Fundamental Data Types

Advanced Programming

ICOM 4015

Lecture 4

Reading: Java Concepts Chapter 4

Lecture 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

Continued…

Lecture Goals

- 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
```

Number Types

- 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

Continued…

Primitive Types

Type	Description	Size
`double`	The double-precision floating-point type, with a range of about ±10^{308} and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about ±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 byte

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
```

Continued…

Number Types: Floating-point Types

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

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

Cast discards fractional part.

- `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

1. Which are the most commonly used number types in Java?
2. When does the cast `(long) x` yield a different result from the call `Math.round(x)`?
3. How do you round the `double` value `x` to the nearest `int` value, assuming that you know that it is less than $2 \cdot 10^9$?

Answers

- `int` and `double`
- When the fractional part of `x` is ≥ 0.5
- 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

File CashRegister.java

```
/**
   A cash register totals up sales and computes change due.
*/
public class CashRegister
{
   /**
      Constructs a cash register with no money in it.
   */
   public CashRegister()
   {
      purchase = 0;
      payment = 0;
   }

```

Continued…

File CashRegister.java

```
   /**
      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
   */
```

Continued…

File CashRegister.java

```
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;
}

/**
   Computes the change due and resets the machine for
      the next customer.
   @return the change due to the customer
*/
```

Continued…

File `CashRegister.java`

```java
   public double giveChange()
   {
      double change = payment - purchase;
      purchase = 0;
      payment = 0;
      return change;
   }

   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;
}
```

File 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());

```

Continued…

File CashRegisterTester.java

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

Output

```
Change=0.25
Change=2.0
```

Self Check

1. 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;
   ```

2. What is wrong with the following statement?

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

Answers

1. The first definition is used inside a method, the second inside a class
2. (1) You should use a named constant, not the "magic number" 3.14
   (2) 3.14 is not an accurate representation of π

