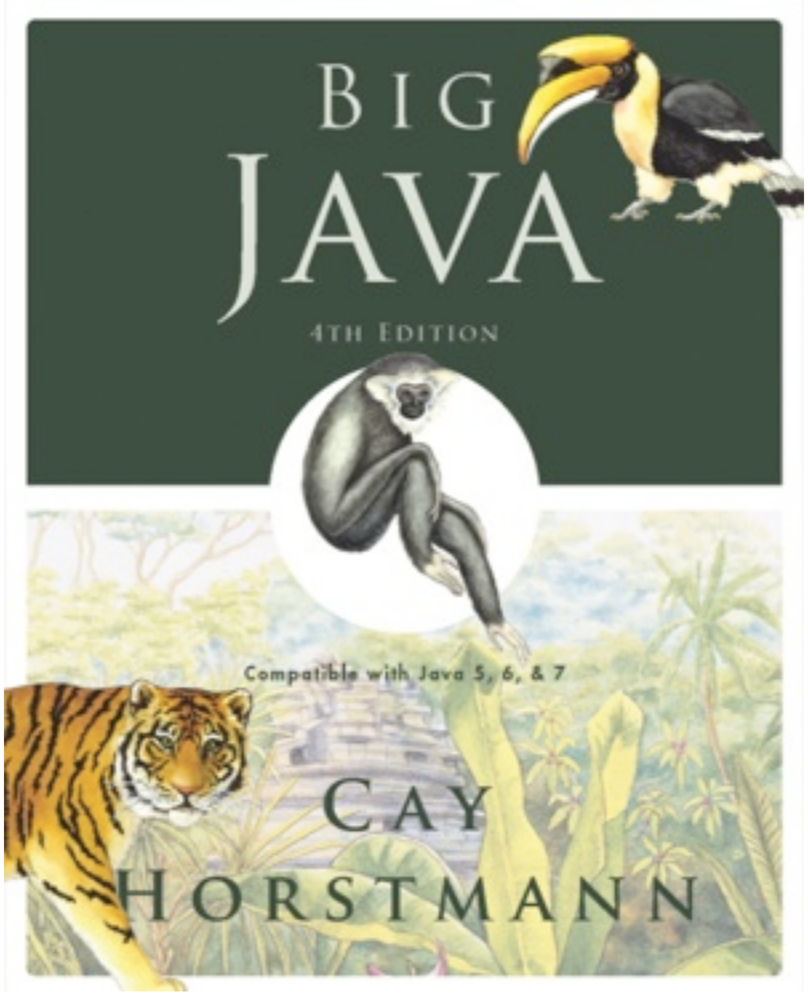


Chapter 4 – Fundamental Data Types

ICOM 4015: Advanced Programming

Lecture 4

Reading: Chapter Four: Fundamental Data Types

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Chapter Goals

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

Key Concepts

- Computer Numbers ≠ Math Numbers
- All computer data is encoded as bit strings
- Use named constants to avoid hardcoding “magic numbers”
- Numbers ≠ Numerals

Number Types

- `int`: integers, no fractional part:

  ```
  1, -4, 0
  ```

- `double`: floating-point numbers (double precision):

  ```
  0.5, -3.11111, 4.3E24, 1E-14
  ```

- A numeric computation overflows if the result falls outside the range for the number type:

  ```
  int n = 1000000;
  System.out.println(n * n); // prints -727379968
  ```

- Java: 8 primitive types, including four integer types and two floating point types

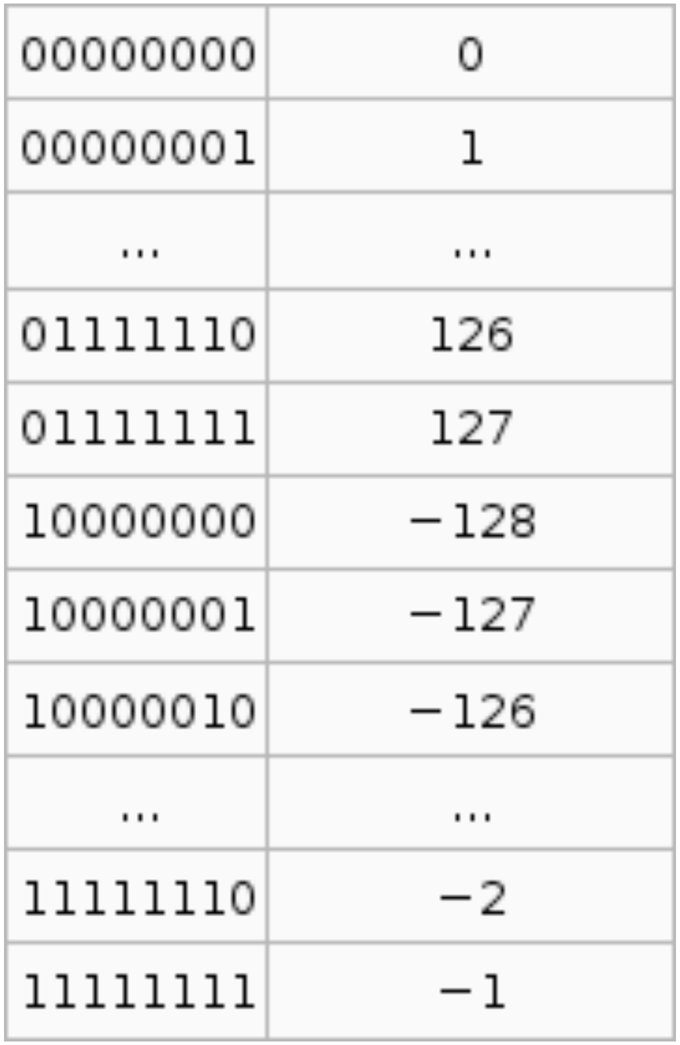
Big Java by Cay Horstmann

Primitive Types

Type	Description	Size
`int`	The integer type, with range -2,147,483,648 . . . 2,147,483,647	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

