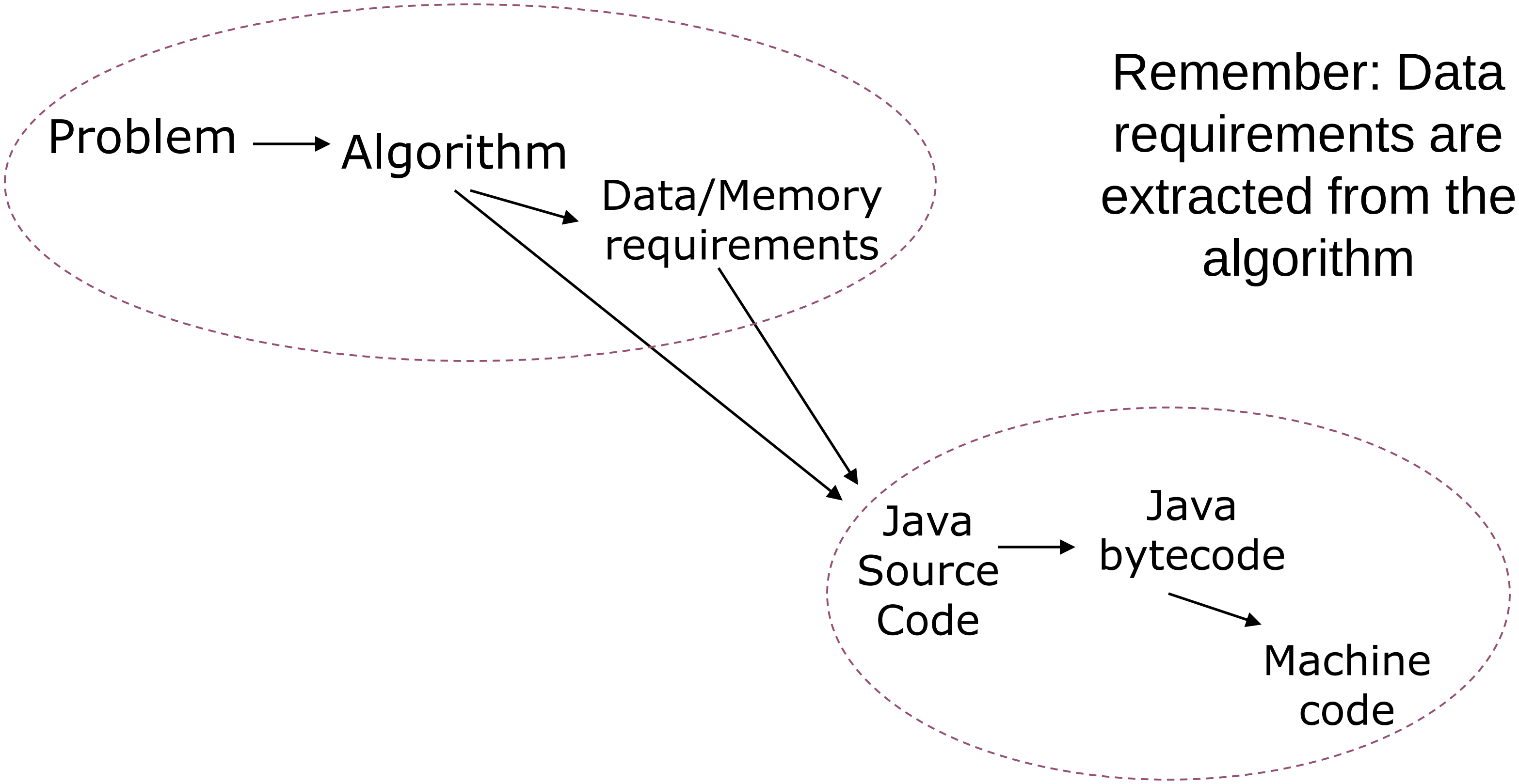
Java Coding

Syntax for Variables & Constants
Input, Output and Assignment
a complete Java program
data representations

From problem to program…

- The story so far…



Remember: Data requirements are extracted from the algorithm

Need Java Syntax for…

- Algorithm *(in pseudo-code)*
    - Sequence, Decision & Repetition, of
    - Data flow operations
        - Input, Output & Assignment

- Data/Memory requirements
    - Meaningfully named memory locations
    - Restriction on data (data types)
    - Variables or Constants & initial value

- Plus comments & methods!

Comments & White space

- *Comments Syntax*:
  - // any text on remainder of current line
  - /* any text across multiple lines */

- Examples:
  - // Author: David.
    // Date: Oct. 2002
  - /*
        This program
        blah, blah,
        blah
     */
- javadoc comments
  - /** description followed by any text, include tags such @author & @version, until */

Comments & White space

Layout program code for ease of reading!

- Java ignores line endings, blanks lines & white space!
- Program can be written on a single line, one word per line or (almost) however you want!
- Use blank lines & indentation (space or tab characters) to layout as per program logical structure

Identifiers

- User-defined names
  - Used for variables, constants, methods, etc.
  - Any sequence of letters, digits and the underscore character only.
  - First character may not be a digit!
  - Upper and lower case are considered different (i.e. case sensitive!)
  - Cannot use Java reserved words
    - i.e. words such as *while*, *for*, *class*, *if*, etc.

CS101 rule: Names must be meaningful!

Identifiers

- “Dog” is not the same identifier as “dog”
- Identifiers such as String, System, out, etc. are not reserved words and could be used
  - BUT doing so might make your program very confusing since you are redefining commonly used terms.

CS101 rule: Names must be meaningful!

Data Types

- For now, use only the following…

- Primitive
    - **int** *(for numeric integer, e.g. 5, -27, 0, 510…)*
    - **double** *(for numeric real, e.g. 5.75, 3.0, -2.6…)*
    - **char** *(for any character, e.g. A, a, B, b, 3, ?, &, …)*
    - **boolean** *(for true / false only)*

- Non-primitive
    - **String** *(for any sequence of zero or more characters e.g. "CS101", "A", "Well done!", …)*

Declaring Variables

- *Syntax*:

```
type name;
```

Notice semicolon (missing it is syntax error!)

- Type
  - Any Java type
- Name *(identifier)*
  - Convention:
    first letter of embedded words capital, except first!

- Examples:
  - int age; double area; long initialSpeed;
  - char letterGrade; char lettergrade;
  - boolean exists;

CAUTION Java is case sensitive!

Declaring Variables

- Variable names cannot have spaces
  - A name such as “speed of sound” or “sum of grades so far” cannot be used.
- Resolve by
  - replacing spaces with “_” (not normally used in Java) or
  - removing spaces (which would make reading it difficult)
  - capitalise first letter of each embedded word, except first
  - E.g. “speedOfSound” & “sumOfGradesSoFar”

CAUTION
Java is case sensitive!

