Day 01

# BASIC CONCEPTS OF JAVA 1

## FUNDAMENTALS OF TELECOMMUNICATIONS LAB

Dr. Huy Nguyen

## AGENDA

- Introduction
- Fundamental Data Types
- Objects

## AGENDA

- Introduction
- Fundamental Data Types
- Objects

## BASIC CONCEPTS OF JAVA 1

## INTRODUCTION

## Chapter Goals

- To understand the activity of programming
- To learn about the architecture of computers
- To learn about machine code and high level programming languages
- To become familiar with your computing environment and your compiler
- To compile and run your first Java program
- To recognize syntax and logic errors
- To write pseudocode for simple algorithms

## What Is Programming?

- Computers are programmed to perform tasks
- Different tasks = different programs
- Program
  - *Sequence of basic operations executed in succession*
  - *Contains instruction sequences for all tasks it can execute*
- Sophisticated programs require teams of highly skilled programmers and other professionals

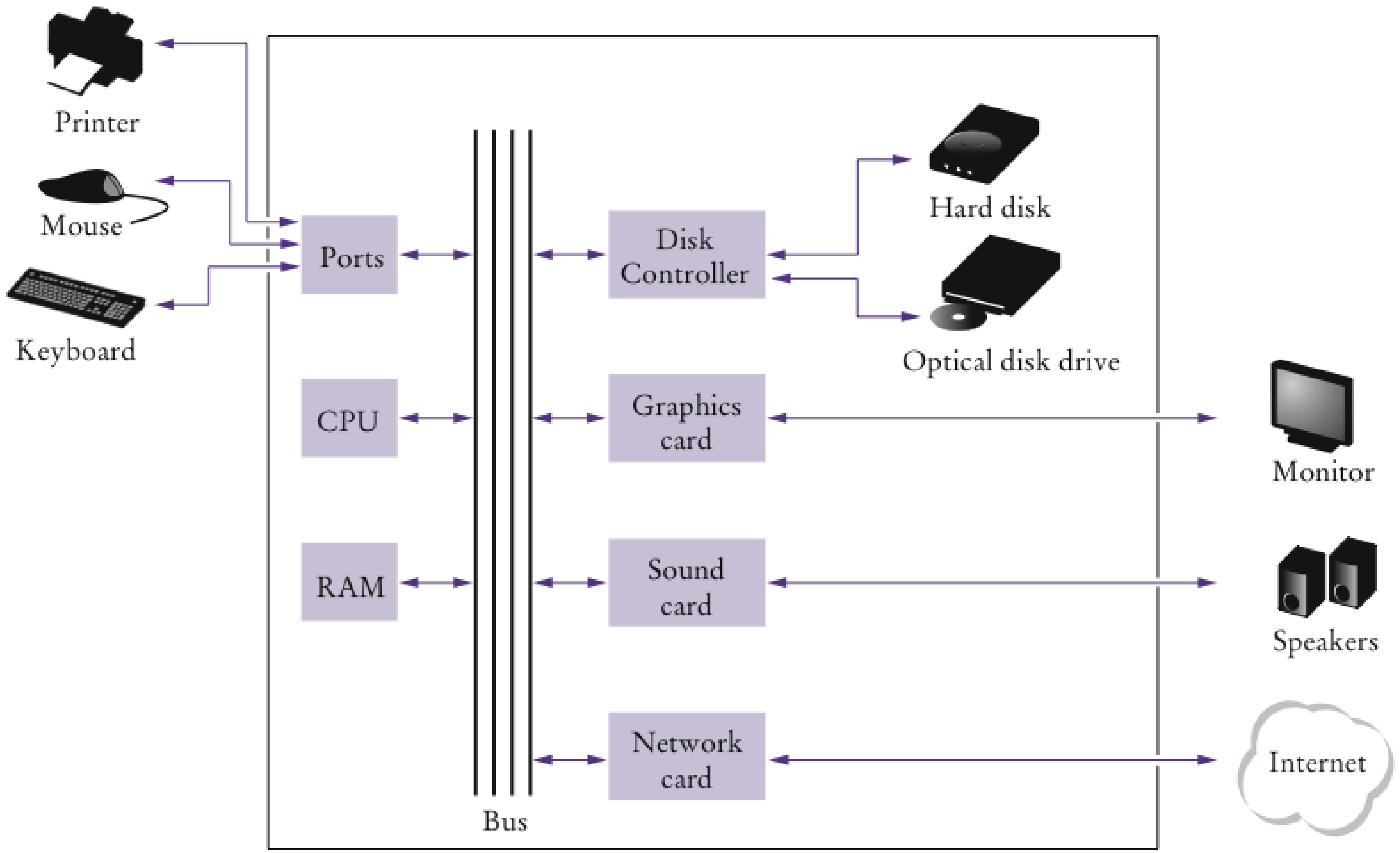
## Self Check

Can a computer program develop the initiative to execute tasks in a better way than its programmers envisioned?

## The Anatomy of a Computer

- Central processing unit
  - *Chip*
  - *Transistors*
- Motherboard
- Storage
  - *Primary storage: Random-access memory (RAM)*
  - *Secondary storage: e.g. HDD*
  - *Removable storage devices: e.g. USBs, CDs*
- Peripherals
- Executes very simple instructions
- Executes instructions very rapidly
- General purpose device

## Schematic Diagram of a Computer



## Self Check

Where is a program stored when it is not currently running?

## Self Check

Which part of the computer carries out arithmetic operations, such as addition and multiplication?

## Machine Code

- Generally, machine code depends on the CPU type
- However, the instruction set of the Java virtual machine (JVM) can be executed on many types of CPU
- Java Virtual Machine (JVM) - a typical sequence of machine instructions is:
  1. *Load the contents of memory location 40.*
  2. *Load the value 100.*
  3. *If the first value is greater than the second value, continue with the instruction that is stored in memory location 240.*
- Machine instructions are encoded as numbers:

```
 21 40
16 100
163 240
```

- Compiler translates high-level language to machine code

## Self Check

Does a person who uses a computer for office work ever run a compiler?

## The Java Programming Language

- Simple
- Safe
- Platform-independent (“write once, run anywhere”)
- Rich library (packages)
- Designed for the internet

## Self Check

What are the two most important benefits of the Java language?

## Self Check

How long does it take to learn the entire Java library?

## HelloPrinter.java

```java
public class HelloPrinter
{
   public static void main(String[] args)
   {
      // Display a greeting in the console window

      System.out.println("Hello, World!");
   }
}
```

## Program Run:

```
Hello, World!
```

## The Structure of a Simple Program: Class Declaration

- Classes are the fundamental building blocks of Java programs:

  `public class HelloPrinter`

  starts a new **class**
- Every source file can contain at most one public class
- The name of the public class must match the name of the file containing the class:
  - *Class* `HelloPrinter` *must be contained in a file named* `HelloPrinter.java`

## The Structure of a Simple Program: `main` Method

- Every Java application contains a class with a main method
  - *When the application starts, the instructions in the main method are executed*
- `public static void main(String[] args)`

  ```
  {
     . . .
  }
  ```

  declares a `main` method

## The Structure of a Simple Program: Comments

- The first line inside the main method is a comment:
  ```
  // Display a greeting in the console window
  ```
- Compiler ignores any text enclosed between `//` and end of the line
- Use comments to help human readers understand your program

## The Structure of a Simple Program: Statements

- The body of the main method contains statements inside the curly brackets (`{ }`)
- Each statement ends in a semicolon (`;`)
- Statements are executed one by one
- Our method has a single statement:

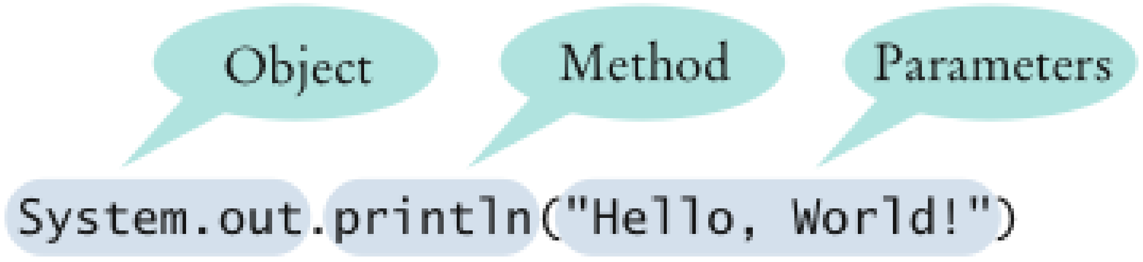
```
System.out.println("Hello, World!");
```

  which prints a line of text:

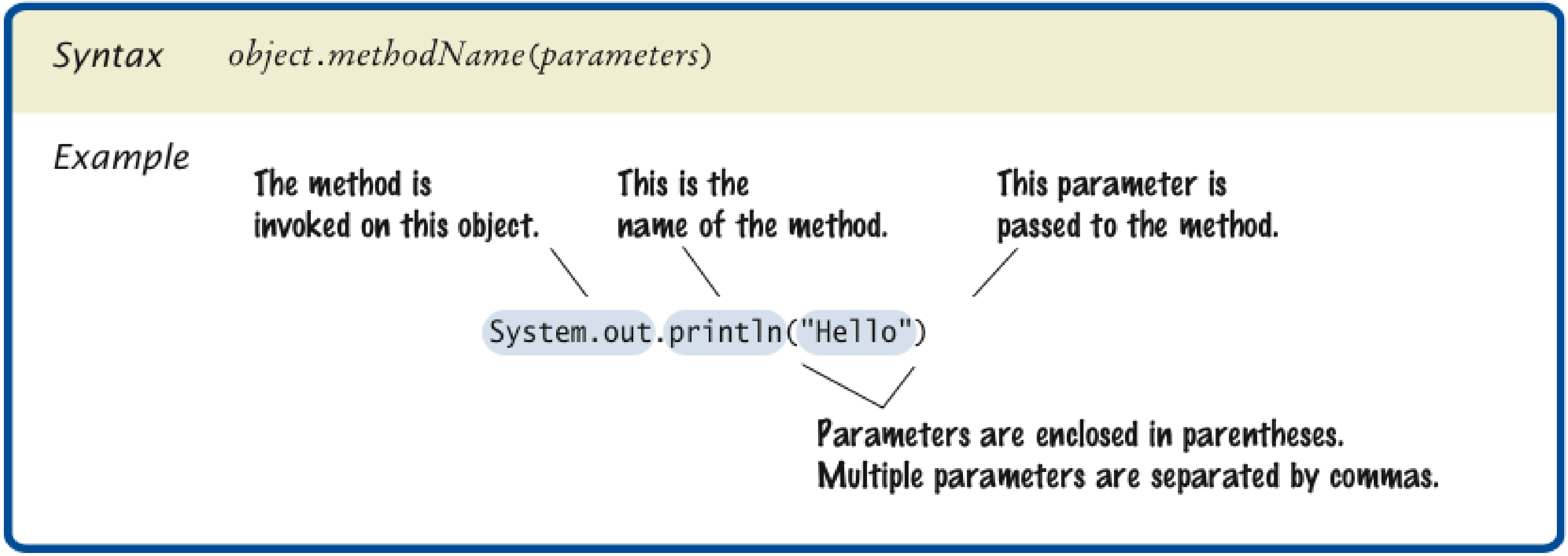
```
Hello, World
```

## The Structure of a Simple Program: Method Call

- `System.out.println("Hello, World!");` is a *method call*
- A method call requires:
  1. *The object that you want to use (in this case, `System.out`)*
  2. *The name of the method you want to use (in this case, `println`)*
  3. ***Parameters** enclosed in parentheses (`()`) containing any other information the method needs (in this case, `"Hello, World!"`)*



## Syntax Method Call



*Syntax* *object.methodName(parameters)*

*Example*

```
System.out.println("Hello")
```

The method is invoked on this object.
This is the name of the method.
This parameter is passed to the method.
Parameters are enclosed in parentheses. Multiple parameters are separated by commas.

## The Structure of a Simple Program: Strings

- **String:** a sequence of characters enclosed in double quotation marks:

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

## Self Check

How would you modify the `HelloPrinter` program to print the words `"Hello,"` and `"World!"` on two lines?

**Answer:**

```
System.out.println("Hello,");
System.out.println("World!");
```

## Self Check

What does the following set of statements print?

```
System.out.print("My lucky number is");
System.out.println(3 + 4 + 5);
```

**Answer:** The printout is

```
My lucky number is12
```

It would be a good idea to add a space after the `is`.

