Chapter 2

Getting Started with Java

Animated Version

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Objectives

After you have read and studied this chapter, you should be able to

- Identify the basic components of Java programs
- Write simple Java programs
- Describe the difference between object declaration and creation
- Describe the process of creating and running Java programs
- Use the Date, SimpleDateFormat, String, and JOptionPane standard classes
- Develop Java programs, using the incremental development approach

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The First Java Program

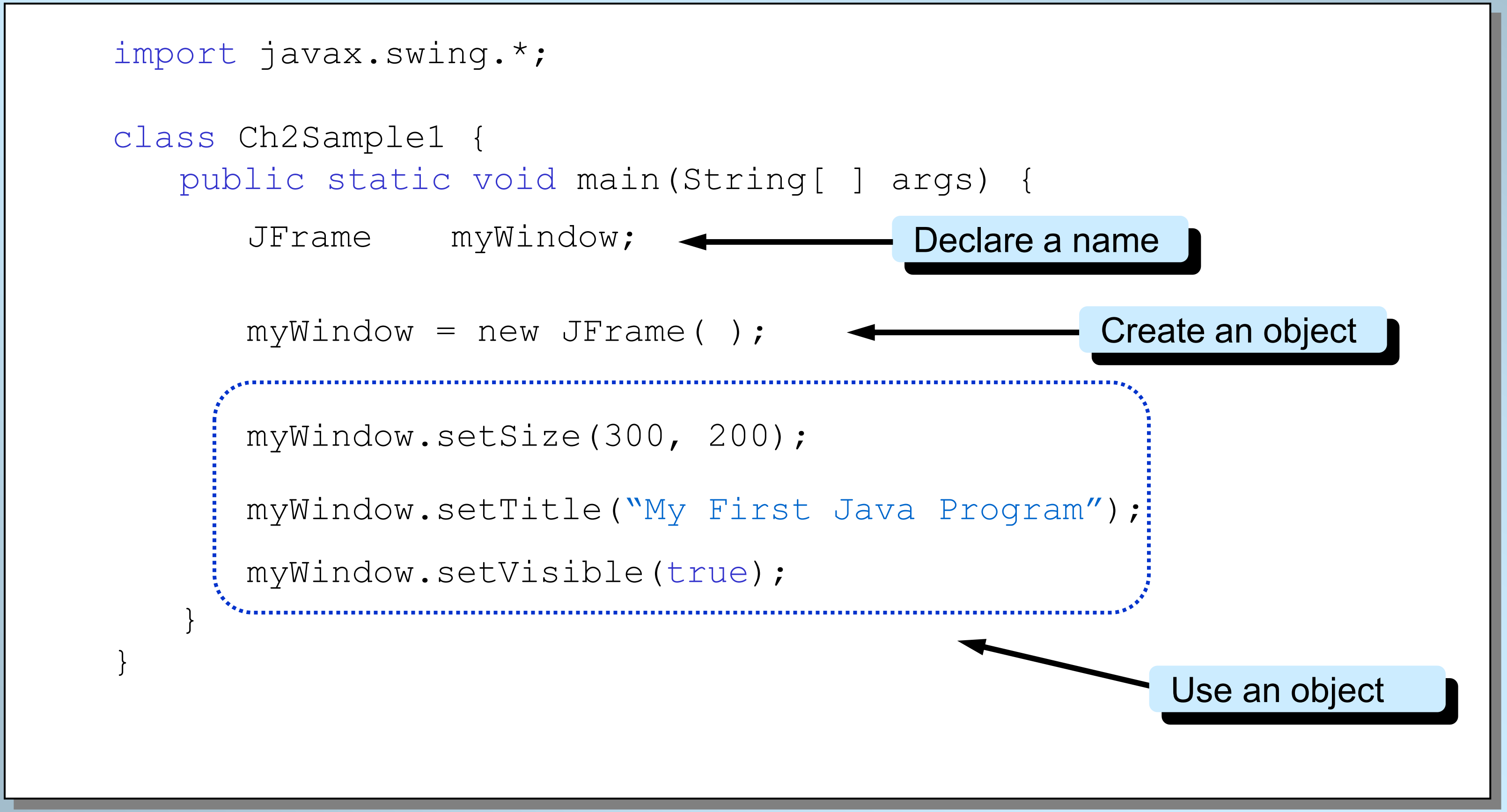
- The fundamental OOP concept illustrated by the program:

  An object-oriented program uses objects.

- This program displays a window on the screen.
- The size of the window is set to 300 pixels wide and 200 pixels high. Its title is set to My First Java Program.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Program Ch2Sample1



```
import javax.swing.*;

class Ch2Sample1 {
   public static void main(String[ ] args) {
      JFrame    myWindow;

      myWindow = new JFrame( );

      myWindow.setSize(300, 200);

      myWindow.setTitle(“My First Java Program”);

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

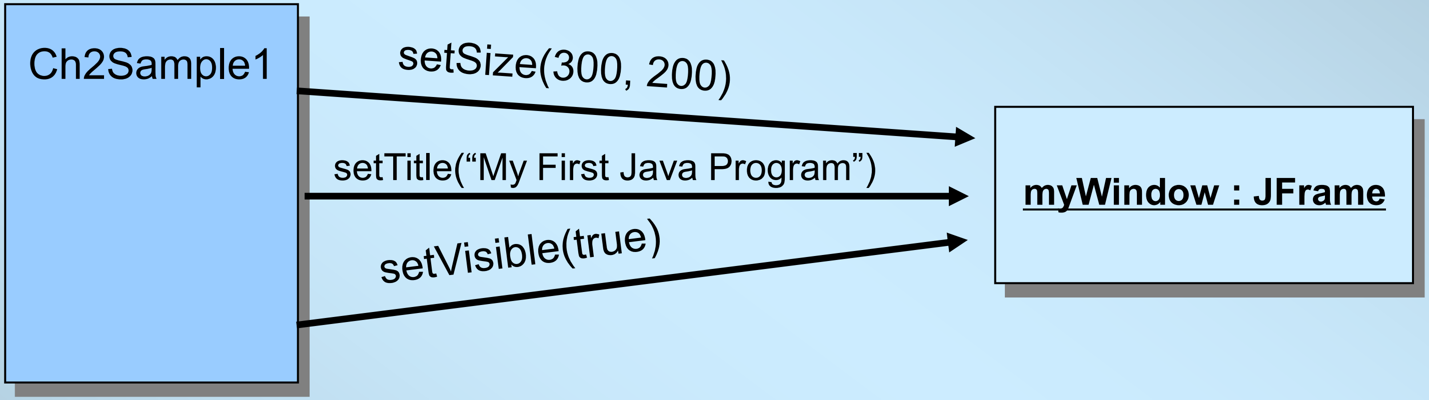
Declare a name

Create an object

Use an object

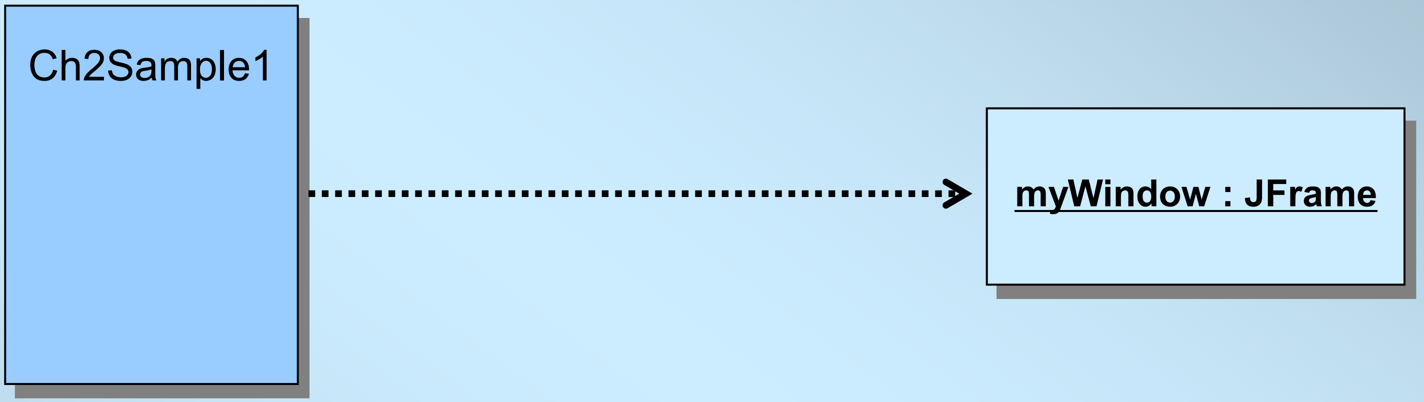
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Program Diagram for Ch2Sample1



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

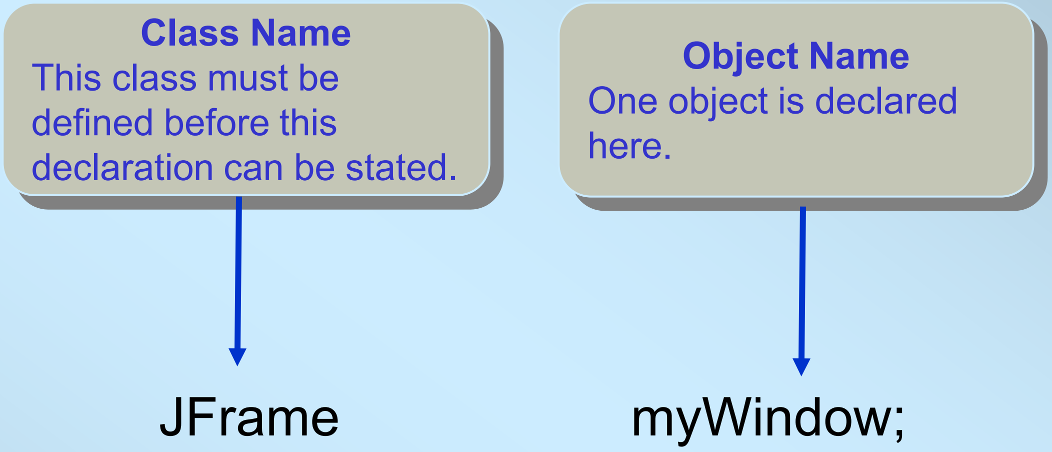
Dependency Relationship



Instead of drawing all messages, we summarize it by showing only the dependency relationship. The diagram shows that Ch2Sample1 “depends” on the service provided by myWindow.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Object Declaration



Class Name
This class must be defined before this declaration can be stated.

Object Name
One object is declared here.

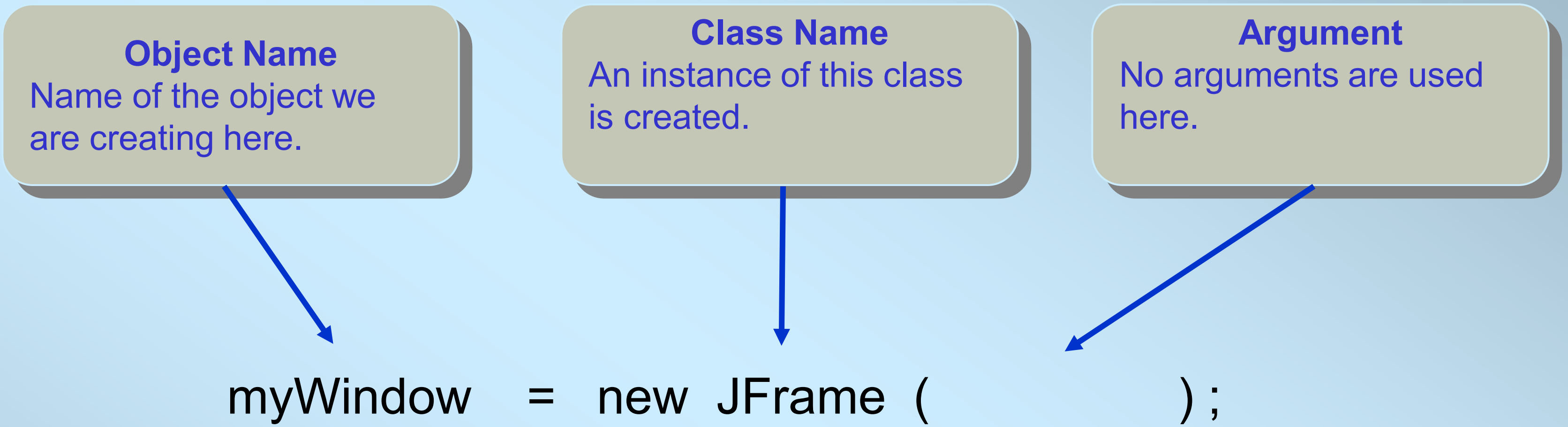
```
JFrame          myWindow;
```

More Examples

```
Account    customer;
Student    jan, jim, jon;
Vehicle    car1, car2;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Object Creation



Object Name
Name of the object we are creating here.

Class Name
An instance of this class is created.

Argument
No arguments are used here.

```
myWindow = new JFrame (         ) ;
```

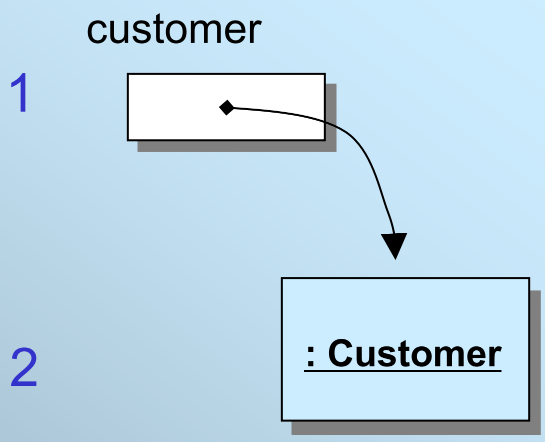
More Examples

```
customer = new Customer( );
jon      = new Student(“John Java”);
car1     = new Vehicle( );
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Declaration vs. Creation