Declaring Constants

- *Syntax*:

```
final type name = value;
```

- Type
  - Any Java type
- Name *(identifier)*
  - Convention: all capital letters (& underscore!)
- Value *(literal, variable, constant, expression)*

- Examples:
  - final int SPEEDOFLIGHT = 300;
  - final float PI = 3.142;
  - final String COMPANY = “Bilkent”;
  - final char LETTER_GRADE = ‘A’;

Literal values
String use “…”
char use ‘.’

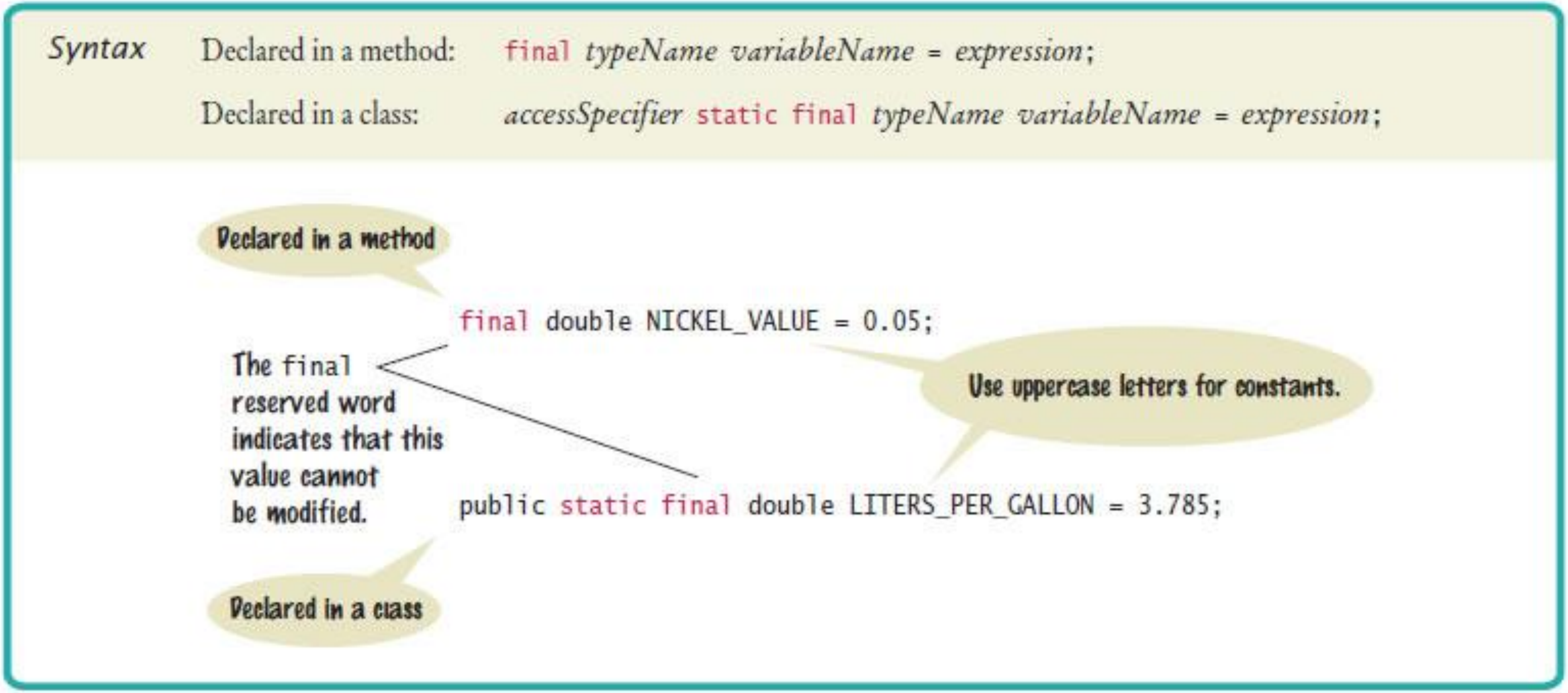
Declaring Constants

- We may also declare constants as static,
  - This requires them to be defined in the class, not the main method.
  - Advantage:
    - If a constant is not static, Java will allocate a memory for that constant in every object of the class (i.e., one copy of the constant per object).
    - If a constant is static, there will be only one copy of the constant for that class (i.e., one copy per class).
  - If the constant has only one value, it should declared static
  - If the constant might have different value for each object, for example the creation time of the object, it should not be declared static

- Naming constants makes maintenance easier
  - Which of these three alternatives would be better if we wished to update PI to 3.0

- PI = 3.142; & circumference = 2 * PI * radius;
- circumference = 2 * 3.142 * radius;
- circumference = 6.284 * radius;

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

Output (1)

- *Syntax*:

```
System.out.println( output );
```

- where output is
  - Literal value
    eg. “The area is “, ‘?’, 12.5, …
  - Named variable or constant
    eg. area, userName, TAXRATE, …
  - Expression
    eg. 2 * PI * radius,
    “The area is “ + area

Value is output exactly as is!

Value in named memory location is output

Resulting value of expression is output

Note use of + for string concatenation

Output (2)

- Use

```
System.out.print( output );
```

To output the value & leave text cursor on current line.

```
System.out.println( “Welcome to CS101”);
System.out.println( “The tax rate is “ + TAXRATE + ‘%’);
```

```
System.out.println( “Welcome to CS101”);
System.out.print( “The tax rate is “);
System.out.print( TAXRATE);
System.out.println( ‘%’);
```

```
System.out.println();
```

Output blank line!

Output (3)

- How can we display double quotes as part of output?
- Problem since they terminate string literal!
- Use escape sequence \”
- but then how about the back slash character?
- Again use \\
- Look at book for others

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:

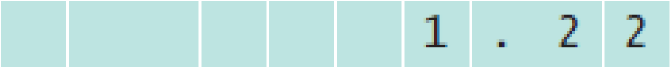
```
System.out.printf("%.2f", price);
```