Big Java by Cay Horstmann

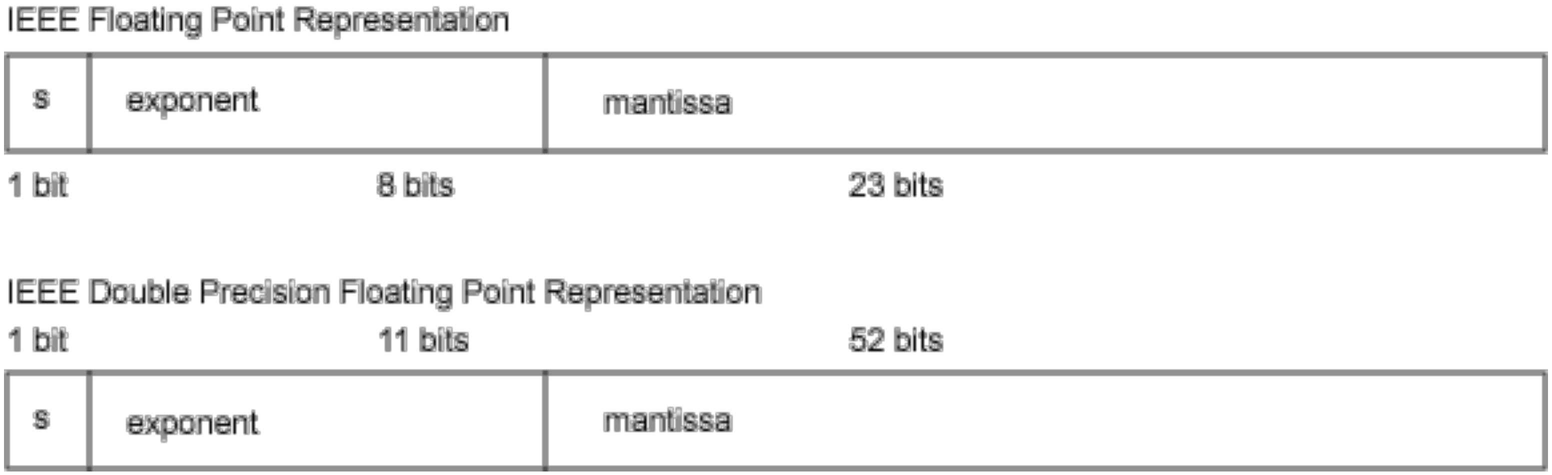
Coding Methods for Numeric Types



00000000	0
00000001	1
...	...
01111110	126
01111111	127
10000000	−128
10000001	−127
10000010	−126
...	...
11111110	−2
11111111	−1

2’s Complement Coding for Positive and Negative Integers

Coding Methods for Numeric Types


IEEE Floating Point Representation
s
exponent
mantissa
1 bit
8 bits
23 bits
IEEE Double Precision Floating Point Representation
1 bit
11 bits
52 bits
s
exponent
mantissa

IEEE 754 Standard for Floating Point Numbers

Number Types: Floating-point Types

- Rounding errors occur when an exact conversion between numbers is not possible:

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

- Java: Illegal to assign a floating-point expression to an integer variable:

```
double balance = 13.75;
int dollars = balance; // Error
```

Self Check 4.1

Which are the most commonly used number types in Java?

Answer: `int` and `double`

Big Java by Cay Horstmann

Self Check 4.2

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

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

Self Check 4.3

Which of the following initializations are incorrect, and why?

```
a. int dollars = 100.0;
b. 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`.

Constants: `final`

- A `final` variable is a constant
- Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: Use all-uppercase names for constants

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

Big Java by Cay Horstmann

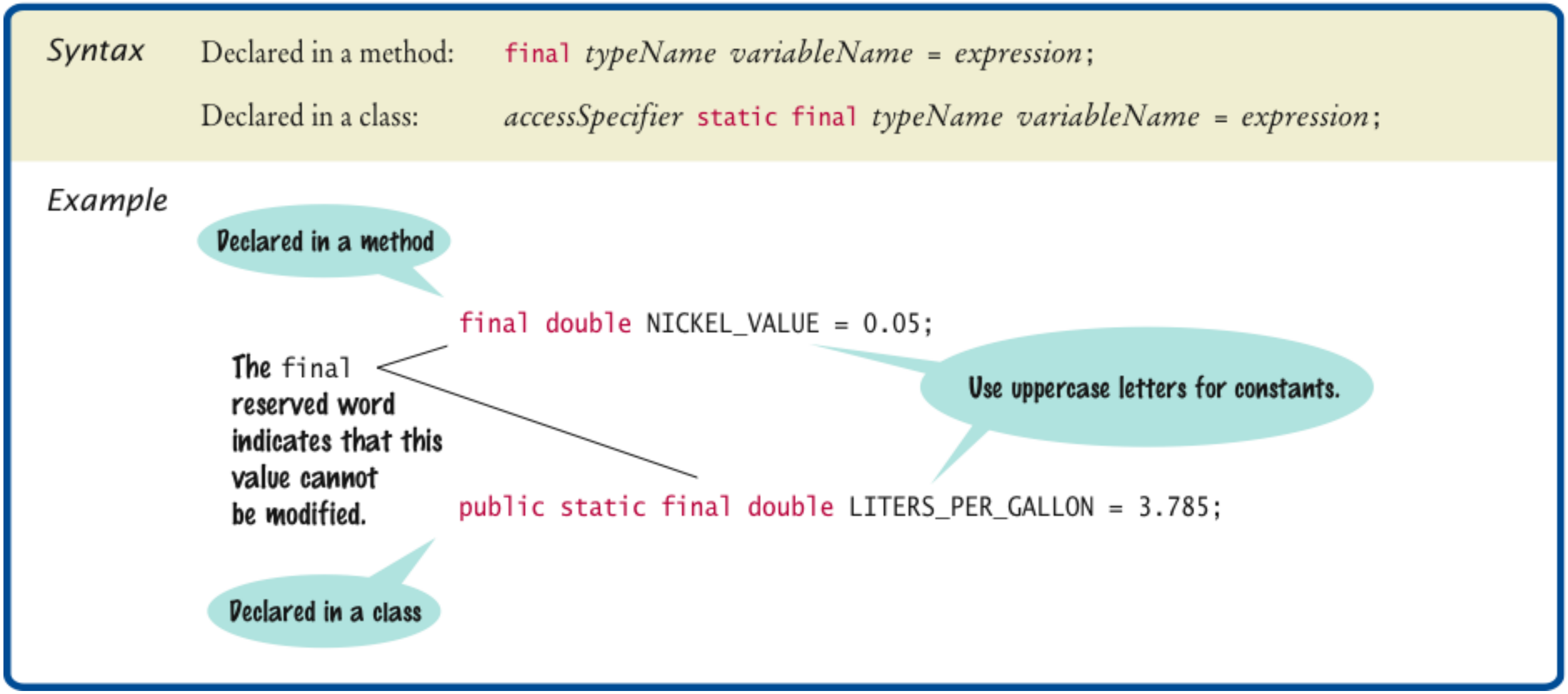
Constants: `static final`

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as `static` and `final`
- Give `static final` constants public access to enable other classes to use them

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

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

Syntax 4.1 Constant Definition



Syntax

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

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

Example

```
final double NICKEL_VALUE = 0.05;

public static final double LITERS_PER_GALLON = 3.785;
```

Declared in a method
Use uppercase letters for constants.
The final reserved word indicates that this value cannot be modified.
Declared in a class

ch04/cashregister/CashRegister.java

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

   private double purchase;
   private double payment;

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

Continued

ch04/cashregister/CashRegister.java (cont.)

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

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

Continued

ch04/cashregister/CashRegister.java (cont.)