```
1   Customer   customer;
2   customer   =  new  Customer( );
```

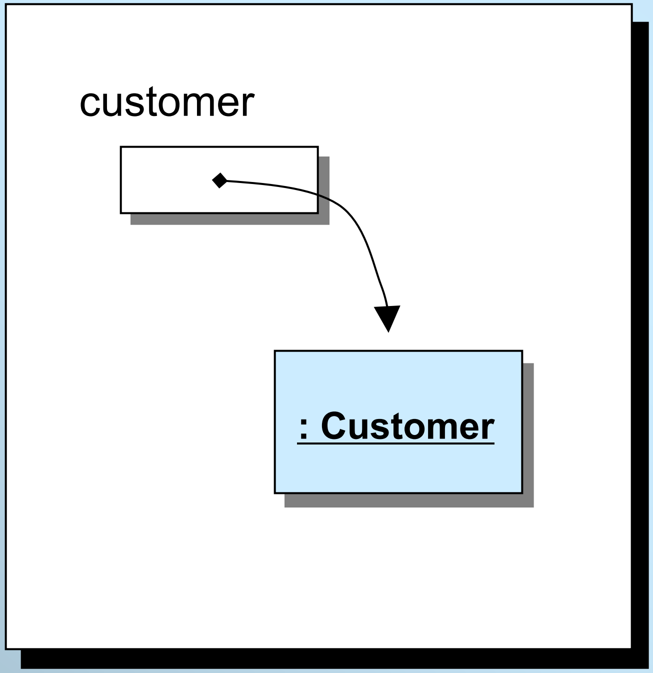


1. The identifier customer is declared and space is allocated in memory.

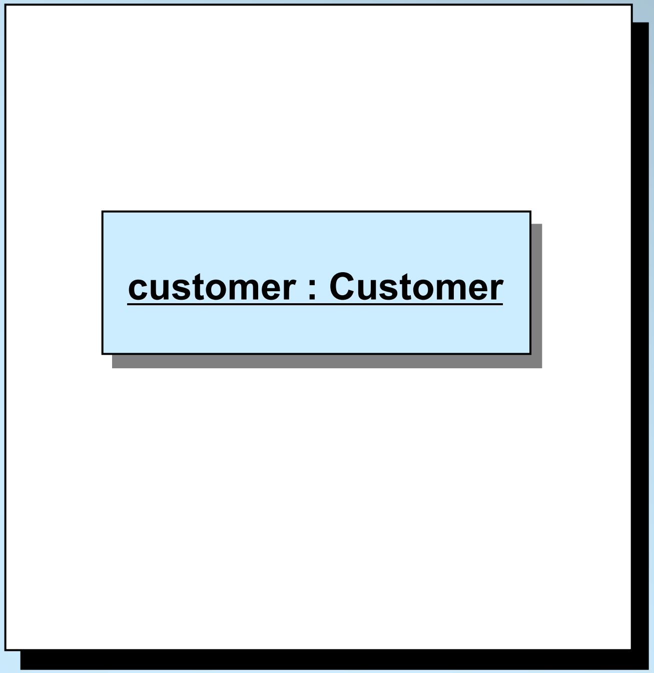
2. A Customer object is created and the identifier customer is set to refer to it.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

State-of-Memory vs. Program



State-of-Memory
Notation

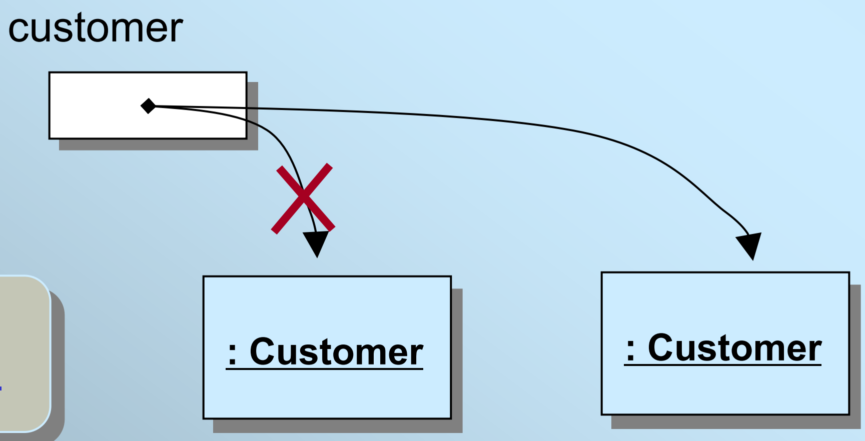


Program Diagram
Notation

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Name vs. Objects

```
Customer   customer;
customer   =  new  Customer( );
customer   =  new  Customer( );
```

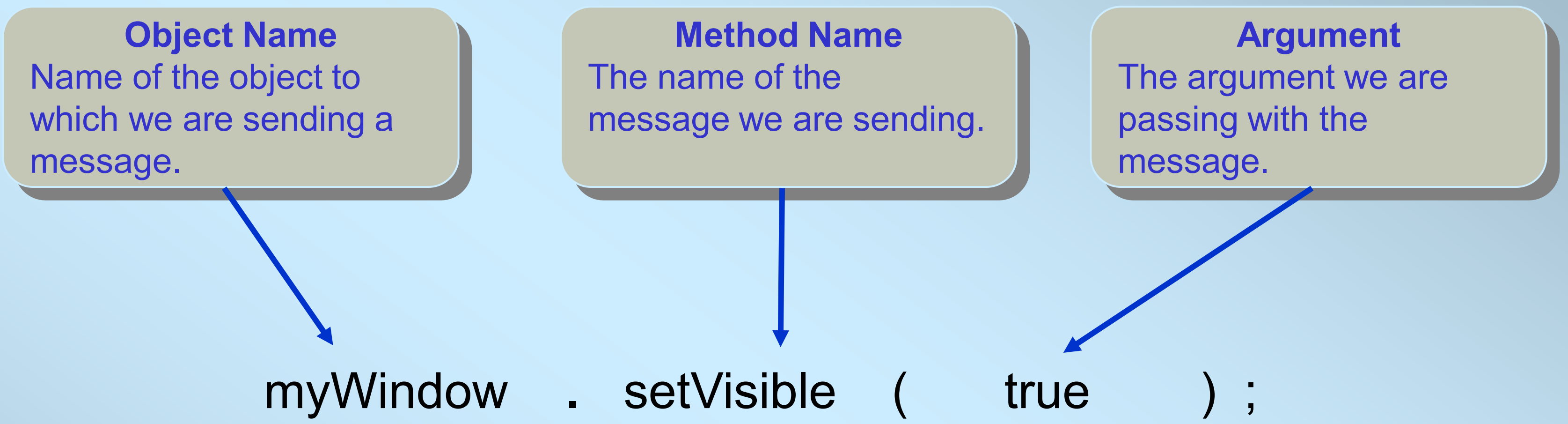


Created with the first **new**.

Created with the second **new**. Reference to the first Customer object is lost.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sending a Message



Object Name
Name of the object to which we are sending a message.

Method Name
The name of the message we are sending.

Argument
The argument we are passing with the message.

myWindow . setVisible (true) ;

More Examples

```
account.deposit( 200.0 );
student.setName(“john”);
car1.startEngine(  );
```

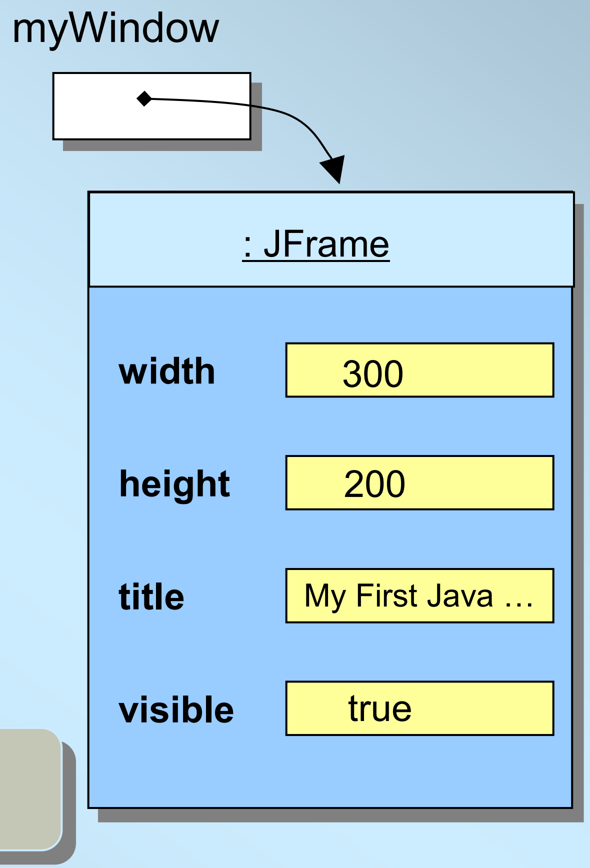
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Execution Flow

Program Code

```
Jframe     myWindow;

myWindow   = new JFrame( );

myWindow.setSize(300, 200);

myWindow.setTitle
         ("My First Java Program");

myWindow.setVisible(true);
```

State-of-Memory Diagram



The diagram shows only four of the many data members of a JFrame object.

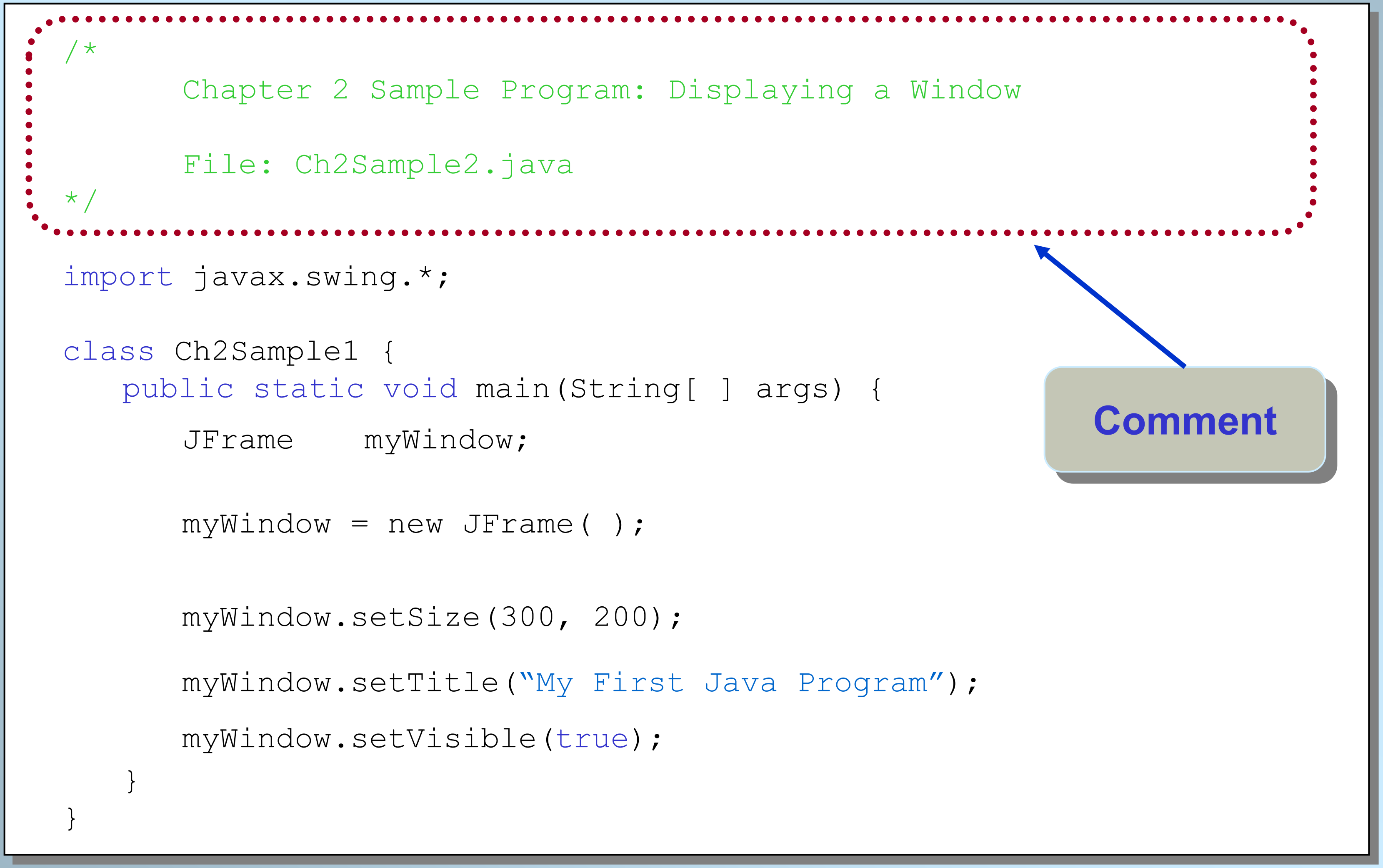
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Program Components

- A Java program is composed of
  - comments,
  - import statements, and
  - class declarations.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Program Component: Comment

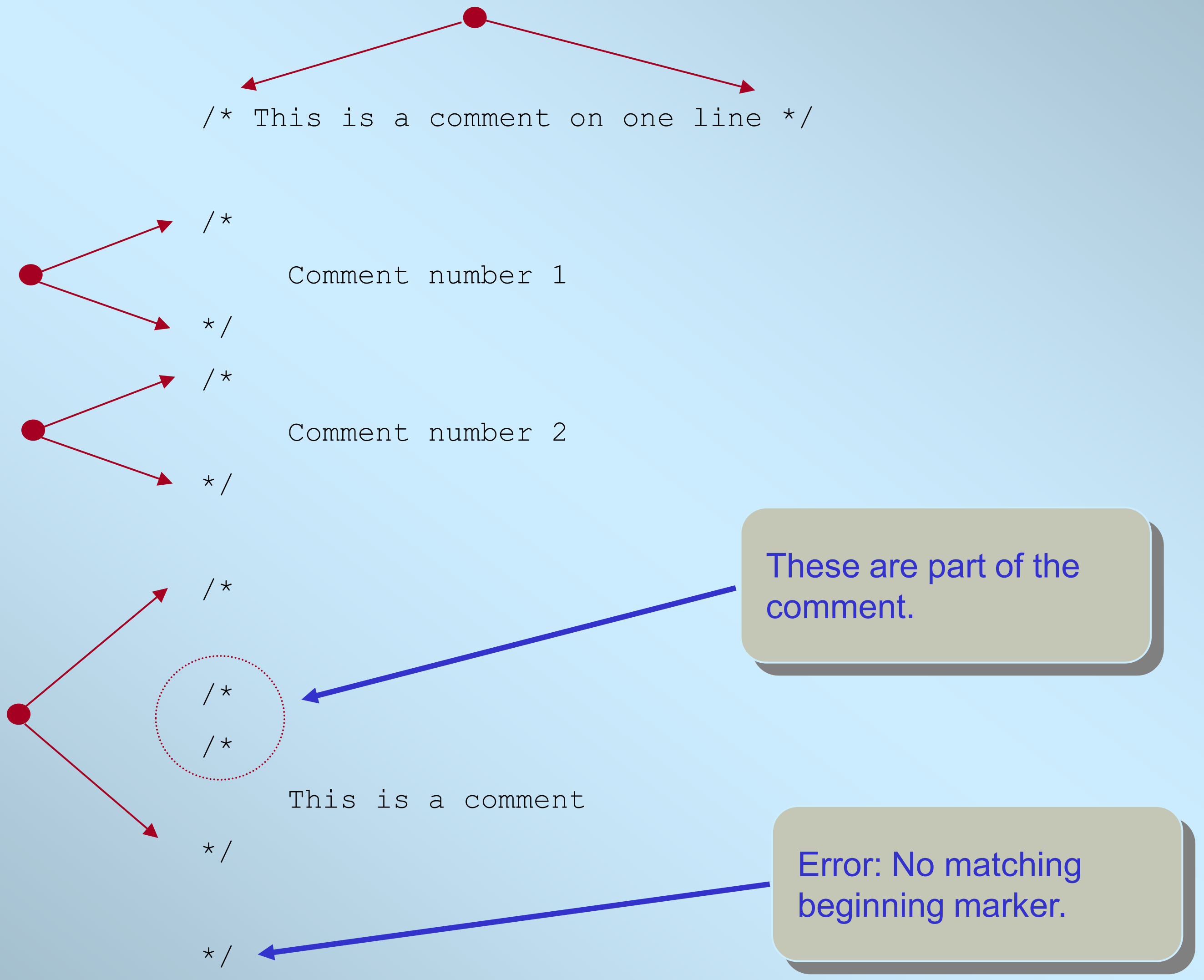


```
/*
      Chapter 2 Sample Program: Displaying a Window

      File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
   public static void main(String[ ] args) {
      JFrame    myWindow;

      myWindow = new JFrame( );


      myWindow.setSize(300, 200);

      myWindow.setTitle(“My First Java Program”);

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

Comment

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Matching Comment Markers



```
/* This is a comment on one line */

/*
     Comment number 1
*/
/*
     Comment number 2
*/

/*

/*
/*
     This is a comment
*/

*/
```

These are part of the comment.