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

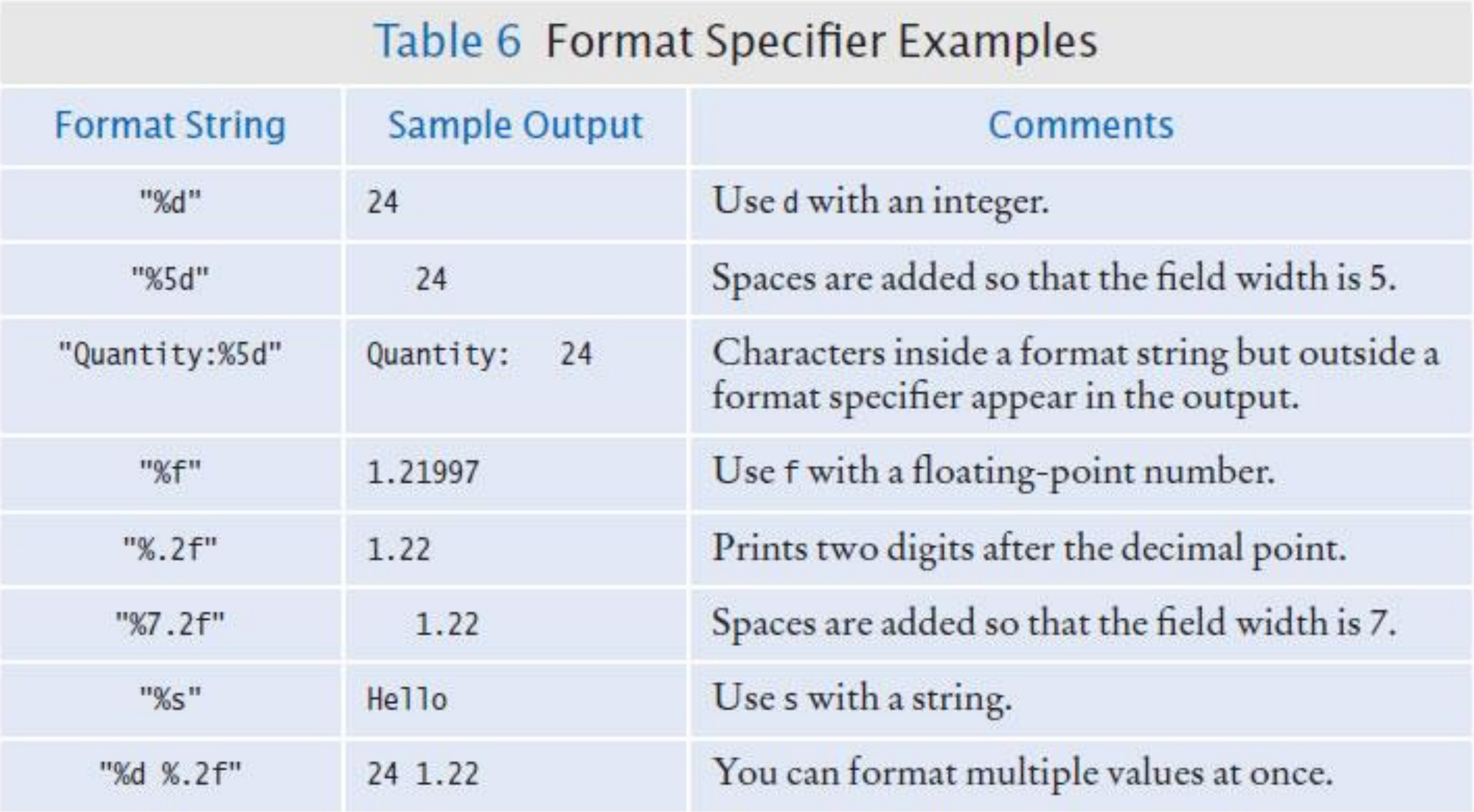


Table 6 Format Specifier Examples

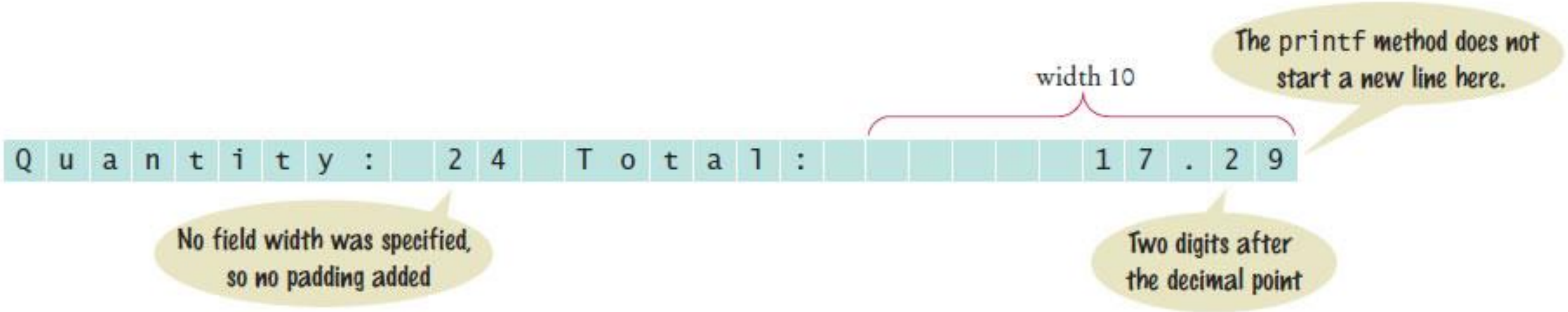
Format String	Sample Output	Comments
`"%d"`	`24`	Use d with an integer.
`"%5d"`	`   24`	Spaces are added so that the field width is 5.
`"Quantity:%5d"`	`Quantity:   24`	Characters inside a format string but outside a format specifier appear in the output.
`"%f"`	`1.21997`	Use f with a floating-point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



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

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

Outline Java Program

- The CS101 console template…

ClassName.java

```
import java.util.Scanner;

/** …description…
    @author …yourname…
    @version 1.00, date
*/
public class ClassName {

  public static void main( String[] args) {
     Scanner scan = new Scanner( System.in);

     // constants

     // variables

     // program code

  }
}
```

Scanner class imported for you!

Header comment is in javadoc format!

Imports must be before this

In Java
  program = class

Classname
Convention:
first letters capitalised

Filename & classname MUST be the same.

Input

- *Syntax*:

```
StringVariable = scan.next();
```

```
intVariable = scan.nextInt();
```

```
doubleVariable = scan.nextDouble();
```

- Examples

```
userName = scan.next();
age = scan.nextInt();
salary = scan.nextDouble();
str = scan.nextLine();
```

Variables must be declared before use

- Standard from Java5.0 on
- Invalid input may give run-time error!
- Program must include:
  - import java.util.Scanner;
  - Scanner scan = new Scanner(System.in);

Input

- Program waits for user to press ENTER (end of line)
- Then takes the user input and stores it in the specified variable.

