

# ICOM 4015: Advanced Programming

## Lecture 4

### Chapter Four: Fundamental Data Types



## Chapter Four: Fundamental Data Types

ICOM 4015 Fall 2008

*Big Java* by Cay Horstmann  
Copyright © 2008 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

# 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

# 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 <code>false</code> and <code>true</code>	1 bit

## 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
```

- **Casts: used to convert a value to a different type**

```
int dollars = (int) balance; // OK
```

Cast discards fractional part.

## Number Types: Floating-point Types (cont.)

---

`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 4.1 Cast

---

*(typeName) expression*

### **Example:**

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

### **Purpose:**

To convert an expression to a different type.



## Self Check 4.1

---

Which are the most commonly used number types in Java?

**Answer:** `int` and `double`

## Self Check 4.2

---

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

**Answer:** When the fractional part of `x` is  $\geq 0.5$

## Self Check 4.3

---

How do you round the `double` value `x` to the nearest `int` value, assuming that you know that it is less than  $2 \cdot 10^9$ ?

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

## 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;
```

## 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 4.2 Constant Definition

---

In a method:

```
final typeName variableName = expression;
```

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;
```

**Purpose:**

To define a constant in a method or a class.

## ch04/cashregister/CashRegister.java

---

```
01: /**
02:     A cash register totals up sales and computes change due.
03: */
04: public class CashRegister
05: {
06:     /**
07:         Constructs a cash register with no money in it.
08:     */
09:     public CashRegister()
10:     {
11:         purchase = 0;
12:         payment = 0;
13:     }
14:
15:     /**
16:         Records the purchase price of an item.
17:         @param amount the price of the purchased item
18:     */
19:     public void recordPurchase(double amount)
20:     {
21:         purchase = purchase + amount;
22:     }
```

## ch04/cashregister/CashRegister.java (cont.)

---

```
23:
24:     /**
25:         Enters the payment received from the customer.
26:         @param dollars the number of dollars in the payment
27:         @param quarters the number of quarters in the payment
28:         @param dimes the number of dimes in the payment
29:         @param nickels the number of nickels in the payment
30:         @param pennies the number of pennies in the payment
31:     */
32:     public void enterPayment(int dollars, int quarters,
33:         int dimes, int nickels, int pennies)
34:     {
35:         payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
36:             + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
37:     }
38:
39:     /**
40:         Computes the change due and resets the machine for the next
customer.
41:         @return the change due to the customer
42:     */
43:     public double giveChange()
44:     {
```



## ch04/cashregister/CashRegister.java (cont.)

---

```
45:     double change = payment - purchase;
46:     purchase = 0;
47:     payment = 0;
48:     return change;
49: }
50:
51: public static final double QUARTER_VALUE = 0.25;
52: public static final double DIME_VALUE = 0.1;
53: public static final double NICKEL_VALUE = 0.05;
54: public static final double PENNY_VALUE = 0.01;
55:
56: private double purchase;
57: private double payment;
58: }
```

## ch04/cashregister/CashRegisterTester.java

---

```
01: /**
02:     This class tests the CashRegister class.
03: */
04: public class CashRegisterTester
05: {
06:     public static void main(String[] args)
07:     {
08:         CashRegister register = new CashRegister();
09:
10:         register.recordPurchase(0.75);
11:         register.recordPurchase(1.50);
12:         register.enterPayment(2, 0, 5, 0, 0);
13:         System.out.print("Change: ");
14:         System.out.println(register.giveChange());
15:         System.out.println("Expected: 0.25");
16:
17:         register.recordPurchase(2.25);
18:         register.recordPurchase(19.25);
19:         register.enterPayment(23, 2, 0, 0, 0);
20:         System.out.print("Change: ");
21:         System.out.println(register.giveChange());
22:         System.out.println("Expected: 2.0");
23: ICOM 4015 Fall 2008
24: }
```

## ch04/cashregister/CashRegisterTester.java (cont.)

---

### Output:

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?

```
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  $\pi$ .