```
   /**
      Computes the change due and resets the machine for the next customer.
      @return the change due to the customer
   */
   public double giveChange()
   {
      double change = payment - purchase;
      purchase = 0;
      payment = 0;
      return change;
   }
}
```

ch04/cashregister/CashRegisterTester.java

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

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

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

ch04/cashregister/CashRegisterTester.java (cont.)

Program Run:

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

Self Check 4.4

What is the difference between the following two statements?

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

and

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

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

Self Check 4.5

What is wrong with the following statement sequence?

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

Answer:

1. You should use a named constant, not the “magic number” 3.14.
2. 3.14 is not an accurate representation of π.

Arithmetic Operators

- Four basic operators:
  - *addition:* `+`
  - *subtraction:* `-`
  - *multiplication:* `*`
  - *division:* `/`
- Parentheses control the order of subexpression computation:

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

- Multiplication and division bind more strongly than addition and subtraction:

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

Increment and Decrement

- `items++` is the same as `items = items + 1`
- `items--` subtracts `1` from `items`

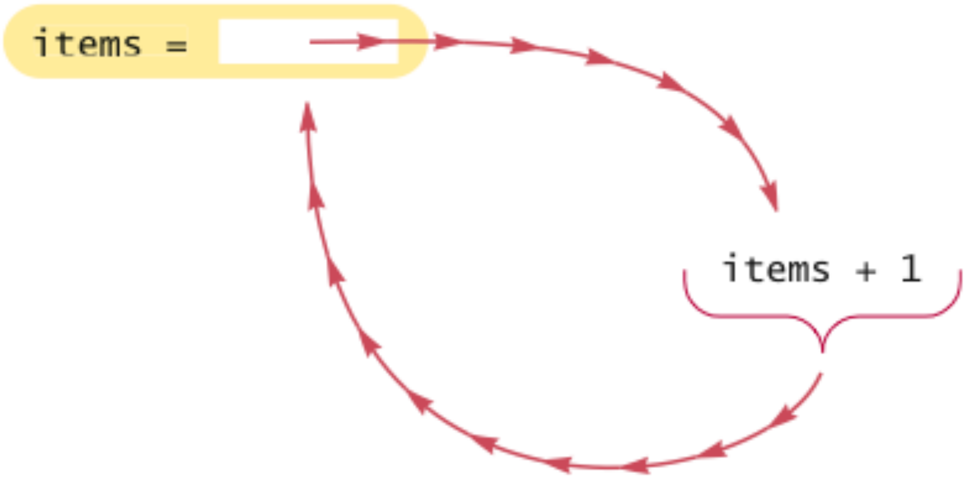


Figure 1 Incrementing a Variable

Big Java by Cay Horstmann

Integer Division

- `/` is the division operator
- If both arguments are integers, the result is an integer. The remainder is discarded
- `7.0 / 4` yields `1.75`
  `7 / 4` yields `1`
- Get the remainder with `%` (pronounced “modulo”)
  `7 % 4` is `3`

Big Java by Cay Horstmann

Integer Division

Example:

```
final int PENNIES_PER_NICKEL = 5;
final int PENNIES_PER_DIME = 10;
final int PENNIES_PER_QUARTER = 25;
final int PENNIES_PER_DOLLAR = 100;

// Compute total value in pennies
int total = dollars * PENNIES_PER_DOLLAR + quarters
   * PENNIES_PER_QUARTER + nickels * PENNIES_PER_NICKEL
   + dimes * PENNIES_PER_DIME + pennies;

// Use integer division to convert to dollars, cents
int dollars = total / PENNIES_PER_DOLLAR;
int cents = total % PENNIES_PER_DOLLAR;
```

Compute Change With Minimal Coins

Powers and Roots

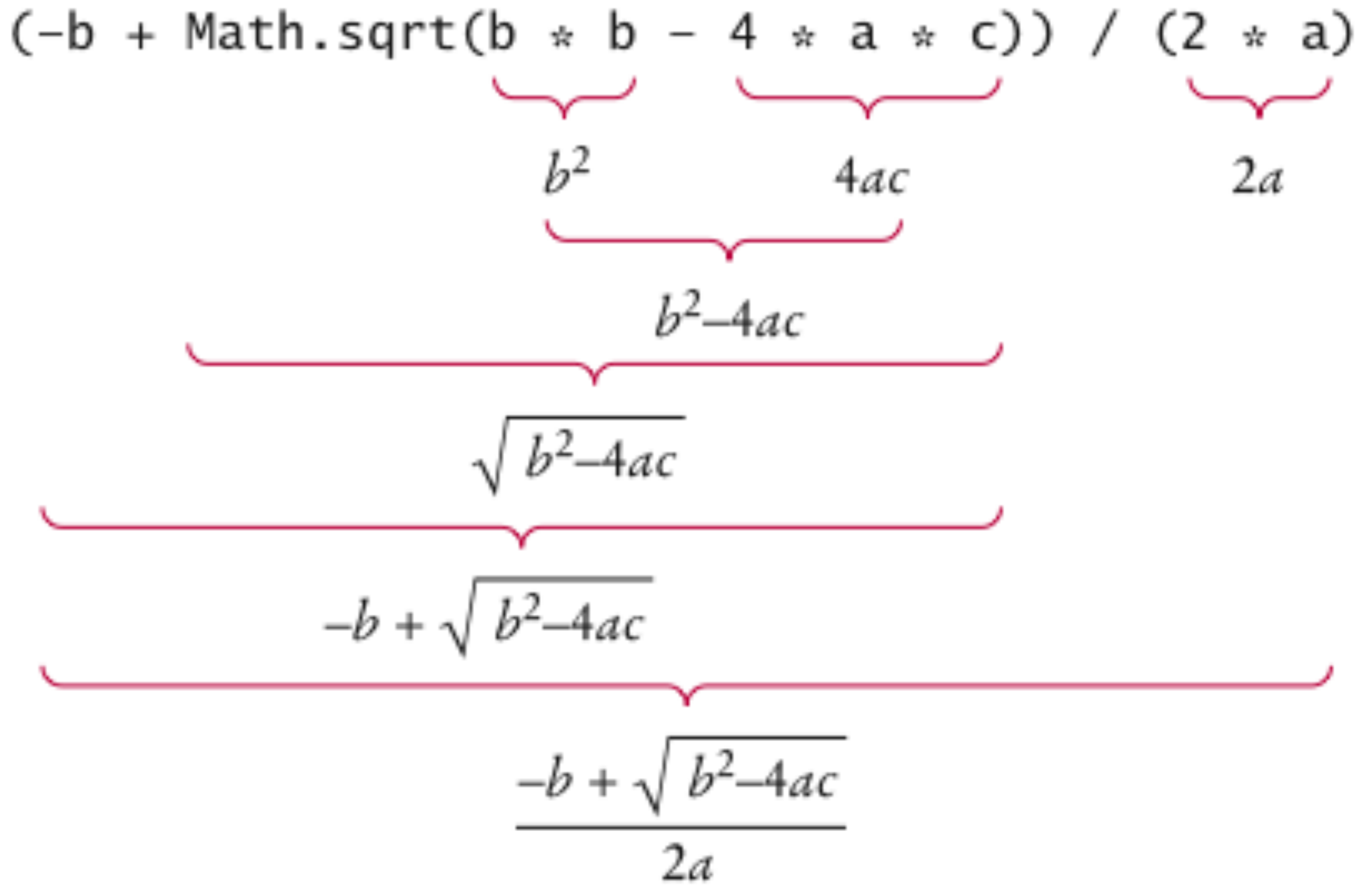
- `Math` class: contains methods `sqrt` and `pow` to compute square roots and powers
- To compute x^n, you write `Math.pow(x, n)`
- However, to compute x^2 it is significantly more efficient simply to compute `x * x`
- To take the square root of a number, use `Math.sqrt`; for example, `Math.sqrt(x)`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

  $$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

  can be represented as