Error: No matching beginning marker.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Three Types of Comments

```
/*
     This is a comment with
     three lines of
     text.
*/
```

Multiline Comment

```
// This is a comment
// This is another comment
// This is a third comment
```

Single line Comments

```
/**
 * This class provides basic clock functions. In addition
 * to reading the current time and today’s date, you can
 * use this class for stopwatch functions.
 */
```

javadoc Comments

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

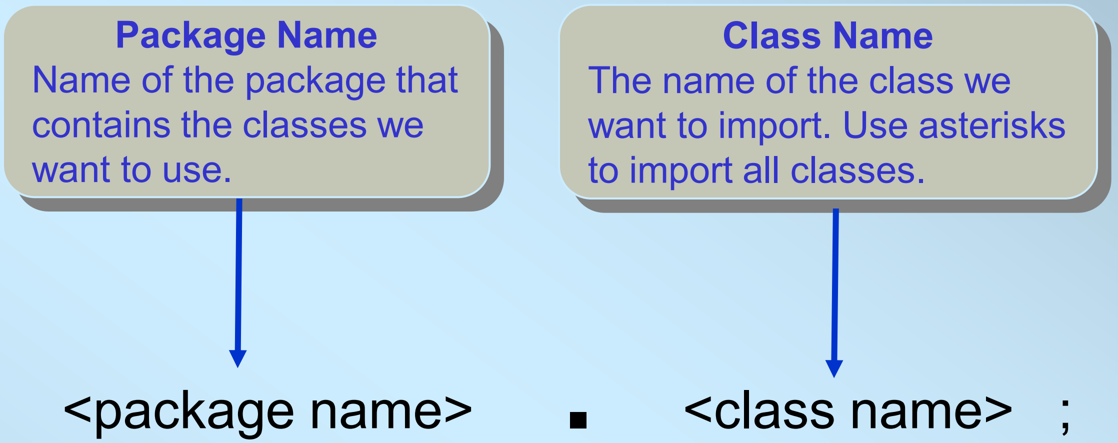
Import Statement

```
/*
       Chapter 2 Sample Program: Displaying a Window

       File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
   public static void main(String[ ] args) {

       JFrame    myWindow;

       myWindow = new JFrame( );

       myWindow.setSize(300, 200);

       myWindow.setTitle("My First Java Program");

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

Import Statement

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Import Statement Syntax and Semantics



Package Name
Name of the package that contains the classes we want to use.

Class Name
The name of the class we want to import. Use asterisks to import all classes.

<package name> . <class name> ;

e.g. dorm . Resident;

More Examples

```
import      javax.swing.JFrame;
import      java.util.*;
import      com.drcaffeine.simplegui.*;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

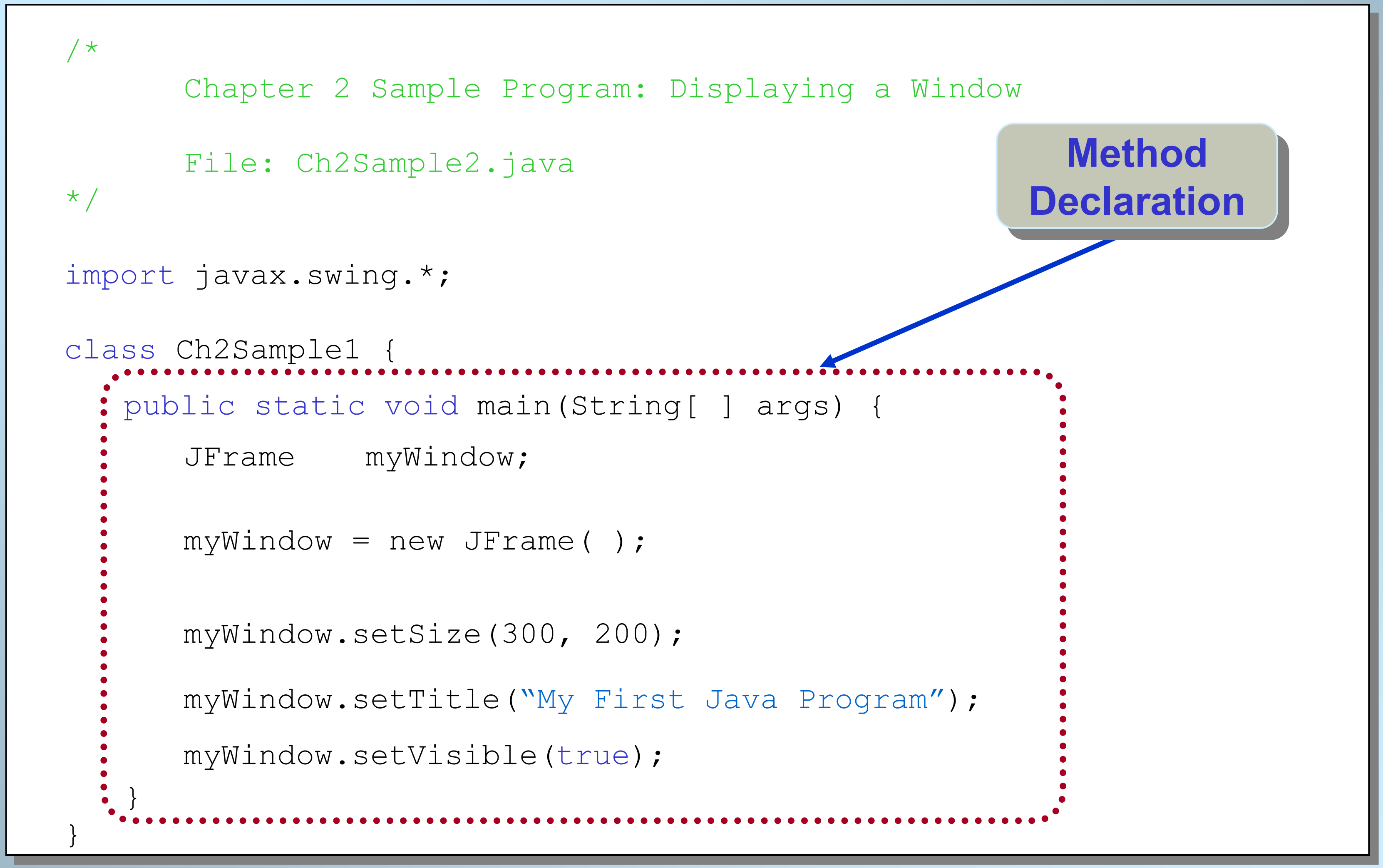
Class Declaration

```
/*
      Chapter 2 Sample Program: Displaying a Window

      File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
   public static void main(String[ ] args) {
      JFrame    myWindow;

      myWindow = new JFrame( );


      myWindow.setSize(300, 200);

      myWindow.setTitle(“My First Java Program”);

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

Class Declaration

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

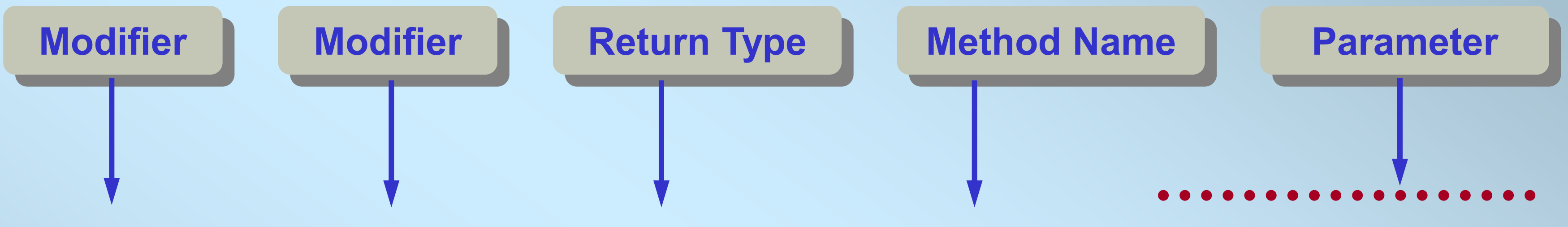
Method Declaration



```
/*
      Chapter 2 Sample Program: Displaying a Window

      File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
   public static void main(String[ ] args) {
      JFrame    myWindow;

      myWindow = new JFrame( );


      myWindow.setSize(300, 200);

      myWindow.setTitle("My First Java Program");

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

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Method Declaration Elements


Modifier

Modifier

Return Type

Method Name

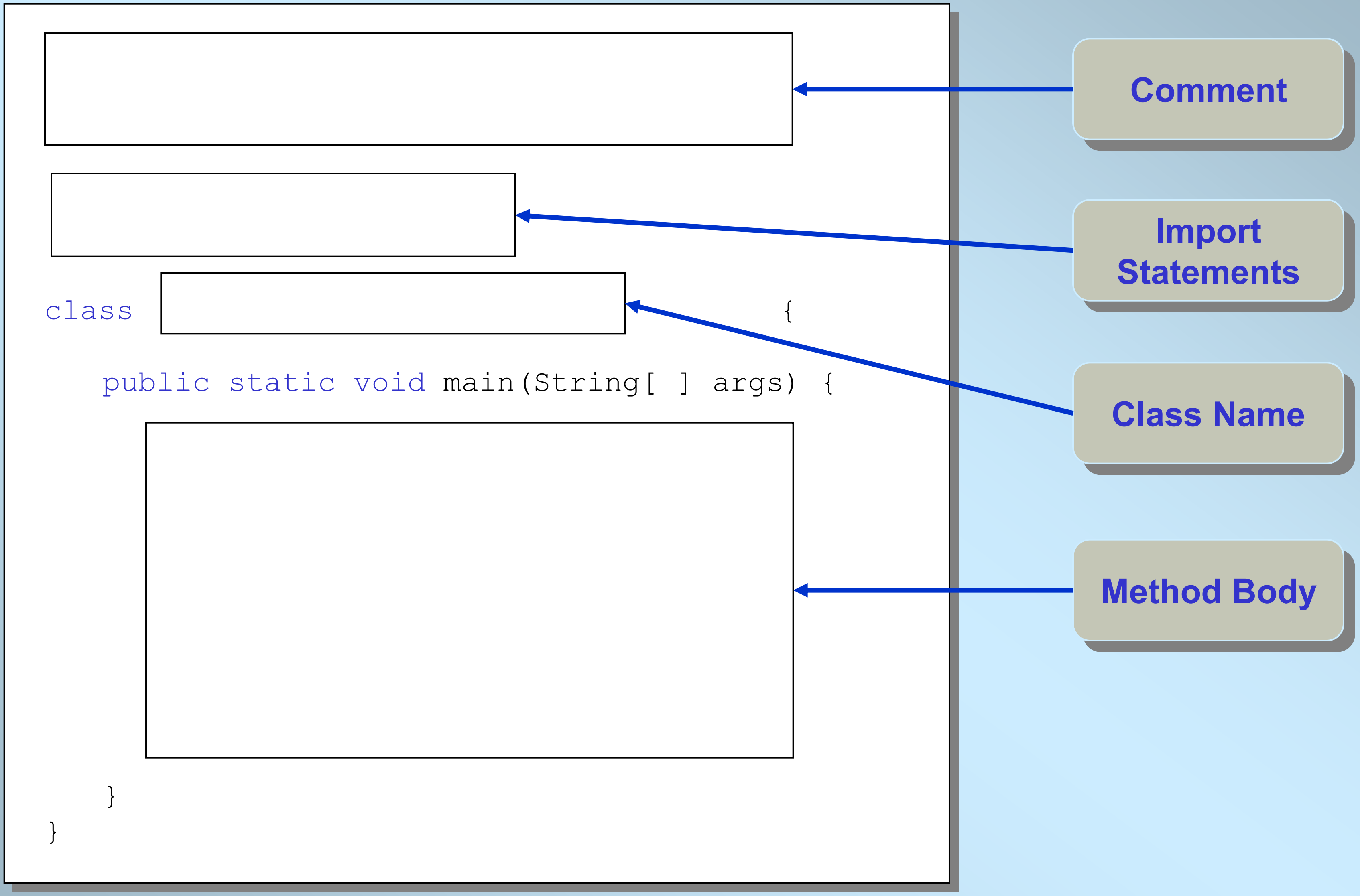
Parameter

```
public     static     void     main(     String[ ] args   ){

    JFrame myWindow;

    myWindow = new JFrame( );

    myWindow.setSize(300, 200);

    myWindow.setTitle("My First Java Program");

    myWindow.setVisible(true);

}
```

Method Body

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Template for Simple Java Programs



```
class                                    {

    public static void main(String[ ] args) {



    }
}
```

Comment

Import Statements

Class Name

Method Body

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Why Use Standard Classes

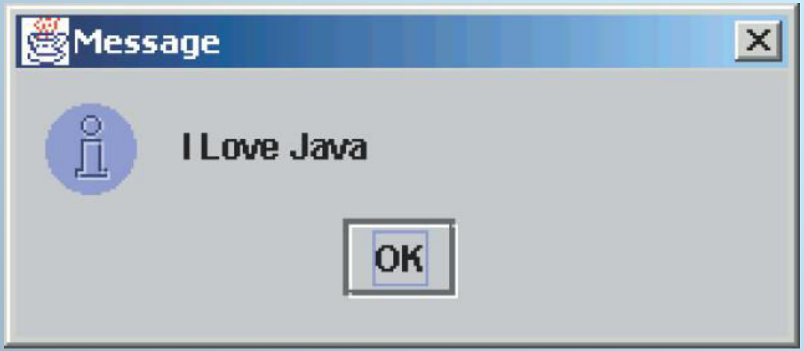
- Don’t reinvent the wheel. When there are existing objects that satisfy our needs, use them.
- Learning how to use standard Java classes is the first step toward mastering OOP. Before we can learn how to define our own classes, we need to learn how to use existing classes
- We will introduce four standard classes here:
    - JOptionPane
    - String
    - Date
    - SimpleDateFormat.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

JOptionPane

- Using **showMessageDialog** of the **JOptionPane** class is a simple way to display a result of a computation to the user.

```
JOptionPane.showMessageDialog(null, “I Love Java”);
```



I Love Java

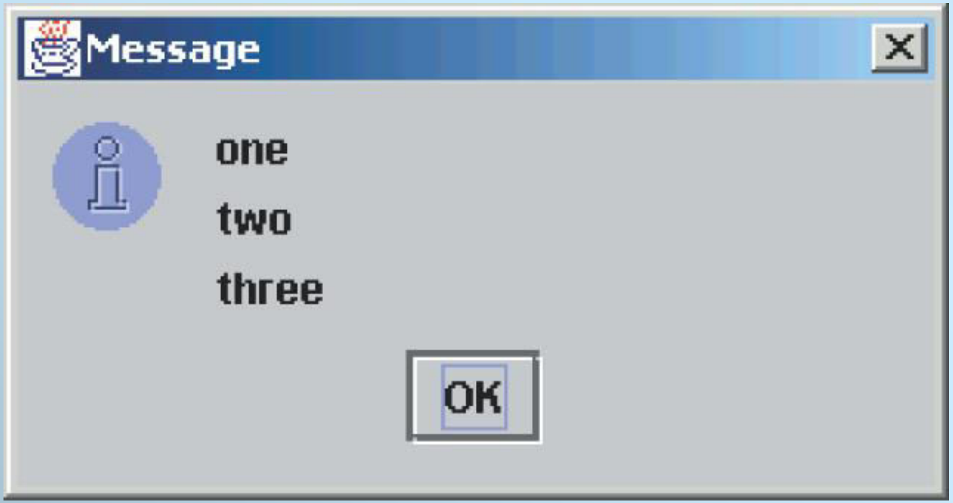
This dialog will appear at the center of the screen.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Displaying Multiple Lines of Text

