Advanced Computer Programming

[Lecture 02]

Saeed Reza Kheradpisheh

kheradpisheh@ut.ac.ir

Department of Computer Science
Shahid Beheshti University
Spring 1397-98

Variables

When your program carries out computations, you will want to store values so that you can use them later.

- In a Java program, you use **variables** to store values.

Example of variable **declaration**:

- `int number = 6;`

Variables

When your program carries out computations, you will want to store values so that you can use them later.

- In a Java program, you use **variables** to store values.

Example of variable **declaration**:

- `int number = 6;`

Definition

A **variable** is a storage location in a computer program. Each variable has a type, name, and holds a value.

Variables

Syntax

Variable declaration:

- `typeName variableName = value;`, or
- `typeName variableName;`

- You usually specify an **initial value**.
- You also specify the **type** (size) of its values.
  - Java supports quite a few data types: numbers, text strings, files, dates, and many others.
- After you have declared and initialized a variable, you can use it

```
int number = 10;
System.out.println(number);
int product = 4 * number;
```

Variables

Real-world example: Parking space

Unnumbered 2 p30b
Javier Larrea/Age Fotostock

Unnumbered 2 p31
© Ingenui/iStockphoto

Variables

Some programming examples:

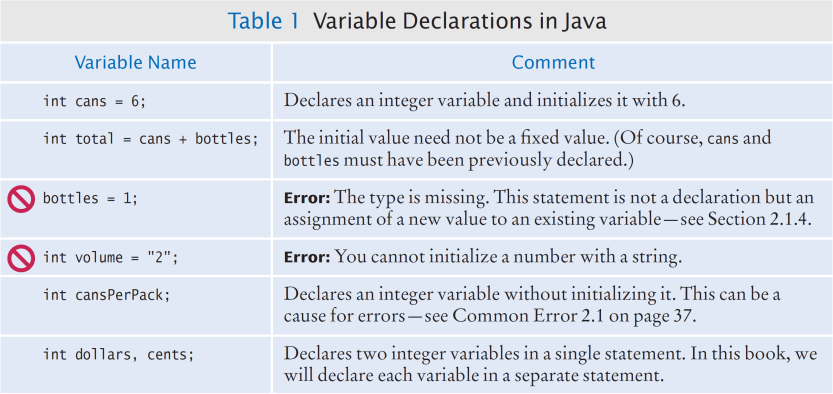


Table 1 Variable Declarations in Java

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.

Type of Variables

Type = Size + Operations

Number Types

Two most commonly used number types:

- **`int`**: for integer numbers.

  ```
  int number = 10;
  ```

- **`double`**: for floating-point numbers.

  ```
  double number2 = 10.55;
  ```

Number Types

Two most commonly used number types:

- **`int`**: for integer numbers.

  ```
  int number = 10;
  ```

- **`double`**: for floating-point numbers.

  ```
  double number2 = 10.55;
  ```

Definition

Numeral value that occurs in a Java program is called a **number literal**.

```
int number = 10;
double number2 = 10.55;
```

Number Literals

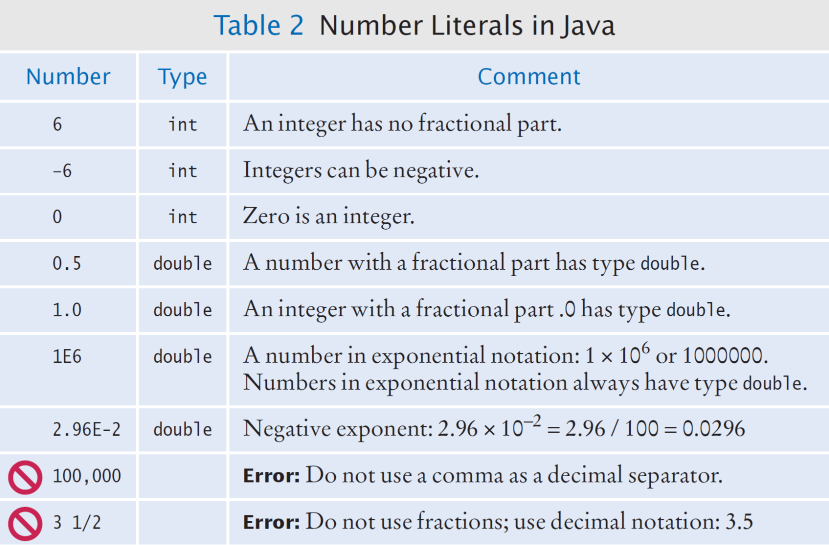


Table 2 Number Literals in Java

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

Variable Name

When you declare a variable, you should pick a name that explains its purpose.