  ```
  (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
  ```

Use Parenthesis to Override Operator Precedence



```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

Figure 2 Analyzing an Expression

Big Java by Cay Horstmann

Some (Static) Methods Available in the Math Built-in Class

Function	Returns
`Math.sqrt(x)`	square root
`Math.pow(x, y)`	power x^y
`Math.exp(x)`	e^x
`Math.log(x)`	natural log

`Math.sin(x), Math.cos(x),`
`Math.tan(x)`
	sine, cosine, tangent
	(x in radians)

`Math.round(x)`	closest integer to x
`Math.min(x, y), Math.max(x, y)`	minimum, maximum

Big Java by Cay Horstmann

Cast and Round

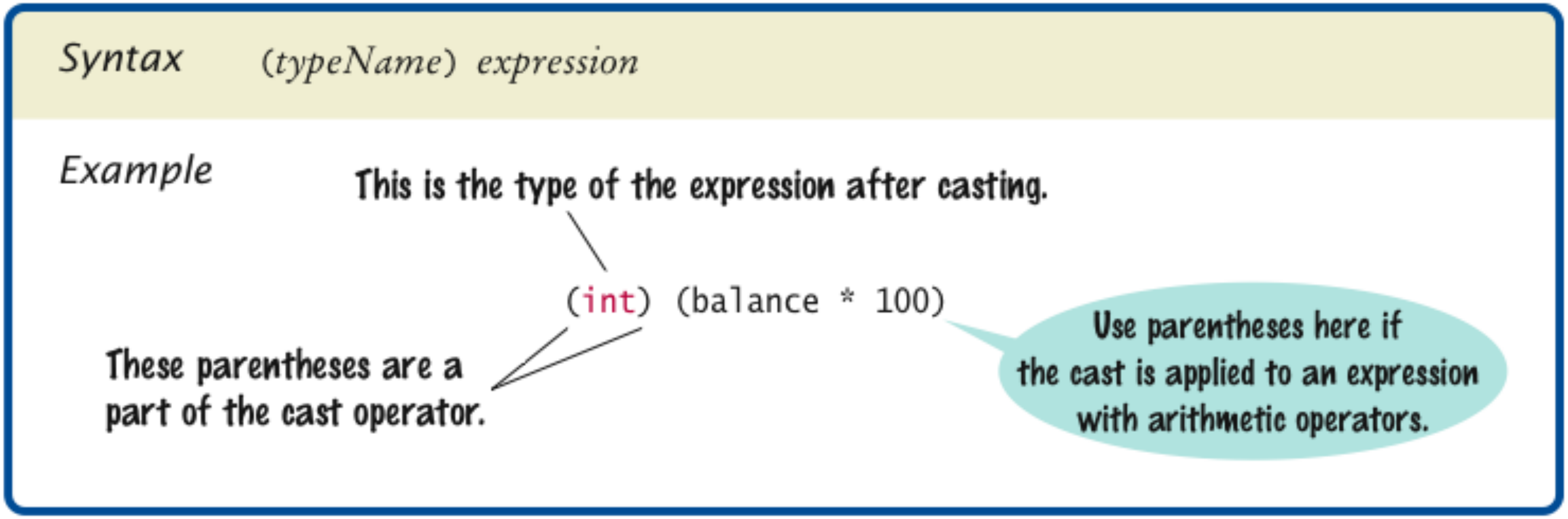
- **Cast** converts a value to a different type:

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

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

```
long rounded = Math.round(balance);
// if balance is 13.75, then rounded is set to 14
```

