 10 inches wide.
  - The number of pairs needed is 95 / 10 = 9.5.
  - Discard the fractional part.
- We need 9 tile pairs or 18 tiles, plus the initial black tile => 19 tiles.
  - Tiles span $19 \times 5 = 95$ inches
  - Gap is $100 - 19 \times 5 = 5$ inches
- Distribute the gap at both ends
  - gap is $(100 - 19 \times 5) / 2 = 2.5$ inches

Problem Solving: First Do It By Hand

- Devise an algorithm with arbitrary values for the total width and tile width.
- The pseudocode

```
number of pairs = integer part of (total width - tile width) / (2 x tile width)
number of tiles = 1 + 2 x number of pairs
gap at each end = (total width - number of tiles x tile width) / 2
```

Self Check 4.17

Translate the pseudocode for computing the number of tiles and the gap width into Java.

Answer:

```
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 4.18

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

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

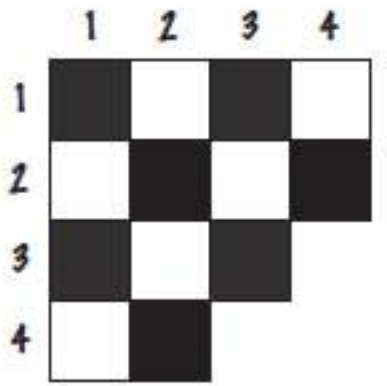
Answer: Now there are groups of four tiles (gray/white/gray/black) following the initial black tile. Therefore, the algorithm is now

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.

Self Check 4.19

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.



Continued

Self Check 4.19 - continued

Answer: 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:

Rows%2	Columns%2	Color
0	0	0
0	1	1
1	0	1
1	1	0

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:

```
color = ((row % 2) + (column % 2)) % 2
```

Self Check 4.20

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 x $156 = $412. The repair cost in year n is $100 + n x $156. To avoid accumulation of roundoff errors, it is actually a good idea to use the original expression that yielded $156, that is,

Repair cost in year n = 100 + n x 1400 / 9

Self Check 4.21

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.

Continued

Self Check 4.21 - continued

Answer: The pseudocode follows 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.

`String` Type

- A string is a sequence of characters.
- You can declare variables that hold strings

```
String name = "Harry";
```

- A string variable is a variable that can hold a string
- String literals are character sequences enclosed in quotes
- A string literal denotes a particular string

```
"Harry"
```

`String` Type

- String *length* is the number of characters in the string
  - The length of `"Harry"` is 5
- The `length` method yields the number of characters in a string
  - `int n = name.length();`
- A string of length 0 is called the *empty string*
  - Contains no characters
  - Is written as `""`

Concatenation

- **Concatenating strings** means to put them together to form a longer string
- Use the + operator
- Example:

```
String fName = "Harry”;
String lName = "Morgan”;
String name = fName + lName;
```

- Result:

```
"HarryMorgan"
```

- To separate the first and last name with a space

```
String name = fName + " " + lName;
```

- Results in

```
"Harry Morgan"
```

Concatenation

- If one of the arguments of the + operator is a string
  - The other is forced to become to a string:
  - Both strings are then concatenated
- Example

```
String jobTitle = "Agent”;
int employeeId = 7;
String bond = jobTitle + employeeId;
```

- Result

```
"Agent7"
```

Concatenation in Print Statements

- Useful to reduce the number of `System.out.print` instructions

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

versus

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

String Input

- Use the `next` method of the `Scanner` class to read a string containing a single word.

```
System.out.print("Please enter your name: ");
String name = in.next();
```

- Only one word is read.
- Use a second call to `in.next` to get a second word.

Escape Sequences

- To include a quotation mark in a literal string, precede it with a backslash (`\`)

```
"He said \"Hello\""
```

- Indicates that the quotation mark that follows should be a part of the string and not mark the end of the string
- Called an **escape sequence**
- To include a backslash in a string, use the escape sequence `\\`