# Assignment, Increment, and Decrement

---

- Assignment is not the same as mathematical equality:

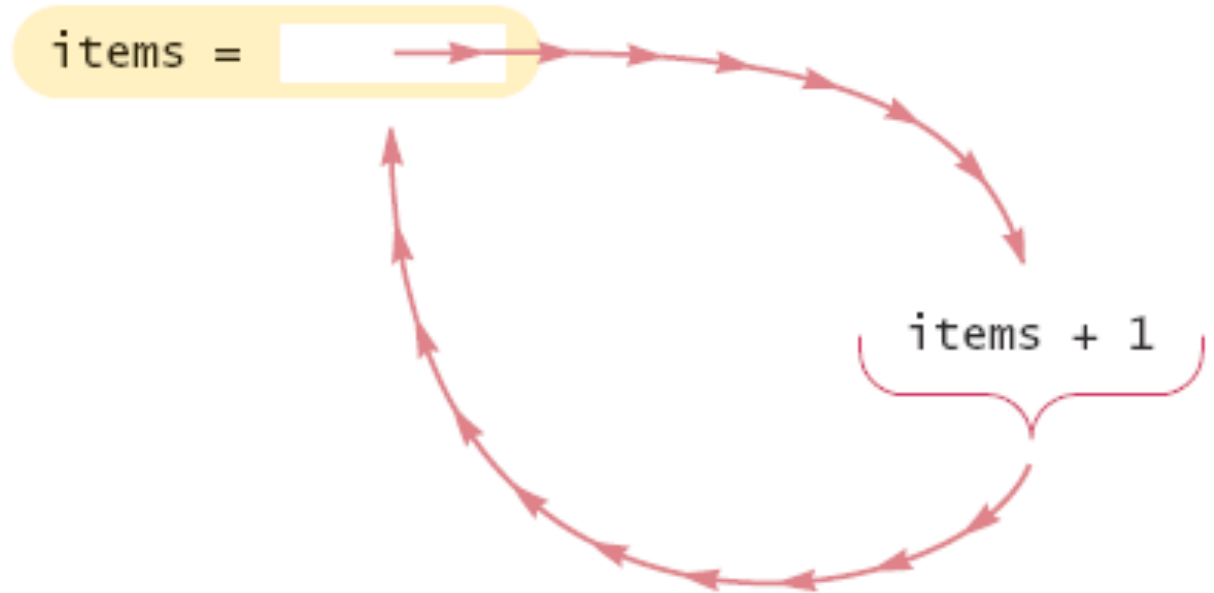
```
items = items + 1;
```

- `items++` **is the same as** `items = items + 1`

- `items--` **subtracts 1 from** `items`

# Assignment, Increment, and Decrement

---



**Figure 1**

Incrementing a Variable

## Self Check 4.6

---

What is the meaning of the following statement?

```
balance = balance + amount;
```

**Answer:** The statement adds the `amount` value to the `balance` variable.



## Self Check 4.7

---

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

```
n--;
```

```
n++;
```

```
n--;
```

**Answer:** One less than it was before.

## Arithmetic Operations

---

- `/` 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`

## Arithmetic Operations

---

```
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;
```

## The `Math` class

---

- `Math` class: contains methods like `sqrt` and `pow`
- 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 the `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)`

# Mathematical Methods

---

Function	Returns
<code>Math.sqrt(x)</code>	square root
<code>Math.pow(x, y)</code>	power $x^y$
<code>Math.exp(x)</code>	$e^x$
<code>Math.log(x)</code>	natural log
<code>Math.sin(x)</code> , <code>Math.cos(x)</code> , <code>Math.tan(x)</code>	sine, cosine, tangent ( $x$ in radians)
<code>Math.round(x)</code>	closest integer to $x$
<code>Math.min(x, y)</code> , <code>Math.max(x, y)</code>	minimum, maximum

## Analyzing an Expression

$$\begin{array}{c} (-b + \text{Math.sqrt}(b * b - 4 * a * c)) / (2 * a) \\ \underbrace{\qquad\qquad\qquad} \quad \underbrace{\qquad\qquad\qquad} \quad \underbrace{\qquad\qquad\qquad} \\ \qquad\qquad b^2 \qquad\qquad\qquad 4ac \qquad\qquad\qquad 2a \\ \underbrace{\qquad\qquad\qquad} \\ \qquad\qquad\qquad b^2 - 4ac \\ \underbrace{\qquad\qquad\qquad} \\ \qquad\qquad\qquad \sqrt{b^2 - 4ac} \\ \underbrace{\qquad\qquad\qquad} \\ \qquad\qquad\qquad -b + \sqrt{b^2 - 4ac} \\ \underbrace{\qquad\qquad\qquad} \\ \qquad\qquad\qquad \frac{-b + \sqrt{b^2 - 4ac}}{2a} \end{array}$$

**Figure 2** Analyzing an Expression

## Self Check 4.8

---

What is the value of  $1729 / 100$ ? Of  $1729 \% 100$ ?