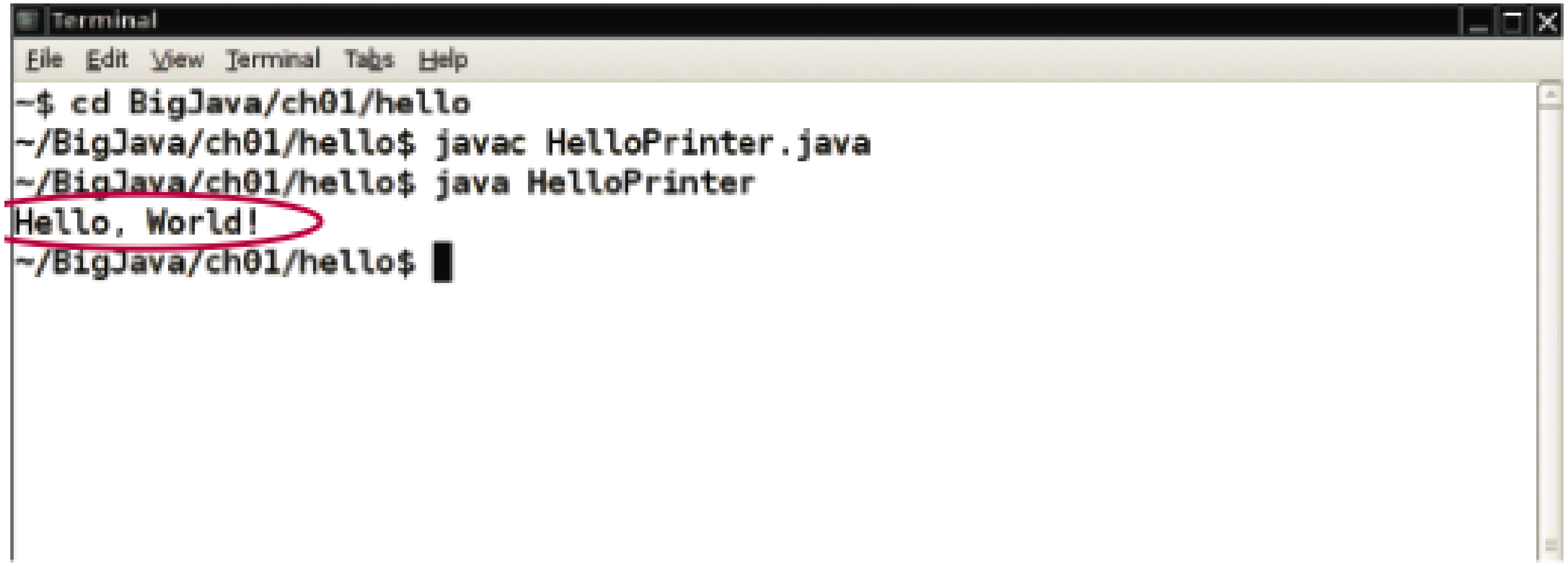
## Editing a Java Program

- Use an editor to enter and modify the program text
- Java is case-sensitive
  - *Be careful to distinguish between upper- and lowercase letters*
- Lay out your programs so that they are easy to read

## Compiling and Running a Java Program

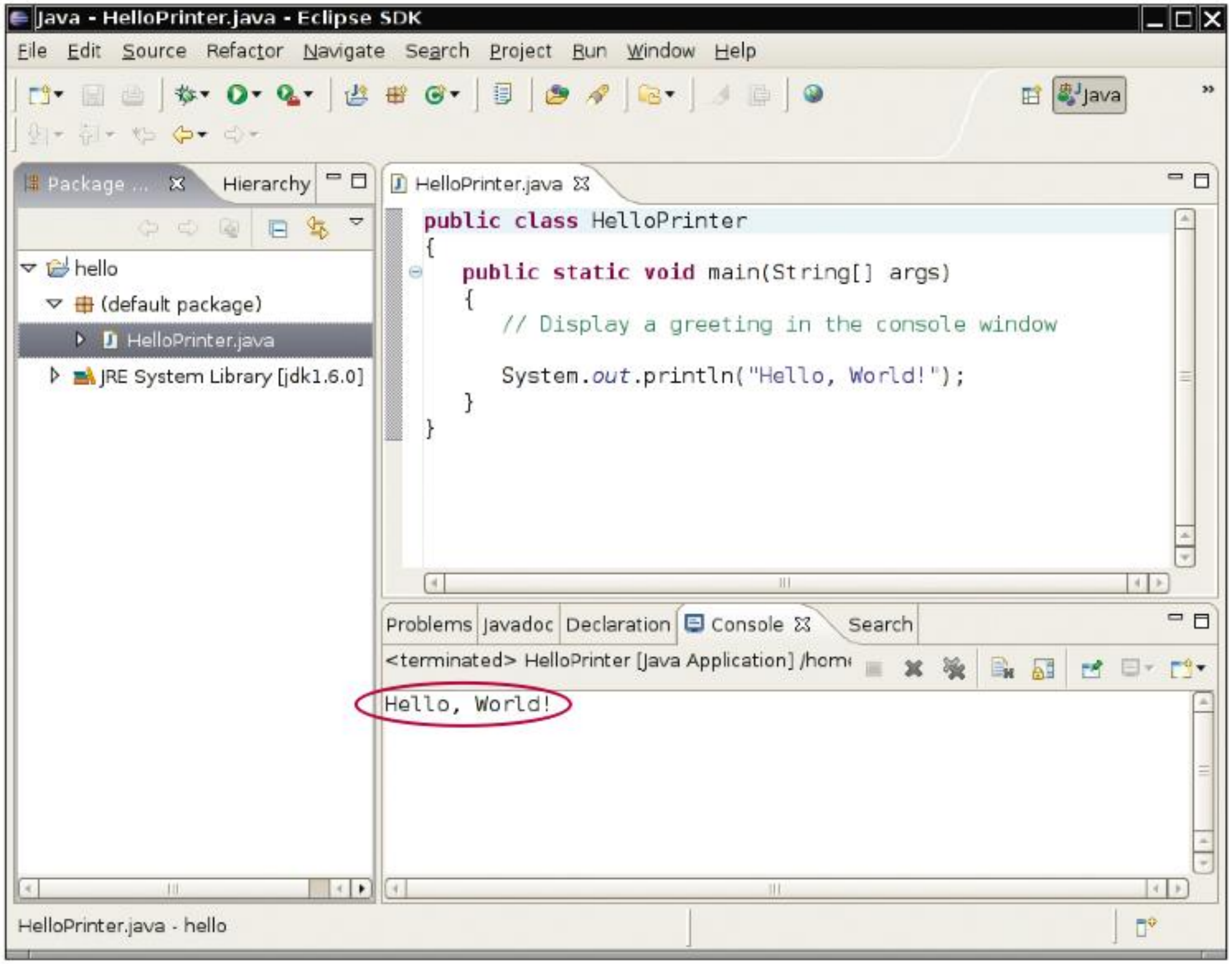
- The Java compiler translates source code into class files that contain instructions for the Java virtual machine
- A class file has extension `.class`
- The compiler does not produce a class file if it has found errors in your program
- The Java virtual machine loads instructions from the program's class file, starts the program, and loads the necessary library files as they are required

## `HelloPrinter` in a Console Window



```
~$ cd BigJava/ch01/hello
~/BigJava/ch01/hello$ javac HelloPrinter.java
~/BigJava/ch01/hello$ java HelloPrinter
Hello, World!
~/BigJava/ch01/hello$ 
```

## `HelloPrinter` in an IDE

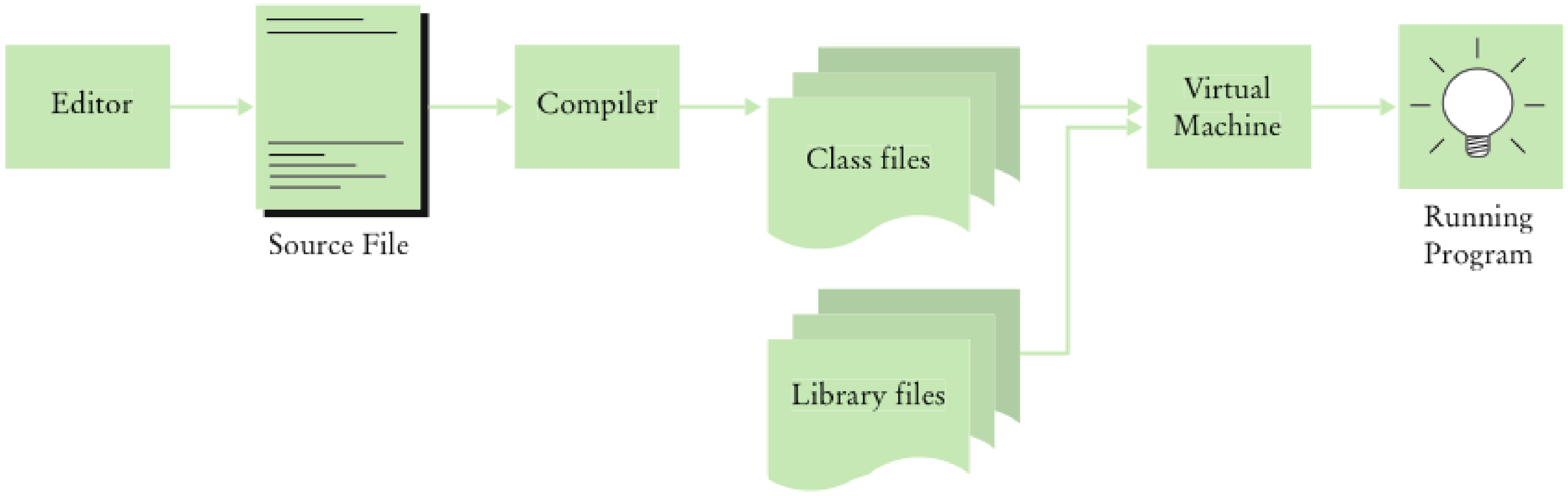

Java - HelloPrinter.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package ... Hierarchy
hello
(default package)
HelloPrinter.java
JRE System Library [jdk1.6.0]
HelloPrinter.java
Problems Javadoc Declaration Console Search
HelloPrinter.java - hello

```java
public class HelloPrinter
{
   public static void main(String[] args)
   {
      // Display a greeting in the console window

      System.out.println("Hello, World!");
   }
}
```

```
<terminated> HelloPrinter [Java Application] /hom
Hello, World!
```

## From Source Code to Running Program



## Self Check

Can you use a word processor for writing Java programs?

## Self Check

What do you expect to see when you load a class file into your text editor?

## Errors

- **Compile-time error:** A violation of the programming language rules that is detected by the compiler
  - *Example*:

    ```
    System.ou.println("Hello, World!);
    ```
  - *Syntax error*
- **Run-time error:** Causes the program to take an action that the programmer did not intend
  - *Examples:*

    ```
    System.out.println("Hello, Word!");
    System.out.println(1/0);
    ```
  - *Logic error*

## Error Management Strategy

- Learn about common errors and how to avoid them
- Use defensive programming strategies to minimize the likelihood and impact of errors
- Apply testing and debugging strategies to flush out those errors that remain

## Self Check

Suppose you omit the `//` characters from the `HelloPrinter.java` program but not the remainder of the comment. Will you get a compile-time error or a run-time error?

```
public class HelloPrinter
{
    public static void main(String[] args)
    {
        Display a greeting in the console window

        System.out.println("Hello, World!");
    }
}
```

## Self Check

When you used your computer, you may have experienced a program that “crashed” (quit spontaneously) or “hung” (failed to respond to your input). Is that behavior a compile-time error or a run-time error?

## Self Check

Why can't you test a program for run-time errors when it has compiler errors?

## Algorithms

- **Algorithm:** A sequence of steps that is:
    - *unambiguous*
    - *executable*
    - *terminating*
- Algorithm for deciding which car to buy, based on total costs:

```
For each car, compute the total cost as follows:
    annual fuel consumed = annual miles driven / fuel efficiency
    annual fuel cost = price per gallon x annual fuel consumed
    operating cost = 10 x annual fuel cost
    total cost = purchase price + operating cost
If total cost1 < total cost2
    Choose car1
Else
    Choose car2
```

## Pseudocode

- **Pseudocode:** An informal description of an algorithm:
  - *Describe how a value is set or changed:*

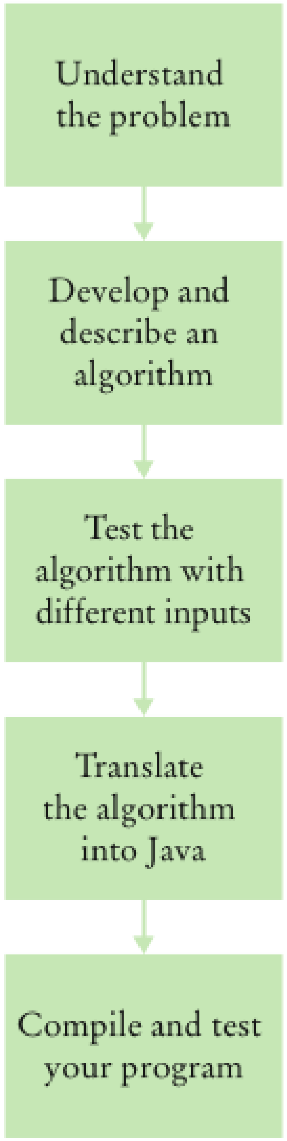
    ```
    total cost = purchase price + operating cost
    ```
  - *Describe decisions and repetitions:*

    ```
    For each car
       operating cost = 10 x annual fuel cost
       total cost = purchase price + operating cost
    ```

    *Use indentation to indicate which statements should be selected or repeated*
  - *Indicate results:*

    ```
    Choose car1
    ```

## Program Development Process


Understand the problem
Develop and describe an algorithm
Test the algorithm with different inputs
Translate the algorithm into Java
Compile and test your program

## Self Check

Investment Problem: You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

Algorithm:

```
Start with a year value of 0 and a balance of $10,000.
Repeat the following steps while the balance is less than $20,000.
Add 1 to the year value.
Multiply the balance value by 1.05 (a 5 percent increase).
```

Suppose the interest rate was 20 percent. How long would it take for the investment to double?

