

CHAPTER 2

FUNDAMENTAL DATA TYPES

FLIGHT	DESTINATION	GATE #
742	LOS ANGELES	A23
801	LONDON	C72
485	MADRID	B34
778	PARIS	A14
54	TOKYO	C89
753	HONG KONG	G1
14	MIAMI	C5
18	NEW YORK	D
74	RIO DEJANEIRO	A
5	SYDNEY	
	BANGKOK	



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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
485	MADRID
770	PARIS
54	TOKYO
753	HONG K
114	

Contents

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



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

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

2.1 Variables

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

FLIGHT	DESTINATI
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	KONG

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

Types introduced in this chapter are the number types **int** and **double** (page 34) and the String type (page 60).

`int cansPerPack = 6;`

See page 35 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.
See page 37.



Use a descriptive variable name.
See page 38.



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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:
 - Number of cans per pack
 - Ounces per can
 - Ounces per bottle
- | | Type of Number |
|-------------------------|----------------------|
| Number of cans per pack | Whole number |
| Ounces per can | Whole number |
| Ounces per bottle | Number with fraction |



FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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;
```

cansPerPack

6

- ❑ Imagine a parking space in a parking garage
 - Identifier: J053
 - Contents: Bob's Chevy

A variable is a storage location with a name



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Example Declarations

Table 1 Variable Declarations in Java

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

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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;
```

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Why different types?

- 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;
```

Use the double type for floating-point numbers.

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



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)

Parts of a floating point number -5:

Sign	Mantissa	Radix exponent
-1	5	10^0

[See JavaWorld article for more detail](#)

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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

□ Legal and illegal variable names

Table 3 Variable Names in Java

Variable Name	Comment
canVolume1	Variable names consist of letters, numbers, and the underscore character.
x	In mathematics, you use short variable names such as x or y . This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 38).
⚠ CanVolume	Caution: Variable names are case sensitive. This variable name is different from <code>canVolume</code> , 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 .

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Assignment Syntax

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

This is an initialization
of a new variable,
NOT an assignment.

The name of a previously
defined variable

```
double total = 0;  
.  
.  
total = bottles * BOTTLE_VOLUME;  
.  
.  
total = total + cans * CAN_VOLUME;
```

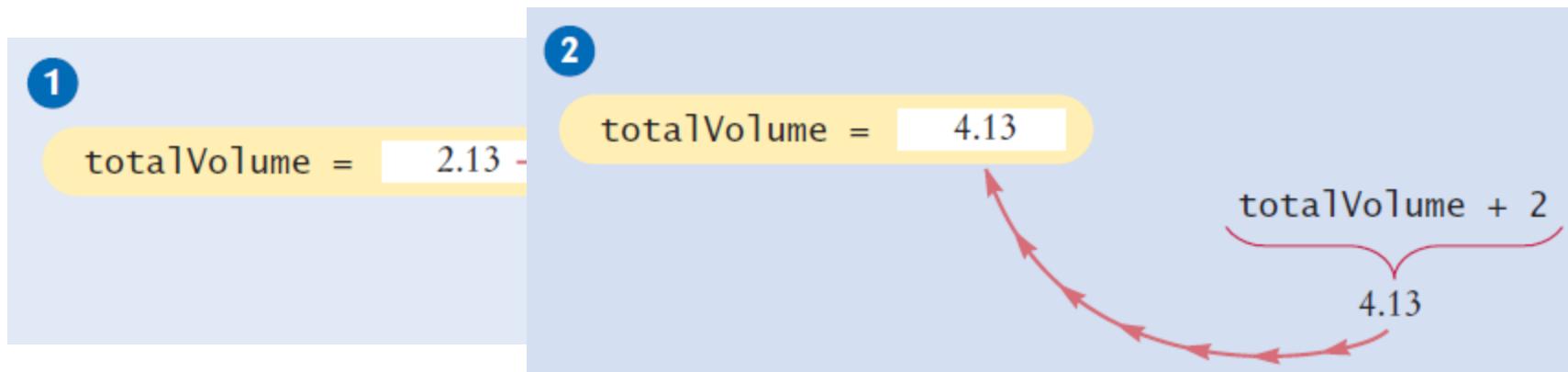
This is an assignment.

The expression that replaces the previous value

The same name
can occur on both sides.
See Figure 1.

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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)

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Java Comments

- There are three forms of comments:

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

2: /*

multi-line – all comment until matching

*/

3: /**

multi-line Javadoc comments

*/

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

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

FLIGHT	DESTINATI
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG K
14	

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; // ??
```