Syntax 4.2 Cast



Syntax *(typeName) expression*

Example

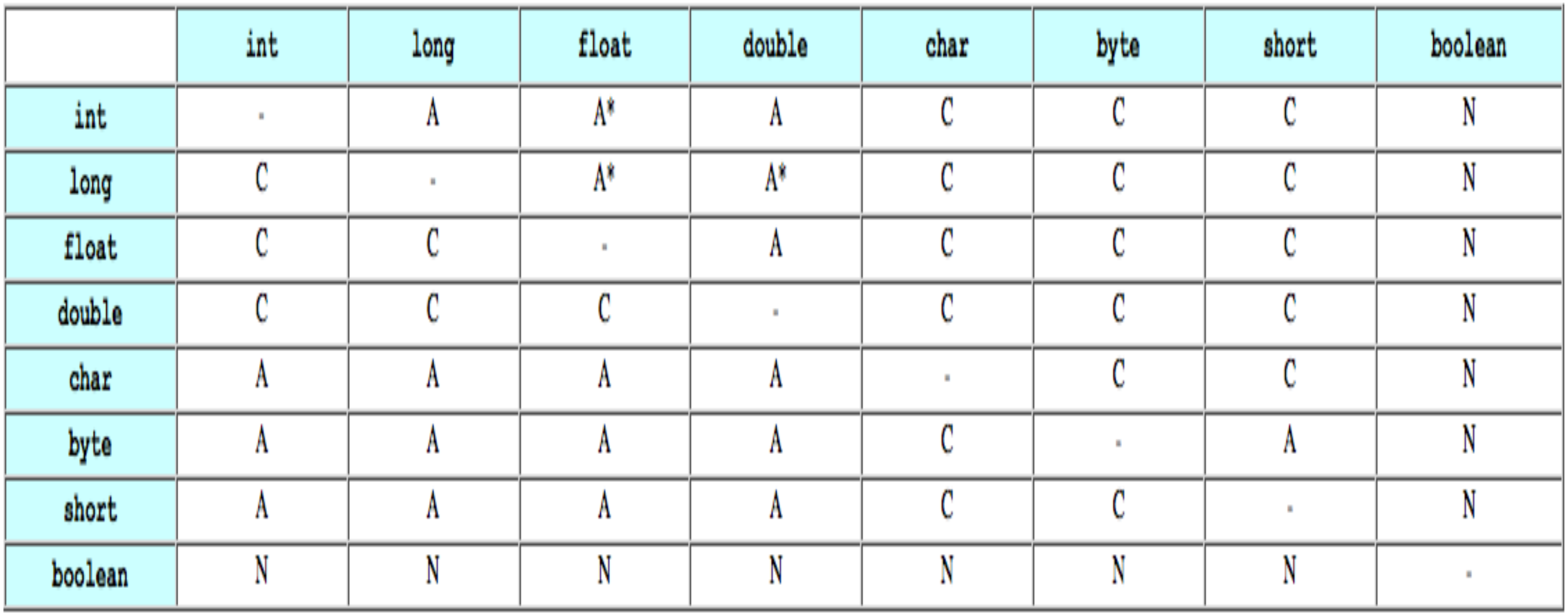
This is the type of the expression after casting.

```
(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.

When is Explicit Casting Required?



	int	long	float	double	char	byte	short	boolean
int	·	A	A*	A	C	C	C	N
long	C	·	A*	A*	C	C	C	N
float	C	C	·	A	C	C	C	N
double	C	C	C	·	C	C	C	N
char	A	A	A	A	·	C	C	N
byte	A	A	A	A	C	·	A	N
short	A	A	A	A	C	C	·	N
boolean	N	N	N	N	N	N	N	·

A = Automatic **C = Required** **N = Not allowed**

Big Java by Cay Horstmann

Arithmetic Expressions

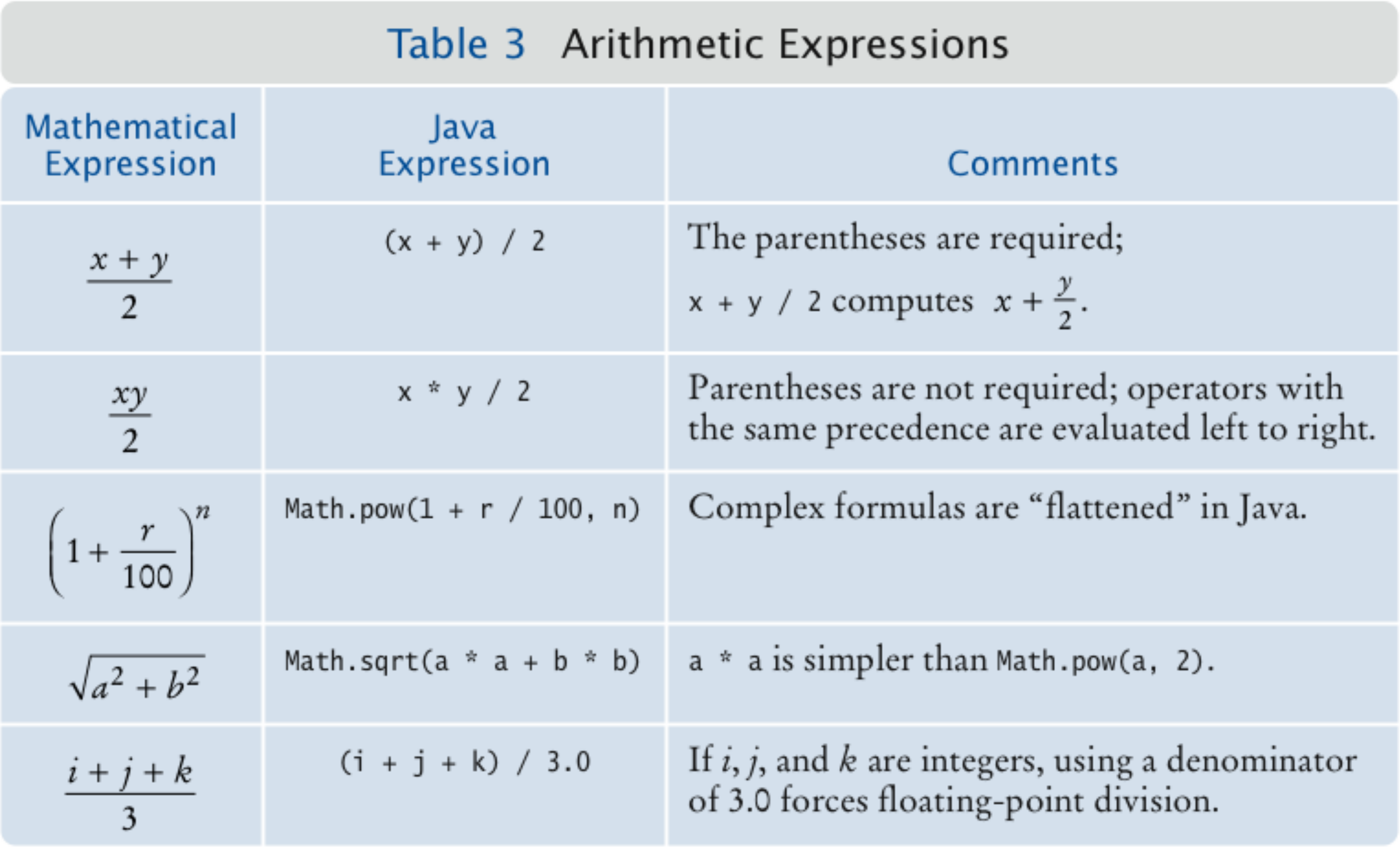


Table 3 Arithmetic Expressions

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

Self Check 4.6

What is the value of `n` after the following sequence of statements?

```
n--;
n++;
n--;
```

Answer: One less than it was before.

Self Check 4.7

What is the value of `1729 / 100`? Of `1729 % 100`?

Answer: `17` and `29`

Self Check 4.8

Why doesn't the following statement compute the average of s1, s2, and s3?