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

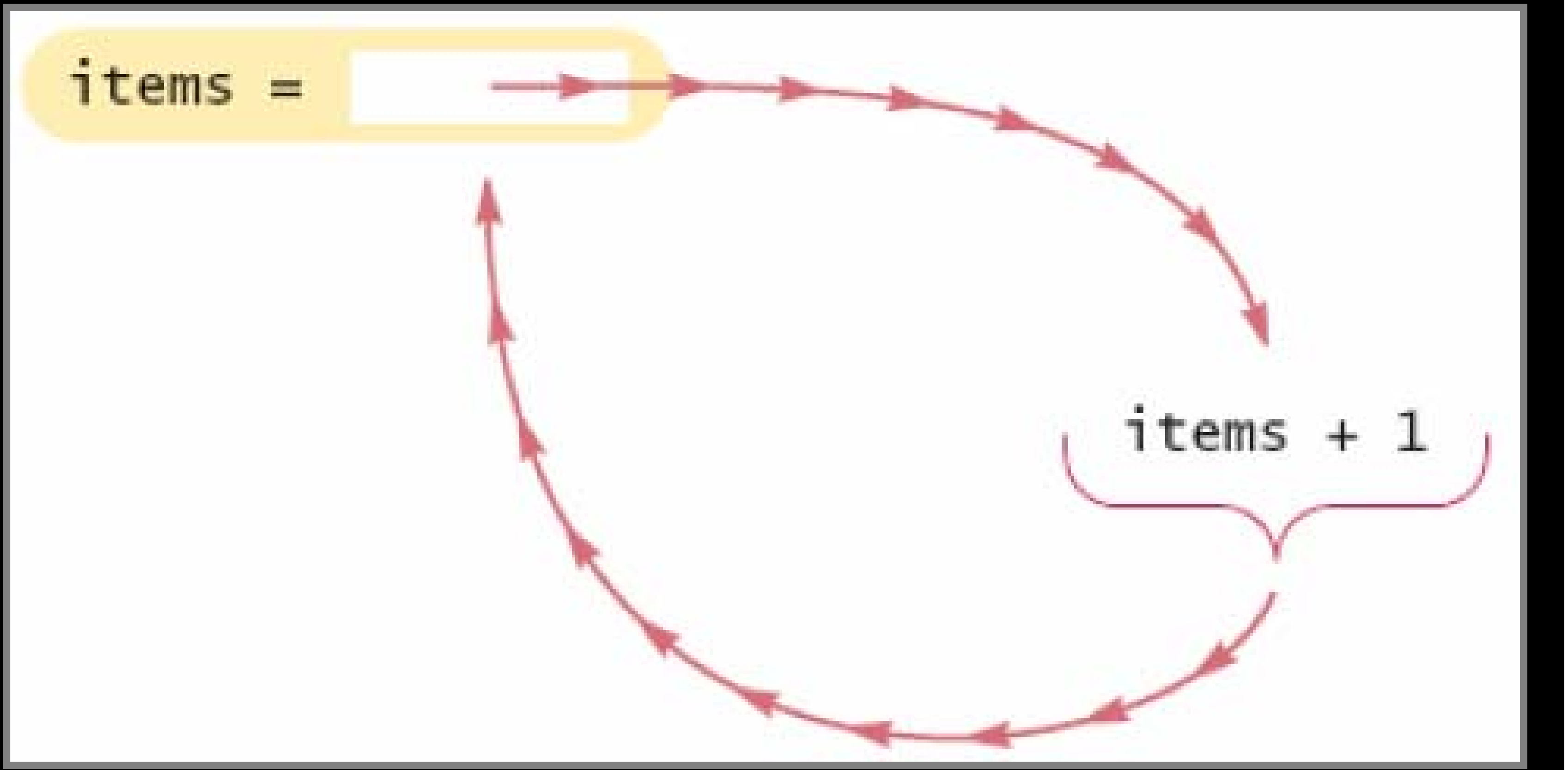


Figure 1:
Incrementing a Variable

Self Check

1. What is the meaning of the following statement?

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

1. What is the value of **n** after the following sequence of statements?

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

Answers

1. The statement adds the `amount` value to the balance `variable`
2. 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)`