## Self Check

Suppose your cell phone carrier charges you $29.95 for up to 300 minutes of calls, and $0.45 for each additional minute, plus 12.5 percent taxes and fees. Give an algorithm to compute the monthly charge for a given number of minutes.

**Answer:**

Is the number of minutes at most 300?

a. If so, the answer is $29.95 × 1.125 = $33.70.

b. If not,

1. Compute the difference: (number of minutes) – 300.
2. Multiply that difference by 0.45.
3. Add $29.95.
4. Multiply the total by 1.125. That is the answer.

## AGENDA

- Introduction
- Fundamental Data Types
- Objects

## AGENDA

- Introduction
- Fundamental Data Types
- Objects

BASIC CONCEPTS OF JAVA 1

## FUNDAMENTAL DATA TYPES

## 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`, `3.3E24`, `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 `false` and `true` | 1 bit |

## Number Types: Floating-point Types

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

```
double f = 3.35;
System.out.println(100 * f); // prints 334.99999999999994
```

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

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

## Self Check

Which are the most commonly used number types in Java?

## Self Check

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

## Self Check

Which of the following initializations are incorrect, and why?

```
a. int dollars = 100.0;
b. double balance = 100;
```

## 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

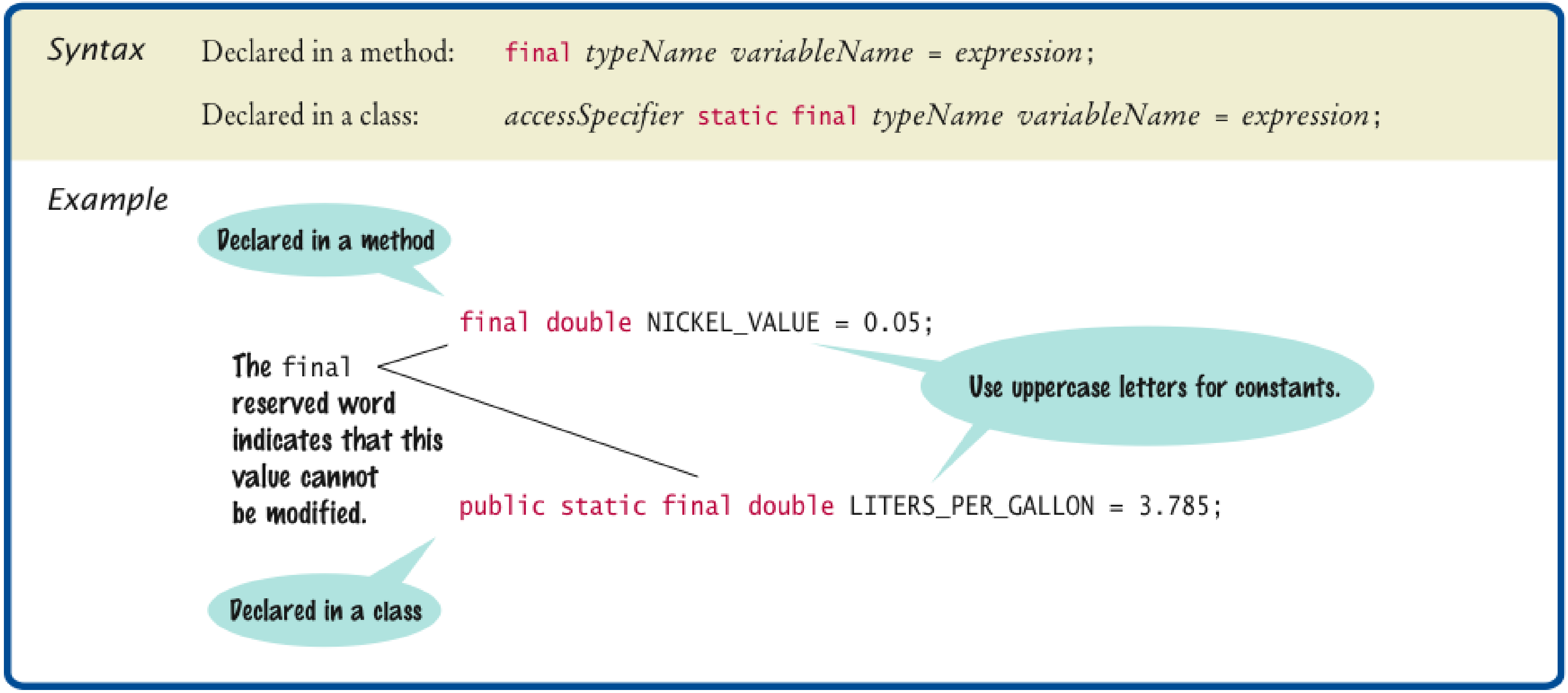
```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;
payment = dollars + quarters * QUARTER_VALUE
   + dimes * DIME_VALUE + nickels * NICKEL_VALUE
   + pennies * PENNY_VALUE;
```

## Constants: `static final` 1 Constant cho nhiều class

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

## 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;
   }

```

## 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;
   }

```

## CashRegister.java (cont.)

```java
    /**
       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;
    }
 }
```

# CashRegisterTester.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");
   }
}
```

## CashRegisterTester.java (cont.)

Program Run:

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

## Self Check

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;
```

## Self Check

What is wrong with the following statement sequence?

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

3.14 phải gán là const.

## Arithmetic Operators

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

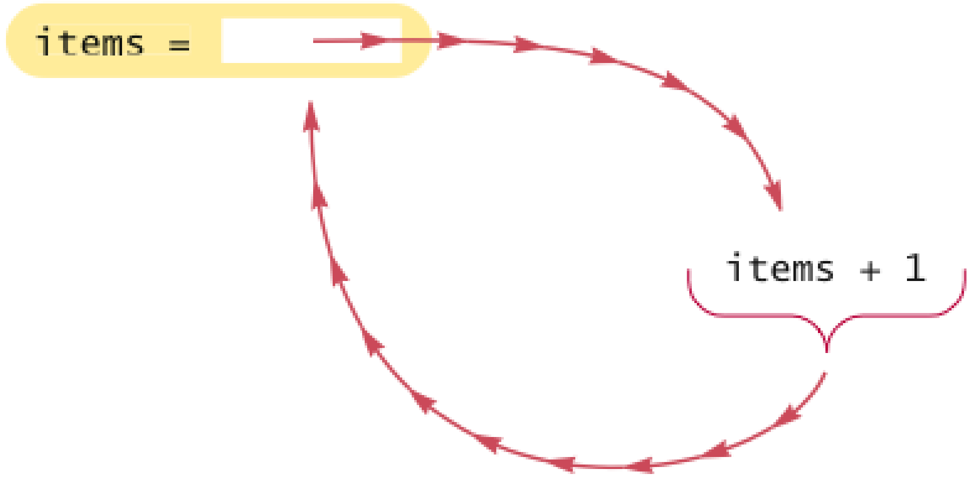
```
(a + b) / 2
```

- 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`



## 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`

## 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;
```

## 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)
```

## Analyzing an Expression

$$\underbrace{\underbrace{\texttt{(-b + }\underbrace{\texttt{Math.sqrt(}\underbrace{\underbrace{\texttt{b * b}}_{b^2}\texttt{ - }\underbrace{\texttt{4 * a * c}}_{4ac}}_{b^2-4ac}\texttt{)}}_{\sqrt{b^2-4ac}}\texttt{)}}_{-b+\sqrt{b^2-4ac}}\texttt{ / (}\underbrace{\texttt{2 * a}}_{2a}\texttt{)}}_{\frac{-b+\sqrt{b^2-4ac}}{2a}}$$

## Mathematical Methods

| 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 |

## Cast and Round

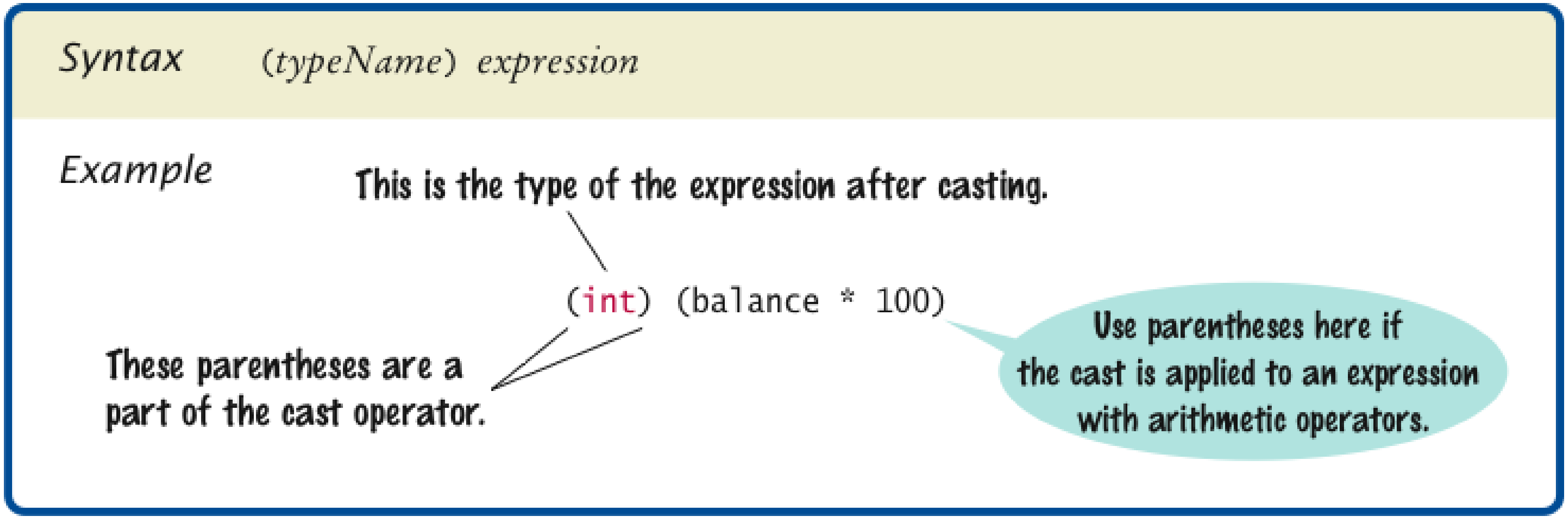
- Cast converts a value to a different type:

```
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 Cast



*Syntax* (*typeName*) *expression*

*Example*

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

This is the type of the expression after casting.
These parentheses are a part of the cast operator.
Use parentheses here if the cast is applied to an expression with arithmetic operators.

## 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

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

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

## Self Check

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

## Self Check

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

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

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

## Self Check

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

## Calling Static Methods

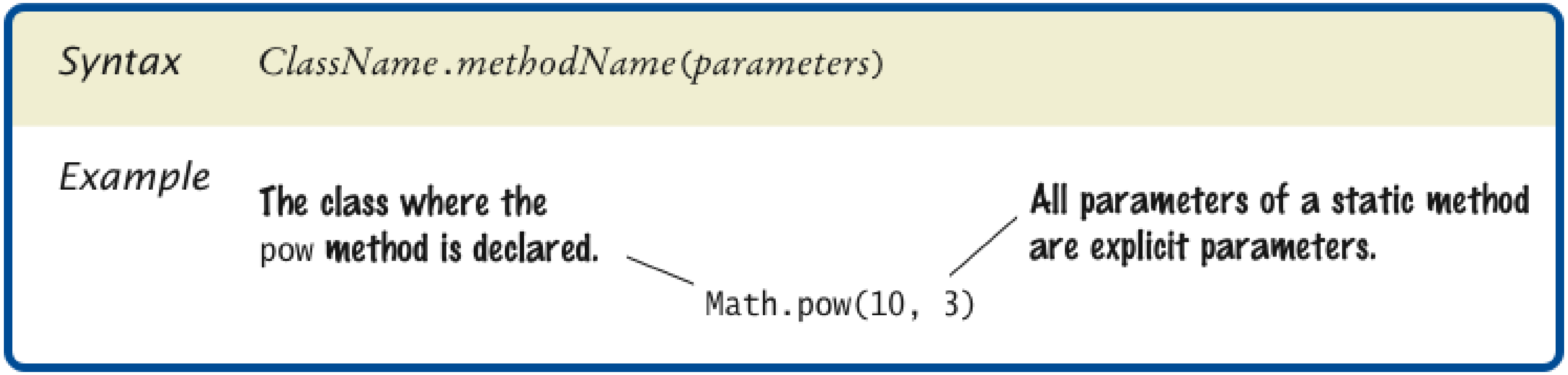
- A `static` method does not operate on an object