```
double average = s1 + s2 + s3 / 3; // Error
```

Answer: Only s3 is divided by 3. To get the correct result, use parentheses. Moreover, if s1, s2, and s3 are integers, you must divide by 3.0 to avoid integer division:

```
(s1 + s2 + s3) / 3.0
```

Self Check 4.9

What is the value of `Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))` in mathematical notation?

Answer: $\sqrt{x^2 + y^2}$

Big Java by Cay Horstmann

Self Check 4.10

When does the cast `(long) x` yield a different result from the call `Math.round(x)`?

Answer: When the fractional part of `x` is ≥ 0.5

Self Check 4.11

How do you round the `double` value `x` to the nearest `int` value, assuming that you know that it is less than 2 · 109?

Answer: By using a cast: `(int) Math.round(x)`

Calling Static Methods

- A `static` method does not operate on an object

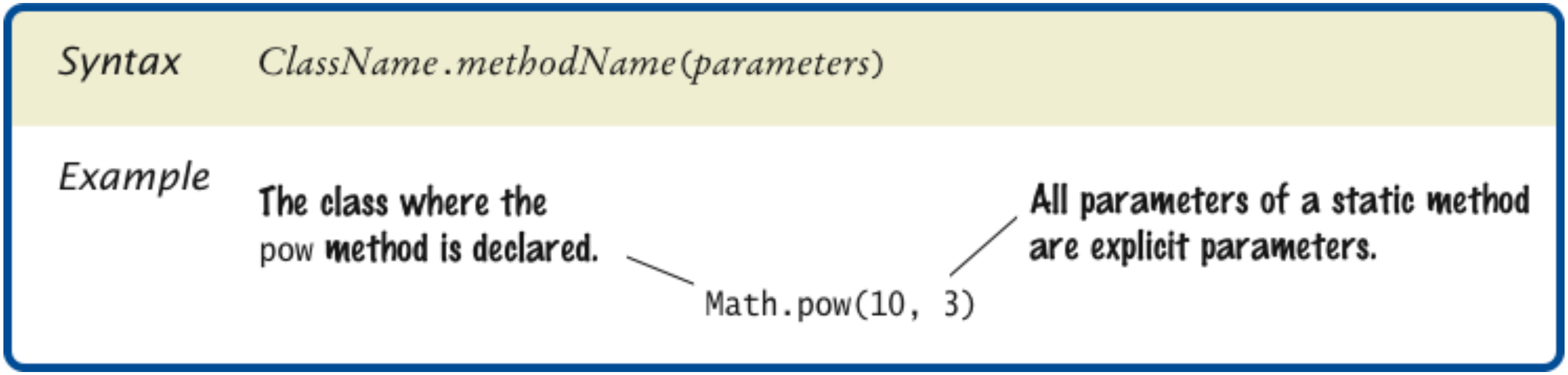
```
double x = 4;
double root = x.sqrt(); // Error
```

- Static methods are declared inside classes
- Naming convention: Classes start with an uppercase letter; objects start with a lowercase letter:

```
Math
System.out
```

Big Java by Cay Horstmann

Syntax 4.3 Static Method Call



Syntax *ClassName.methodName(parameters)*

Example

```
Math.pow(10, 3)
```

The class where the pow method is declared.
All parameters of a static method are explicit parameters.

Self Check 4.12

Why can't you call `x.pow(y)` to compute x^y?

Answer: `x` is a number, not an object, and you cannot invoke methods on numbers.

Self Check 4.13

Is the call `System.out.println(4)` a static method call?

Answer: No – the `println` method is called on the object `System.out`.

Big Java by Cay Horstmann

The `String` Class

- A string is a sequence of characters
- Strings are objects of the `String` class
- A string *literal* is a sequence of characters enclosed in double quotation marks:

  ```
  "Hello, World!"
  ```

- String *length* is the number of characters in the String
  - *Example:* `"Harry".length()` *is 5*
- Empty string: `""`

Big Java by Cay Horstmann

Concatenation

- Use the `+` operator:

```
String name = "Dave";
String message = "Hello, " + name;
// message is "Hello, Dave"
```

- If one of the arguments of the `+` operator is a string, the other is converted to a string

```
String a = "Agent”;
int n = 7;
String bond = a + n; // bond is "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);
```

Converting between Strings and Numbers

- Convert to number:

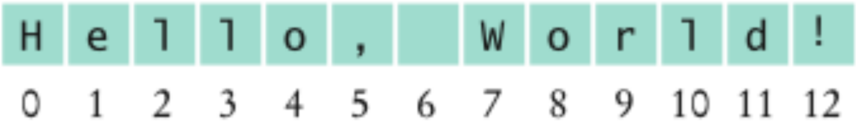
```
int n = Integer.parseInt(str);
double x = Double.parseDouble(string);
```

- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

Substrings