- Scanner splits input stream up at whitespace boundaries by default
  - next() gets a word, and nextInt() may leave text on current line
  - nextLine() gets rest of current line
- May use
  - scan.useDelimiter(System.getProperty("line.separator"));
  - To break on line rather than on whitespace boundaries

- Common problem:
  - User enters Turkish values, but machine is in English locale for example, 23,75 vs. 23.75

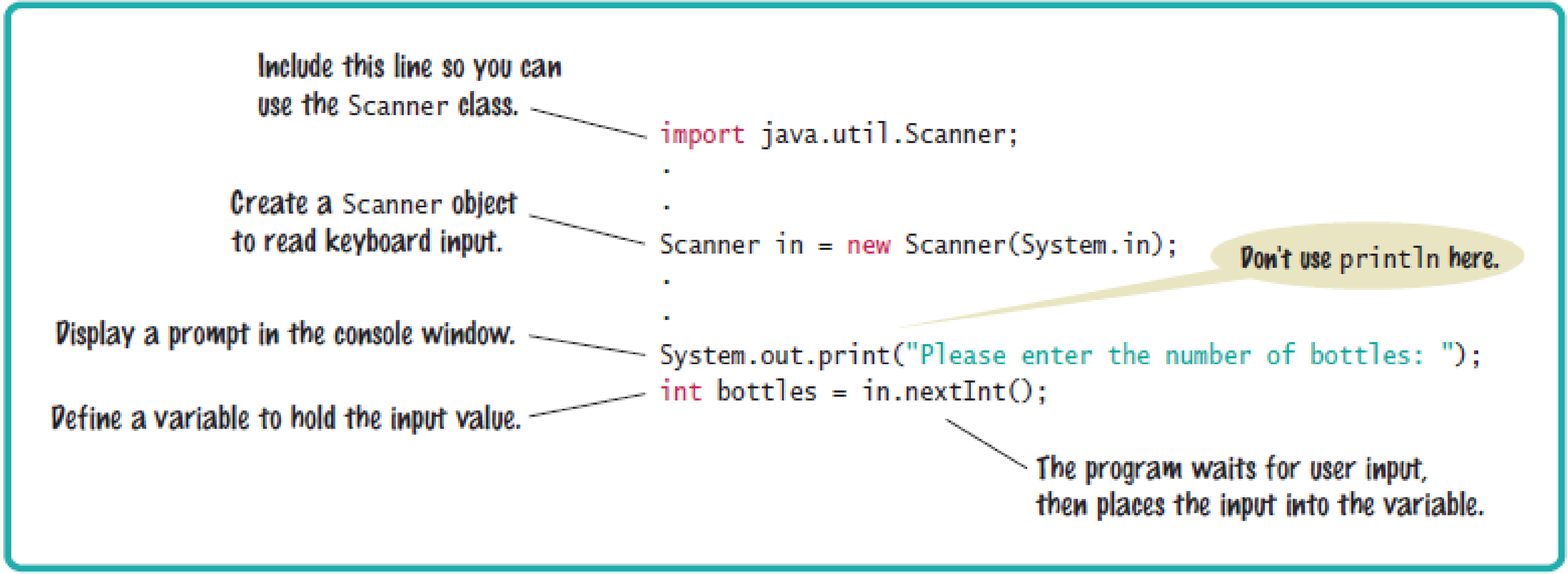
Input

- Scanner class includes hasNext(), hasNextInt(), hasNextDouble(), etc.
- Can’t use “scan.nextChar()” use “scan.nextLine().charAt(0)” instead

- Problem of reading String after number,
  - Eg. i = scan.nextInt(); s = scan.nextLine();
  - Will usually give an empty String s!
  - Add extra scan.nextLine(); inbetween so as to consume \n character

- Can also use to read from string, files or url’s
- But may need exception handling, for example “throws java.io.IOException” added to method (usually main)

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.

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.

Assignment

… is assigned the result of …

- *Syntax*:

```
resultVariable = expression;
```

Result of expression must be of suitable type to put into resultVariable!

- where expression is
  - operand or
  - operand operator operand
- &
  - Operand is
    - Literal value
    - Named Variable or constant
    - Result of method call
    - Expression *(can use brackets to disambiguate)*!
  - Operator is
    - +, -, *, /, % *(modulus, remainder after integer division)*

Assignment

… is assigned the result of …

- Examples

```
total = 0;          x = y;
sum = firstNumber + secondNumber;
netPay = grossPay * ( 1 - TAX_RATE/100);
count = count + 1;
c = Math.sqrt( a * a + b * b );
```

- What is the result of this?

```
   4 + 2 / 3 - 1
```

- Evaluation rules
  - Bracketed sub-expressions first
  - Operator precedence (* / % *before* + -)
  - Left to right

Assignment - compatibility

- d = i;   // ok
- but i = d; // not ok!
- can force with i = (int) d; // typecast!

- Type compatibility
  - Can put narrower types into wider ones
  - Usually no danger of loss (eg. int into long or double)
  - Going the other way is dangerous (serious loss of info.)
  - Compiler issues error message.

- Can use type cast to force compiler to accept
- Useful when dividing two ints since result type is int not real!
  - double = (double) int/int;

Self Check

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`.

CS101 console template

- The CS101 console template…

ClassName.java

```
import java.util.Scanner;

/** …description…
    @author …yourname…
    @version 1.00, date
*/
public class ClassName {

  public static void main( String[] args) {
     Scanner scan = new Scanner( System.in);

     // constants

     // variables

     // program code

  }
}
```

In Java
  program = class

Classname
Convention:
first letters capitalised

Filename & classname
MUST be the same.

A Complete Example (1)

- Problem – find area & circumference…
- Algorithm

1. Print welcome message
2. Ask for & get radius from user
3. Compute area as pi.radius.radius
4. Compute circumference as 2.pi.radius
5. Report area, circumference & radius

- Data requirements

L radius - int
L area, circumference - double
PI – double, constant = 3.142

A Complete Example (2)

AreaCircum.java

```
import java.util.Scanner;

/** …description…
    @author …yourname…
    @version 1.00, 2005/10/07
*/

public class AreaCircum {

    public static void main( String[] args) {

        // constants


        // variables


        // 1. Print welcome message
        // 2. Ask for & get radius from user
        // 3. Compute area as pi.radius.radius
        // 4. Compute circumference as 2.pi.radius
        // 5. Report area, circumference & radius
    }
}
```

A Complete Example (3)

AreaCircum.java

```
import java.util.Scanner;

/**
* AreaCircum - computes area & circum of circle given radius
*
* @author  David
* @version 1.00, 2005/10/07
*/
public class AreaCircum
{

   public static void main( String[] args)
   {
      // constants
      final double PI = 3.142;

      // variables
      int      radius;
      double   area;
      double   circumference;
```