In Java, there are a few simple rules for variable names:

- Variable names must start with a letter or the underscore (_) character, and the remaining characters must be letters, numbers, or underscores.

Variable Name

When you declare a variable, you should pick a name that explains its purpose.

In Java, there are a few simple rules for variable names:

- Variable names must start with a letter or the underscore (_) character, and the remaining characters must be letters, numbers, or underscores.
- Spaces are not permitted inside names either. You can use uppercase letters to denote word boundaries, as in cans**P**er**P**ack (camel casing).

Variable Name

When you declare a variable, you should pick a name that explains its purpose.

In Java, there are a few simple rules for variable names:

- Variable names must start with a letter or the underscore (_) character, and the remaining characters must be letters, numbers, or underscores.
- Spaces are not permitted inside names either. You can use uppercase letters to denote word boundaries, as in `cans`**`P`**`er`**`P`**`ack` (camel casing).
- Variable names are case sensitive.

Variable Name

When you declare a variable, you should pick a name that explains its purpose.

In Java, there are a few simple rules for variable names:

- Variable names must start with a letter or the underscore (_) character, and the remaining characters must be letters, numbers, or underscores.
- Spaces are not permitted inside names either. You can use uppercase letters to denote word boundaries, as in `cansPerPack` (camel casing).
- Variable names are case sensitive.
- You cannot use reserved words such as `double` or `int` as names (Appendix C).

Variable Name

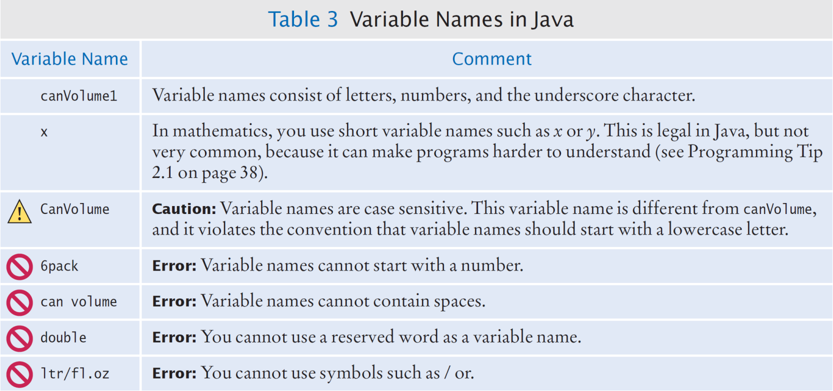


Table 3 Variable Names in Java

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

You use the **assignment statement** to place a new value into a variable. That value is stored in the variable, overwriting its previous contents.

```
variableName = value;
```

(direction is important)

The Assignment Statement

You use the **assignment statement** to place a new value into a variable. That value is stored in the variable, overwriting its previous contents.

```
variableName = value;
```

(direction is important)

Note

Assignment is different from variable declaration;

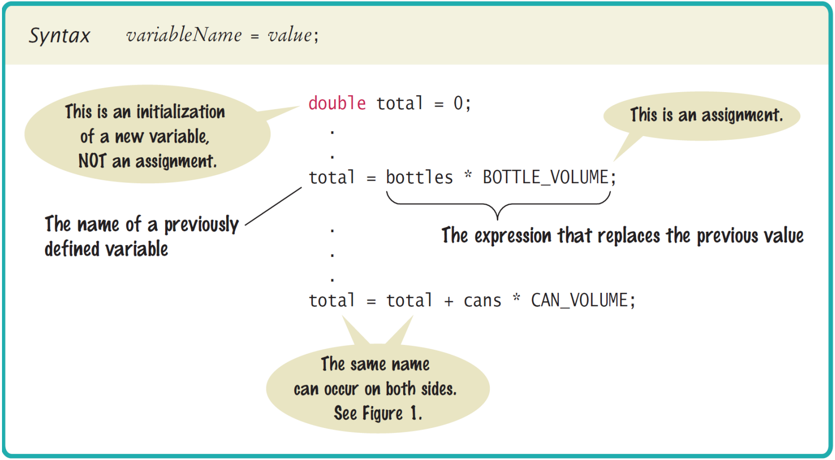
- Variable declaration (an instruction to create a new variable of an specific type)

  ```
  int number = 10;
  ```

- Assignment statement (an instruction to replace the contents of the existing variable with another value)

  ```
  number = 100;
  ```

The Assignment Statement



Syntax *variableName* = *value*;

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

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

This is an assignment.