-
  ```
  String greeting = "Hello, World!";
  String sub = greeting.substring(0, 5); // sub is "Hello"
  ```
- Supply start and “past the end” position
- First position is at `0`



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

Figure 3 String Positions

Substrings

- `String sub2 = greeting.substring(7, 12); // sub2 is "World"`
- Substring length is "past the end" - start

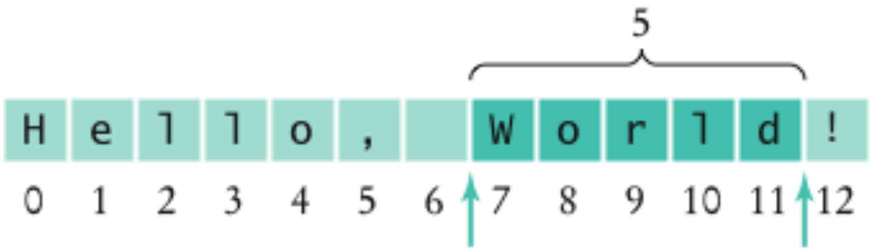


Figure 4 Extracting a Substring

Self Check 4.14

Assuming the `String` variable `s` holds the value `"Agent"`, what is the effect of the assignment `s = s + s.length()`?

> **Answer:** `s` is set to the string `Agent5`

Big Java by Cay Horstmann

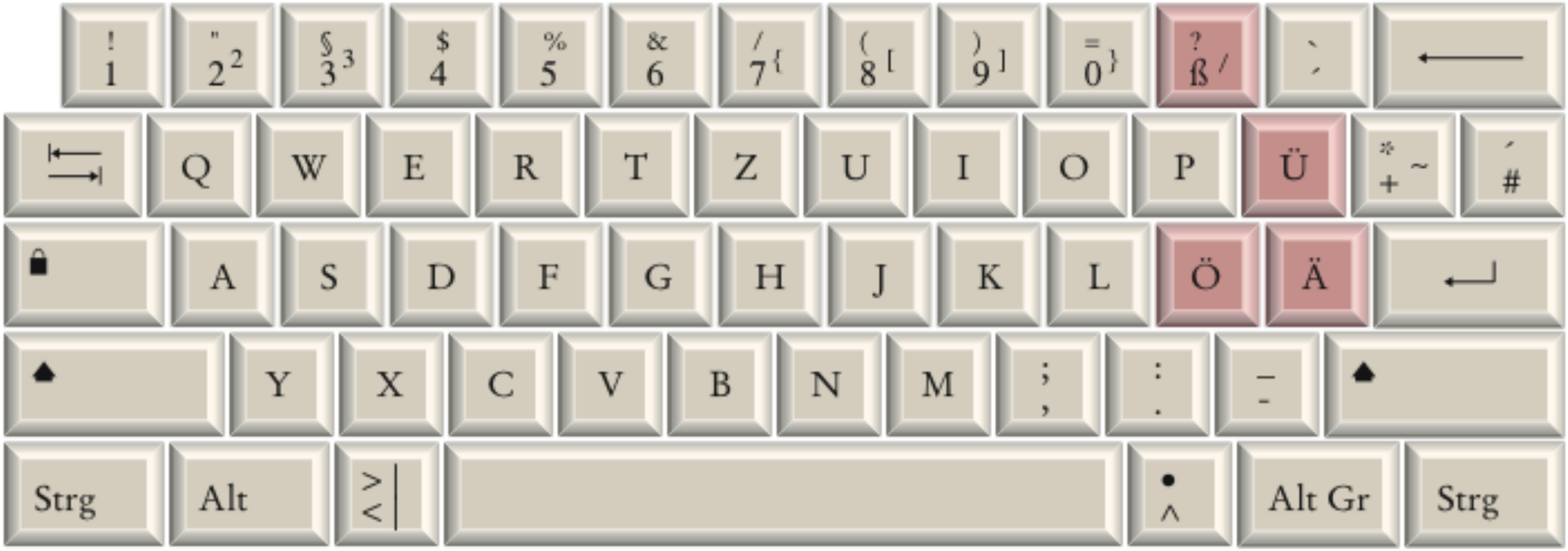
Self Check 4.15

Assuming the String variable `river` holds the value `"Mississippi "`, what is the value of `river.substring(1, 2)`? Of `river.substring(2, river.length() - 3)`?

Answer: The strings `"i"` and `"ssissi"`

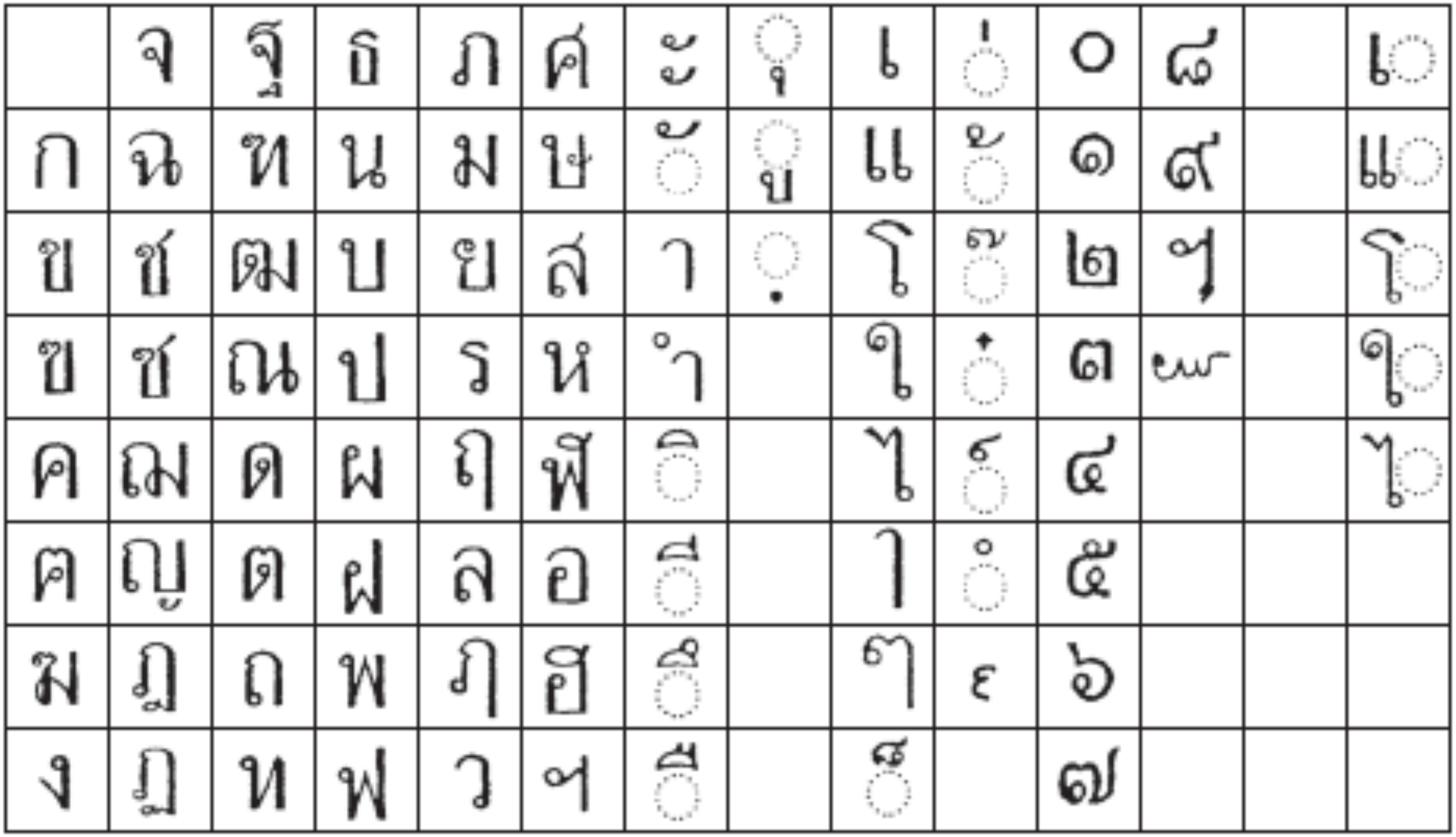
Big Java by Cay Horstmann

German Keyboard



A German Keyboard

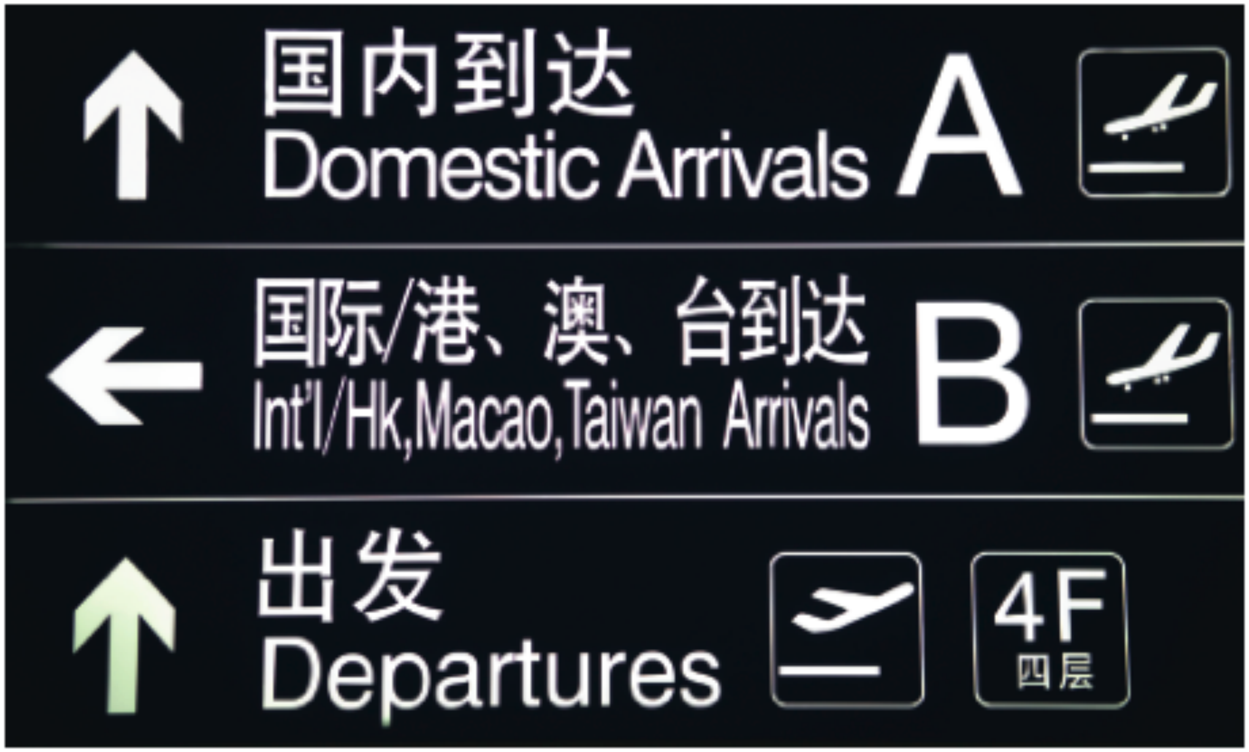
Thai Alphabet



	จ	ฐ	ธ	ภ	ศ	ะ	◌ุ	เ	◌่	๐	๘		เ◌
ก	ฉ	ฑ	น	ม	ษ	◌ั	◌ู	แ	◌้	๑	๙		แ◌
ข	ช	ฒ	บ	ย	ส	า	◌ฺ	โ	◌๊	๒	๚		โ◌
ฃ	ซ	ณ	ป	ร	ห	◌ำ		ใ	◌๋	๓	๛		ใ◌
ค	ฌ	ด	ผ	ฤ	ฬ	◌ิ		ไ	◌์	๔			ไ◌
ฅ	ญ	ต	ฝ	ล	อ	◌ี		ๅ	◌ํ	๕			
ฆ	ฎ	ถ	พ	ฦ	ฮ	◌ึ		ๆ	๎	๖			
ง	ฏ	ท	ฟ	ว	ฯ	◌ื		◌็		๗			

The Thai Alphabet

Chinese Ideographs



Chinese Ideographs

Reading Input

- `System.in` has minimal set of features — it can only read one byte at a time
- In Java 5.0, Scanner class was added to read keyboard input in a convenient manner