Header has been edited to include program description & author name

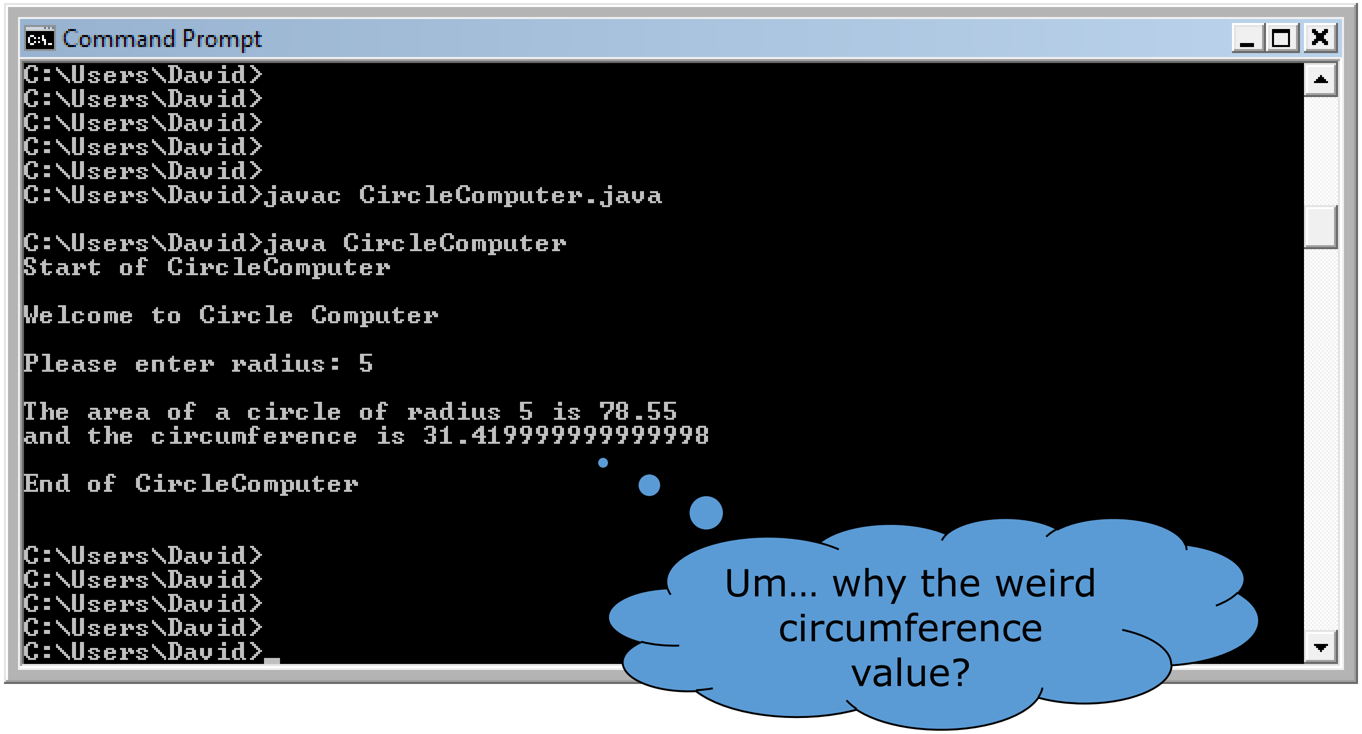
A Complete Example (3)

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

      // 1. Print welcome message
      System.out.println( "Welcome to area circumference finder.");

      // 2. Ask for & get radius from user
      System.out.print( "Please enter the radius: ");
      radius = scan.nextInt();

      // 3. Compute area as pi.radius.radius
      area = PI * radius * radius;

      // 4. Compute circumference as 2.pi.radius
      circumference = 2 * PI * radius;

      // 5. Report area, circumference & radius
      System.out.print( "The area of a circle of radius ");
      System.out.print( radius);
      System.out.print( " is ");
      System.out.println( area);
      System.out.print( "and its circumference is ");
      System.out.print( circumference);
      System.out.println();
   }

} // end of class AreaCircum
```

Template line required for Keyboard input.

Steps 2 & 5 expanded as per original algorithm.

Compile & Run…



```
C:\Users\David>
C:\Users\David>
C:\Users\David>
C:\Users\David>
C:\Users\David>
C:\Users\David>javac CircleComputer.java

C:\Users\David>java CircleComputer
Start of CircleComputer

Welcome to Circle Computer

Please enter radius: 5

The area of a circle of radius 5 is 78.55
and the circumference is 31.419999999999998