Continued…

The Math class

- In Java,

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

can be represented as

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

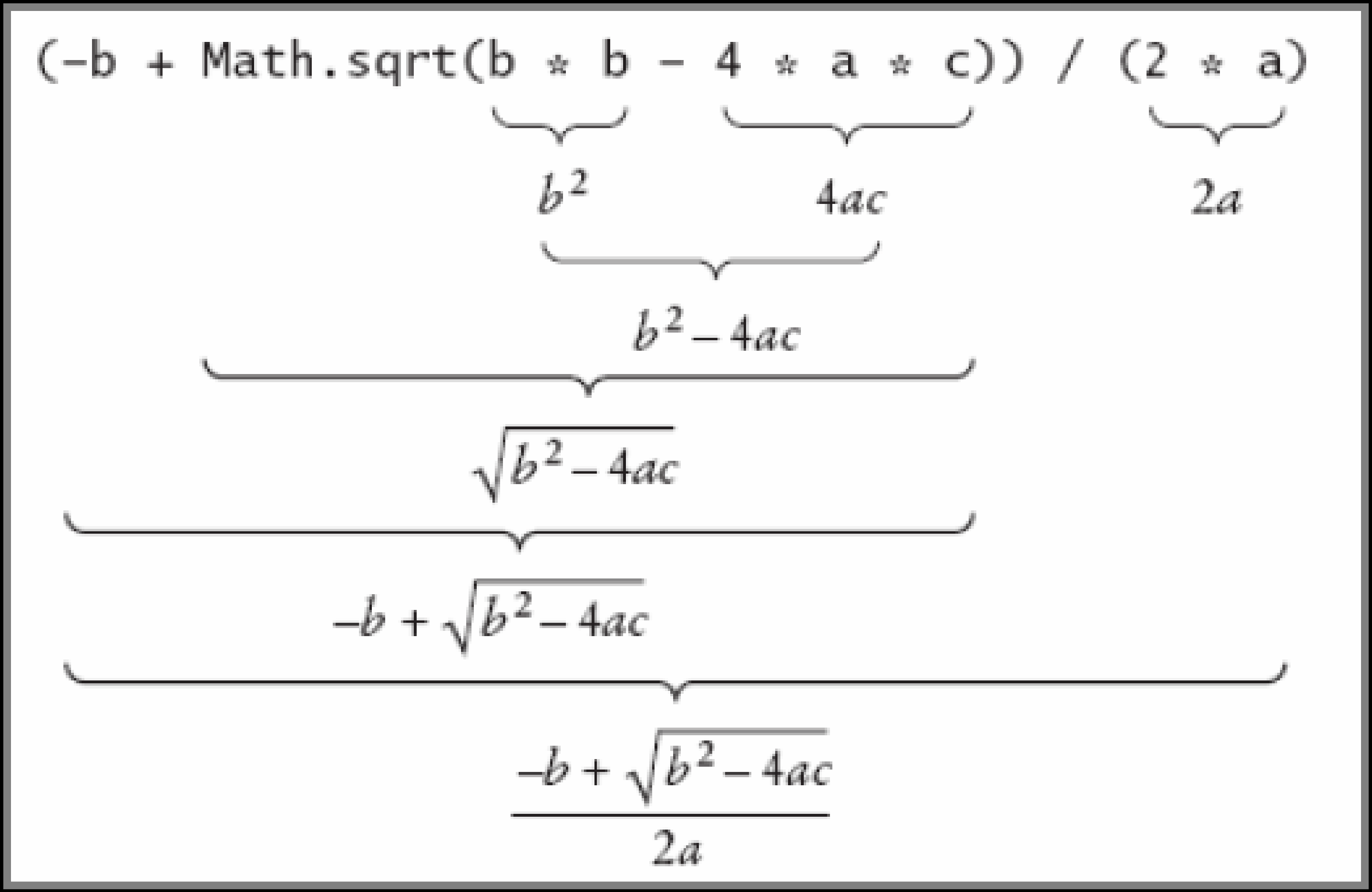
Mathematical Methods in Java

`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 radian)

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

Analyzing an Expression



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

b^2

$4ac$

$2a$

$b^2 - 4ac$

$\sqrt{b^2 - 4ac}$

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

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

Figure 3:
Analyzing an Expression

Self Check

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

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

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

3. What is the value of

```
Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))
```

in mathematical notation?

Answers

1. 17 and 29

2. 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
```

3. $$\sqrt{x^2 + y^2}$$

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
```

Fall 2 adapted from Java Concepts companion slides 36

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

1. Why can't you call `x.pow(y)` to compute x^y?
2. Is the call `System.out.println(4)` a static method call?

Answers

1. `x` is a number, not an object, and you cannot invoke methods on numbers
2. 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(str);
```

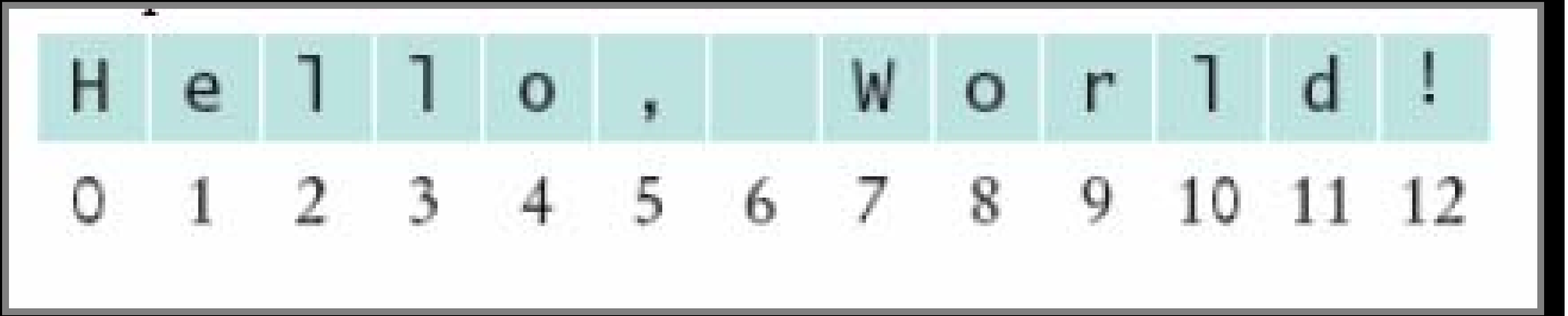
- **Convert to string:**