**Answer:** 17 and 29

## Self Check 4.9

---

Why doesn't the following statement compute the average of  $s_1$ ,  $s_2$ , and  $s_3$ ?

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

**Answer:** Only  $s_3$  is divided by 3. To get the correct result, use parentheses. Moreover, if  $s_1$ ,  $s_2$ , and  $s_3$  are integers, you must divide by 3.0 to avoid integer division:

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



## Self Check 4.10

---

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

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

## Calling Static Methods

---

- A `static` method does not operate on an object `double x = 4;`  
`double root = x.sqrt(); // Error`
- Static methods are defined inside classes
- Naming convention: Classes start with an uppercase letter;  
objects start with a lowercase letter

`Math`

`System.out`

## Syntax 4.3 Static Method Call

---

*ClassName.methodName(parameters)*

### **Example:**

`Math.sqrt(4)`

### **Purpose:**

To invoke a static method (a method that does not operate on an object) and supply its parameters.

## Self Check 4.11

---

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 4.12

---

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

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

# Strings

---

- A string is a sequence of characters
- Strings are objects of the String class
- String constants:  
`"Hello, World!"`
- String variables:  
`String message = "Hello, World!";`
- String length:  
`int n = message.length();`
- Empty string: `""`

# 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(x);
```

- 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

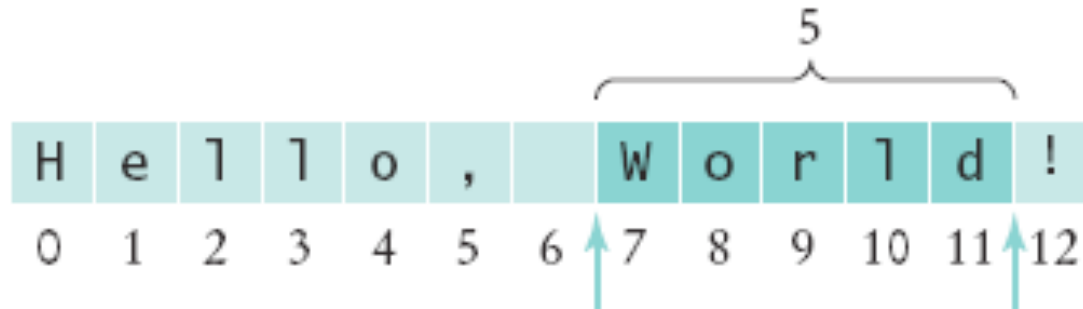
H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

**Figure 3** String Positions

## Substrings (cont.)

---

Substring length is “past the end” - start



**Figure 4** Extracting a Substring

## Self Check 4.13

---

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`

## Self Check 4.14

---

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"`

# International Alphabets

---



A German Keyboard

# International Alphabets

	จ	ฐ	ถ	ภ	ค	ช	ฌ	เ	อ	ง	ฌ		เ
ก	ฌ	ท	น	ม	ษ	ั	ุ	แ	็	๑	๔		แ
ข	ช	ฌ	บ	ย	ล	า	ุ	โ	็	๒	๕		โ
ช	ช	ฌ	ป	ร	ห	า		ใ	็	๓	๗		ใ
ค	ฌ	ด	ผ	ฤ	ฬ	็		ไ	็	๕			ไ
ค	ฌ	ต	ฝ	ล	อ	็		า	็	๕			
ฆ	ฌ	ถ	พ	ภ	ฮ	็		ใ	็	๕			
ง	ฌ	ท	พ	ว	๕	็		็		๕			

## The Thai Alphabet

# International Alphabets

## CLASSIC SOUPS

					Sm.	Lg.	
清	燉	雞	湯	57.	House Chicken Soup (Chicken, Celery, Potato, Onion, Carrot) .....	1.50 2.75	
雞	飯	湯	58.	Chicken Rice Soup .....	1.85 3.25		
雞	麵	湯	59.	Chicken Noodle Soup .....	1.85 3.25		
廣	東	雲	吞	60.	Cantonese Wonton Soup.....	1.50 2.75	
蕃	茄	蛋	湯	61.	Tomato Clear Egg Drop Soup .....	1.65 2.95	
雲	吞	湯	62.	Regular Wonton Soup .....	1.10 2.10		
酸	辣	湯	63.	Hot & Sour Soup .....	1.10 2.10		
蛋	花	湯	64.	Egg Drop Soup.....	1.10 2.10		
雲	蛋	湯	65.	Egg Drop Wonton Mix.....	1.10 2.10		
豆	腐	菜	湯	66.	Tofu Vegetable Soup .....	NA 3.50	
雞	玉	米	湯	67.	Chicken Corn Cream Soup .....	NA 3.50	
蟹	肉	玉	米	湯	68.	Crab Meat Corn Cream Soup.....	NA 3.50
海	鮮	湯	69.	Seafood Soup.....	NA 3.50		

A Menu with Chinese Characters



## 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)**
- `nextWord` **reads a word (until any white space)**

## ch04/cashregister/CashRegisterSimulator.java

---