End of CircleComputer


C:\Users\David>
C:\Users\David>
C:\Users\David>
C:\Users\David>
C:\Users\David>_
```

Um… why the weird circumference value?

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

Testing…

- It compiled & ran, but…**is it correct?**
- How can you tell?
  - Enter input & check results it outputs
    (e.g. radius 5 → area 78.55 & circumference 31.42)

    are these actually the right answers?
- Really need more input/output sets
  - what input values should we use?
  - & how many do we need?
- Thinking about testing during design can help produce better programs!

Testing…

- Trying every possible value for the input is a waste of time
- Choose some key values
  - a couple of normal cases, e.g. 5 & 10
  - any special cases, e.g. 0
  - unusual / exceptional cases, e.g. -2 & 5.25
  - really bad cases, e.g. xyz
- Don’t assume the user will give you sensible values
- What you do in such exceptional cases is a matter for the customer/software producer to decide
  - You may just leave it to generate a run-time error (exception) or
  - Handle it and output a nice user-friendly error message

Edsgar Djkstra (Dutch computer scientist. Turing award winner 1972, died 2002):
“Program testing can be used to show the presence of bugs, but never to show their absence”

Data & DATA Types

Data Types

- For now, use only the following…

- Primitive
  - **int** *(for numeric integer, e.g. 5, -27, 0, 510…)*
  - **double** *(for numeric real, e.g. 5.75, 3.0, -2.6…)*
  - **char** *(for any character, e.g. A, a, B, b, 3, ?, &, …)*
  - **boolean** *(for true / false only)*

- Non-primitive
  - **String** *(for any sequence of zero or more characters e.g. “CS101”, “A”, “Well done!”, …)*

Numeric representations

Number bases

- 583_{10}

  $5.10^2 + 8.10^1 + 3.10^0$

- 417_8

  $4.8^2 + 1.8^1 + 7.8^0$

- 110_2

  $1.2^2 + 1.2^1 + 0.2^0$

Base 2 - binary

0
1

00
01
10
11

000
001
010
011
100
101
110
111

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

- digits 0 & 1
- 2^n values
- 0 → $(2^n – 1)$

Characters…

- Coding

0000 — ‘a’	1000 — ‘0’
0001 — ‘b’	1001 — ‘1’
0010 — ‘c’	1010 — ‘2’
0011 — ‘d’	1011 — ‘3’
0100 — ‘+’	1100 — ‘ ’
0101 — ‘-’	1101 — ‘x’
0110 — ‘*’	1110 — ‘y’
0111 — ‘/’	1111 — ‘z’

- Size…?

‘A’.. ‘Z’	→ 26
‘a’.. ‘z’	→ 26
‘0’.. ‘9’	→ 10
punc.	→ ??

62

$2^6 = 64$
$2^7 = 128$
$2^8 = 256$

Standard Codes

- ASCII
  - 8 bit
  - 128 characters,
  - English only!
- UNICODE
  - 16 bit
  - first 128 characters same as ASCII
  - All languages!

Data Types

- Primitive
  - **byte, short, int, long** *(numeric integer)*
  - **float, double** *(numeric real)*

  - **char** - any character, e.g. A, a, B, b, 3, ?, &, …
    (Java uses ISO Unicode standard, 16 bit/char)
  - **boolean** - true / false

- Non-primitive
  - **String** - any sequence of zero or more characters
  - **enum** – an ordered set of user-defined values

  - anything & everything else!
    (we will come to these shortly)

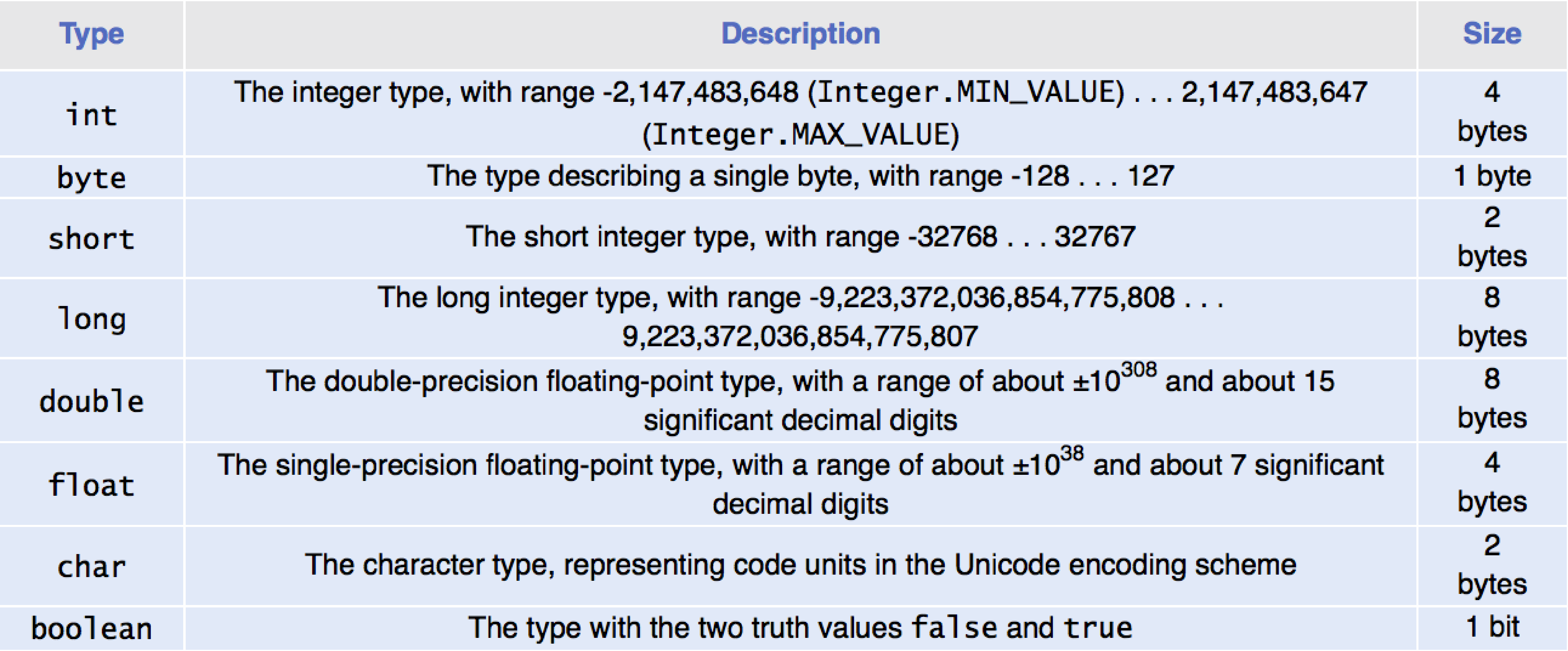
Number Types

- 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 Numeric Types

	Type	Storage	Min Value	Max Value
integer	byte	8 bits	-128	127
	short	16 bits	-32,768	32,767
	int	32 bits	-2,147,483,648	2,147,483,647
	long	64 bits	-9 x 10^{18}	9 x 10^{18}
real	float	32 bits	$\mp$ 3.4 x $10^{\mp 38}$	7 significant digits
	double	64 bits	$\mp$ 1.7 x $10^{\mp 308}$	15 significant digits

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

Primitive Numeric Types

- Integer types stored as binary (base two) values
- Actually 2’s complement… why?

- So that addition doesn't need to have any special logic for dealing with negative numbers
- Say you have two numbers, 2 and -1
- Intuitive way of representing numbers, would be 0010 and 1001
- In the two's complement way, they are 0010 and 1111
- Two's complement addition is very simple
  - Add numbers normally and any carry bit at the end is discarded
  - 0010 + 1111 =10001 = 0001 (discard the carry)
  - 0001 is 1, which is what we expected
- But in intuitive method, adding is more complicated:
  - 0010 + 1001 = 1011, which is -3

Misc…

- Why so many numeric types?
  - memory, processing time, error, …
- Error in reals?
- Typecasting
  - int into double, but not double into int!
- Overflow/underflow
  - What happens if add one to maxint or subtract one from –maxint?
- Division by zero
- Why not use String for everything?

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

`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`:

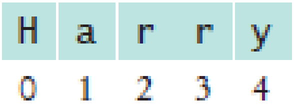
  ```
  System.out.printf("Price: %10.2f\n", price);
  ```

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

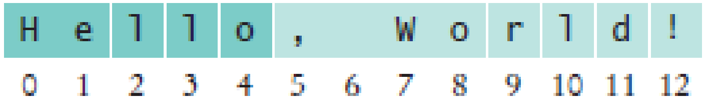
```
String name = "Harry”;
char start = name.charAt(0);
char last = name.charAt(4);
```

  - 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”
```