- We can display multiple lines of text by separating lines with a new line marker \n.

```
JOptionPane.showMessageDialog(null,
           "one\ntwo\nthree");
```



```
one
two
three
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

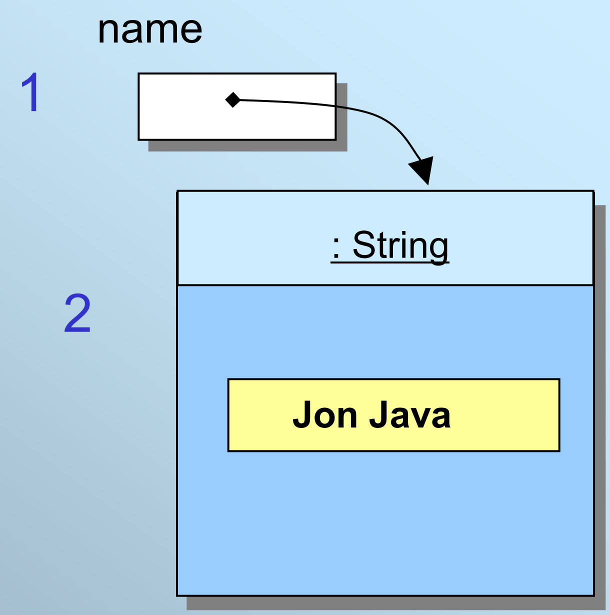
String

- The textual values passed to the showMessageDialog method are instances of the String class.
- A sequence of characters separated by double quotes is a String constant.
- There are close to 50 methods defined in the String class. We will introduce three of them here: substring, length, and indexOf.
- We will also introduce a string operation called concatenation.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

String is an Object

```
1   String   name;
2   name  =  new  String(“Jon Java”);
```

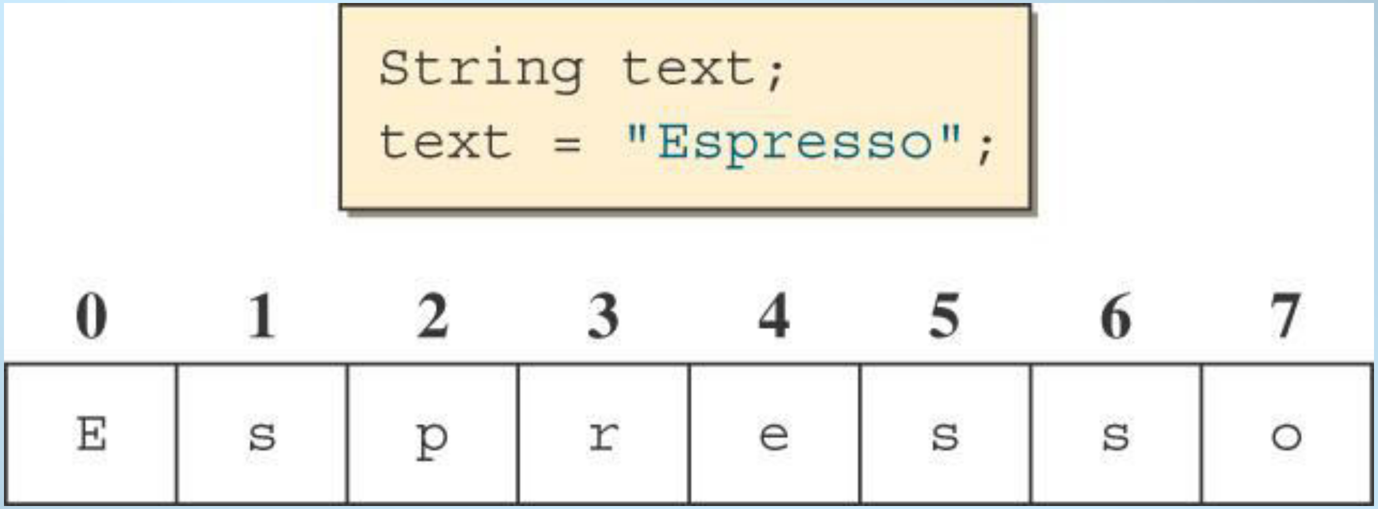


1. The identifier name is declared and space is allocated in memory.

2. A String object is created and the identifier name is set to refer to it.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

String Indexing



```
String text;
text = "Espresso";
```

0	1	2	3	4	5	6	7
E	s	p	r	e	s	s	o

The position, or index, of the first character is 0.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Definition: substring

- Assume str is a String object and properly initialized to a string.
- str.substring(i, j) will return a new string by extracting characters of str from position i to j-1 where $0 \leq i < $ length of str, $0 < j \leq$ length of str, and $i \leq j$.
- If str is "programming" , then str.substring(3, 7) will create a new string whose value is "gram" because g is at position 3 and m is at position 6.
- The original string str remains unchanged.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Examples: substring

```
String text = “Espresso”;
```

```
text.substring(6,8)    →    “so”
text.substring(0,8)    →    “Espresso”
text.substring(1,5)    →    “spre”
text.substring(3,3)    →    “”
text.substring(4,2)    →    error
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

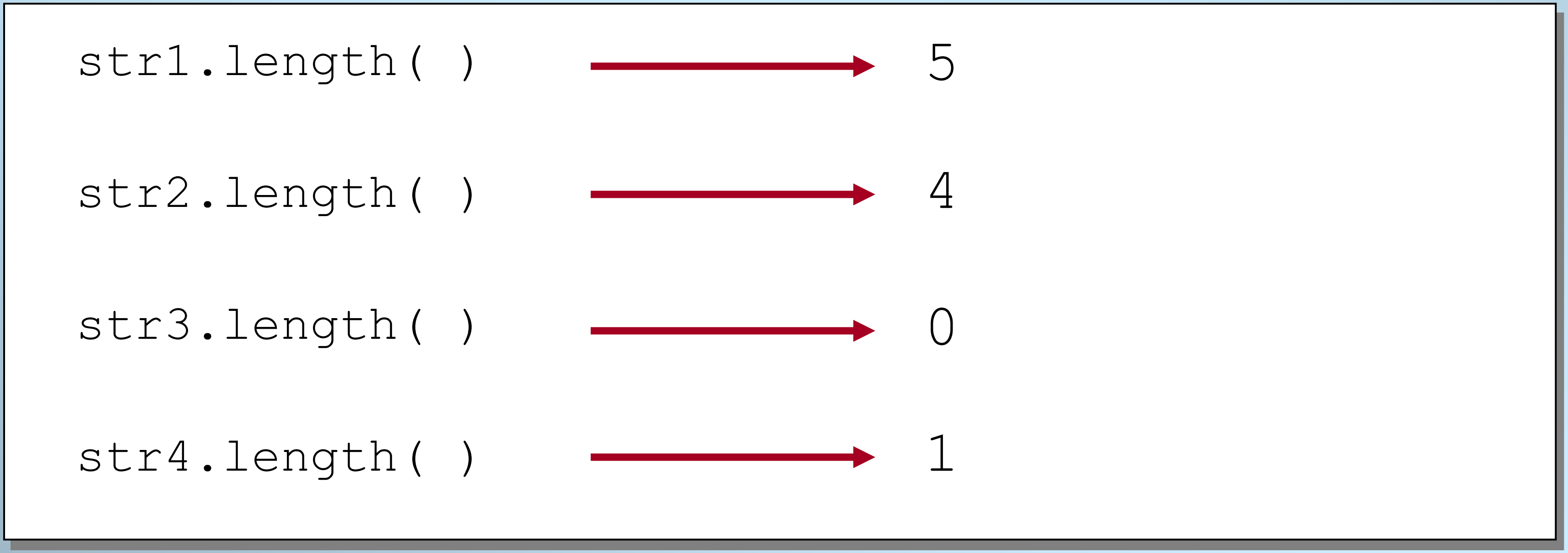
Definition: length

- Assume str is a String object and properly initialized to a string.
- str.length( ) will return the number of characters in str.
- If str is “programming” , then str.length( ) will return 11 because there are 11 characters in it.
- The original string str remains unchanged.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Examples: length

```
String str1, str2, str3, str4;
str1 = “Hello” ;
str2 = “Java” ;
str3 = “” ; //empty string
str4 = “ “ ; //one space
```



```
str1.length( )  →  5

str2.length( )  →  4

str3.length( )  →  0

str4.length( )  →  1
```

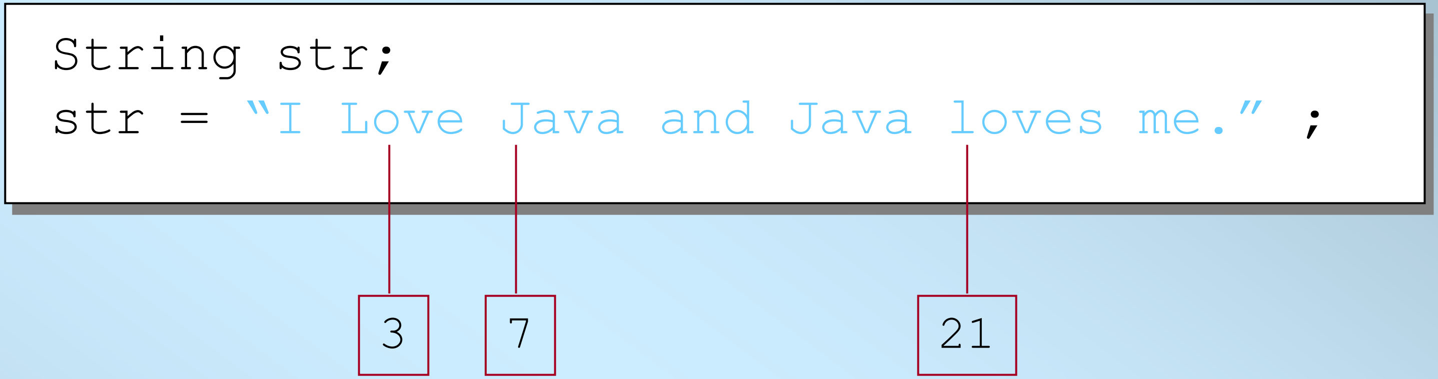
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Definition: indexOf

- Assume str and substr are String objects and properly initialized.
- str.indexOf(substr) will return the first position substr occurs in str.
- If str is “programming” and substr is “gram” , then str.indexOf(substr) will return 3 because the position of the first character of substr in str is 3.
- If substr does not occur in str, then –1 is returned.
- The search is case-sensitive.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Examples: indexOf



```
String str;
str = “I Love Java and Java loves me.” ;
```

```
str.indexOf( “J” )            →  7

str2.indexOf( “love” )        →  21

str3. indexOf( “ove” )        →  3

str4. indexOf( “Me” )         →  -1
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Definition: concatenation

- Assume str1 and str2 are String objects and properly initialized.
- str1 + str2 will return a new string that is a concatenation of two strings.
- If str1 is “pro” and str2 is “gram” , then str1 + str2 will return “program”.
- Notice that this is an operator and not a method of the String class.
- The strings str1 and str2 remains the same.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Examples: concatenation

```
String str1, str2;
str1 = “Jon” ;
str2 = “Java” ;
```

```
str1 + str2                   →   “JonJava”

str1 + “ “ + str2             →   “Jon Java”

str2 + “, “ + str1            →   “Java, Jon”

“Are you “ + str1 + “?”       →   “Are you Jon?”
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Date

- The Date class from the java.util package is used to represent a date.
- When a Date object is created, it is set to today (the current date set in the computer)
- The class has toString method that converts the internal format to a string.

```
Date today;
today = new Date( );

today.toString( );

      →  "Fri Oct 31 10:05:18 PST 2003"
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

SimpleDateFormat

- The SimpleDateFormat class allows the Date information to be displayed with various format.
- Table 2.1 page 68 shows the formatting options.

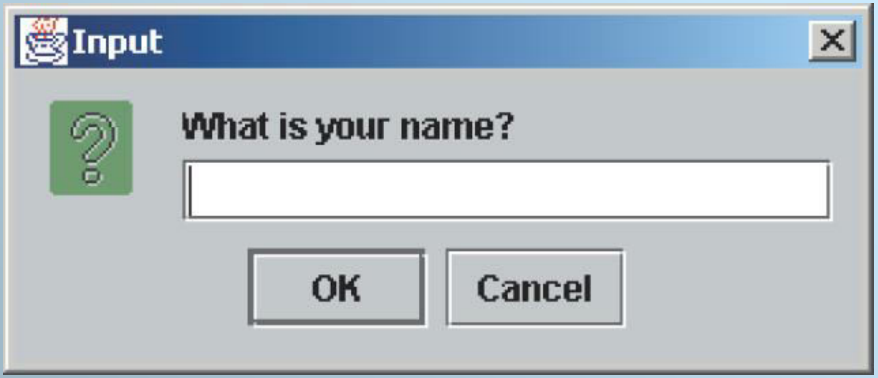
```
Date today = new Date( );
SimpleDateFormat sdf1, sdf2;
sdf1 = new SimpleDateFormat( “MM/dd/yy” );
sdf2 = new SimpleDateFormat( “MMMM dd, yyyy” );

sdf1.format(today);   →   “10/31/03”

sdf2.format(today);   →   “October 31, 2003”
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

JOptionPane for Input

- Using **showInputDialog** of the **JOptionPane** class is a simple way to input a string.

```
String name;

name = JOptionPane.showInputDialog
                  (null, “What is your name?”);
```



This dialog will appear at the center of the screen ready to accept an input.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Problem Statement

- Problem statement:

  Write a program that asks for the user's first, middle, and last names and replies with their initials.