The name of a previously defined variable

The expression that replaces the previous value

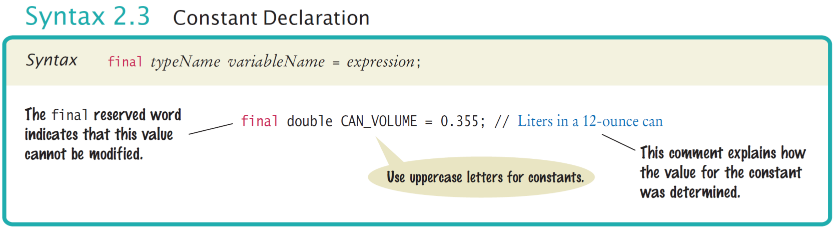
The same name can occur on both sides. See Figure 1.

Constants

Definition

When a variable is defined with the reserved word `final`, its value can never change, so it is a **constant**.

Constants are commonly written using CAPITAL letters to be distinguished.



Syntax 2.3 Constant Declaration

Syntax `final typeName variableName = expression;`

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

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

Use uppercase letters for constants.

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

Comments

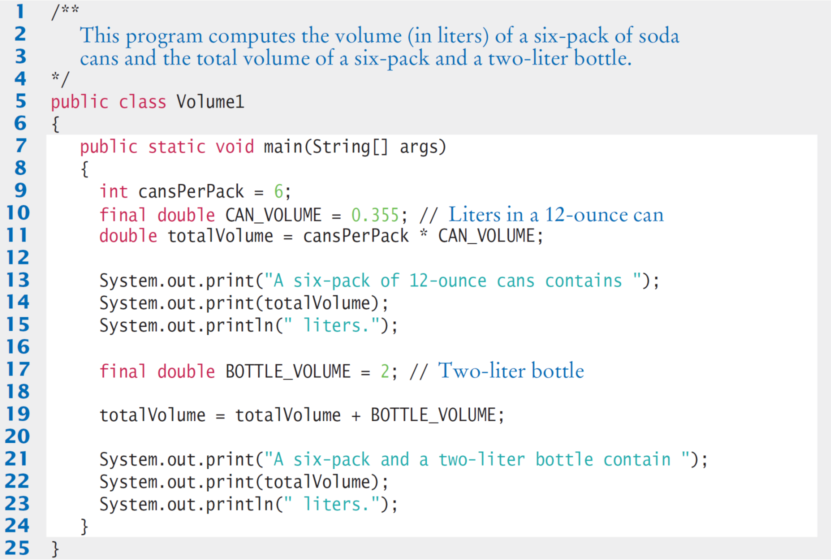
Definition

As your programs get more complex, you should add **comments**, explanations for human readers of your code. The compiler does not process comments at all.

Types of commenting:

- Line: `//comment begins to the end of line`
- Block: `/*all comments in between*/`

Example

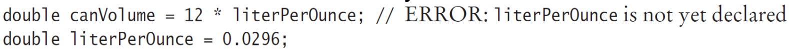


```
/**
   This program computes the volume (in liters) of a six-pack of soda
   cans and the total volume of a six-pack and a two-liter bottle.
*/
public class Volume1
{
   public static void main(String[] args)
   {
      int cansPerPack = 6;
      final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
      double totalVolume = cansPerPack * CAN_VOLUME;

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

      final double BOTTLE_VOLUME = 2; // Two-liter bottle

      totalVolume = totalVolume + BOTTLE_VOLUME;

      System.out.print("A six-pack and a two-liter bottle contain ");
      System.out.print(totalVolume);
      System.out.println(" liters.");
   }
}
```