-
  ```
  Scanner in = new Scanner(System.in);
  System.out.print("Enter quantity:");
  int quantity = in.nextInt();
  ```
- `nextDouble` reads a `double`
- `nextLine` reads a line (until user hits Enter)
- `next` reads a word (until any white space)

ch04/cashregister/CashRegisterSimulator.java

```
import java.util.Scanner;

/**
   This program simulates a transaction in which a user pays for an item
   and receives change.
*/
public class CashRegisterSimulator
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      CashRegister register = new CashRegister();

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

      System.out.print("Enter dollars:  ");
      int dollars = in.nextInt();
```

Continued

ch04/cashregister/CashRegisterSimulator.java (cont.)

```
      System.out.print("Enter quarters: ");
      int quarters = in.nextInt();
      System.out.print("Enter dimes: ");
      int dimes = in.nextInt();
      System.out.print("Enter nickels: ");
      int nickels = in.nextInt();
      System.out.print("Enter pennies: ");
      int pennies = in.nextInt();
      register.enterPayment(dollars, quarters, dimes, nickels, pennies);

      System.out.print("Your change: ");
      System.out.println(register.giveChange());
   }
}
```

Continued

ch04/cashregister/CashRegisterSimulator.java (cont.)

Program Run:

```
Enter price: 7.55
Enter dollars: 10
Enter quarters: 2
Enter dimes: 1
Enter nickels: 0
Enter pennies: 0
Your change: is 3.05
```

Self Check 4.16

Why can’t input be read directly from `System.in`?

Answer: The class only has a method to read a single byte. It would be very tedious to form characters, strings, and numbers from those bytes.

Self Check 4.17

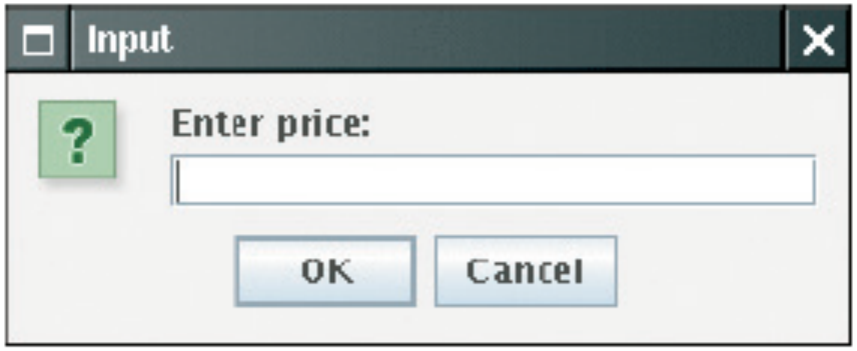
Suppose `in` is a `Scanner` object that reads from `System.in`, and your program calls

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

What is the value of name if the user enters `John Q. Public`?

Answer: The value is `"John"`. The `next` method reads the next *word*.

Reading Input From a Dialog Box



Enter price:

An Input Dialog Box

Reading Input From a Dialog Box

- `String input = JOptionPane.showInputDialog(`*`prompt`*`)`
- Convert strings to numbers if necessary:

  ```
  int count = Integer.parseInt(input);
  ```

- Conversion throws an exception if user doesn't supply a number — see Chapter 11
- Add `System.exit(0)` to the `main` method of any program that uses `JOptionPane`