Example:

input: Andrew Lloyd Weber
output: ALW

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Overall Plan

- Identify the major tasks the program has to perform.
    - We need to know what to develop before we develop!
- Tasks:
    – Get the user’s first, middle, and last names
    – Extract the initials and create the monogram
    – Output the monogram

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Development Steps

- We will develop this program in two steps:

1. Start with the program template and add code to get input

2. Add code to compute and display the monogram

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 1 Design

- The program specification states “get the user’s name” but doesn’t say how.
- We will consider “how” in the Step 1 design
- We will use JOptionPane for input
- Input Style Choice #1

  Input first, middle, and last names separately
- Input Style Choice #2

  Input the full name at once
- We choose Style #2 because it is easier and quicker for the user to enter the information

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 1 Code

```java
/*
   Chapter 2 Sample Program: Displays the Monogram
   File: Step1/Ch2Monogram.java
*/
import javax.swing.*;

class Ch2Monogram {
   public static void main (String[ ] args) {

       String name;

       name = JOptionPane.showInputDialog(null,
             "Enter your full name (first, middle, last):“);

       JOptionPane.showMessageDialog(null, name);
   }
}
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 1 Test

- In the testing phase, we run the program and verify that
  - we can enter the name
  - the name we enter is displayed correctly

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Design

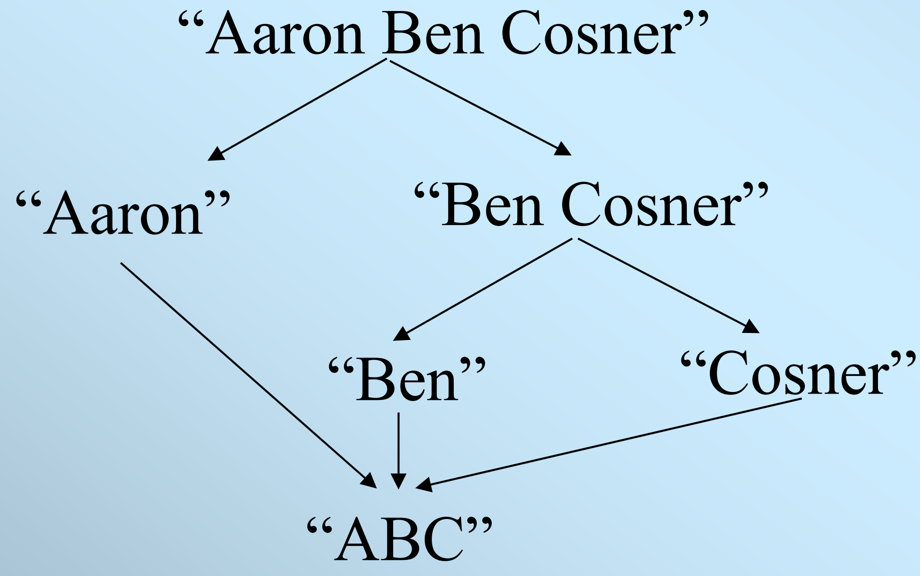
- Our programming skills are limited, so we will make the following assumptions:
  - input string contains first, middle, and last names
  - first, middle, and last names are separated by single blank spaces
- Example

John Quincy Adams	(okay)
John Kennedy	(not okay)
Harrison, William Henry	(not okay)

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Design (cont'd)

- Given the valid input, we can compute the monogram by
  - breaking the input name into first, middle, and last
  - extracting the first character from them
  - concatenating three first characters



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Code

```
/*
   Chapter 2 Sample Program: Displays the Monogram
   File: Step 2/Ch2MonogramStep2.java
*/
import javax.swing.*;

class Ch2Monogram {

   public static void main (String[ ] args) {
       String name, first, middle, last,
             space, monogram;

       space = " “;