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
763	HONG KONG
14	

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)

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG K
14	

Common Error 2.3



❑ Roundoff Errors

- Floating point values are not exact
 - This is a limitation of binary values (no fractions):

```
double price = 4.35;  
double quantity = 100;  
double total = price * quantity;  
// Should be 100 * 4.35 = 435.00  
System.out.println(total); // Prints 434.9999999999999
```

- ❑ 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).

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
PES	HONG
14	

All of the Java Numeric Types

Type	Description	
int	The integer type, with range -2,147,483,648 (<code>Integer.MIN_VALUE</code>) . . . 2,147,483,647 (<code>Integer.MAX_VALUE</code> , 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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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

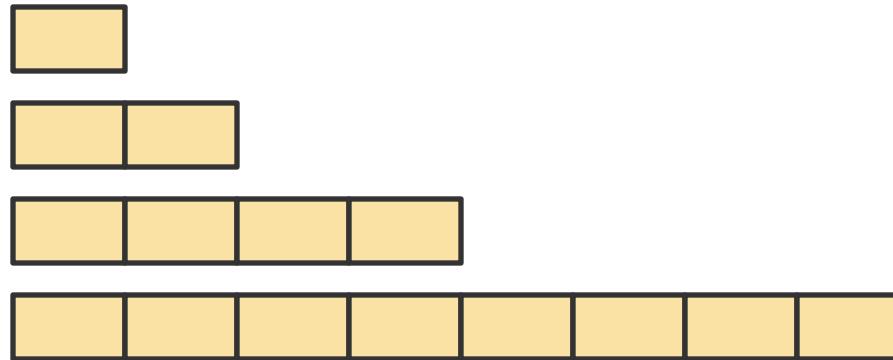
- **boolean**: `true` or `false`
- **char**: One symbol in single quotes ‘a’

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Storage per Type (in bytes)

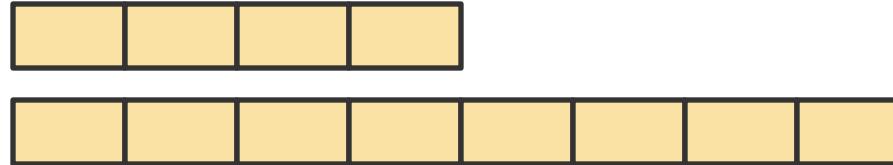
□ Integer Types

- **byte:**
- **short:**
- **int:**
- **long:**



□ Floating Point Types

- **float:**
- **double:**



□ Other Types

- **boolean:**
- **char:**



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG K
14	

2.2 Arithmetic

- Java supports all of the same basic math as a calculator:

- Addition + ■ Multiplication *
- Subtraction - ■ 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



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Mixing Numeric Types

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

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

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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-- ;

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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;
```

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

- 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

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Integer Division and Remainder Examples

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.

- Handy to use for making change:

```
int pennies = 1729;
int dollars = pennies / 100;    // 17
int cents = pennies % 100;      // 29
```

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Powers and Roots

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

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

`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).

$$\begin{aligned}
 & b * \underbrace{\mathbf{Math.pow(1 + \frac{r}{100}, n)}}_{\text{The Java library declares many Mathematical functions, such as } \mathbf{Math.sqrt} \text{ (square root) and } \mathbf{Math.pow} \text{ (raising to a power).}} \\
 & \quad \underbrace{\mathbf{1 + \frac{r}{100}}}_{\text{ }} \\
 & \quad \underbrace{\mathbf{\left(1 + \frac{r}{100}\right)^n}}_{\text{ }} \\
 & \quad \underbrace{\mathbf{b \times \left(1 + \frac{r}{100}\right)^n}}_{\text{ }}
 \end{aligned}$$

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Mathematical Methods

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

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)

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Cast Syntax

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.

- ❑ 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);`

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
PES	HONG
14	

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.
$\left(1 + \frac{r}{100}\right)^n$	<code>Math.pow(1 + r / 100, n)</code>	Use <code>Math.pow(x, n)</code> to compute x^n .
$\sqrt{a^2 + b^2}$	<code>Math.sqrt(a * a + b * b)</code>	$a * a$ is simpler than <code>Math.pow(a, 2)</code> .
$\frac{i + j + k}{3}$	$(i + j + k) / 3.0$	If i , j , and k are integers, using a denominator of 3.0 forces floating-point division.
π	<code>Math.PI</code>	<code>Math.PI</code> is a constant declared in the <code>Math</code> class.

FLIGHT	DESTINATI
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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

FLIGHT	DESTINATI
742	LOS ANGE
801	PARIS
425	LONDON
770	MADRID
54	TOKYO
763	HONG K
14	KONG

Common Error 2.5



□ Unbalanced Parenthesis

- Which is correct?

$(-(b * b - 4 * a * c) / (2 * a)) // 3 (, 2)$

$-(b * b - (4 * a * c))) / 2 * a) // 2 (, 2)$

□ The count of (and) must match

□ Unfortunately, it is hard for humans to keep track

- Here's a handy trick

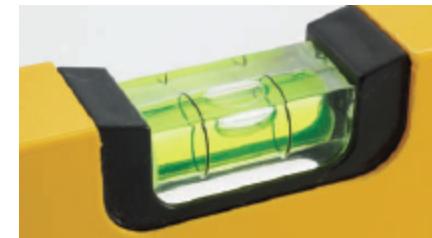
- Count (as +1, and) as -1: Goal: 0

$-(b * b - (4 * a * c))) / 2 * a)$

1 2

1 0 -1

-2



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

2.3 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.

Include this line so you can use the Scanner class.

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

Create a Scanner object to read keyboard input.

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

Don't use **println** here.

Display a prompt in the console window.

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

Define a variable to hold the input value.

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

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



Formatted Output

- Outputting floating point values can look strange:

Price per liter: 1.21997

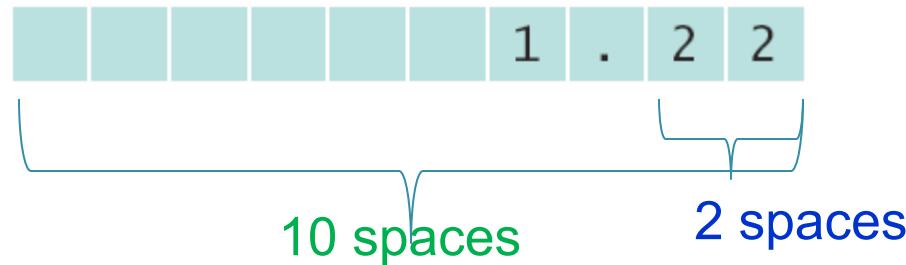
- To control the output appearance of numeric variables, use formatted output tools such as:

```
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 Types

Code	Type	Example
d	Decimal integer	123
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation is used for very large or very small values)	12.3
s	String	Tax:

- You can also include text inside the quotes:

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

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Format Flags

- You can also use format flags to change the way text and numeric values are output:

Table 9 Format Flags

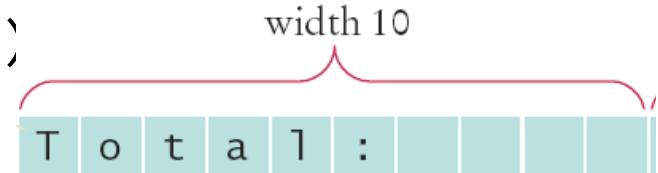
Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG K
14	ONG