```
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
```

## Syntax Static Method Call



*Syntax* *ClassName.methodName(parameters)*

*Example*

The class where the pow method is declared.

All parameters of a static method are explicit parameters.

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

## Self Check

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

## Self Check

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

## 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: `""`

## 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:

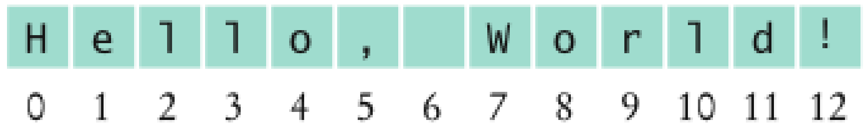
```
int n = Integer.parseInt(str);
double x = Double.parseDouble(x);
```

- Convert to string:

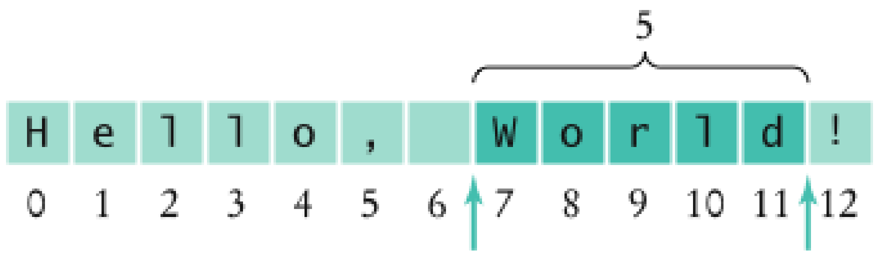
```
String str = "" + n;
str = Integer.toString(n);
```

## 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`



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



## Self Check

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

## Self Check

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)`?

## 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)

## 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();
```

## 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());
    }
}
```

## 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

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

## Self Check

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`?

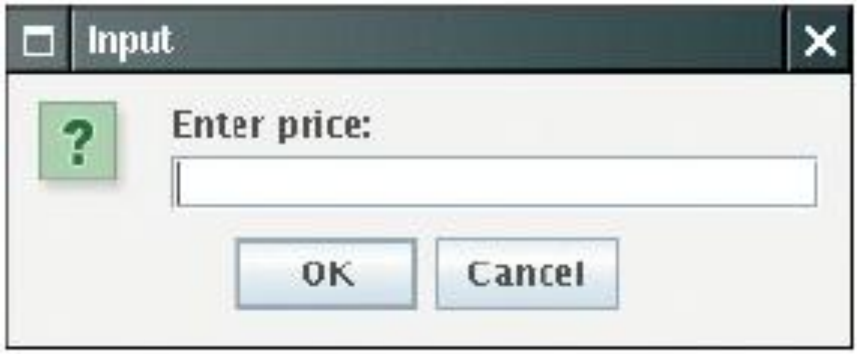
**Answer:** The value is `"John"`. The `next` method reads the next *word*.

## Reading Input From a Dialog Box

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



## AGENDA

- Introduction
- Fundamental Data Types
- Objects

## AGENDA

- Introduction
- Fundamental Data Types
- Objects

## BASIC CONCEPTS OF JAVA 1

## **OBJECTS**

## Chapter Goals

- To learn about variables
- To understand the concepts of classes and objects
- To be able to call methods
- To learn about parameters and return values
- To be able to browse the API documentation
- To implement test programs
- To understand the difference between objects and object references
- To write programs that display simple shapes

## Types

- A **type** defines a set of values and the operations that can be carried out on the values
- Examples:
  - *`13` has type `int`*
  - *`"Hello, World"` has type `String`*
  - *`System.out` has type `PrintStream`*
- Java has separate types for **integers** and **floating-point numbers**
  - *The `double` type denotes floating-point numbers*
- A value such as `13` or `1.3` that occurs in a Java program is called a **number literal**

## Number Literals

| Number | Type | Comment |
|---|---|---|
| 6 | `int` | An integer has no fractional part. |
| –6 | `int` | Integers can be negative. |
| 0 | `int` | Zero is an integer. |
| 0.5 | `double` | A number with a fractional part has type `double`. |
| 1.0 | `double` | An integer with a fractional part .0 has type `double`. |
| `1E6` | `double` | A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type `double`. |
| `2.96E-2` | `double` | Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$ |
| 🚫 100,000 | | **Error:** Do not use a comma as a decimal separator. |
| 🚫 3 1/2 | | **Error:** Do not use fractions; use decimal notation: 3.5. |

## Number Types

- A **type** defines a set of values and the operations that can be carried out on the values
- Number types are **primitive types**
    - *Numbers are not objects*
- Numbers can be combined by arithmetic operators such as `+`, `-`, and `*`

## Self Check

What is the type of the values `0` and `"0"`?

## Self Check

Which number type would you use for storing the area of a circle?

## Self Check

Why is the expression `13.println()` an error?

## Self Check

Write an expression to compute the average of the values `x` and `y`.

## Variables

- Use a **variable** to store a value that you want to use at a later time
- A variable has a type, a name, and a value:

```
String greeting = "Hello, World!”
PrintStream printer = System.out;
int width = 13;
```

- Variables can be used in place of the values that they store:

```
printer.println(greeting);
// Same as System.out.println("Hello, World!”)
printer.println(width);
// Same asSystem.out.println(20)
```

- It is an error to store a value whose type does not match the type of the variable:

```
String greeting = 20; // ERROR: Types don’t match
```

## Variable Declarations

| Variable Name | Comment |
|---|---|
| `int width = 10;` | Declares an integer variable and initializes it with 10. |
| `int area = width * height;` | The initial value can depend on other variables. (Of course, `width` and `height` must have been previously declared.) |
| 🚫 `height = 5;` | **Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.3. |
| 🚫 `int height = "5";` | **Error:** You cannot initialize a number with a string. |
| `int width, height;` | Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement. |

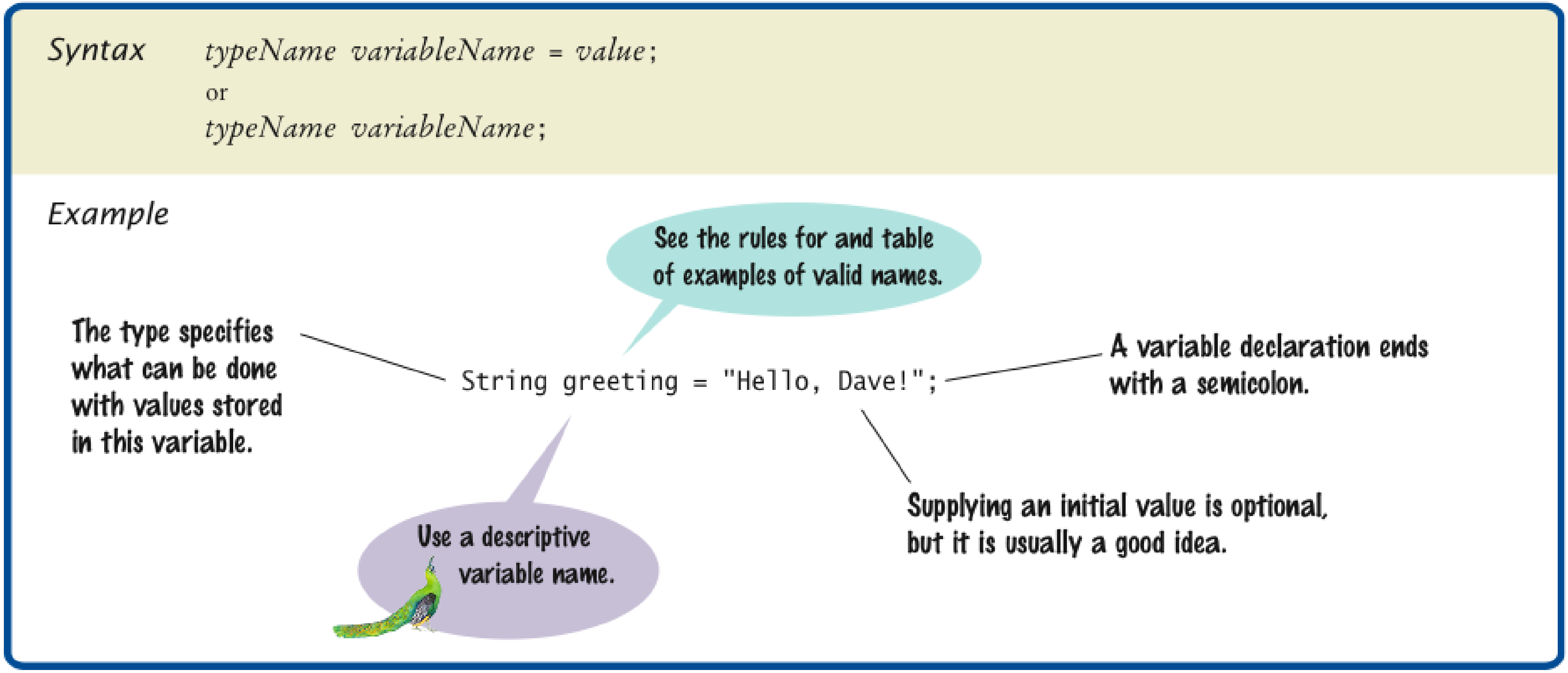
## Identifiers

- **Identifier:** name of a variable, method, or class
- Rules for identifiers in Java:
  - *Can be made up of letters, digits, and the underscore (_) and dollar sign (`$`) characters*
  - *Cannot start with a digit*
  - *Cannot use other symbols such as `?` or `%`*
  - *Spaces are not permitted inside identifiers*
  - *You cannot use reserved words such as public*
  - *They are case sensitive*

## Identifiers

- By convention, variable names start with a lowercase letter
  - *“Camel case”: Capitalize the first letter of a word in a compound word such as* `farewellMessage`
- By convention, class names start with an uppercase letter
- Do not use the `$` symbol in names - it is intended for names that are automatically generated by tools

## Syntax Variable Declaration



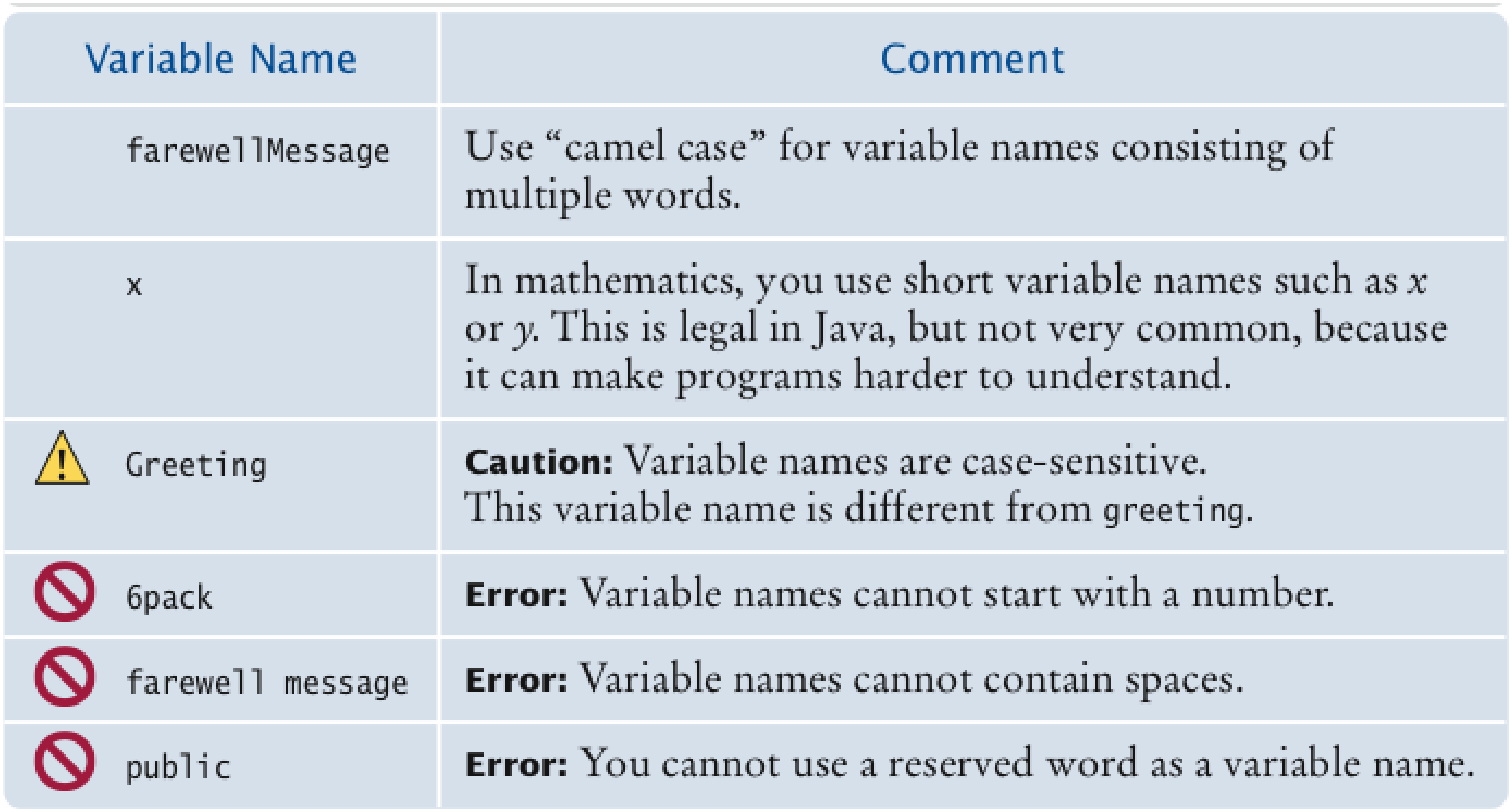
*Syntax* *typeName variableName = value;*
or
*typeName variableName;*

*Example*