       //Input the full name
       name = JOptionPane.showInputDialog(null,
             "Enter your full name (first, middle, last):“ );
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Code (cont’d)

```
      //Extract first, middle, and last names
      first = name.substring(0, name.indexOf(space));
      name = name.substring(name.indexOf(space)+1,
                                       name.length());

      middle = name.substring(0, name.indexOf(space));
      last = name.substring(name.indexOf(space)+1,
                                       name.length());

      //Compute the monogram
      monogram = first.substring(0, 1) +
                   middle.substring(0, 1) +
                          last.substring(0,1);
      //Output the result
      JOptionPane.showMessageDialog(null,
                   "Your monogram is " + monogram);
   }
}
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Test

- In the testing phase, we run the program and verify that, for all valid input values, correct monograms are displayed.
- We run the program numerous times. Seeing one correct answer is not enough. We have to try out many different types of (valid) input values.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Program Review

- The work of a programmer is not done yet.
- Once the working program is developed, we perform a critical review and see if there are any missing features or possible improvements
- One suggestion
  - Improve the initial prompt so the user knows the valid input format requires single spaces between the first, middle, and last names

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 1

Introduction to Object-Oriented Programming and Software Development

Animated Version

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Objectives

After you have read and studied this chapter, you should be able to

- Name the basic components of object-oriented programming
- Differentiate classes and objects.
- Differentiate class and instance methods.
- Differentiate class and instance data values.
- Draw program diagrams using icons for classes and objects
- Describe significance of inheritance in object-oriented programs
- Name and explain the stages of the software lifecycle

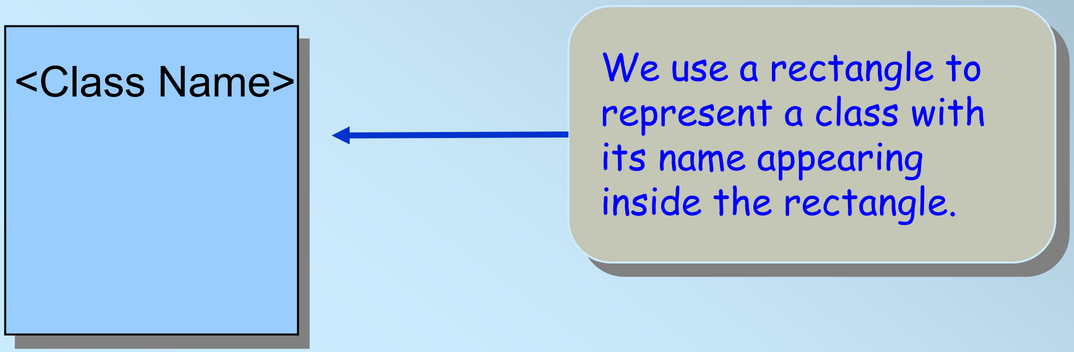
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Classes and Objects

- Object-oriented programs use objects.
- An *object* is a thing, both tangible and intangible. Account, Vehicle, Employee, etc.
- To create an object inside the computer program, we must provide a definition for objects—how they behave and what kinds of information they maintain —called a *class*.
- An object is called an *instance* of a class.

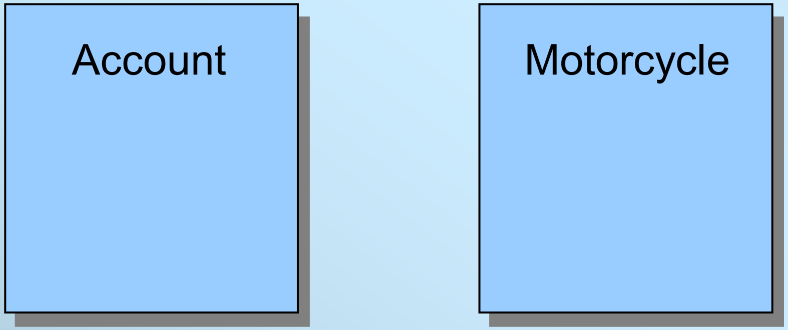
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Graphical Representation of a Class



We use a rectangle to represent a class with its name appearing inside the rectangle.

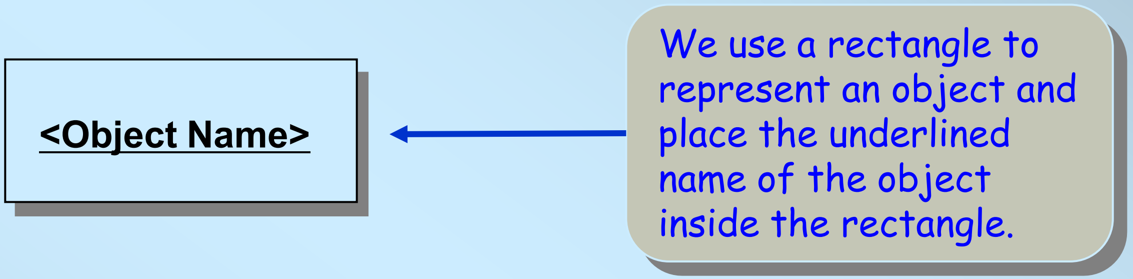
Example:



The notation we used here is based on the industry standard notation called *UML*, which stands for Unified Modeling Language.

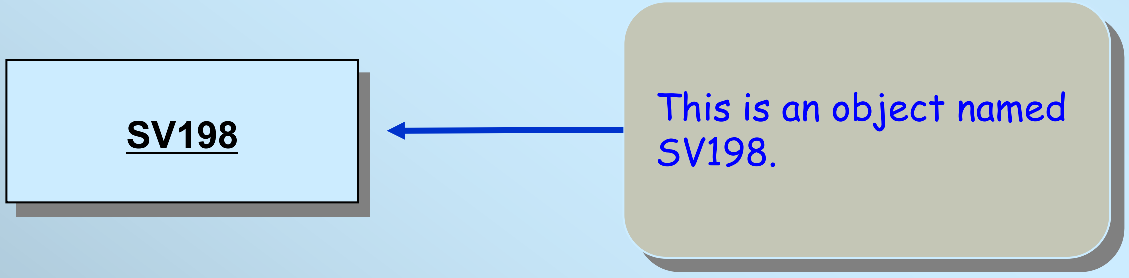
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Graphical Representation of an Object



We use a rectangle to represent an object and place the underlined name of the object inside the rectangle.

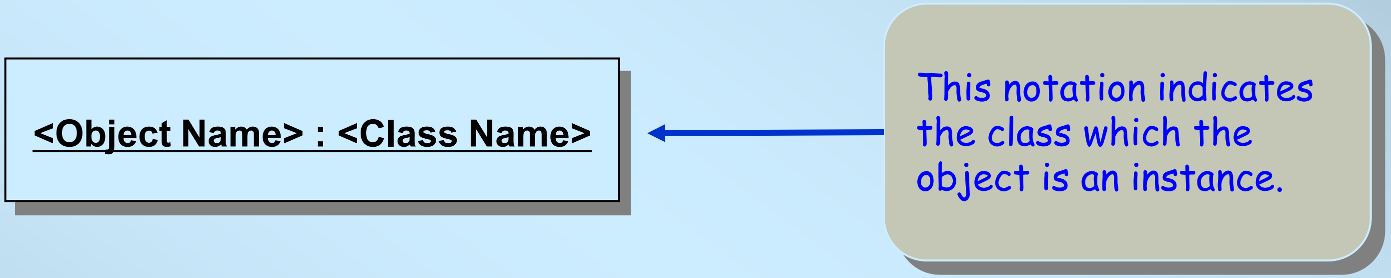
Example:



This is an object named SV198.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

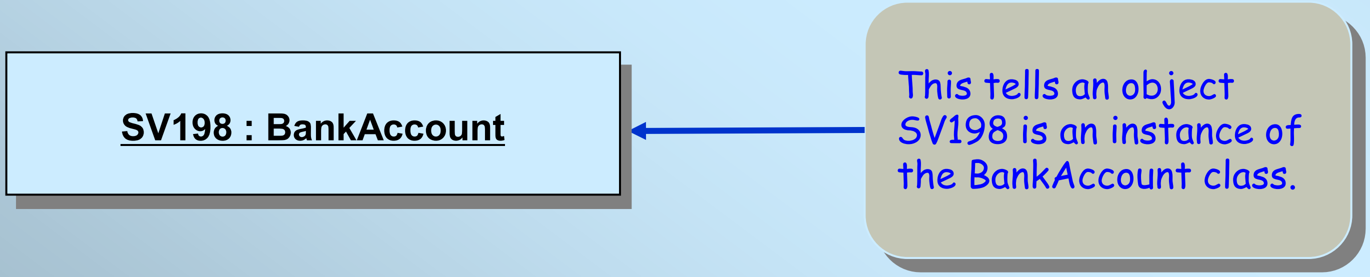
An Object with the Class Name



<Object Name> : <Class Name>

This notation indicates the class which the object is an instance.

Example:



SV198 : BankAccount

This tells an object SV198 is an instance of the BankAccount class.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

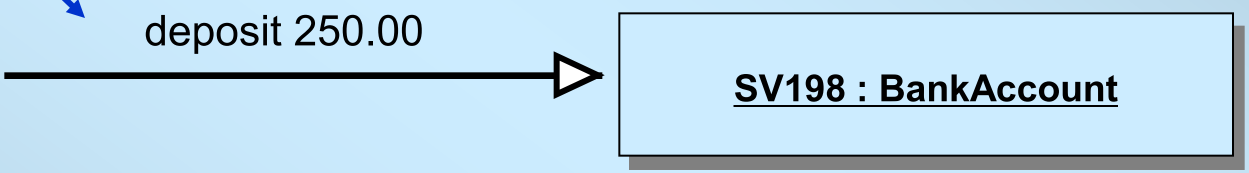
Messages and Methods

- To instruct a class or an object to perform a task, we send a *message* to it.
- You can send a message only to the classes and objects that understand the message you sent to them.
- A class or an object must possess a matching *method* to be able to handle the received message.
- A method defined for a class is called a *class method*, and a method defined for an object is called an *instance method*.
- A value we pass to an object when sending a message is called an *argument* of the message.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sending a Message

Message deposit with the argument 250.00 is sent to a BankAccount object SV198.

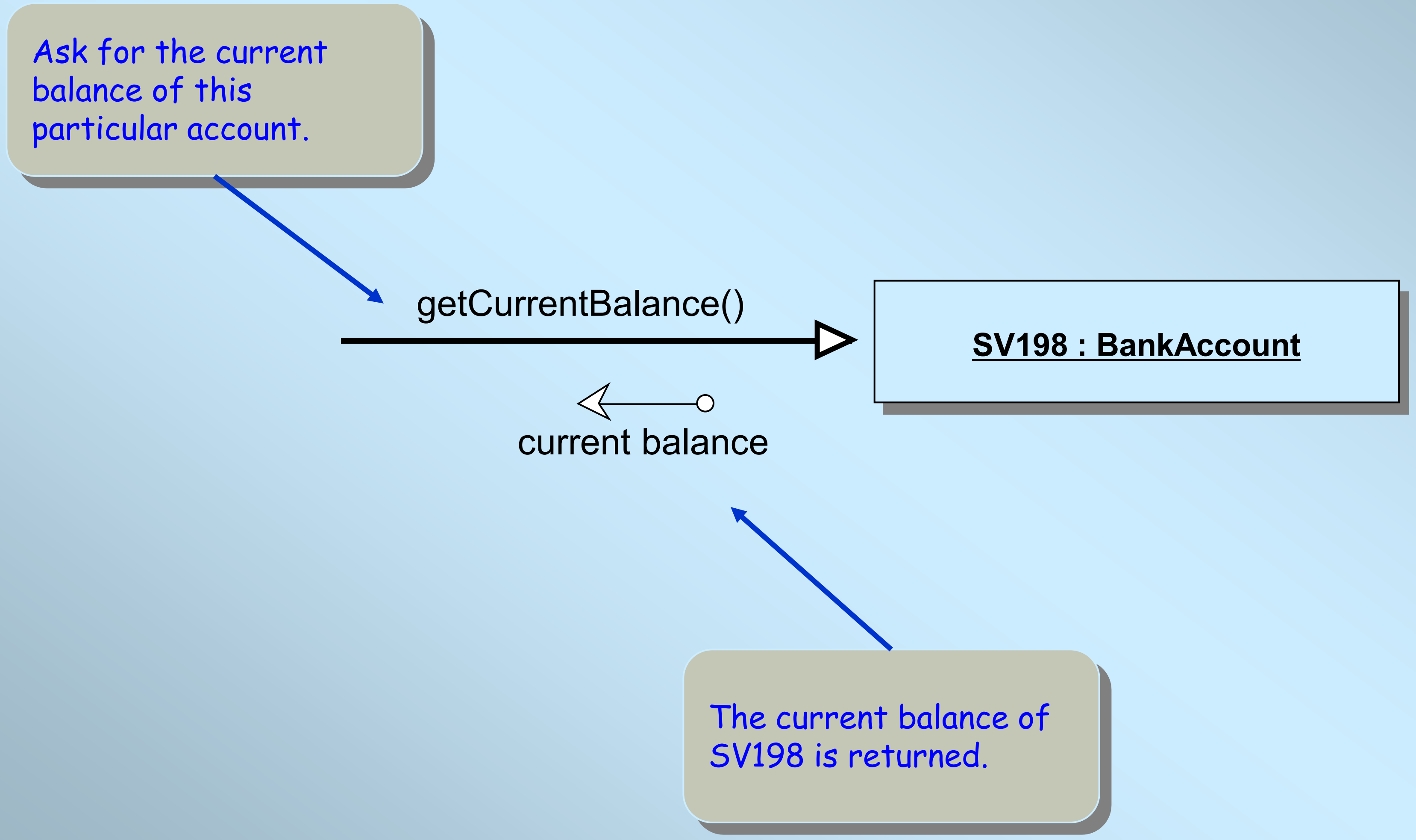


deposit 250.00

SV198 : BankAccount

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sending a Message and Getting an Answer



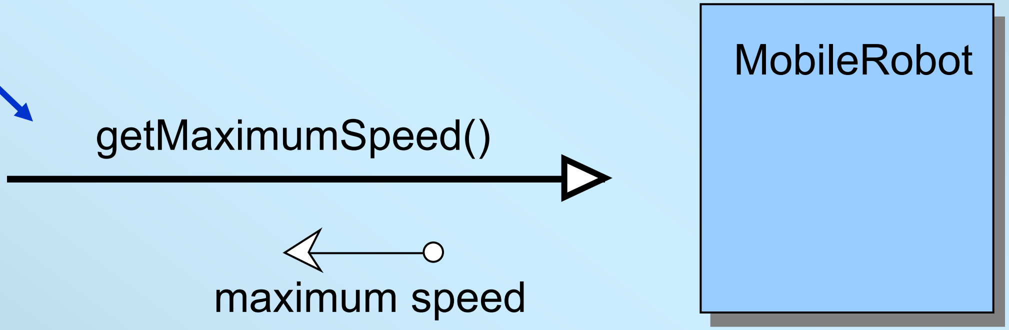
Ask for the current balance of this particular account.

The current balance of SV198 is returned.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Calling a Class Method

Ask for the maximum possible speed for all MobileRobot objects is returned.



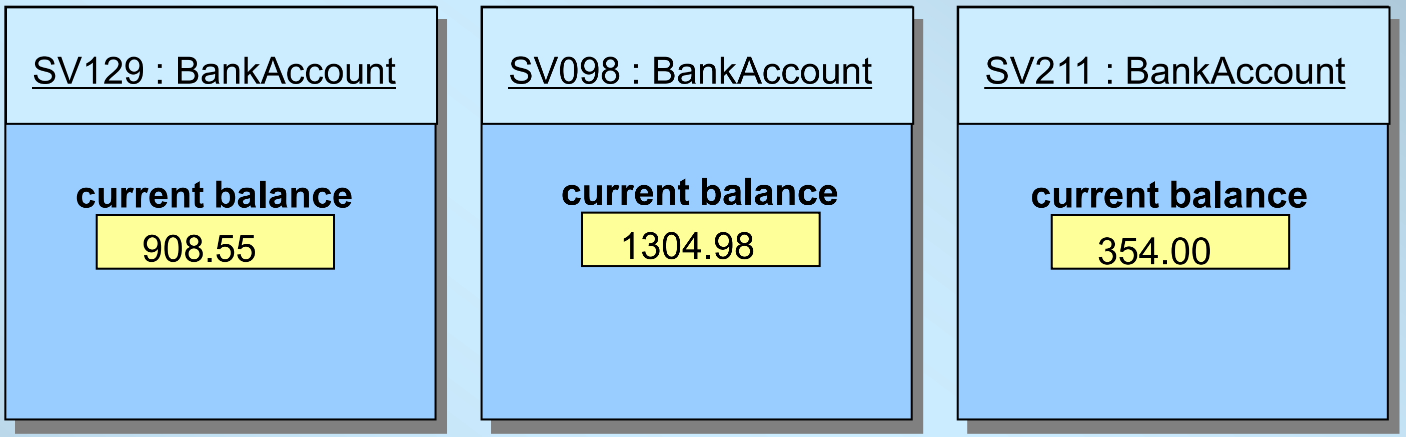
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Class and Instance Data Values

- An object is comprised of data values and methods.
- An *instance data value* is used to maintain information specific to individual instances. For example, each BankAccount object maintains its balance.
- A *class data value* is used to maintain information shared by all instances or aggregate information about the instances.
- For example, minimum balance is the information shared by all Account objects, whereas the average balance of all BankAccount objects is an aggregate information.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sample Instance Data Value



SV129 : BankAccount

current balance

908.55

SV098 : BankAccount

current balance

1304.98

SV211 : BankAccount

current balance

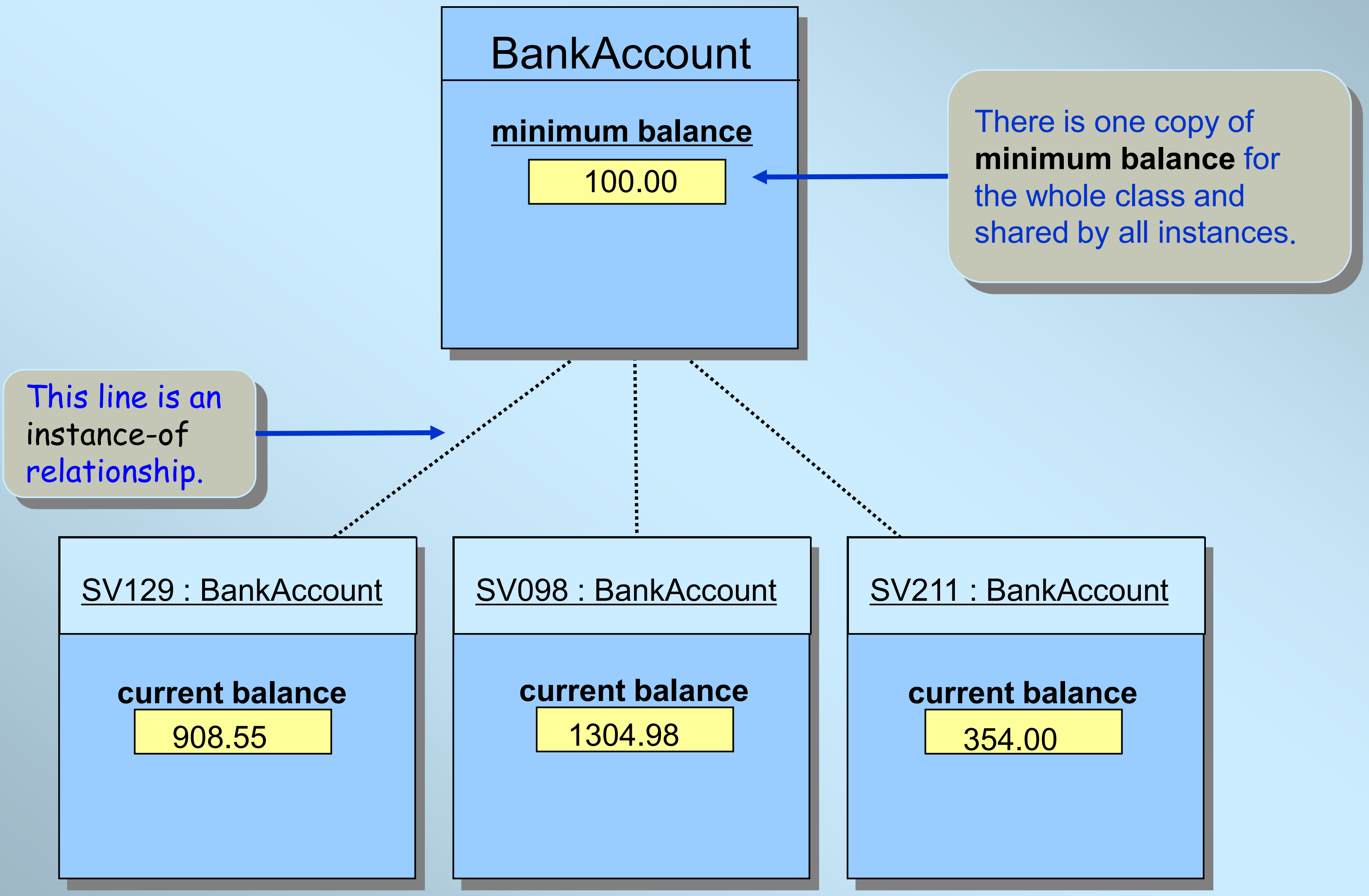
354.00

All three BankAccount objects possess the same instance data value current balance.

The actual dollar amounts are, of course, different.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sample Class Data Value

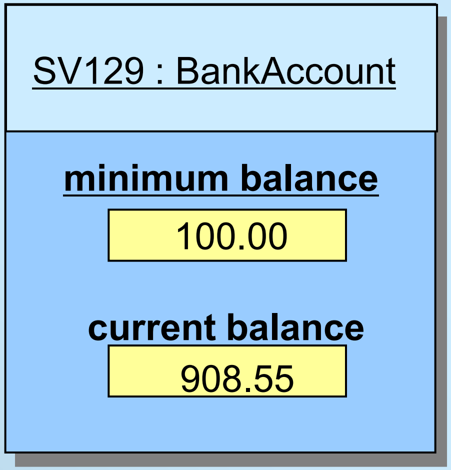


There is one copy of **minimum balance** for the whole class and shared by all instances.

This line is an instance-of relationship.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Object Icon with Class Data Value



When the class icon is not shown, we include the class data value in the object icon itself.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

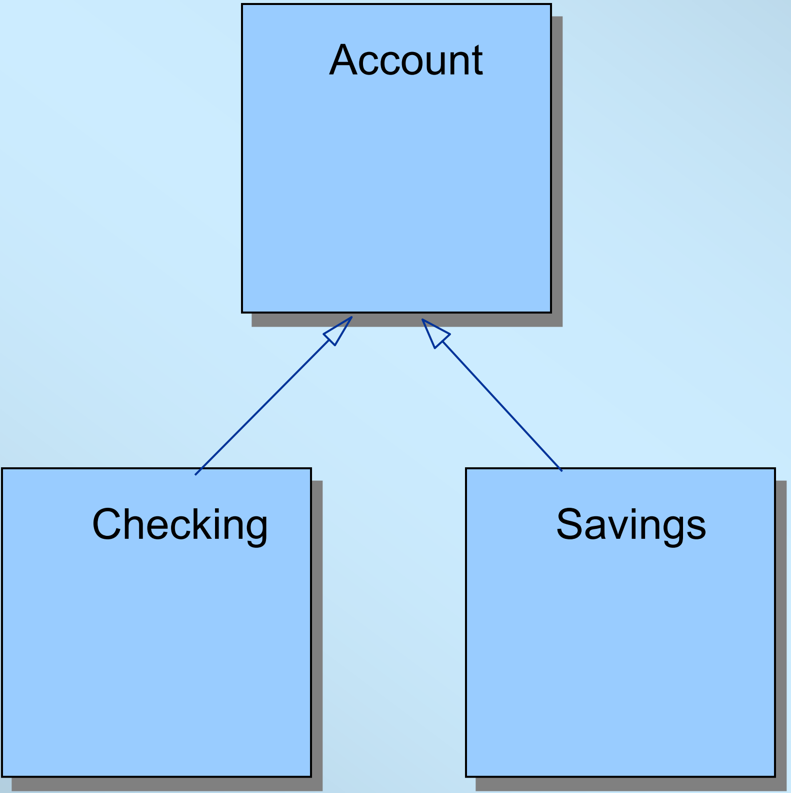
Inheritance

- *Inheritance* is a mechanism in OOP to design two or more entities that are different but share many common features.
  - Features common to all classes are defined in the *superclass.*
  - The classes that inherit common features from the superclass are called *subclasses*.
    - We also call the superclass an *ancestor* and the subclass a *descendant*.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

A Sample Inheritance

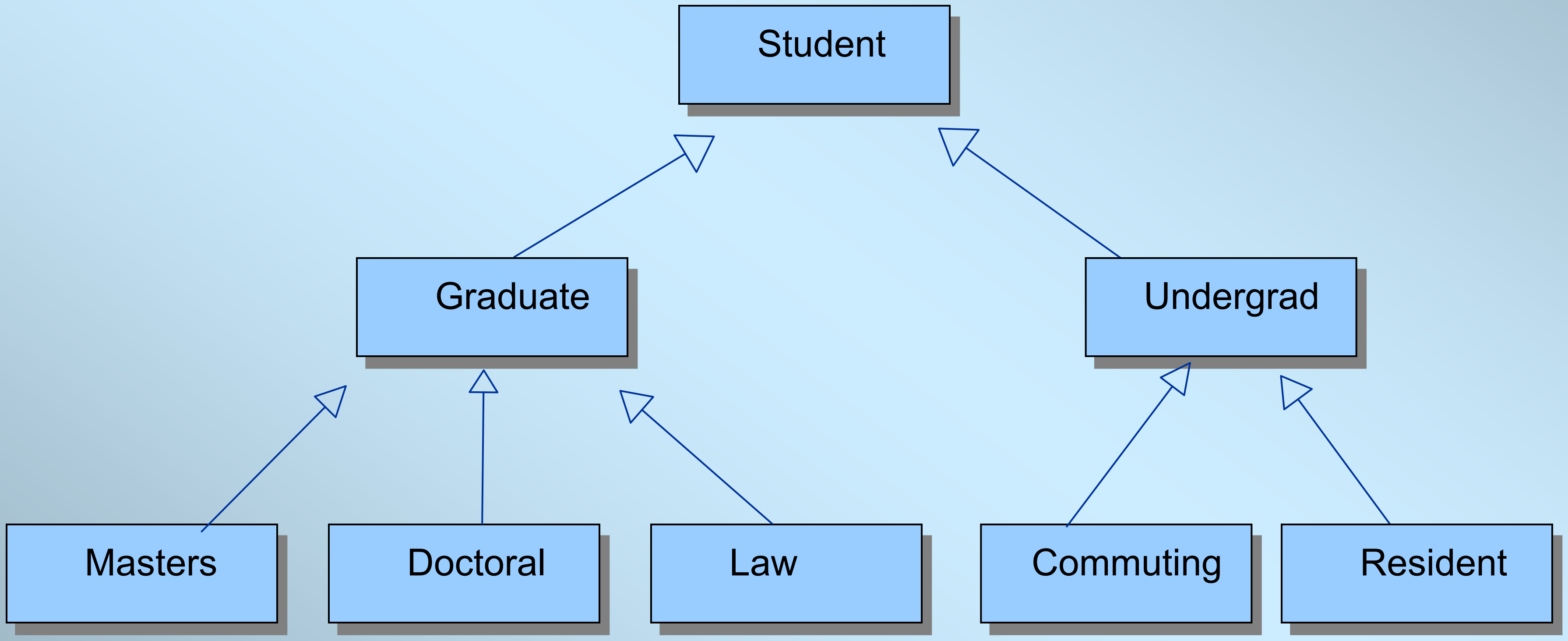
- Here are the superclass **Account** and its subclasses **Savings** and **Checking**.



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Inheritance Hierarchy

- An example of inheritance hierarchy among different types of students.



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Software Engineering

- Much like building a skyscraper, we need a disciplined approach in developing complex software applications.
- *Software engineering* is the application of a systematic and disciplined approach to the development, testing, and maintenance of a program.
- In this class, we will learn how to apply sound software engineering principles when we develop sample programs.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Software Life Cycle

- The sequence of stages from conception to operation of a program is called *software life cycle*.
- Five stages are
  - Analysis
  - Design
  - Coding
  - Testing
  - Operation and Maintenance

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 3

Numerical Data

Animated Version

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Objectives

After you have read and studied this chapter, you should be able to

- Select proper types for numerical data.
- Write arithmetic expressions in Java.
- Evaluate arithmetic expressions using the precedence rules.
- Describe how the memory allocation works for objects and primitive data values.
- Write mathematical expressions, using methods in the Math class.
- Use the GregorianCalendar class in manipulating date information such as year, month, and day.
- Use the DecimalFormat class to format numerical data
- Convert input string values to numerical data
- Perform input and output by using System.in and System.out

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Manipulating Numbers

- In Java, to add two numbers x and y, we write