```
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

Continued…

Substrings

- Substring length is “past the end” - start

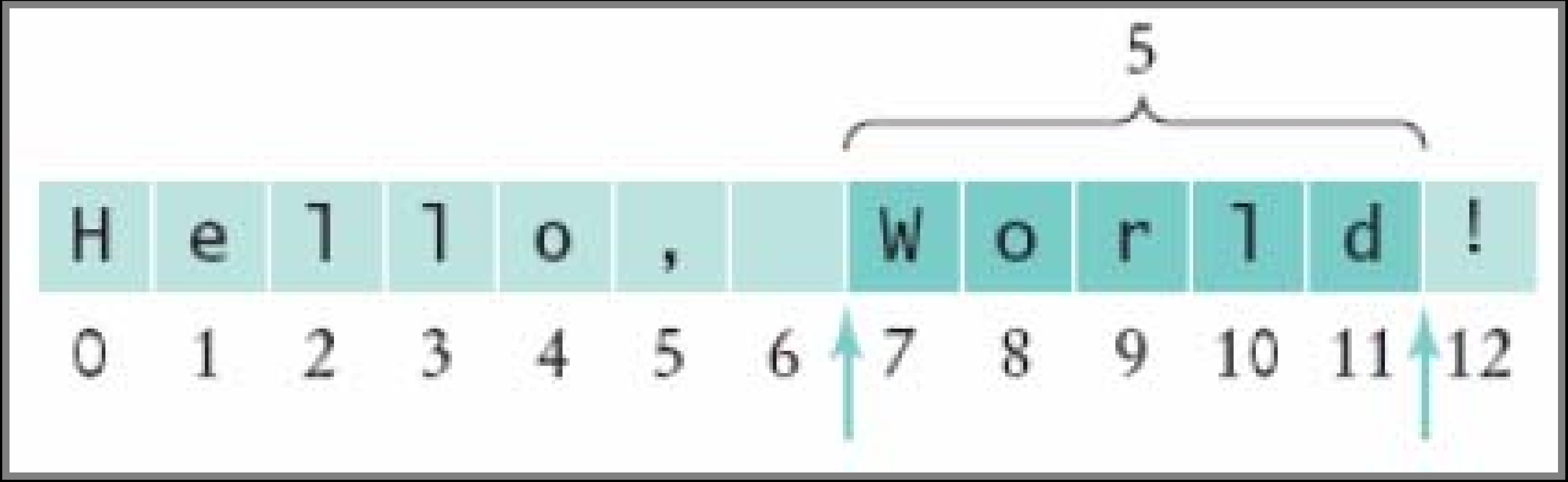


Figure 4:
Extracting a Substring

Self Check

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

2. 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)`?