Common Error

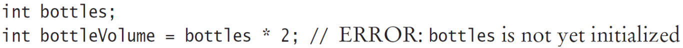
Using Undeclared or Uninitialized Variables

- You must declare a variable before you use it for the first time.



```
double canVolume = 12 * literPerOunce; // ERROR: literPerOunce is not yet declared
double literPerOunce = 0.0296;
```

- A related error is to leave a variable uninitialized.



```
int bottles;
int bottleVolume = bottles * 2; // ERROR: bottles is not yet initialized
```

Numeric Types in Java

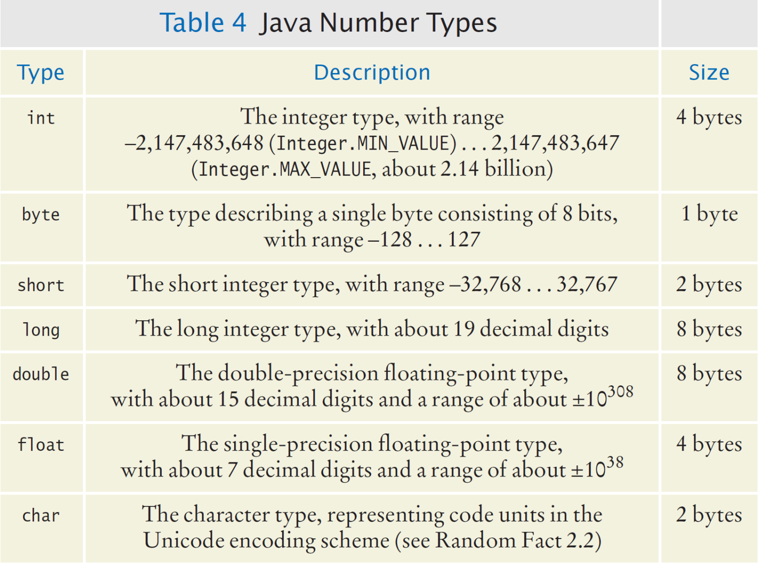


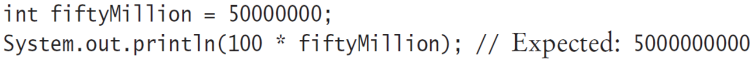
Table 4 Java Number Types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)	4 bytes
`byte`	The type describing a single byte consisting of 8 bits, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32,768 . . . 32,767	2 bytes
`long`	The long integer type, with about 19 decimal digits	8 bytes
`double`	The double-precision floating-point type, with about 15 decimal digits and a range of about $\pm 10^{308}$	8 bytes
`float`	The single-precision floating-point type, with about 7 decimal digits and a range of about $\pm 10^{38}$	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme (see Random Fact 2.2)	2 bytes

Problems with Binary Representation

- **Overflow**

  Because numbers are represented in the computer with a limited number of digits, they cannot represent arbitrary numbers.



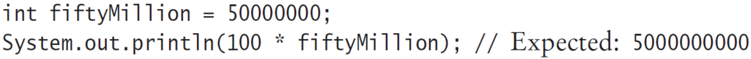
```
int fiftyMillion = 50000000;
System.out.println(100 * fiftyMillion); // Expected: 5000000000
```

output: `705032704`

Problems with Binary Representation

- **Overflow**

  Because numbers are represented in the computer with a limited number of digits, they cannot represent arbitrary numbers.

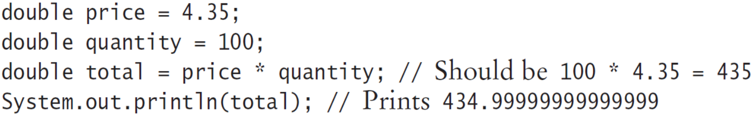


```
int fiftyMillion = 50000000;
System.out.println(100 * fiftyMillion); // Expected: 5000000000
```

  output: `705032704`

- **Roundoff**

  As with decimal numbers, you can get roundoff errors when binary digits are lost.



```
double price = 4.35;
double quantity = 100;
double total = price * quantity; // Should be 100 * 4.35 = 435
System.out.println(total); // Prints 434.99999999999999
```

Arithmetic Operators

- All of the four basic arithmetic operators are available here:
  - addition (`+`)
  - subtraction (`-`)
  - multiplication (`*`)
  - division (`/`)

Definition

The combination of variables, literals, operators, and/or method calls is called an **expression**.

e.g. `a + b / 2`

Arithmetic Operators

Notes

- As in regular algebraic notation, multiplication and division have a higher **precedence** than addition and subtraction.
- **Parentheses** are used to indicate in which order the parts of the expression should be computed.
- **Mixing** integers and floating-point values in an arithmetic expression **yields a floating-point** value.