```
String greeting = "Hello, Dave!";
```

See the rules for and table of examples of valid names.
The type specifies what can be done with values stored in this variable.
A variable declaration ends with a semicolon.
Use a descriptive variable name.
Supplying an initial value is optional, but it is usually a good idea.

## Variable Names



| | Variable Name | Comment |
|---|---|---|
| | `farewellMessage` | Use "camel case" for variable names consisting of multiple words. |
| | `x` | In mathematics, you use short variable names such as *x* or *y*. This is legal in Java, but not very common, because it can make programs harder to understand. |
| ⚠ | `Greeting` | **Caution:** Variable names are case-sensitive. This variable name is different from `greeting`. |
| 🚫 | `6pack` | **Error:** Variable names cannot start with a number. |
| 🚫 | `farewell message` | **Error:** Variable names cannot contain spaces. |
| 🚫 | `public` | **Error:** You cannot use a reserved word as a variable name. |

## Self Check

Which of the following are legal identifiers?

```
Greeting1
g
void          NO
101dalmatians
Hello, World    NO
<greeting>     NO
```

## Self Check

Define a variable to hold your name. Use camel case in the variable name.

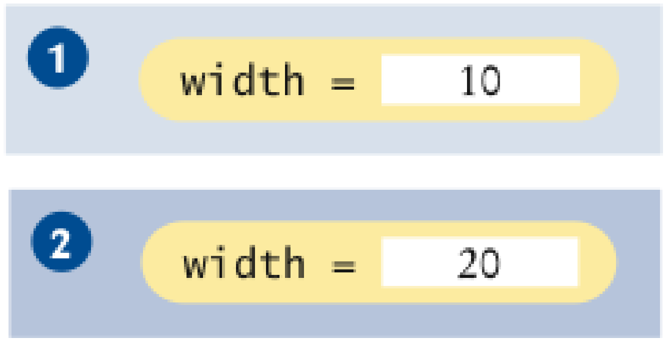
**Answer:**

```
String myName = “Huy Nguyen";
```

## The Assignment Operator

- Assignment operator: =
- Used to change the value of a variable:

```
int width= 10; 1
width = 20; 2
```



## Uninitialized Variables

- It is an error to use a variable that has never had a value assigned to it:

```
int height;
width = height; // ERROR-uninitialized variable height
```

height =
No value has been assigned.

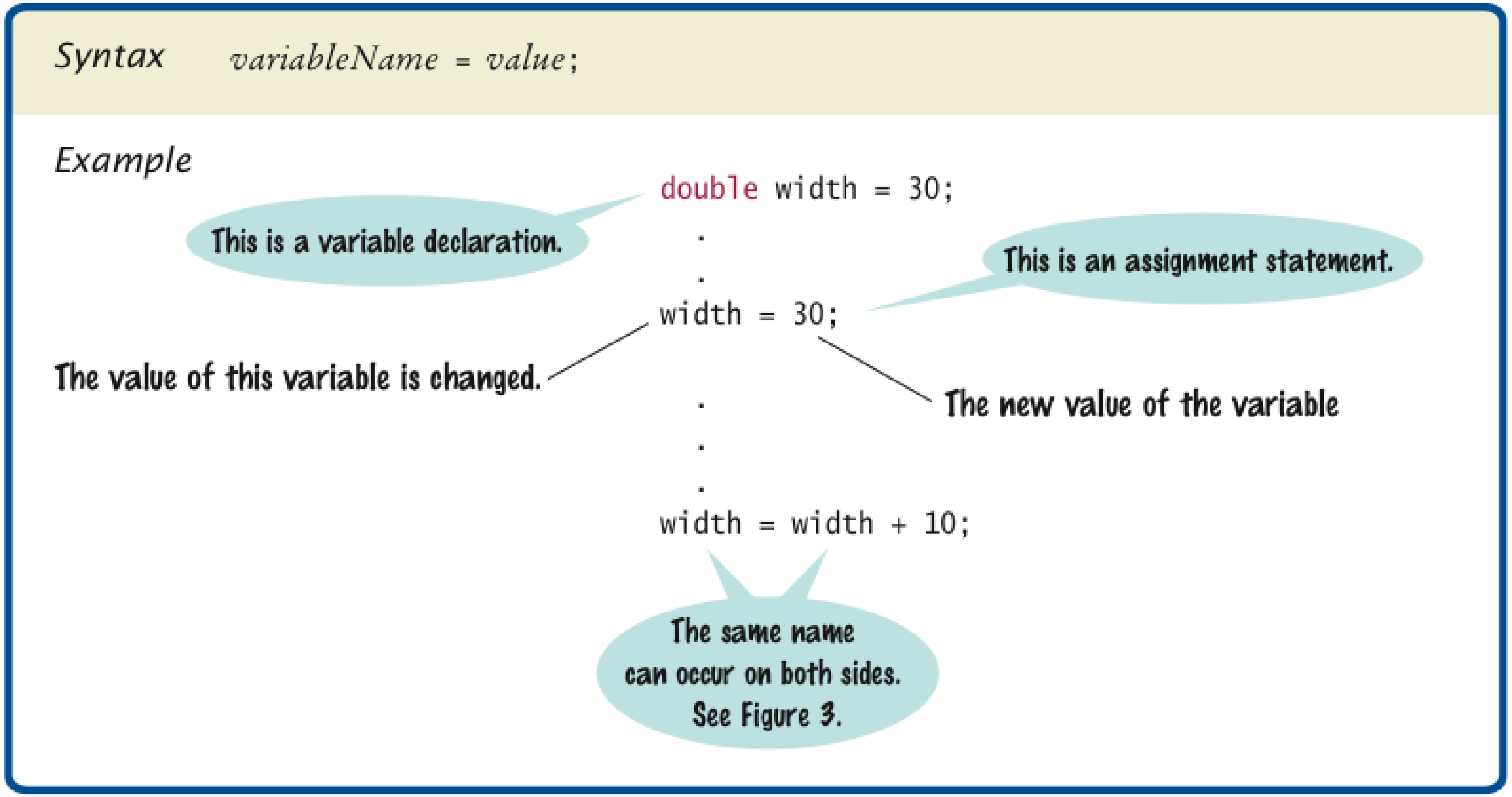
- Remedy: assign a value to the variable before you use it:

```
int height = 30;
width = height; // OK
```

- Even better, initialize the variable when you declare it:

```
int height = 30;
int width = height; // OK
```

## Syntax Assignment


Syntax variableName = value;

Example

```
double width = 30;
   .
   .
width = 30;
   .
   .
   .
width = width + 10;
```

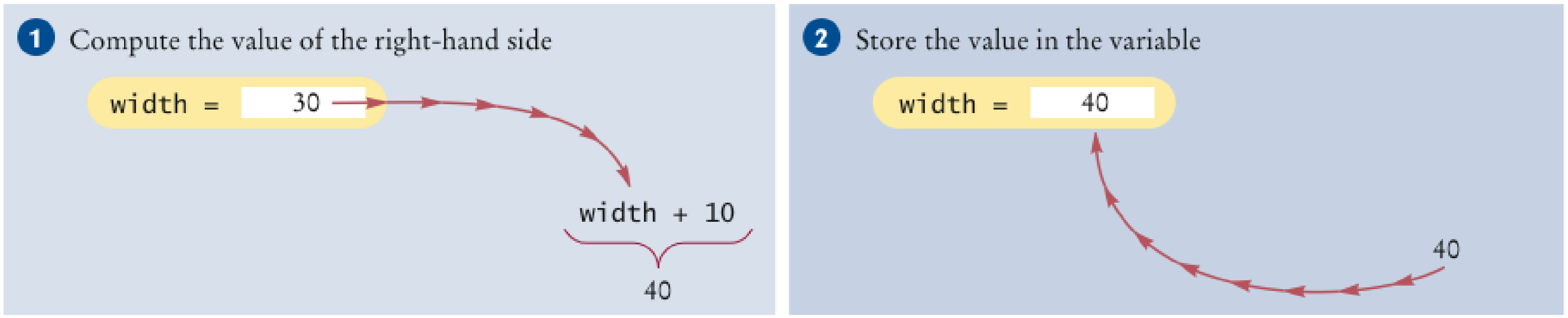
This is a variable declaration.
This is an assignment statement.
The value of this variable is changed.
The new value of the variable
The same name can occur on both sides. See Figure 3.

## Assignment

- The right-hand side of the = symbol can be a mathematical expression:

```
width = height + 10;
```

- Means:
  1. *compute the value of `width + 10`*
  2. *store that value in the variable `width`*


1 Compute the value of the right-hand side
width = 30
width + 10
40
2 Store the value in the variable
width = 40
40

## Self Check

Is `12 = 12` a valid expression in the Java language?

## Self Check

How do you change the value of the `greeting` variable to `"Hello, Nina!"`?

**Answer:**

```
greeting = "Hello, Nina!";
```

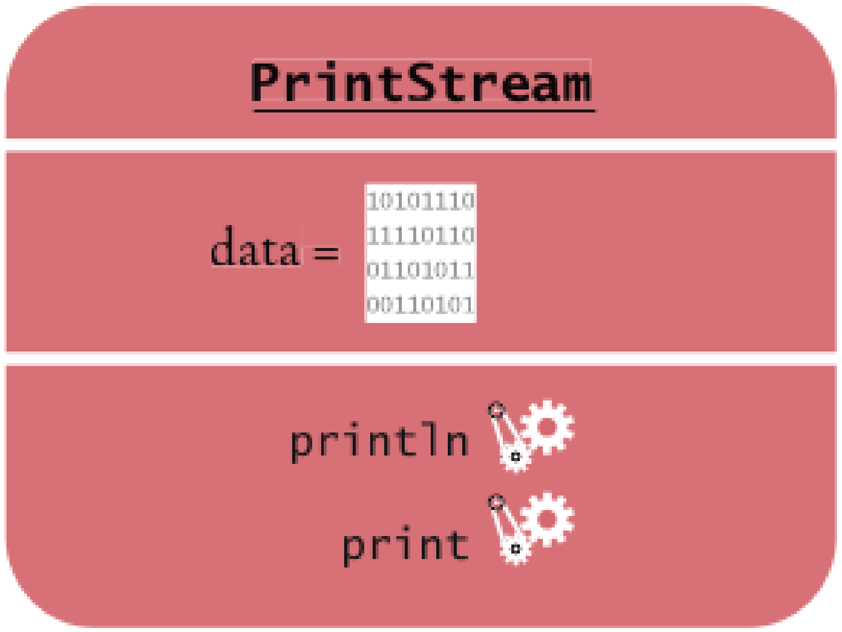
Note that

```
String greeting = "Hello, Nina!“;
```

is not the right answer - that statement defines a new variable.

## Objects and Classes

- **Object:** entity that you can manipulate in your programs (by calling methods)
- Each object belongs to a **class**
- Example: `System.out` belongs to the class `PrintStream`



## Methods

- **Method:** sequence of instructions that accesses the data of an object
- You manipulate objects by calling its methods
- **Class:** declares the methods that you can apply to its objects
- Class determines legal methods:

```
String greeting = "Hello";
greeting.println() // Error
greeting.length() // OK
```

- **Public Interface:** specifies what you can do with the objects of a class

## Overloaded Method

- **Overloaded method:** when a class declares two methods with the same name, but different parameters
- Example: the `PrintStream` class declares a second method, also called `println`, as

```
public void println(int output)
```

## String Methods

- `length`: counts the number of characters in a string:

```
String greeting = "Hello, World!";
int n = greeting.length(); // sets n to 13
```

- `toUpperCase`: creates another String object that contains the characters of the original string, with lowercase letters converted to uppercase:

```
 String river = "Mississippi";
 String bigRiver = river.toUpperCase();
 // sets bigRiver to "MISSISSIPPI"
```

- When applying a method to an object, make sure method is defined in the appropriate class:

```
System.out.length(); // This method call is an error
```

## Self Check

How can you compute the length of the string `"Mississippi"`?

**Answer:** `river.length()` or `"Mississippi".length()`

## Self Check

How can you print out the uppercase version of `"Hello, World!"`?

**Answer:**

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

## Self Check

Is it legal to call `river.println()`? Why or why not?

**Answer:** It is not legal. The variable `river` has type `String`. The `println` method is not a method of the `String` class.

## Parameters

- **Parameter:** an input to a method
- **Implicit parameter:** the object on which a method is invoked:

  ```
  System.out.println(greeting)
  ```

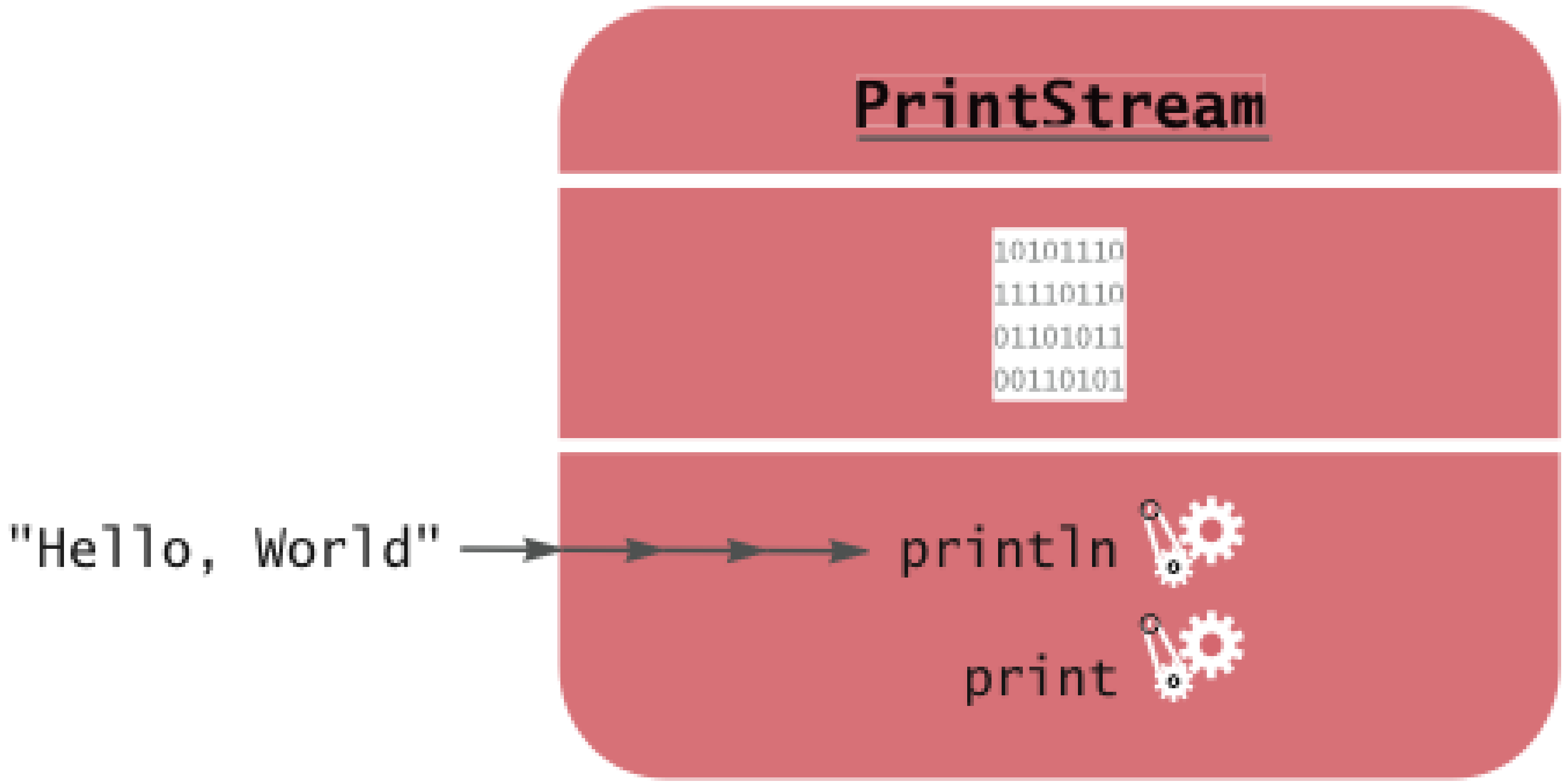
- **Explicit parameters:** all parameters except the implicit parameter:

  ```
  System.out.println(greeting)
  ```

- Not all methods have explicit parameters:

  ```
  greeting.length() // has no explicit parameter
  ```

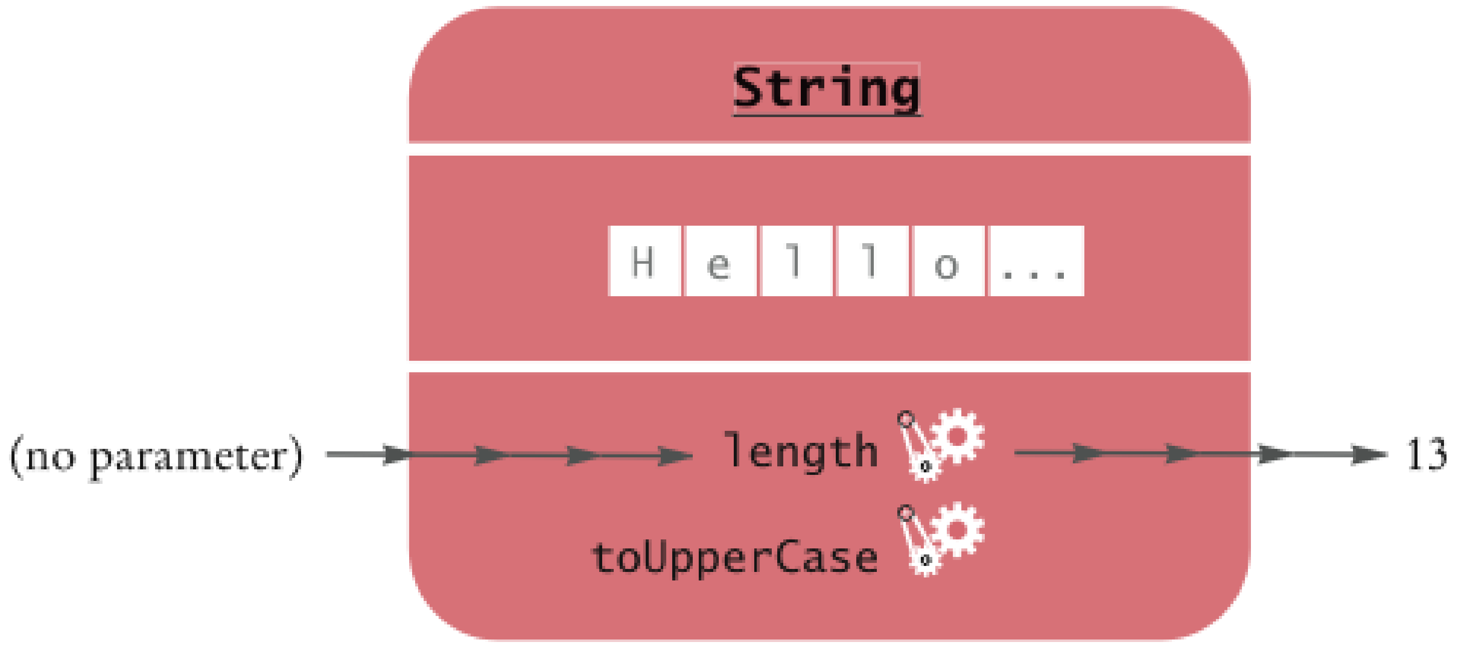
## Passing a Parameter



## Return Values

- **Return value:** a result that the method has computed for use by the code that called it:

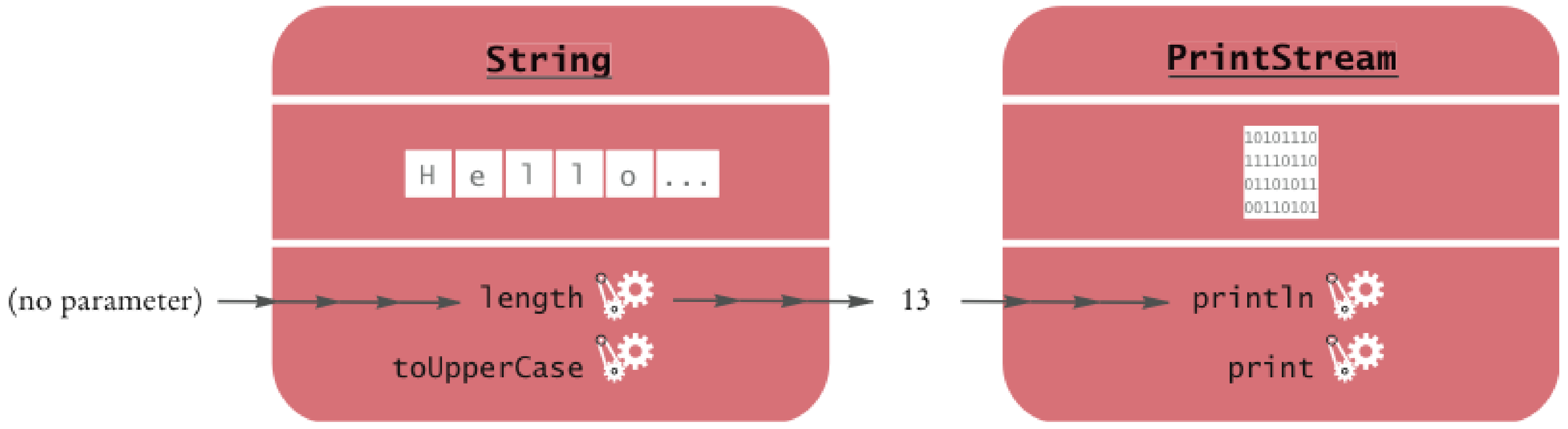
```
int n = greeting.length(); // return value stored in n
```



## Passing Return Values

- You can also use the return value as a parameter of another method:

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

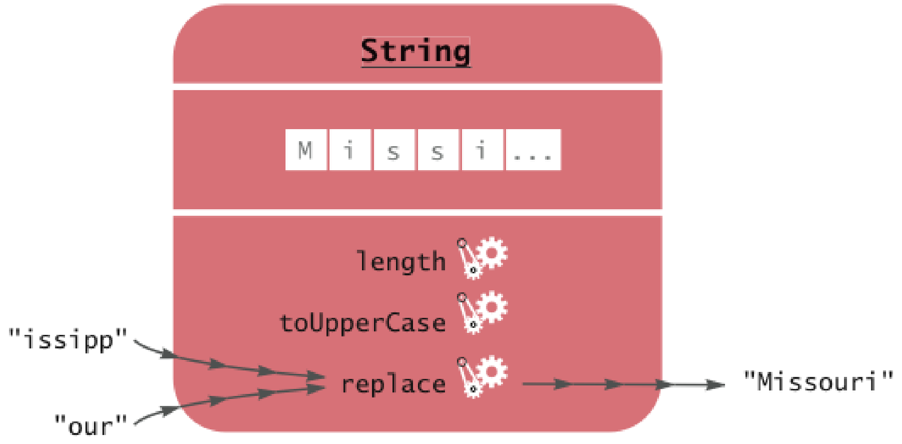


- Not all methods return values. Example: `println`

## A More Complex Call

- `String` method `replace` carries out a search-and-replace operation:

  ```
  river.replace("issipp", "our”)
  // constructs a new string ("Missouri")
  ```



- This method call has
  - *one implicit parameter: the string* `"Mississippi"`
  - *two explicit parameters: the strings* `"issipp"` *and* `"our"`
  - *a return value: the string* `"Missouri"`

## Self Check

What are the implicit parameters, explicit parameters, and return values in the method call `river.length()`?

## Self Check

What is the result of the call `river.replace("p", "s")`?

**Answer:** `"Missississi"`.

## Self Check

What is the result of the call

```
    String greeting = "Hello, World!";
    greeting.replace("World", "Dave").length()?
```

## Self Check

How is the `toUpperCase` method defined in the `String` class?

## Constructing Objects

```
new Rectangle(5, 10, 20, 30)
```

- Detail:
  1. *The `new` operator makes a `Rectangle` object*
  2. *It uses the parameters (in this case, `5`, `10`, `20`, and `30`) to initialize the data of the object*
  3. *It returns the object*
- Usually the output of the new operator is stored in a variable:

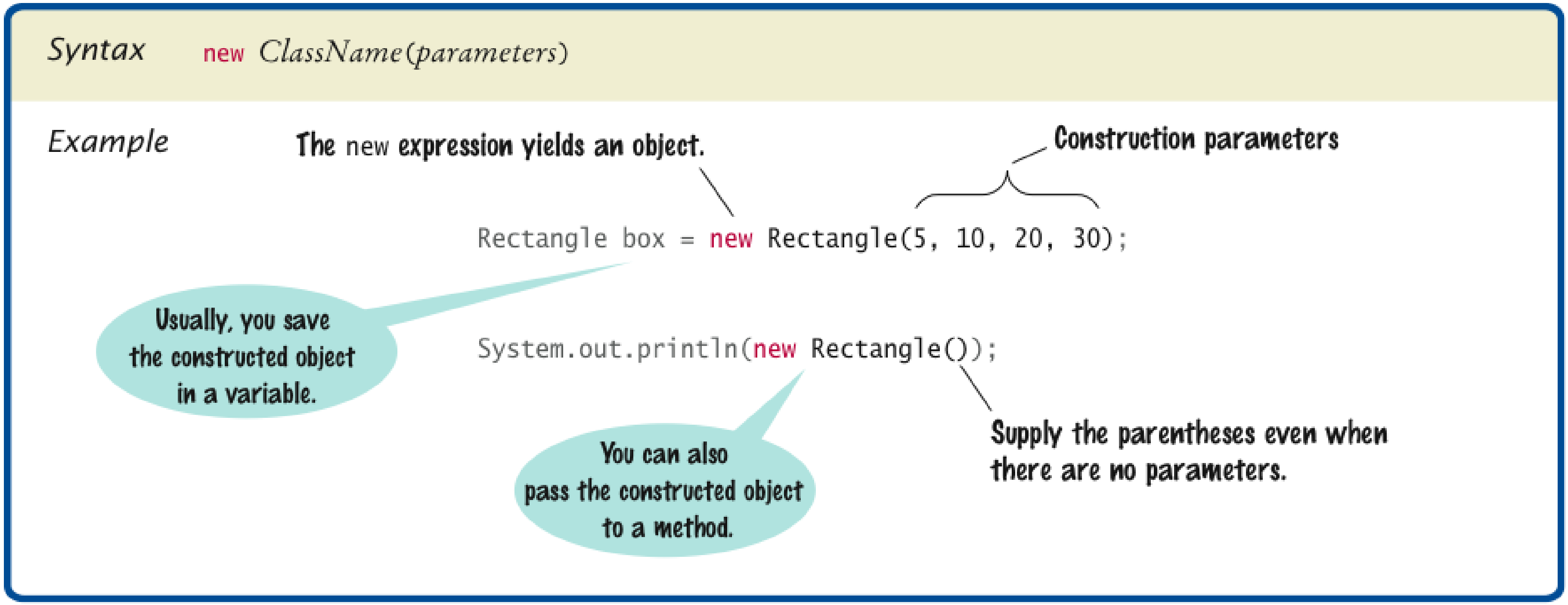
```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

## Constructing Objects

- **Construction:** the process of creating a new object
- The four values `5`, `10`, `20`, and `30` are called the *construction parameters*
- Some classes let you construct objects in multiple ways:

```
new Rectangle()
// constructs a rectangle with its top-left corner
// at the origin (0, 0), width 0, and height 0
```

## Syntax Object Construction



*Syntax* `new` *ClassName(parameters)*

*Example*

The `new` expression yields an object.

Construction parameters

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Usually, you save the constructed object in a variable.

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

You can also pass the constructed object to a method.

Supply the parentheses even when there are no parameters.

## Self Check

The `getWidth` method returns the width of a `Rectangle` object. What does the following statement print?

```
    System.out.println(new  Rectangle().getWidth());
```

## Accessor and Mutator Methods

- **Accessor method**: does not change the state of its implicit parameter:

```
double width = box.getWidth();
```

- **Mutator method**: changes the state of its implicit parameter:

```
box.translate(15, 25);
```

## Self Check

Is the `toUpperCase` method of the `String` class an accessor or a mutator?

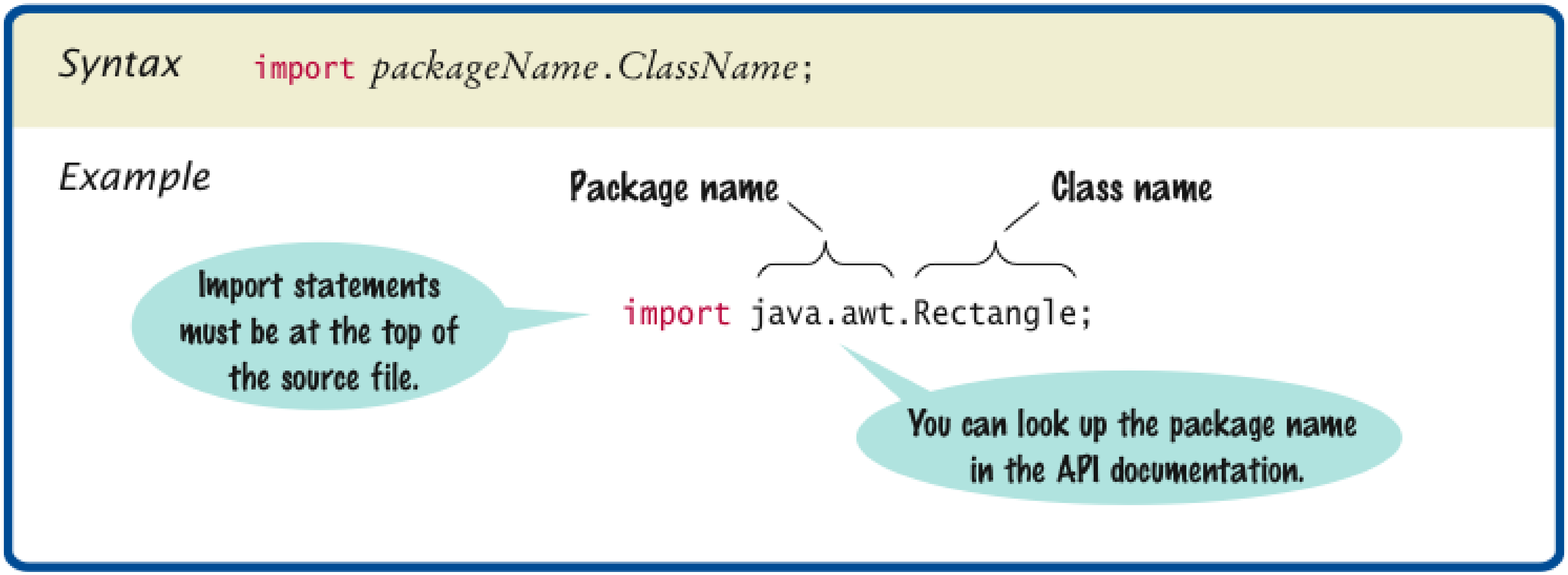
## Packages

- **Package**: a collection of classes with a related purpose
- Import library classes by specifying the package and class name:

```
import java.awt.Rectangle;
```

- You don’t need to import classes in the `java.lang` package such as `String` and `System`

## Syntax Importing a Class from a Package


*Syntax* `import` *packageName.ClassName*;

*Example*

```
import java.awt.Rectangle;
```

Package name
Class name
Import statements must be at the top of the source file.
You can look up the package name in the API documentation.

## The API Documentation

- **API:** Application Programming Interface
- **API documentation:** lists classes and methods in the Java library

## Detailed Method Description

The detailed description of a method shows:

- The action that the method carries out
- The parameters that the method receives
- The value that it returns (or the reserved word void if the method doesn’t return any value)

## Self Check

Look at the API documentation of the `String` class. Which method would you use to obtain the string `"hello, world!"` from the string `"Hello, World!“`?

**Answer:** `toLowerCase`

## Self Check

In the API documentation of the `String` class, look at the description of the `trim` method. What is the result of applying trim to the string `"  Hello, Space !  "`? (Note the spaces in the string.)

> **Answer:** `"Hello, Space !"` - only the leading and trailing spaces are trimmed.

## Self Check

The `Random` class is defined in the `java.util` package. What do you need to do in order to use that class in your program?

**Answer:** Add the statement

```
import java.util.Random;
```

at the top of your program.

## Implementing a Test Program

1. Provide a tester class.
2. Supply a `main` method.
3. Inside the `main` method, construct one or more objects.
4. Apply methods to the objects.
5. Display the results of the method calls.
6. Display the values that you expect to get.

## MoveTester.java

```
import java.awt.Rectangle;

public class MoveTester
{
   public static void main(String[] args)
   {
      Rectangle box = new Rectangle(5, 10, 20, 30);

      // Move the rectangle
      box.translate(15, 25);

      // Print information about the moved rectangle
      System.out.print("x: ");
      System.out.println(box.getX());
      System.out.println("Expected: 20");

      System.out.print("y: ");
      System.out.println(box.getY());
      System.out.println("Expected: 35");
   }
}
```

## MoveTester.java (cont.)

### Program Run:

```
x: 20
Expected: 20
y: 35
Expected: 35
```

## Self Check

Suppose we had called `box.translate(25, 15)` instead of `box.translate(15, 25)`. What are the expected outputs?

## Object References

- **Object reference:** describes the location of an object
- The `new` operator returns a reference to a new object:

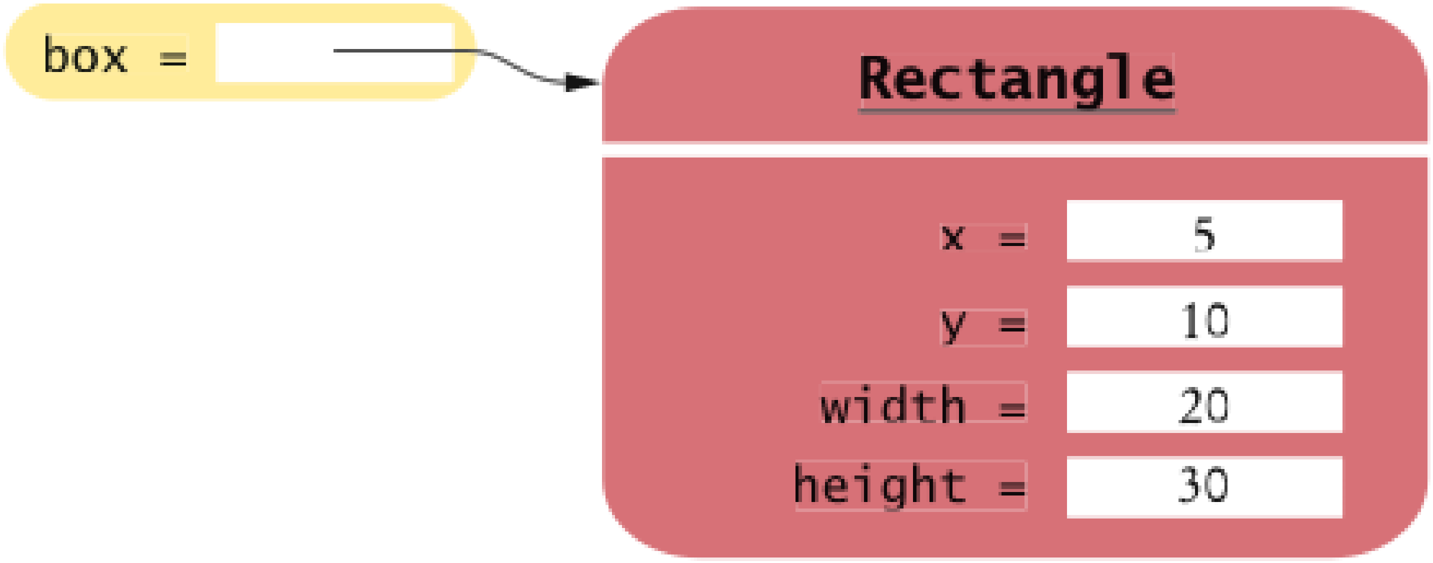
```
Rectangle box = new Rectangle();
```