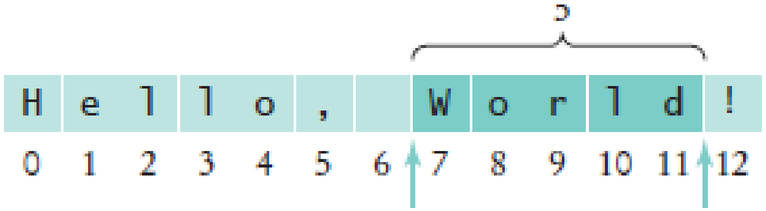


H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

Substrings

- To extract `"World"`

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



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

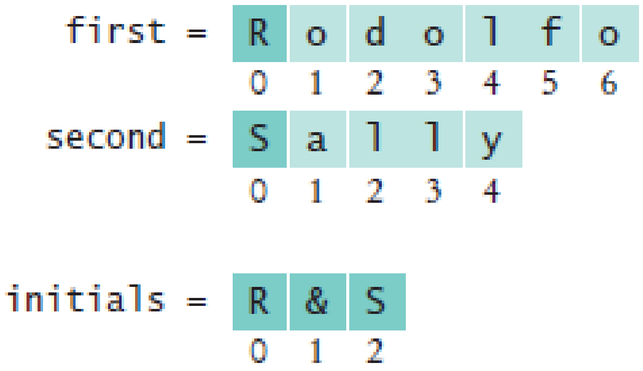


Figure 3 Building the initials String

String Operations

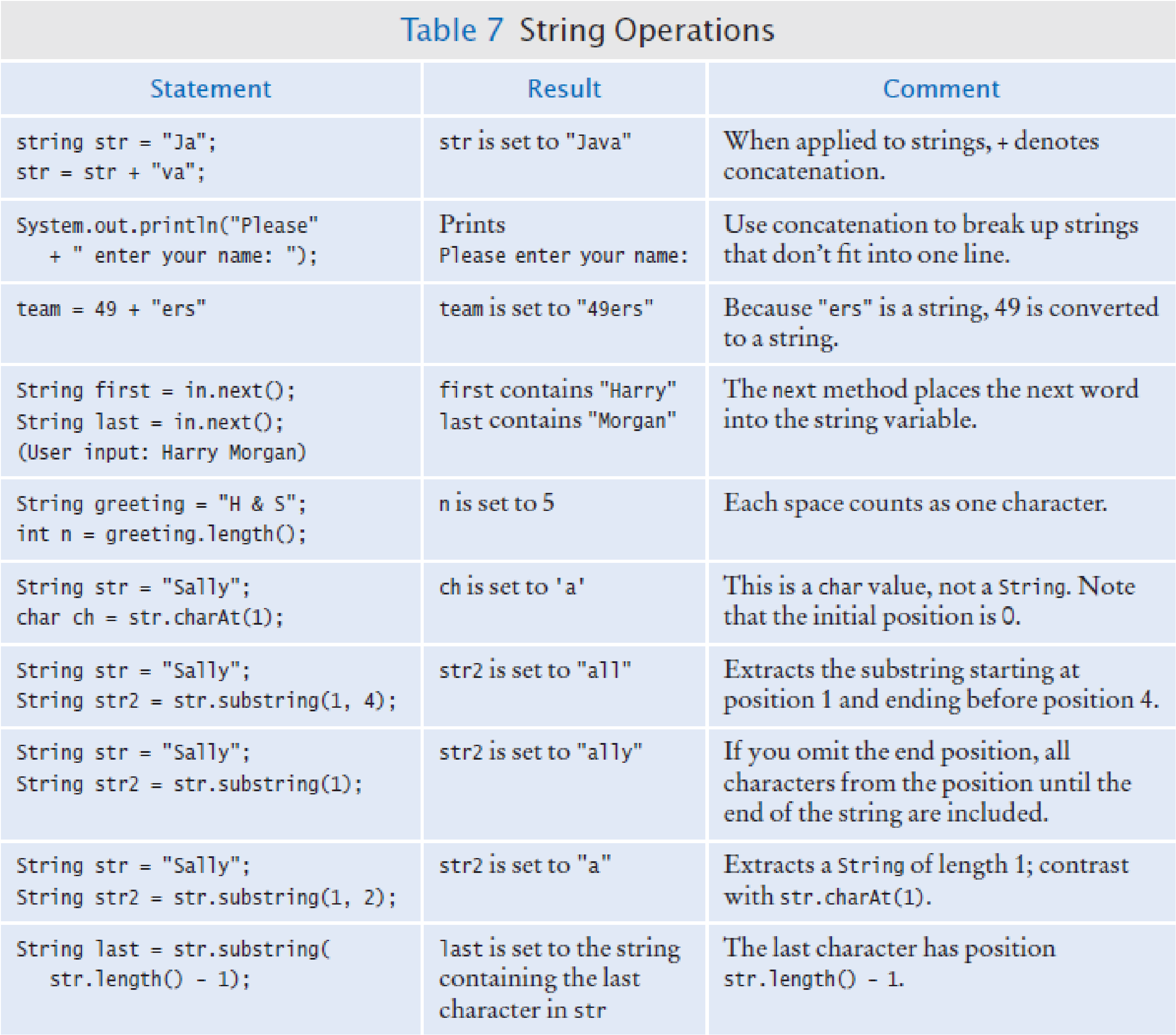


Table 7 String Operations

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`.

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
```

- 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
```