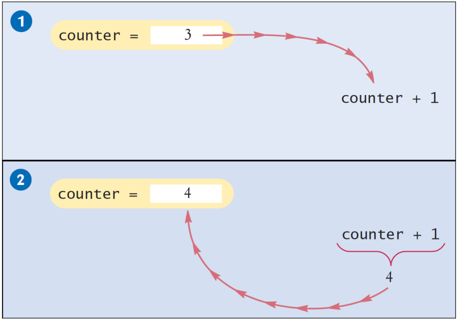
Increment and Decrement

Changing a variable by adding or subtracting 1 is so common that there is a special shorthand for it;

- Increment: `++` operator

  `counter++;` equals to `counter = counter + 1;`
- Decrement: `--` operator

  `counter--;` equals to `counter = counter - 1;`



Combining Assignment and Arithmetic

In Java you can combine arithmetic and assignment:

- `counter = counter + 10;` can be written as: `counter += 10;`
- `counter = counter - 10;` can be written as: `counter -= 10;`
- `counter = counter * 10;` can be written as: `counter *= 10;`
- `counter = counter / 10;` can be written as: `counter /= 10;`

Integer Division and Remainder

- Division works as you would expect, as long as at least one of the numbers involved is a floating-point number.

  `7.0/4 = 7/4.0 = 7.0/4.0 = 1.75`

- If both numbers are integers, then the result of the division is always an integer, with the remainder discarded.

  `7/4 = 1`

- If you are interested in the remainder only, use the `%` operator.

  `7 % 4 = 3`

Variable Swapping

Q & A

Q: How can we swap the values of two variables?

Variable Swapping

Q & A

Q: How can we swap the values of two variables?

A: Using a temporary variables;

```
int a = 10;
int b = 20;
int c = a;
a = b;
b = c;
```

Variable Swapping

Q & A

Q: How can we swap the values of two variables?

A: Using a temporary variables;

```
int a = 10;
int b = 20;
int c = a;
a = b;
b = c;
```

Q: Can we swap integer values without a temporary variable?

A: Your task!

Power and Roots

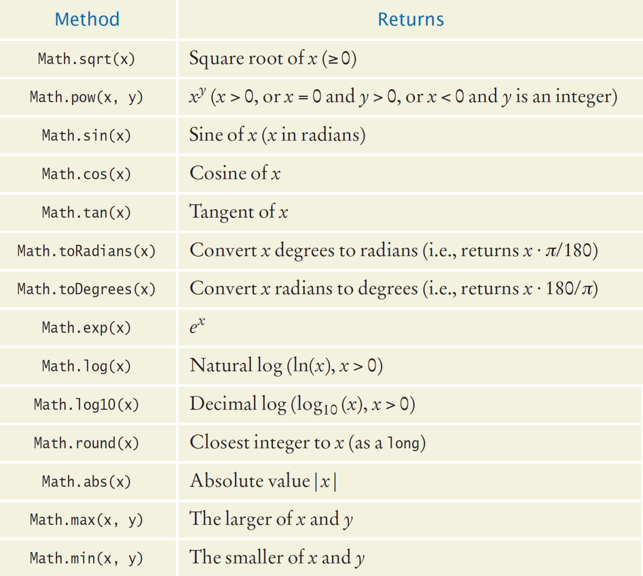
- In Java, there are no symbols for powers and roots.
- To take the square root of a number, you use the `Math.sqrt` $\sqrt{x}$ equals to `Math.sqrt(x)`
- To compute x^n you write `Math.pow(x,n)`.
- In Java you should write linear mathematic expressions. e.g.

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

should be written as

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

The `Math` Library



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

Converting Floting-Point Numbers to Integers

You have a value of type `double` that you need to convert to the type `int`.

- It is an error to assign a floating-point value to an integer.