Answers

1. `s` is set to the string `Agent5`
2. The strings `"i"` and `"ssissi"`

International Alphabets

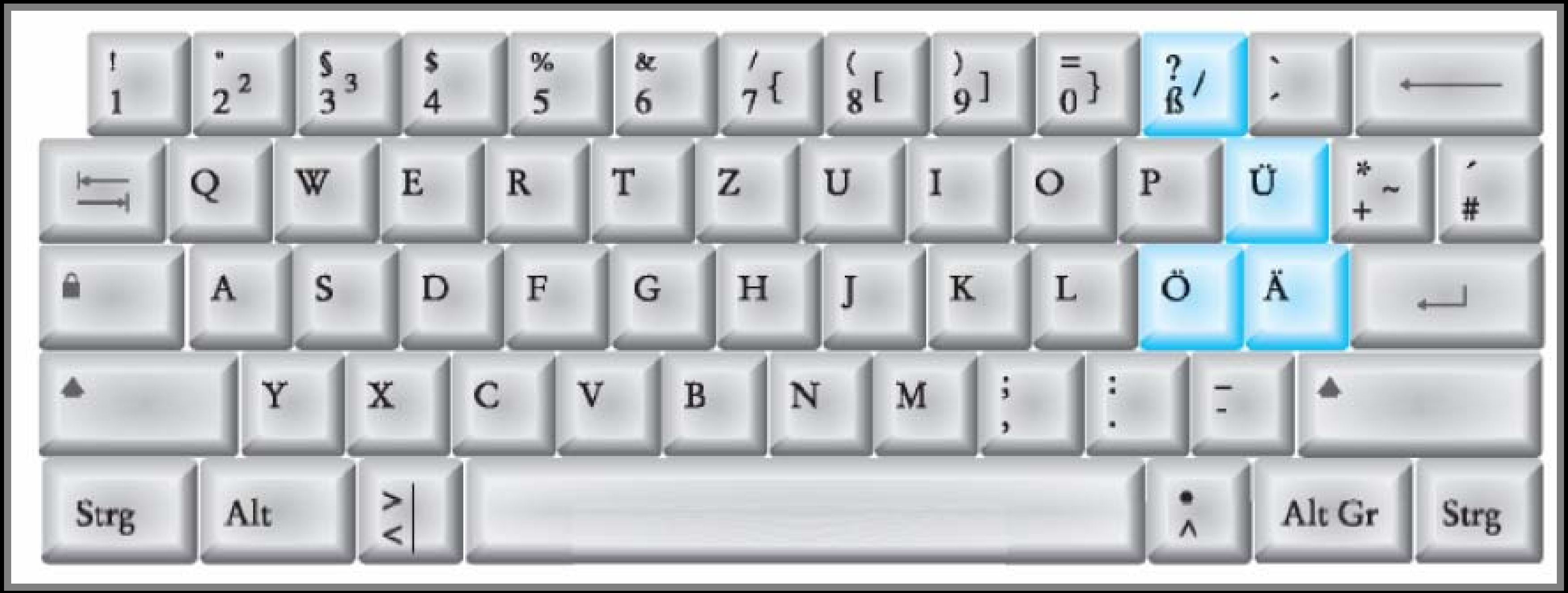
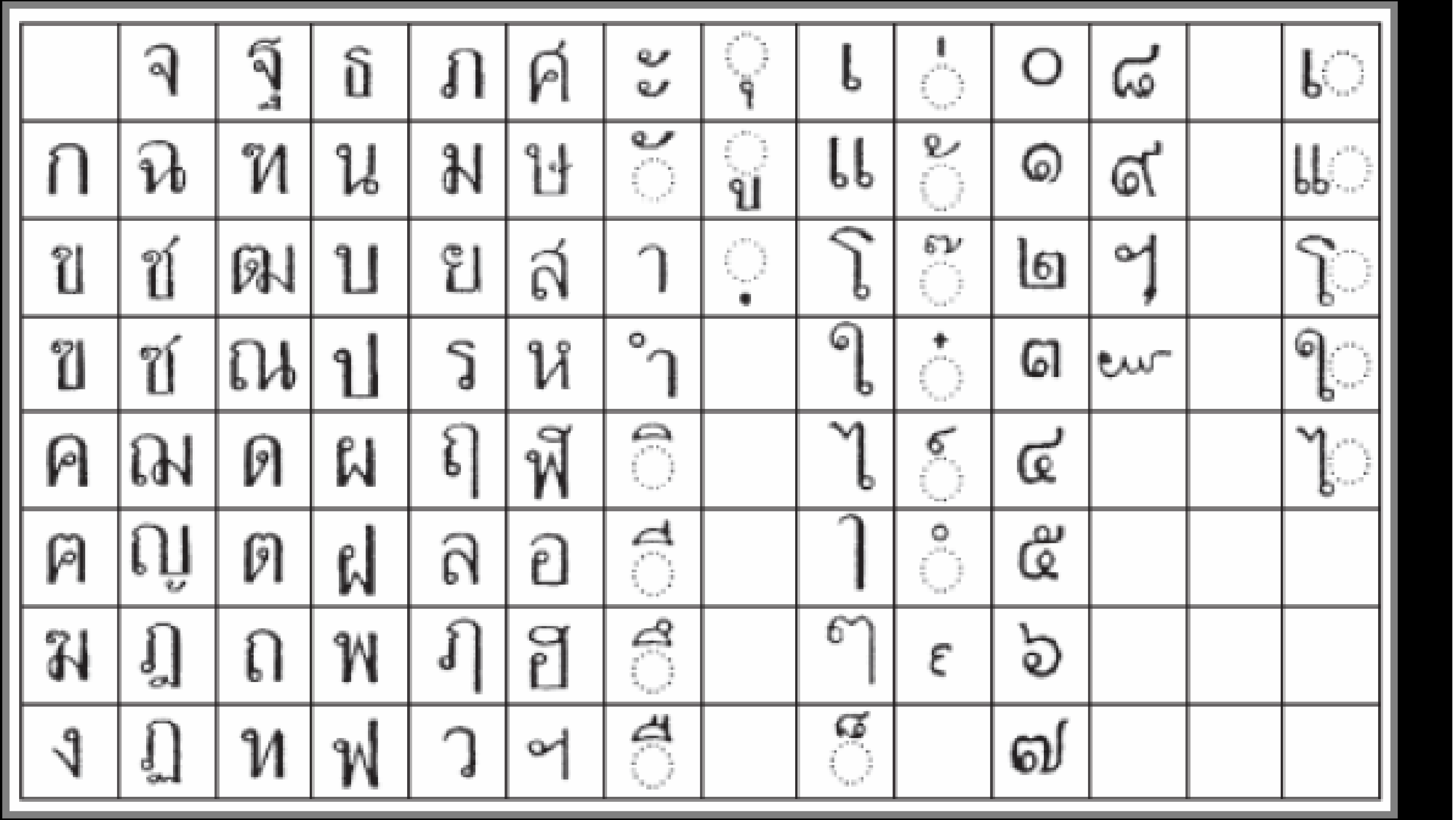