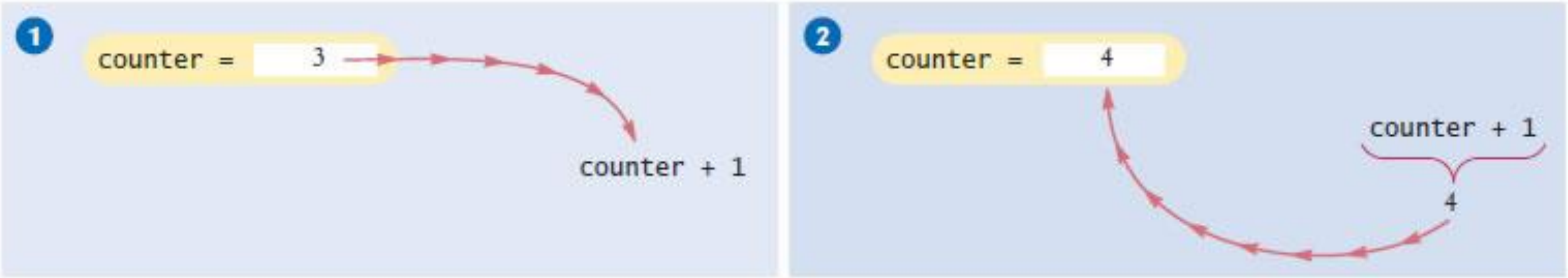


Figure 1 Incrementing a Variable

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

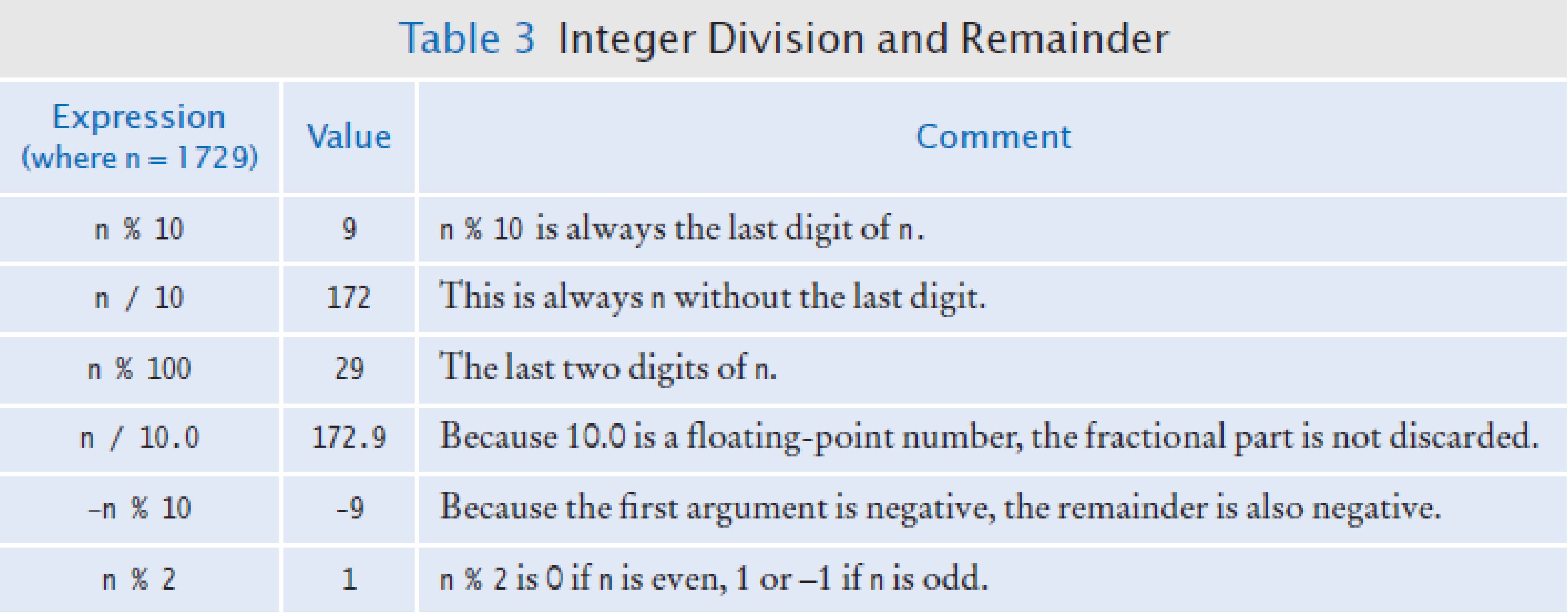


Table 3 Integer Division and Remainder

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)
```

Mathematical Methods

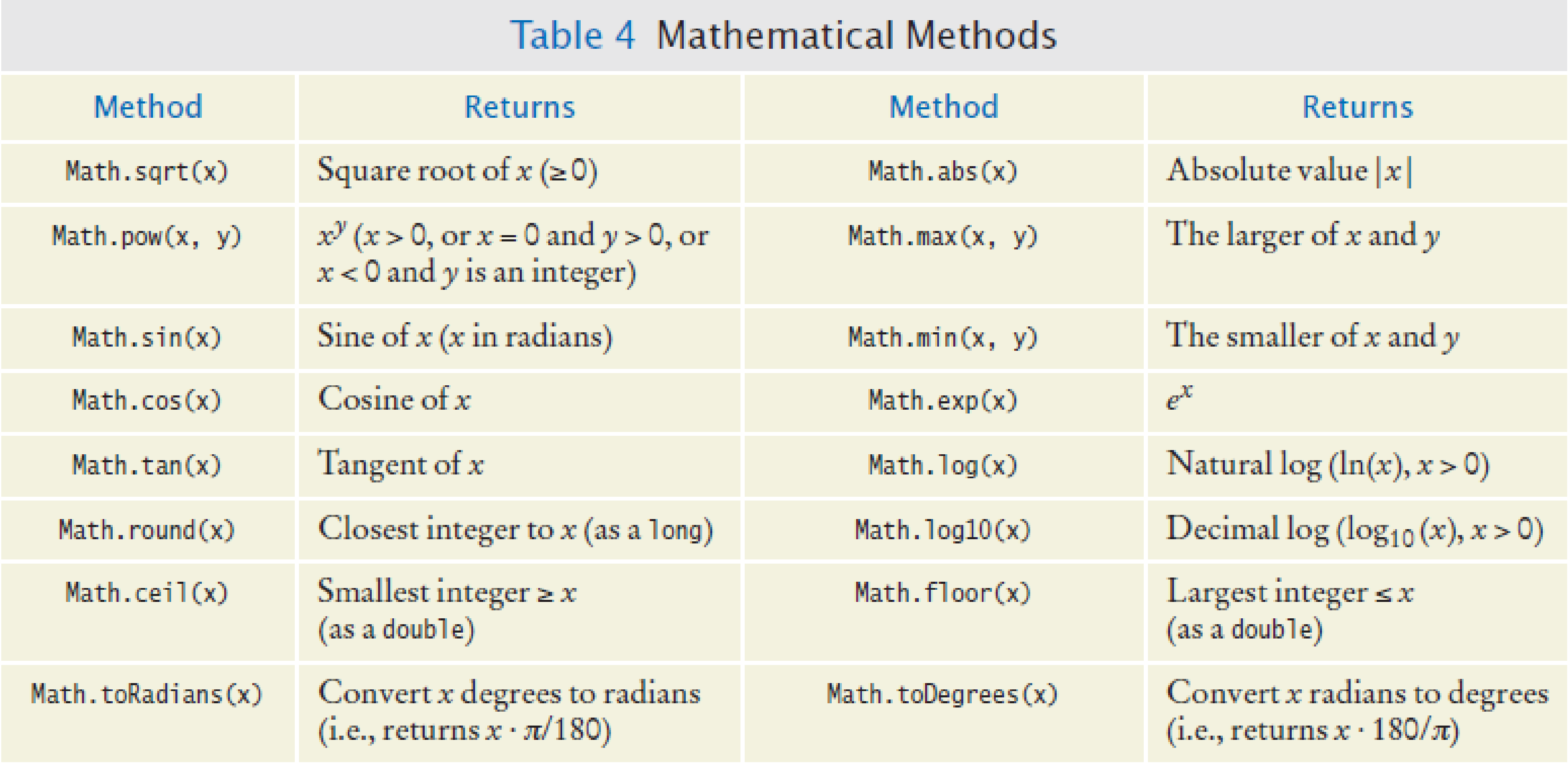


Table 4 Mathematical Methods

Method	Returns	Method	Returns
`Math.sqrt(x)`	Square root of x (≥ 0)	`Math.abs(x)`	Absolute value $\|x\|$
`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`.