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

- The compiler disallows this assignment because it is potentially dangerous:
  - The fractional part is lost.
  - The magnitude may be too large.

Converting Floting-Point Numbers to Integers

You have a value of type `double` that you need to convert to the type `int`.

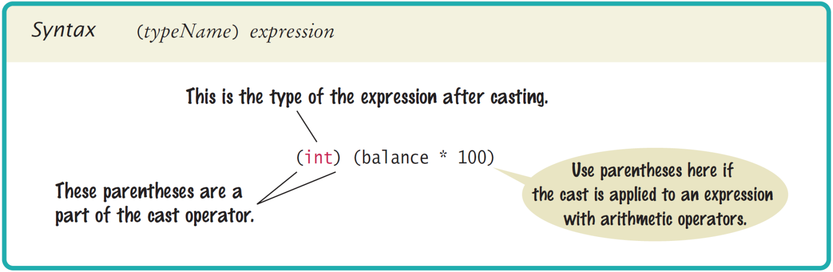
- It is an error to assign a floating-point value to an integer.

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

- The compiler disallows this assignment because it is potentially dangerous:
  - The fractional part is lost.
  - The magnitude may be too large.
- You must use the **cast operator** to convert a convert floating-point value to an integer.

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

Cast Operator



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.

Input and Output

Output

- `System.out.println(arg);`
- `System.out.print(arg);`

Input

- An `Scanner` must be created first

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

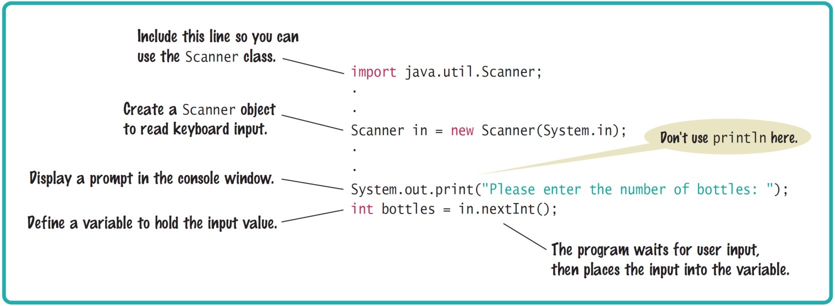
- To use `Scanner`, the package `java.util.Scanner` must be imported

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

- Use `next...` methods to read inputs, e.g.

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

Reading Input



```
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.

Don't use println here.

Display a prompt in the console window.

Define a variable to hold the input value.

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

Formatted Output

When you print the result of a computation, you often want to **control its appearance**. For example:

- Rounding to a number of significant digits.
  (`System.out.printf("%.2f", price);`)
- Specifying a field width.
  (`System.out.printf("%10.2f", price);`)

Formatted Output

When you print the result of a computation, you often want to **control its appearance**. For example:

- Rounding to a number of significant digits.
  (`System.out.printf("%.2f", price);`)
- Specifying a field width.
  (`System.out.printf("%10.2f", price);`)

Use the **`printf`** method and **format specifiers** to specify how values should be formatted,

- `%...f`, formating a floating-point number.
- `%...d`, formating an integer number.
- `%...s`, formating a string.

Format Specifiers

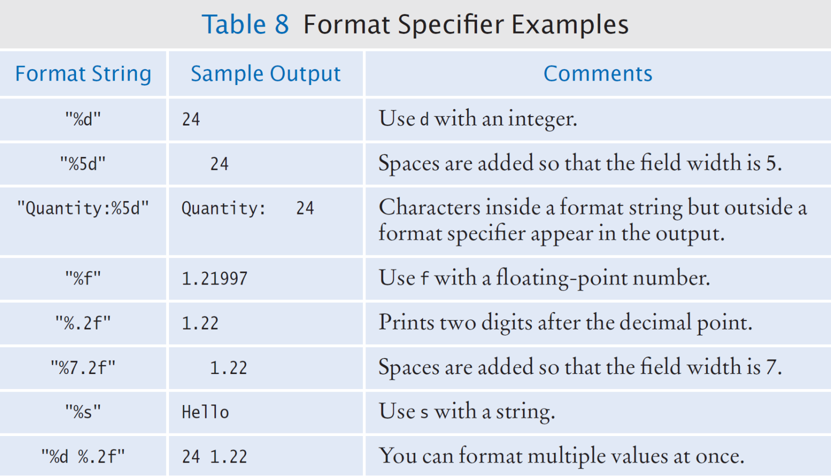


Table 8 Format Specifier Examples

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.

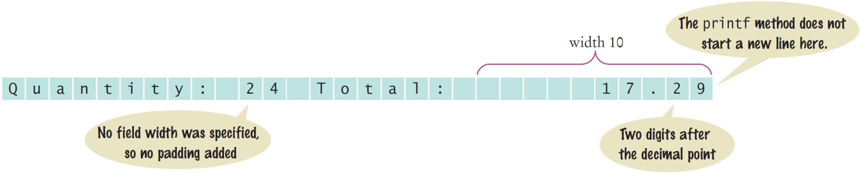
Format String

Definition

A **format string** is a string contains format specifiers and literal characters. Any characters that are not format specifiers are printed verbatim.

Examples:

- `System.out.printf("Price per liter:%10.2f", price);`
  ```
  Price per liter:   1.22
  ```
- `System.out.printf("Quantity:  %d Total:  %10.2f", quantity, total);`