Figure 5:
A German Keyboard

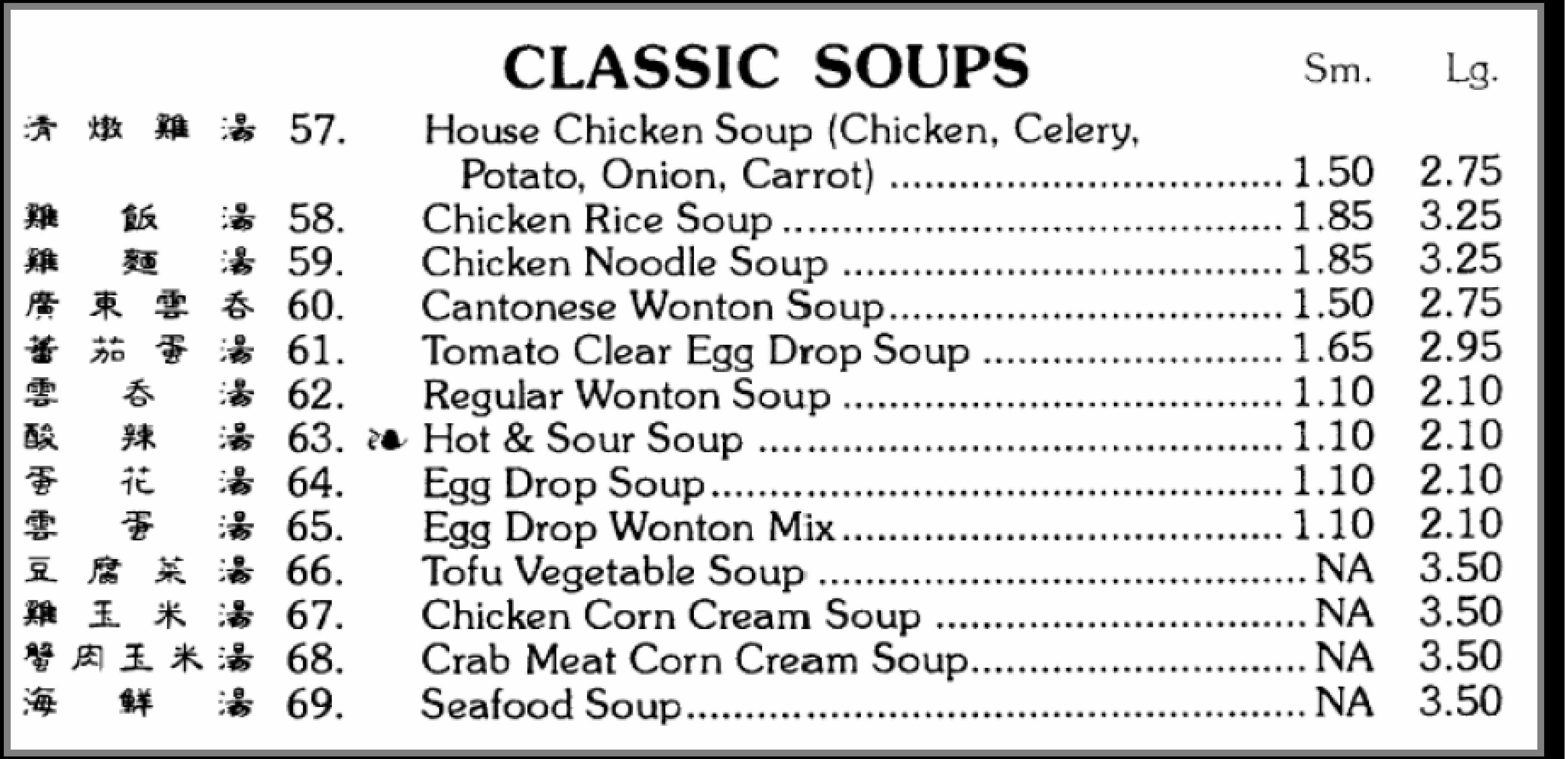
International Alphabets



	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			

Figure 6:
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

Figure 7:
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)