- Multiple object variables can refer to the same object:

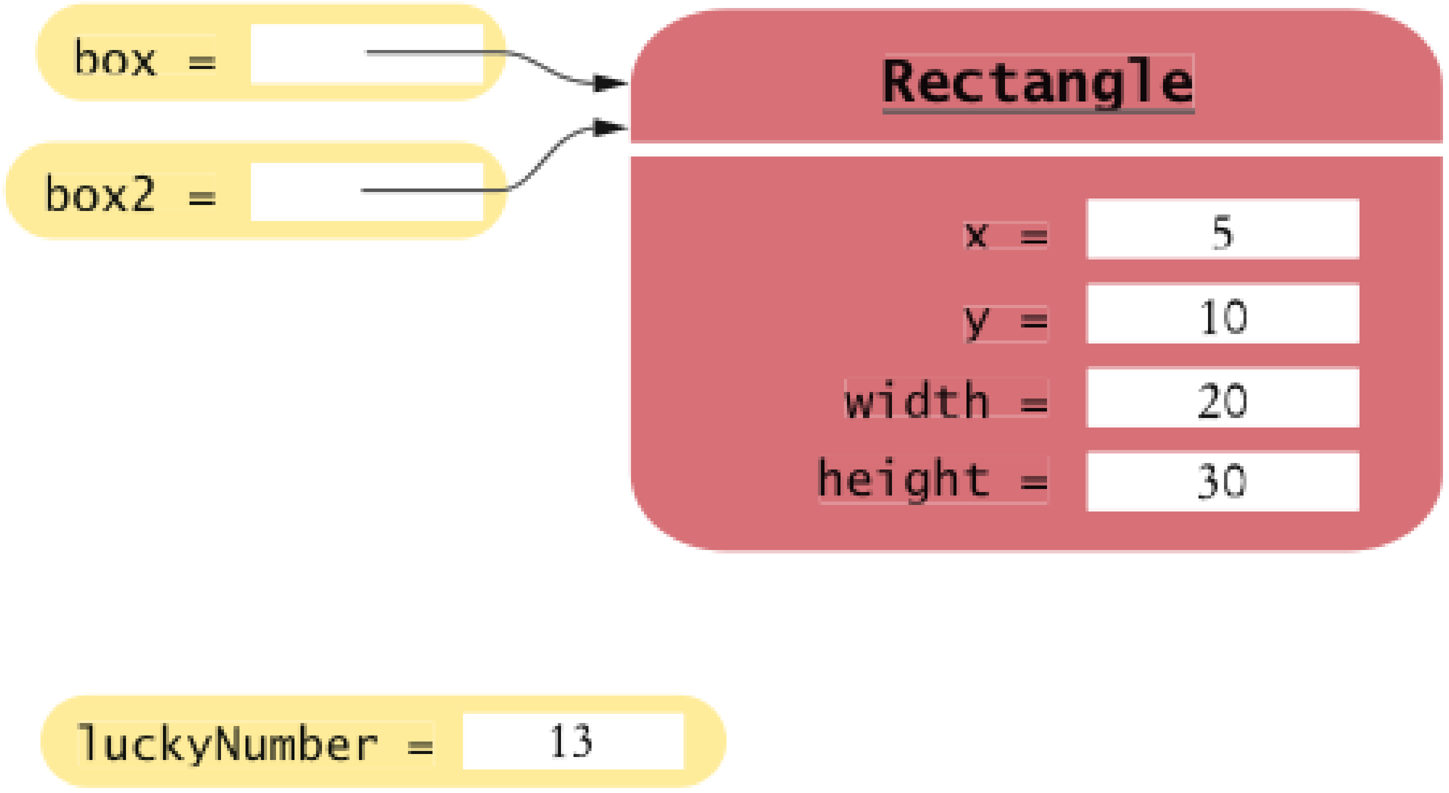
```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
```

- Primitive type variables ≠ object variables

## Object Variables and Number Variables

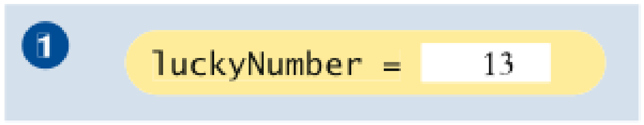


## Object Variables and Number Variables



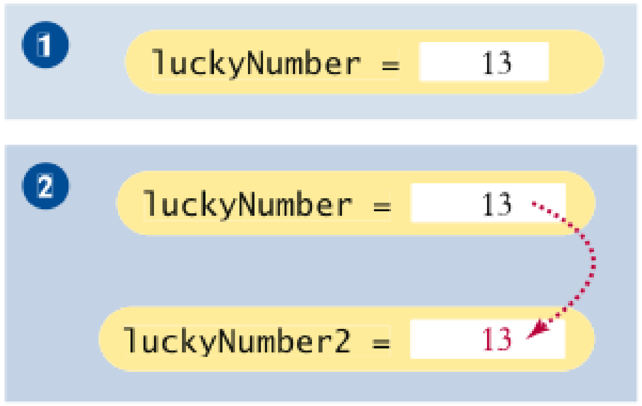
## Copying Numbers

```
int luckyNumber = 13; 1
```



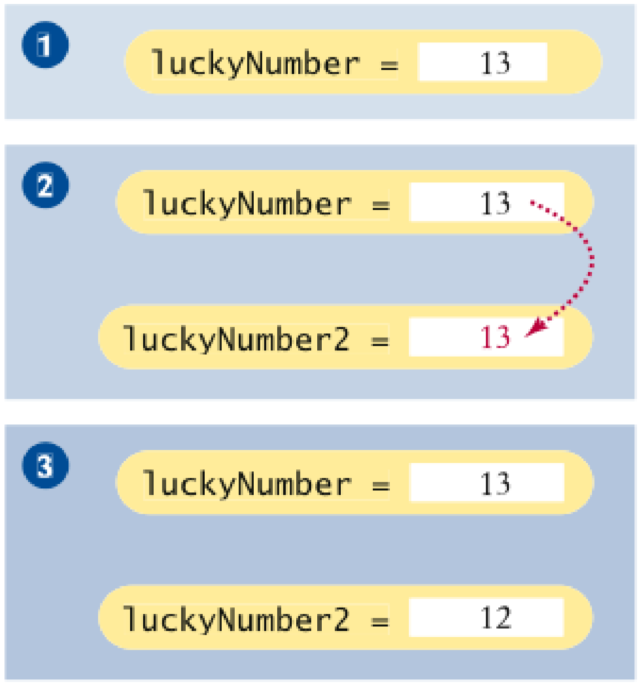
## Copying Numbers (cont.)

```
int luckyNumber = 13; ❶
int luckyNumber2 = luckyNumber; ❷
```



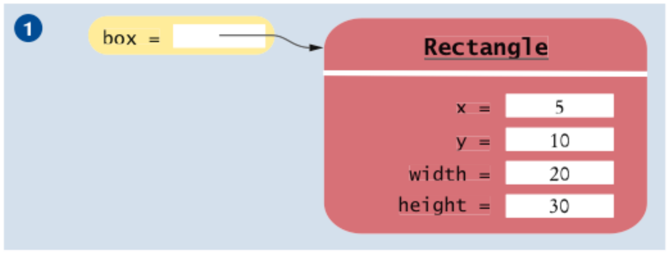
## Copying Numbers (cont.)

```
int luckyNumber = 13; ❶
int luckyNumber2 = luckyNumber; ❷
luckyNumber2 = 12; ❸
```



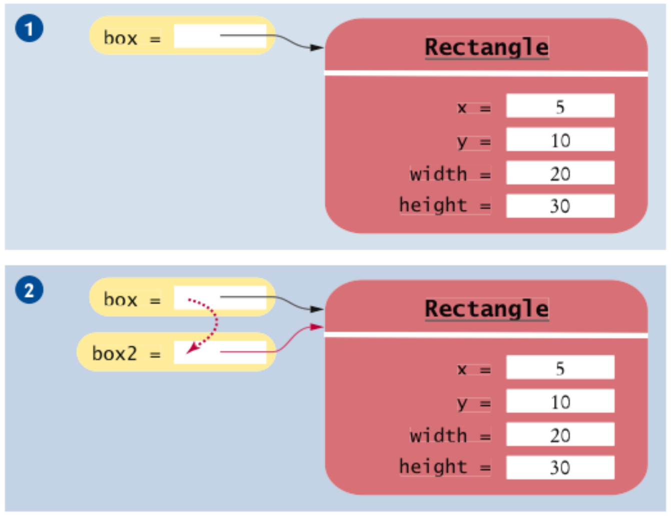
## Copying Object References

```
Rectangle box = new Rectangle(5, 10, 20, 30); 1
```



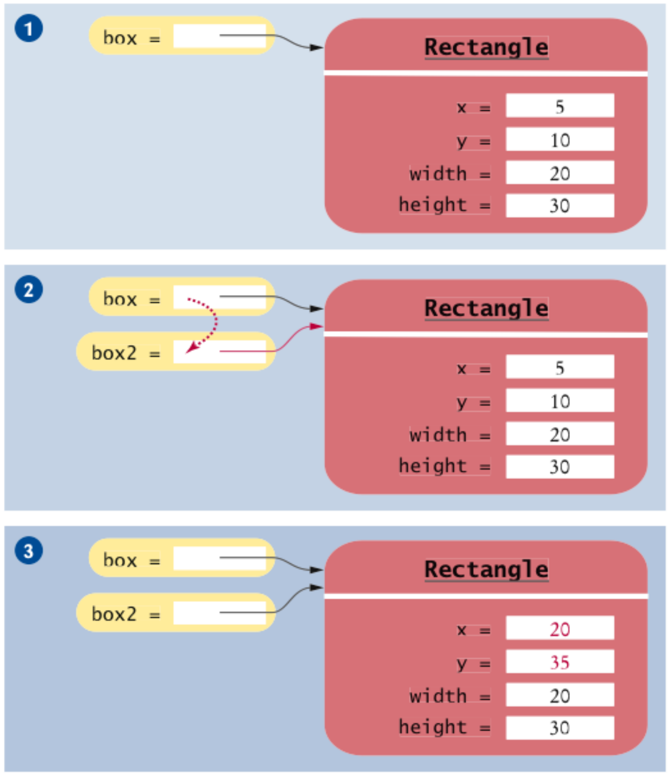
## Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30); 1
Rectangle box2 = box; 2
```



## Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30);1
Rectangle box2 = box; 2
Box2.translate(15, 25); 3
```



## Self Check

What is the effect of the assignment `greeting2 = greeting`?

## Self Check

After calling `greeting2.toUpperCase()`, what are the contents of `greeting` and `greeting2`?

## Graphical Applications and Frame Windows

To show a frame:

1. Construct an object of the `JFrame` class:
```
JFrame frame = new JFrame();
```
2. Set the size of the frame:
```
frame.setSize(300, 400);
```
3. If you'd like, set the title of the frame:
```
frame.setTitle("An Empty Frame");
```
4. Set the "default close operation":
```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```
5. Make the frame visible:
```
frame.setVisible(true);
```

## EmptyFrameViewer.java

```
import javax.swing.JFrame;

public class EmptyFrameViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      frame.setSize(300, 400);
      frame.setTitle("An Empty Frame");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.setVisible(true);
   }
}
```

## Self Check

How do you display a square frame with a title bar that reads `"Hello, World!”`?

**Answer:** Modify the `EmptyFrameViewer` program as follows:

```
frame.setSize(300, 300);
frame.setTitle("Hello, World!");
```

## Self Check

How can a program display two frames at once?

## Using a Component

1. Construct a frame.
2. Construct an object of your component class: `RectangleComponent component = new RectangleComponent();`
3. Add the component to the frame:
   `frame.add(component);`
4. Make the frame visible.

## RectangleComponent.java

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

/**
   A component that draws two rectangles.
*/
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;

      // Construct a rectangle and draw it
      Rectangle box = new Rectangle(5, 10, 20, 30);
      g2.draw(box);

      // Move rectangle 15 units to the right and 25 units down
      box.translate(15, 25);

      // Draw moved rectangle
      g2.draw(box);
   }
}
```

## RectangleViewer.java

```java
import javax.swing.JFrame;

public class RectangleViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      frame.setSize(300, 400);
      frame.setTitle("Two rectangles");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      RectangleComponent component = new RectangleComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

## Self Check

How do you modify the program to draw two squares?

**Answer:**

```
Rectangle box = new Rectangle(5, 10, 20, 20);
```

## Self Check

What happens if you call `g.draw(box)` instead of `g2.draw(box)`?

THANK YOU FOR YOUR ATTENTION !