```
Quantity: 24 Total:      17.29
```

width 10
The printf method does not start a new line here.
No field width was specified, so no padding added
Two digits after the decimal point

Strings

Many programs process **text**, not numbers.

Definition

A **string** is a sequence of characters, characters like letters, numbers, punctuation, spaces, and so on.

Strings

Many programs process **text**, not numbers.

Definition

A **string** is a sequence of characters, characters like letters, numbers, punctuation, spaces, and so on.

Declaring a variable that can hold strings:

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

- **String variable**: A variable that can hold a string.
- **String literal**: Character sequences enclosed in quotes.
- **Length of string**: The number of characters in a string.
  `int n = name.length();`
- **Empty string**: A string of length zero (`""`).

Concating Strings

In Java, you use the **+ operator** to concatenate two strings. For example:

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

Concating Strings

In Java, you use the **+ operator** to concatenate two strings.
For example:

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

results in the string

```
"HarryMorgan"
```

Concating Strings

In Java, you use the **+ operator** to concatenate two strings. For example:

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

results in the string

```
"HarryMorgan"
```

You can concatenate multiple strings

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

Concating Strings

In Java, you use the **+ operator** to concatenate two strings.
For example:

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

results in the string

```
"HarryMorgan"
```

You can concatenate multiple strings

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

results in the string

```
"Harry Morgan"
```

Concating Strings

- When the expression to the left or the right of a + operator is a string, the other one is automatically forced to become a string as well.

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

  `bond`'s value will be `"Agent7"`.

- concatenation is very useful for reducing the number of `System.out.print` instructions.

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

String Input

You can read a string from the console:

```
String name = in.next();
```

where `in` is a scanner.

Note

When a string is read with the `next` method, **only one word is read**.

Escape Sequences

Definition

Escape sequences are used to represent certain special characters within string literals and character literals.

Escape Sequences

Definition

Escape sequences are used to represent certain special characters within string literals and character literals.

Common examples:

- Include a quotation mark:

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

- Include a backslash:

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

- Printing a **newline** (useful with `printf`):

  ```
  System.out.print("*\n**\n***\n");
  ```

  Prints the characters

  ```
  *
  **
  ***
  ```

Strings and Characters

H	a	r	r	y
0	1	2	3	4

- Strings are sequences of characters, each has a position starting from 0 to its length - 1.
- In Java, a character is a value of the type `char`.
- Character literals are delimited by **single quotes**, and you should not confuse them with strings.
    - 'H' is a character, a value of type `char`.
    - "H" is a string containing a single character, a value of type `String`.

Strings and Characters

H	a	r	r	y
0	1	2	3	4

- Strings are sequences of characters, each has a position starting from 0 to its length - 1.
- In Java, a character is a value of the type `char`.
- Character literals are delimited by **single quotes**, and you should not confuse them with strings.
  - ’H’ is a character, a value of type `char`.
  - ”H” is a string containing a single character, a value of type `String`.
- The `charAt` method returns a char value from a string.

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

Substrings

- Once you have a string, you can extract substrings by using the `substring` method.

  `str.substring(start, pastEnd);`

- Example:

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

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

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`.