File InputTester.java

```java
import java.util.Scanner;

/**
   This class tests console input.
*/
public class InputTester
{
   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);

```

Continued…

File InputTester.java

```java
      System.out.print("Enter dollars: ");
      int dollars = in.nextInt();
      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 is ");
      System.out.println(register.giveChange());
   }
}
```

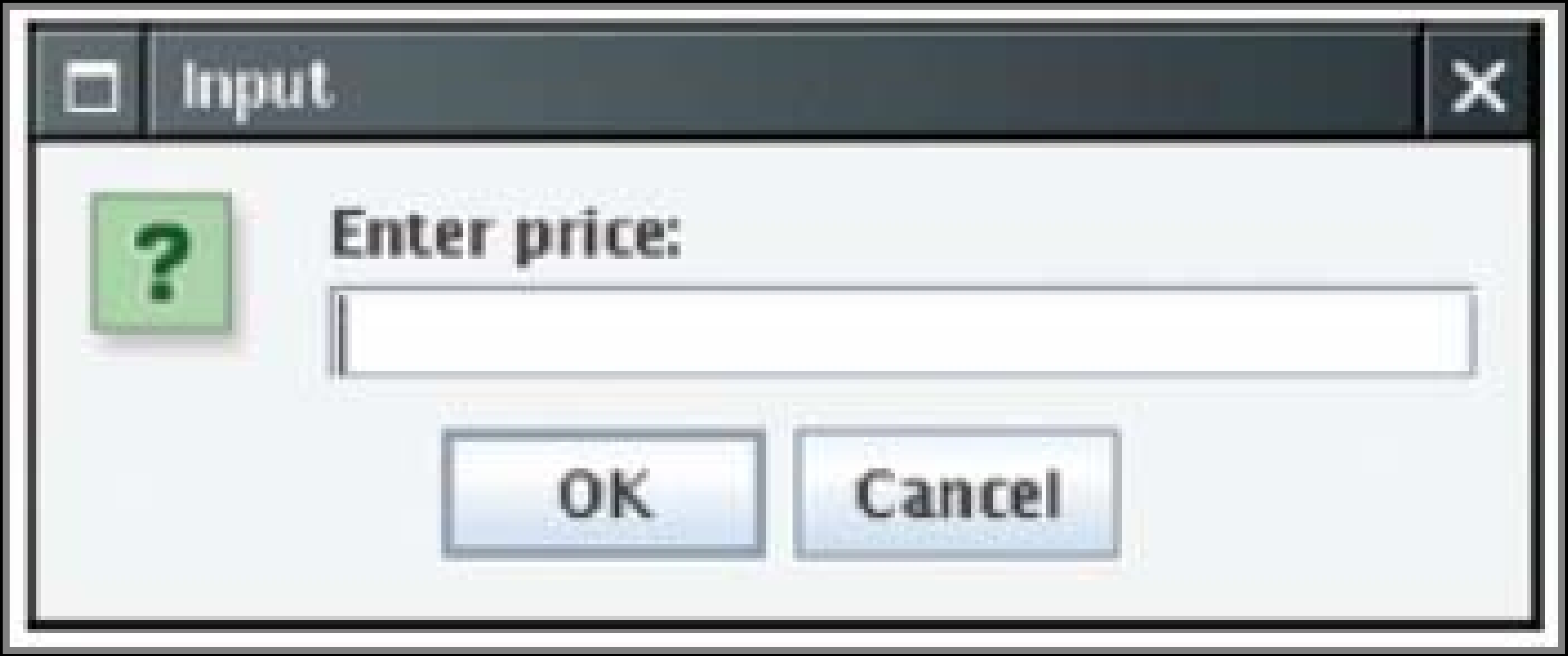
Continued…

File `InputTester.java`

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



Enter price:

Figure 8:
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 15

- Add `System.exit(0)` to the `main` method of any program that uses `JOptionPane`

Self Check

1. **Why can't input be read directly from `System.in`?**

2. **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`?

Answers

1. 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.
2. The value is `"John"`. The `next` method reads the next *word*.