```
x + y
```

- But before the actual addition of the two numbers takes place, we must declare their data type. If x and y are integers, we write

```
int x, y;
```

or

```
int x;
int y;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Variables

- When the declaration is made, memory space is allocated to store the values of x and y.
- x and y are called *variables*. A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location.
- Sample variable declarations:

```
int x;
int v, w, y;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Numerical Data Types

- There are six numerical data types: byte, short, int, long, float, and double.
- Sample variable declarations:

```
int      i,  j,  k;
float    numberOne, numberTwo;
long     bigInteger;
                  double   bigNumber;
```

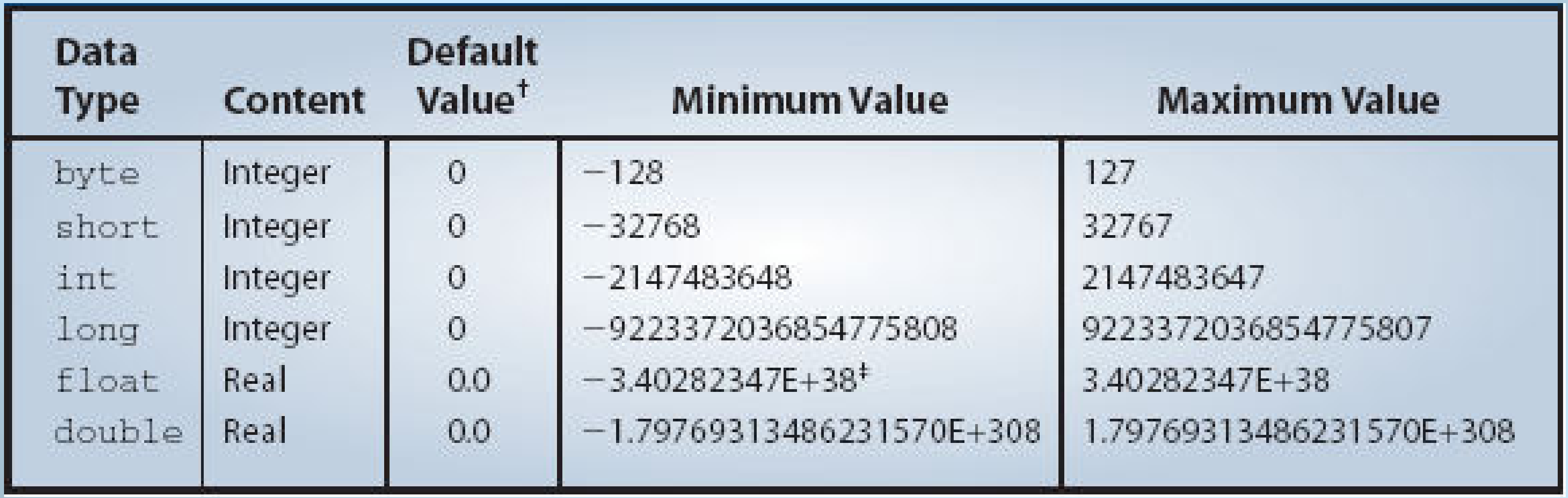
- At the time a variable is declared, it also can be initialized. For example, we may initialize the integer variables count and height to 10 and 34 as

```
int count = 10, height = 34;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Data Type Precisions

The six data types differ in the precision of values they can store in memory.



Data Type	Content	Default Value[†]	Minimum Value	Maximum Value
byte	Integer	0	−128	127
short	Integer	0	−32768	32767
int	Integer	0	−2147483648	2147483647
long	Integer	0	−9223372036854775808	9223372036854775807
float	Real	0.0	−3.40282347E+38[‡]	3.40282347E+38
double	Real	0.0	−1.79769313486231570E+308	1.79769313486231570E+308

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Assignment Statements

- We assign a value to a variable using an *assignment statements*.
- The syntax is

```
<variable> = <expression> ;
```

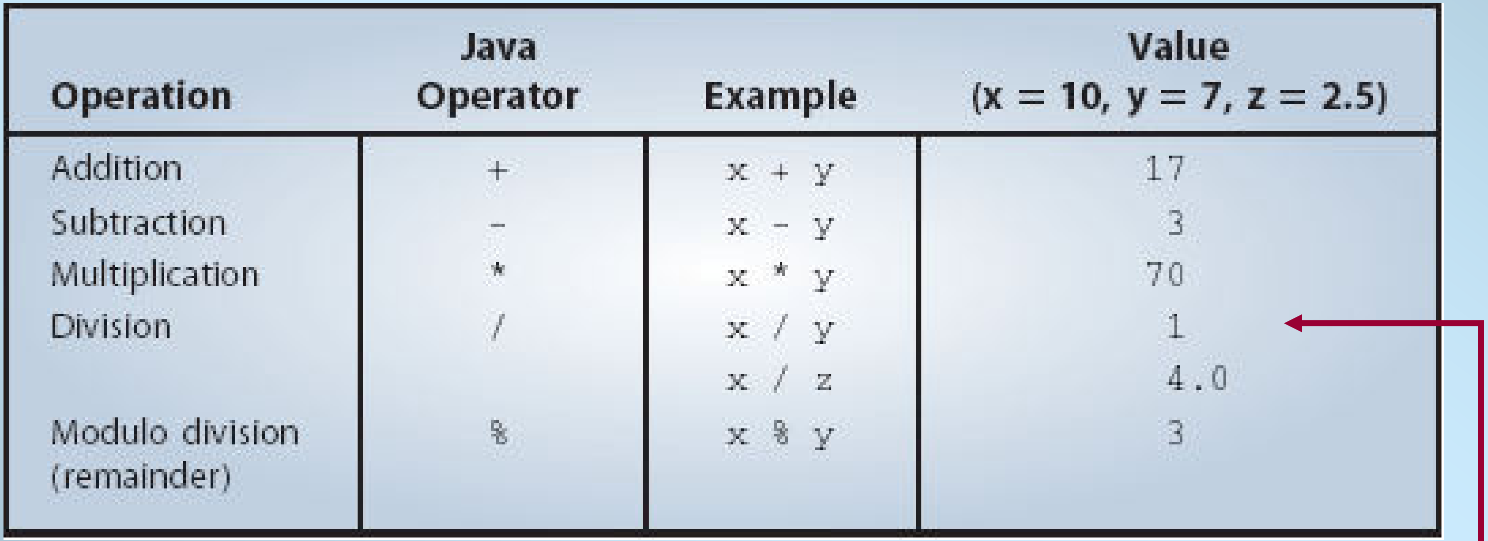
- Examples:

```
sum = firstNumber + secondNumber;
avg = (one + two + three) / 3.0;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Arithmetic Operators

- The following table summarizes the arithmetic operators available in Java.



Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	−	x − y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

This is an integer division where the fractional part is truncated.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Arithmetic Expression

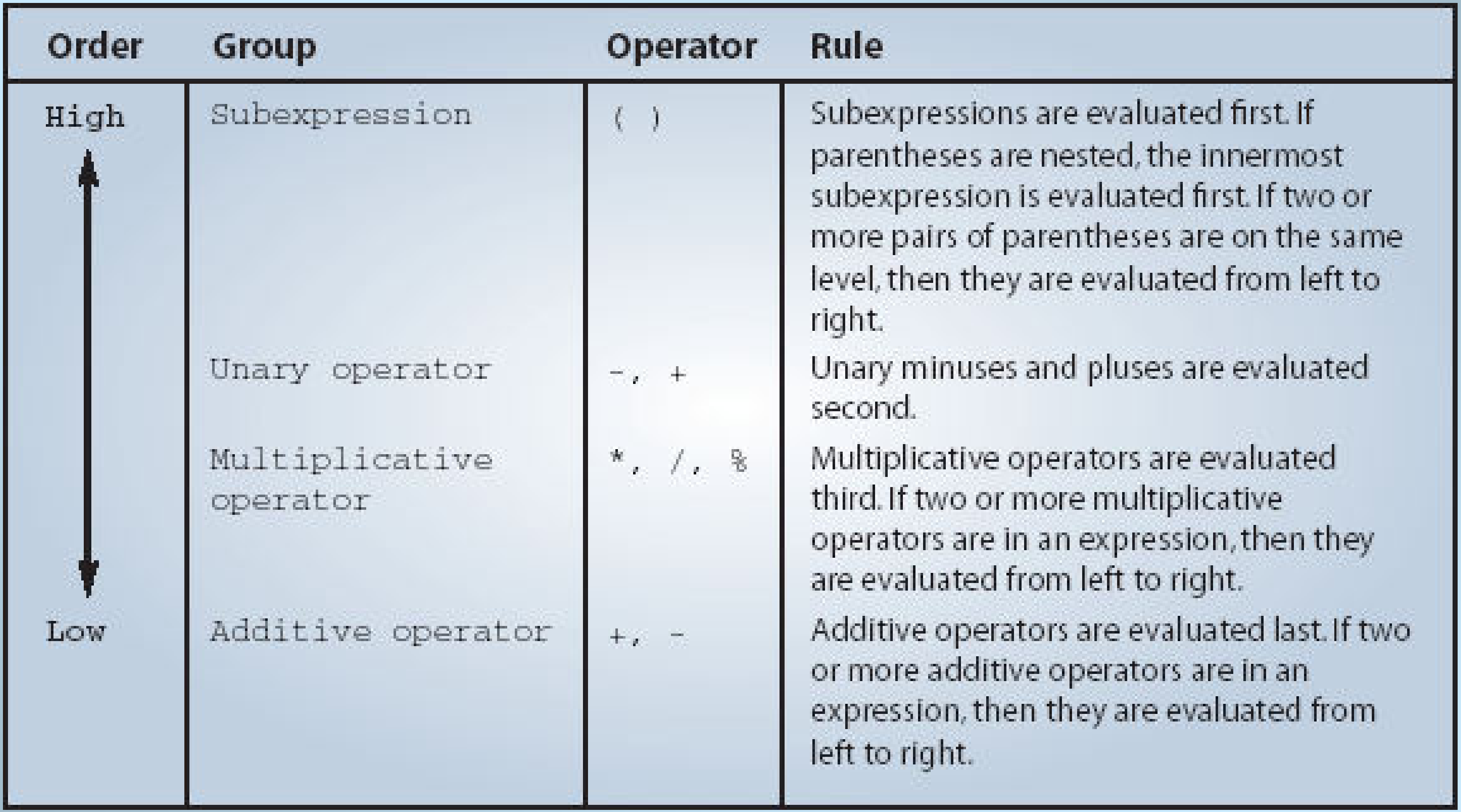
- How does the expression

  ```
  x + 3 * y
  ```

  get evaluated? Answer: x is added to 3*y.
- We determine the order of evaluation by following the *precedence rules*.
- A higher precedence operator is evaluated before the lower one. If two operators are the same precedence, then they are evaluated left to right for most operators.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Precedence Rules



Order	Group	Operator	Rule
High	Subexpression	( )	Subexpressions are evaluated first. If parentheses are nested, the innermost subexpression is evaluated first. If two or more pairs of parentheses are on the same level, then they are evaluated from left to right.
	Unary operator	-, +	Unary minuses and pluses are evaluated second.
	Multiplicative operator	*, /, %	Multiplicative operators are evaluated third. If two or more multiplicative operators are in an expression, then they are evaluated from left to right.
Low	Additive operator	+, -	Additive operators are evaluated last. If two or more additive operators are in an expression, then they are evaluated from left to right.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Type Casting

- If **x** is a **float** and **y** is an **int**, what will be the data type of the following expression?

  ```
  x * y
  ```

  The answer is **float**.
- The above expression is called a *mixed expression*.
- The data types of the operands in mixed expressions are converted based on the *promotion rules*. The promotion rules ensure that the data type of the expression will be the same as the data type of an operand whose type has the highest precision.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Explicit Type Casting

- Instead of relying on the promotion rules, we can make an explicit type cast by prefixing the operand with the data type using the following syntax:

```
( <data type> ) <expression>
```

- Example

```
(float) x / 3
```

Type case **x** to **float** and then divide it by 3.

```
(int) (x / y * 3.0)
```

Type cast the result of the expression **x / y * 3.0** to **int**.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Implicit Type Casting

- Consider the following expression:

```
double x = 3 + 5;
```

- The result of 3 + 5 is of type **int**. However, since the variable **x** is **double**, the value 8 (type **int**) is promoted to 8.0 (type **double**) before being assigned to **x**.
- Notice that it is a promotion. Demotion is not allowed.

```
int x = 3.5;
```

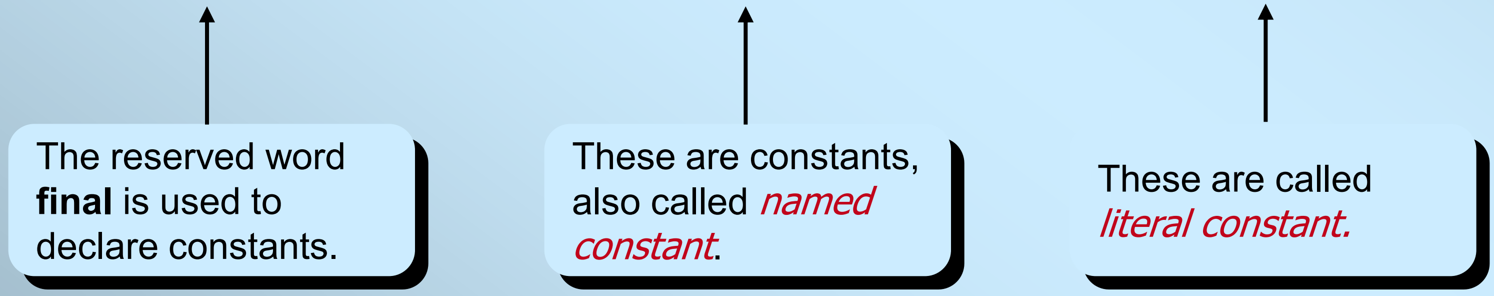
A higher precision value cannot be assigned to a lower precision variable.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Constants

- We can change the value of a variable. If we want the value to remain the same, we use a *constant*.