```
01: import java.util.Scanner;
02:
03: /**
04:     This program simulates a transaction in which a user pays for an
item
05:     and receives change.
06: */
07: public class CashRegisterSimulator
08: {
09:     public static void main(String[] args)
10:     {
11:         Scanner in = new Scanner(System.in);
12:
13:         CashRegister register = new CashRegister();
14:
15:         System.out.print("Enter price: ");
16:         double price = in.nextDouble();
17:         register.recordPurchase(price);
18:
19:         System.out.print("Enter dollars: ");
20:         int dollars = in.nextInt();
```

***Continued***

## ch04/cashregister/CashRegisterSimulator.java (cont.)

---

### Output:

Enter price: 7.55

Enter dollars: 10

Enter quarters: 2

Enter dimes: 1

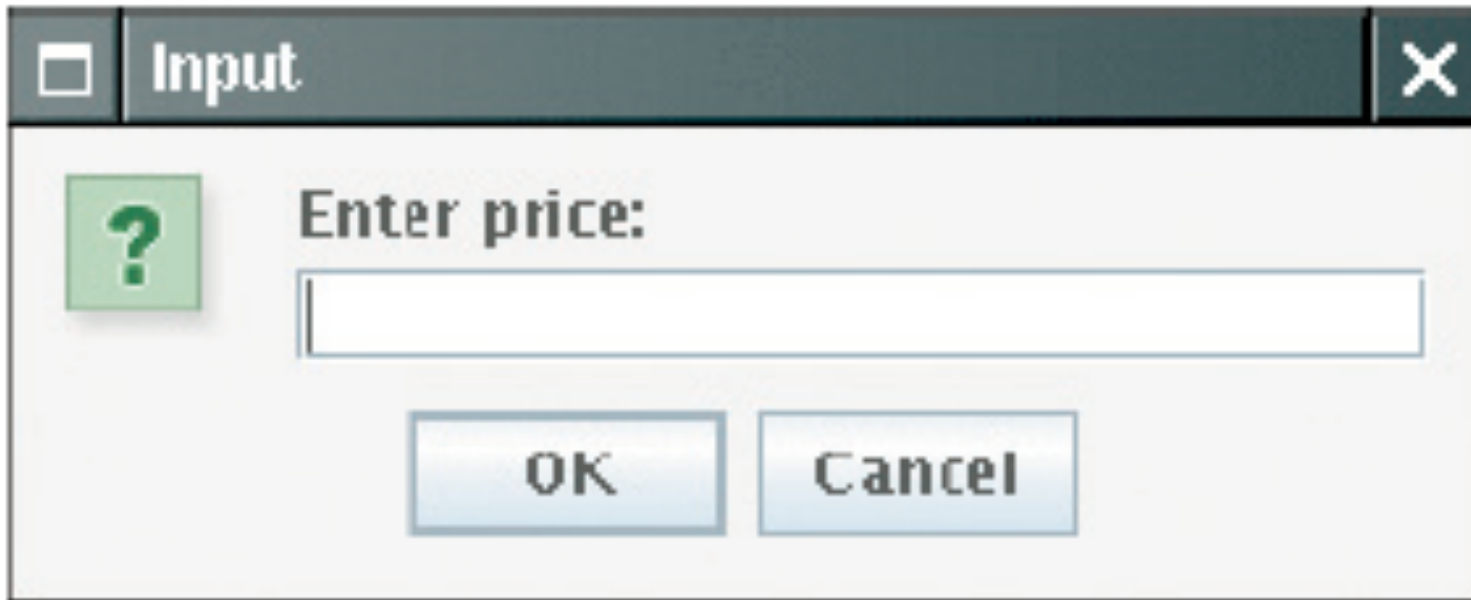
Enter nickels: 0

Enter pennies: 0

Your change: is 3.05

## Reading Input From a Dialog Box

---



### 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`**

## Self Check 4.15

---

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 4.16

---

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*.