Format Flag Examples

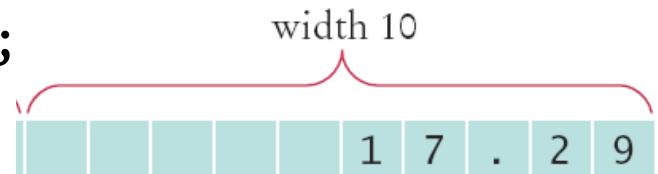
- Left Justify a String:

```
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);
```



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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         final double CANS_PER_PACK = 6;
13         double packVolume = canVolume * CANS_PER_PACK;
14
15         // Compute and print price per ounce
16
17         double pricePerOunce = packPrice / packVolume;
18
19         System.out.printf("Price per ounce: %8.2f", pricePerOunce);
20         System.out.println();
21
22     }
23
24 }
```



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>.

The screenshot shows a Mozilla Firefox browser window with the title "Scanner (Java Platform SE 7) - Mozilla Firefox". The address bar contains the URL <http://download.oracle.com/javase/7/docs/api>. The page content is the Java API documentation for the `Scanner` class. The navigation bar at the top includes links for Overview, Package, Class (which is highlighted), Use, Tree, Deprecated, Index, and Help. A sidebar on the left lists various Java packages. The main content area displays the `Scanner` class definition, its inheritance from `Object`, and its implementation of `Closeable` and `Iterator<String>`. Three yellow arrows point to specific parts of the page: one points to the "Packages" section in the sidebar, another points to the "Classes" section in the sidebar, and a third points to the "Methods" section in the main content area.

Packages

Classes

Methods

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