```
"C:\\Temp\\Secret.txt"
```

- A newline character is denoted with the escape sequence `\n`
- A newline character is often added to the end of the format string when using `System.out.printf`:

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

Strings and Characters

- A string is a sequences of **Unicode** characters.
- A character is a value of the type `char`.
  - Characters have numeric values
- Character literals are delimited by single quotes.
  - `'H'` is a character. It is a value of type `char`
- Don't confuse them with strings
  - `"H"` is a string containing a single character. It is a value of type `String`.

Strings and Characters

- String positions are counted starting with 0.

H	a	r	r	y
0	1	2	3	4

- The position number of the last character is always one less than the length of the string.
- The last character of the string `"Harry"` is at position 4
- The `charAt` method returns a char value from a string
- The example

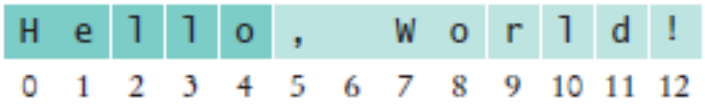
```
String name = "Harry”;
char start = name.charAt(0);
char last = name.charAt(4);
```

  - Sets start to the value `'H'` and last to the value `'y'`.

Substrings

- Use the `substring` method to extract a part of a string.
- The method call `str.substring(start, pastEnd)`
  - returns a string that is made up of the characters in the string `str`,
    - starting at position `start`, and
    - containing all characters up to, but not including, the position `pastEnd`.
- Example:

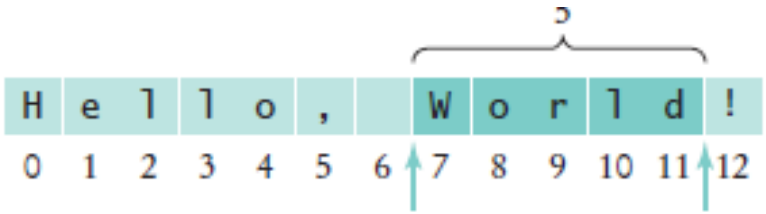
```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub is "Hello"
```



Substrings

- To extract "World"

```
String sub2 = greeting.substring(7, 12);
```



- Substring length is “past the end” - start

Substrings

- If you omit the end position when calling the substring method, then all characters from the starting position to the end of the string are copied.
- Example

```
String tail = greeting.substring(7); // Copies all characters from position 7 on
```

- Result
  - Sets `tail` to the string `"World!"`.

Substrings

- To make a string of one character, taken from the start of first

```
first.substring(0, 1)
```

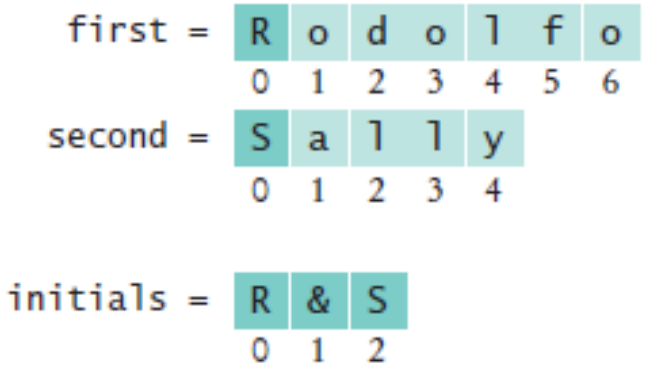


Figure 3 Building the `initials` String

section_5/Initials.java

```
import java.util.Scanner;

/**
   This program prints a pair of initials.
*/
public class Initials
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      // Get the names of the couple

      System.out.print("Enter your first name: ");
      String first = in.next();
      System.out.print("Enter your significant other&apos;s first name: ");
      String second = in.next();

      // Compute and display the inscription

      String initials = first.substring(0, 1)
         + "&" + second.substring(0, 1);
      System.out.println(initials);
   }
}
```

Continued

section_5/Initials.java

Program Run

```
Enter your first name: Rodolfo
Enter your significant other's first name: Sally
R&S
```

String Operations

Table 7 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 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`.

Self Check 4.22

What is the length of the string `"Java Program"`?

Answer: The length is 12. The space counts as a character.

Self Check 4.23

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 4.24

Use string concatenation to turn the string variable str from Self Check 23 into `"Java Programming"`.

Answer:

```
str = str + "ming”;
```

Self Check 4.25

What does the following statement sequence print?

```
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 4.26

Give an input statement to read a name of the form "John Q. Public".

Answer:

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

International Alphabets and Unicode: German Keyboard

© pvachier/iStockphoto.

Hebrew, Arabic, and English

© Joel Carillet/iStockphoto.

Chinese Script

© Saipg/iStockphoto.