Copyright © 2013 by John Wiley & Sons. All rights reserved.

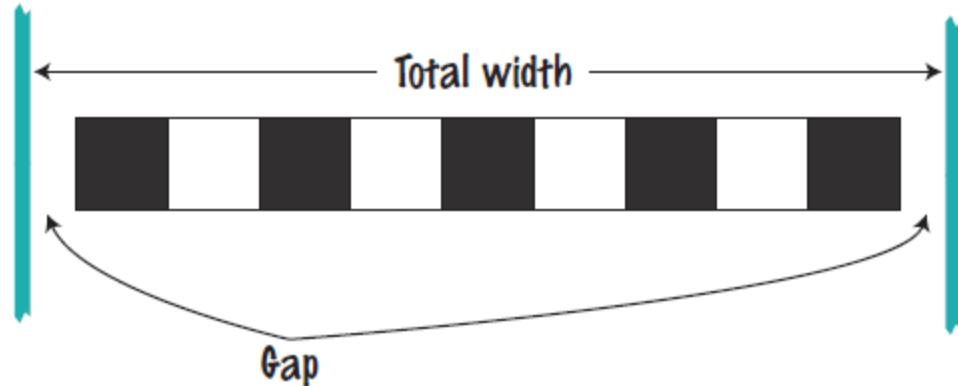
FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

2.4 Problem Solving: First By Hand

- 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.



FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Start with example values

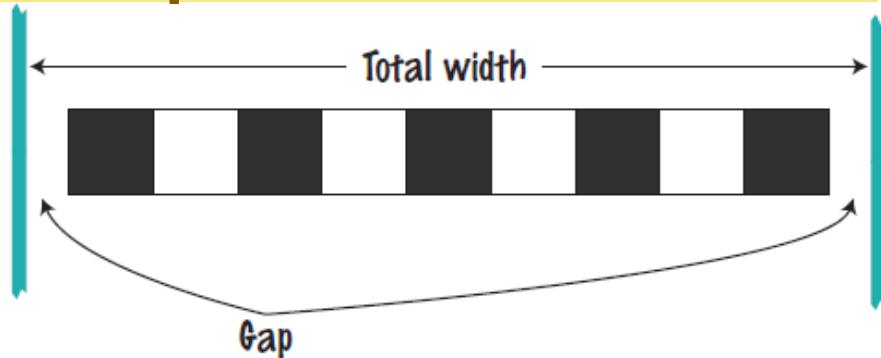
❑ Givens

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



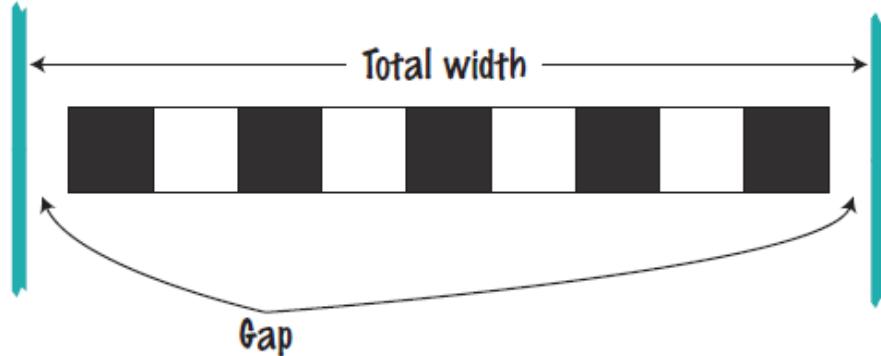
- Observation: each pair is 2x width of 1 tile
 - In our example, $2 \times 5 = 10$ inches

FLIGHT	DESTINAT
742	LOS ANGE
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	KONG

Keep Applying Your Solution

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

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG
14	

Now Devise an Algorithm

- Use your example to see how you calculated values
- How many pairs?

■ Note: must be a whole number

Integer part of:

$(\text{total width} - \text{tile width}) / 2 \times \text{tile width}$

- How many tiles?
 $1 + 2 \times \text{the number of pairs}$
- Gap at each end
 $(\text{total width} - \text{number of tiles} \times \text{tile width}) / 2$



2.5 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 Strings 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");`

*
**

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

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

- 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

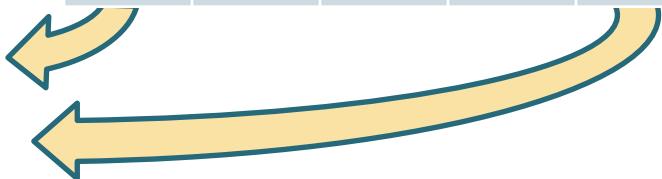
- The first char is index zero (0)
- The **charAt** method returns a char at a given index inside a String:

```
String greeting = "Harry";
```

```
char start = greeting.charAt(0);
```

```
char last = greeting.charAt(4);
```

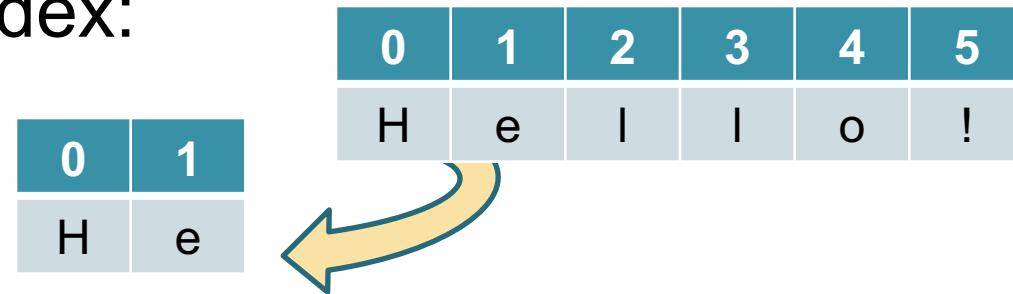
0	1	2	3	4
H	a	r	r	y





Copying portion of 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);
```

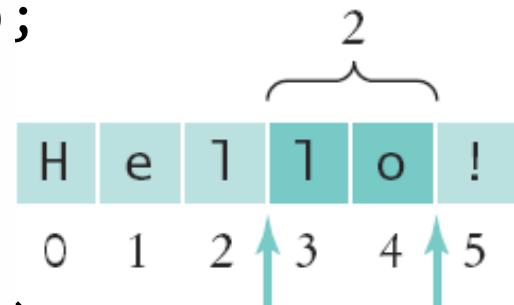




Table 9: String Operations (1)

Table 9 String Operations

Statement	Result	Comment
<pre>String str = "Ja"; str = str + "va";</pre>	str is set to "Java"	When applied to strings, + denotes concatenation.
<pre>System.out.println("Please" + " enter your name: ");</pre>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<pre>team = 49 + "ers"</pre>	team is set to "49ers"	Because "ers" is a string, 49 is converted to a string.
<pre>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</pre>	first contains "Harry" last contains "Morgan"	The next method places the next word into the string variable.
<pre>String greeting = "H & S"; int n = greeting.length();</pre>	n is set to 5	Each space counts as one character.
<pre>String str = "Sally"; char ch = str.charAt(1);</pre>	ch is set to 'a'	This is a char value, not a String. Note that the initial position is 0.

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
PES	HONG
14	

Table 9: String Operations (2)

Statement	Result	Comment
<pre>String str = "Sally"; String str2 = str.substring(1, 4);</pre>	str2 is set to "all"	Extracts the substring starting at position 1 and ending before position 4.
<pre>String str = "Sally"; String str2 = str.substring(1);</pre>	str2 is set to "ally"	If you omit the end position, all characters from the position until the end of the string are included.
<pre>String str = "Sally"; String str2 = str.substring(1, 2);</pre>	str2 is set to "a"	Extracts a String of length 1; contrast with str.charAt(1).
<pre>String last = str.substring(str.length() - 1);</pre>	last is set to the string containing the last character in str	The last character has position str.length() - 1.

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Summary: Variables

- ❑ A variable is a storage location with a name.
- ❑ When declaring a variable, you usually specify an initial value.
- ❑ When declaring a variable, you also specify the type of its values.
- ❑ Use the `int` type for numbers that cannot have a fractional part.
- ❑ Use the `double` type for floating-point numbers.
- ❑ By convention, variable names should start with a lower case letter.
- ❑ An assignment statement stores a new value in a variable, replacing the previously stored value

FLIGHT	DESTINAT
742	LOS ANGEL
801	LONDON
425	MADRID
770	PARIS
54	TOKYO
753	HONG KONG
14	

Summary: Operators

- ❑ The assignment operator = does not denote mathematical equality.
- ❑ You cannot change the value of a variable that is defined as final.
- ❑ The ++ operator adds 1 to a variable; the -- operator subtracts 1.
- ❑ If both arguments of / are integers, the remainder is discarded.
- ❑ The % operator computes the remainder of an integer division.



Summary: Java API

- ❑ The Java library declares many mathematical functions, such as `Math.sqrt` and `Math.pow`.
- ❑ You use a cast (`typeName`) to convert a value to a different type.
- ❑ Java classes are grouped into packages. Use the `import` statement to use classes from packages.
- ❑ Use the `Scanner` class to read keyboard input in a console window.
- ❑ Use the `printf` method to specify how values should be formatted.
- ❑ The API (Application Programming Interface) documentation lists the classes and methods of the Java library.



Summary: Strings

- ❑ Strings are sequences of characters.
- ❑ The length method yields the number of characters in a String.
- ❑ Use the + operator to concatenate Strings; that is, to put them together to yield a longer String.
- ❑ Use the next (one word) or nextLine (entire line) methods of the Scanner class to read a String.
- ❑ Whenever one of the arguments of the + operator is a String, the other argument is converted to a String.
- ❑ If a String contains the digits of a number, you use the Integer.parseInt or Double.parseDouble method to obtain the number value.
- ❑ String index numbers are counted starting with 0.
- ❑ Use the substring method to extract a part of a String