```
final double PI               = 3.14159;
final int    MONTH_IN_YEAR    = 12;
final short  FARADAY_CONSTANT = 23060;
```



The reserved word **final** is used to declare constants.

These are constants, also called *named constant*.

These are called *literal constant.*

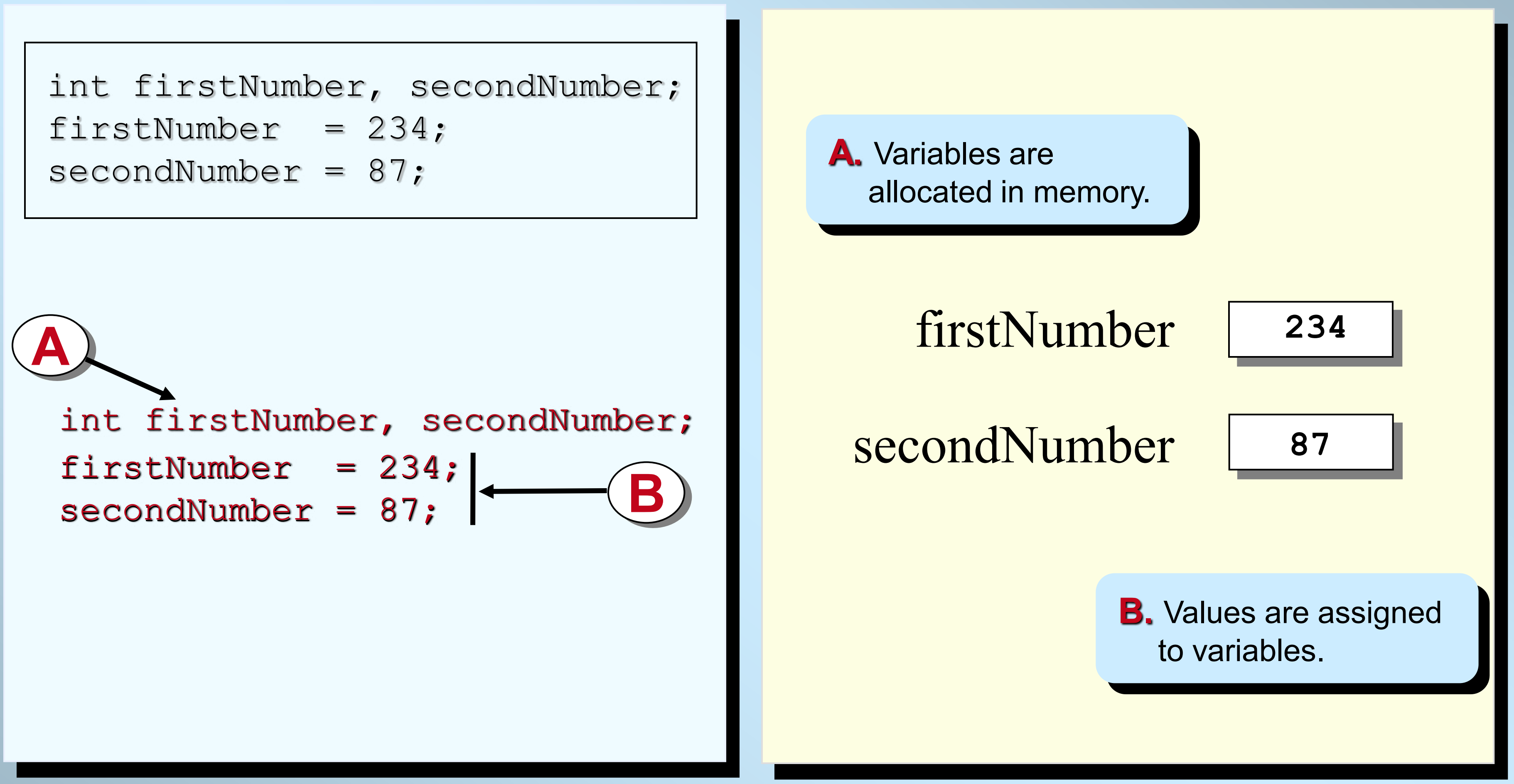
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Primitive vs. Reference

- Numerical data are called *primitive data types*.
- Objects are called *reference data types*, because the contents are addresses that refer to memory locations where the objects are actually stored.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Primitive Data Declaration and Assignments



```
int firstNumber, secondNumber;
firstNumber  = 234;
secondNumber = 87;
```

```
int firstNumber, secondNumber;
firstNumber  = 234;
secondNumber = 87;
```

A. Variables are allocated in memory.

B. Values are assigned to variables.

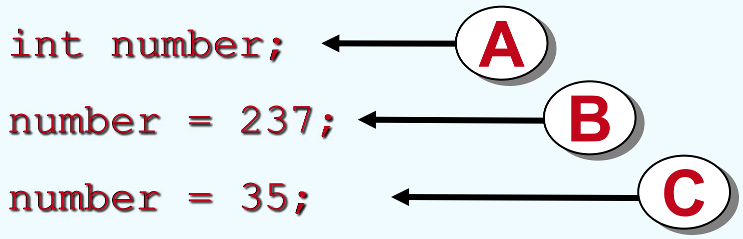
Code

State of Memory

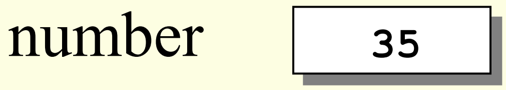
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Assigning Numerical Data

```
int number;
number = 237;
number = 35;
```



Code



A. The variable is allocated in memory.

B. The value **237** is assigned to **number**.

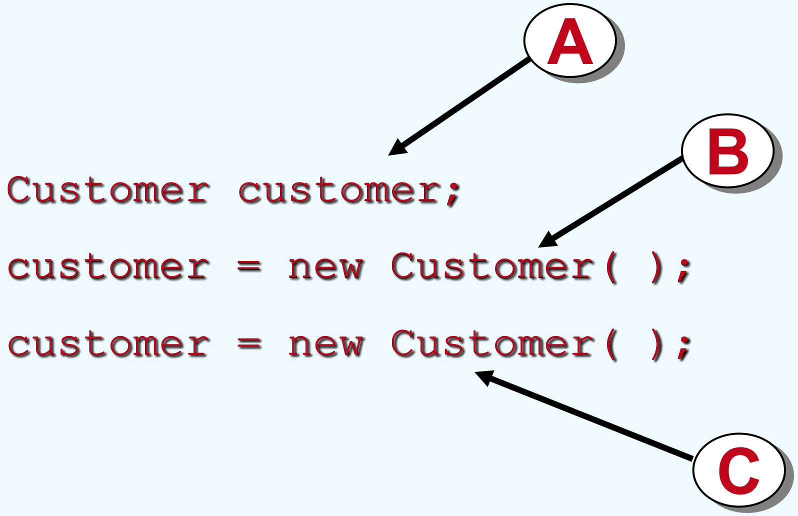
C. The value **35** overwrites the previous value **237**.

State of Memory

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

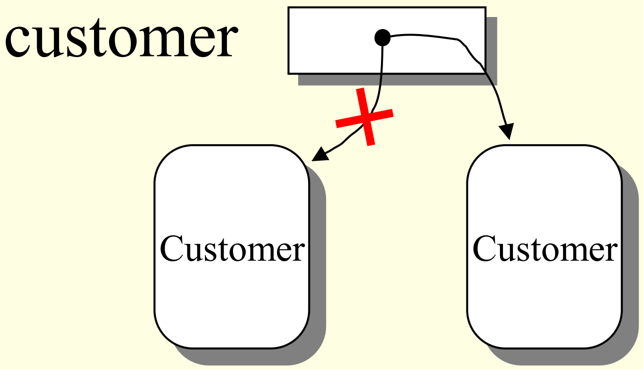
Assigning Objects

```
Customer customer;
customer = new Customer( );
customer = new Customer( );
```



```
Customer customer;

customer = new Customer( );

customer = new Customer( );
```

Code



A. The variable is allocated in memory.

B. The reference to the new object is assigned to **customer**.

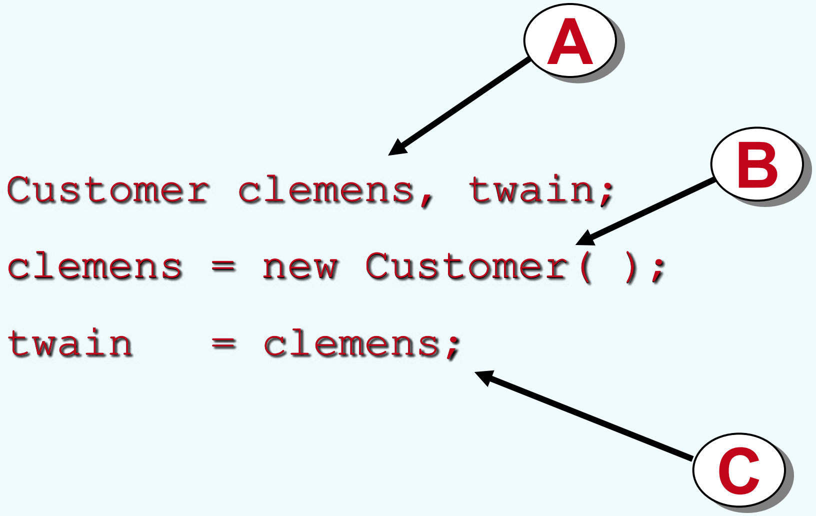
C. The reference to another object overwrites the reference in **customer.**

State of Memory

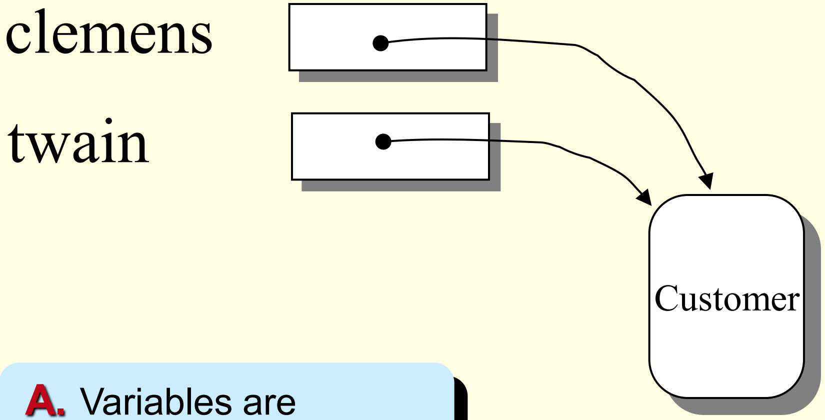
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Having Two References to a Single Object

```
Customer clemens, twain;
clemens = new Customer( );
twain   = clemens;
```



```
Customer clemens, twain;

clemens = new Customer( );

twain   = clemens;
```



A. Variables are allocated in memory.

B. The reference to the new object is assigned to clemens.

C. The reference in clemens is assigned to customer.

Code

State of Memory

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Type Mismatch

- Suppose we want to input an age. Will this work?

```
int      age;

age = JOptionPane.showInputDialog(
                  null, “Enter your age”);
```

- No.
  String value cannot be assigned directly to an int variable.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

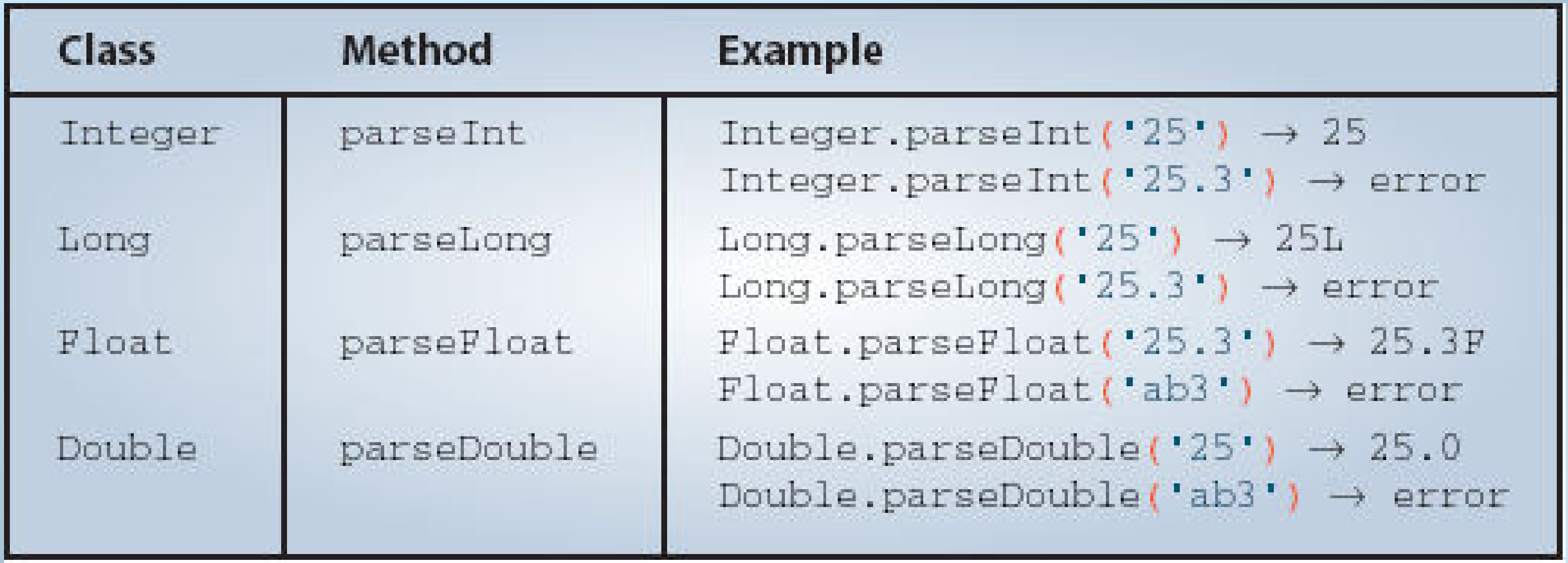
Type Conversion

- *Wrapper classes* are used to perform necessary type conversions, such as converting a String object to a numerical value.

```
int      age;
String   inputStr;

inputStr = JOptionPane.showInputDialog(
                null, “Enter your age”);

age = Integer.parseInt(inputStr);
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Other Conversion Methods



Class	Method	Example
Integer	parseInt	Integer.parseInt('25') → 25 Integer.parseInt('25.3') → error
Long	parseLong	Long.parseLong('25') → 25L Long.parseLong('25.3') → error
Float	parseFloat	Float.parseFloat('25.3') → 25.3F Float.parseFloat('ab3') → error
Double	parseDouble	Double.parseDouble('25') → 25.0 Double.parseDouble('ab3') → error

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sample Code Fragment

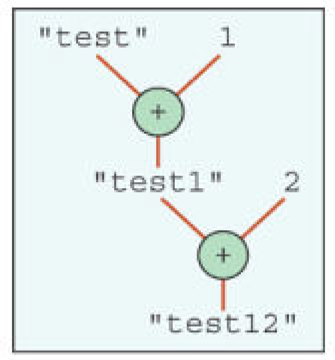
```
//code fragment to input radius and output
//area and circumference
double radius, area, circumference;

   radiusStr = JOptionPane.showInputDialog(
                         null, "Enter radius: " );

   radius = Double.parseDouble(radiusStr);

   //compute area and circumference
   area = PI * radius * radius;
   circumference = 2.0 * PI * radius;

   JOptionPane.showMessageDialog(null,
             "Given Radius:  " + radius + "\n" +
             "Area:          " + area   + "\n" +
             "Circumference: " + circumference);
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

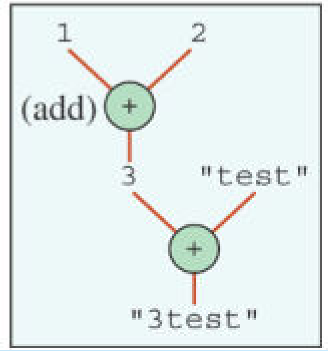
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- <val1> + <val2> is an addition if both are numbers. If either one of them is a String, the it is a concatenation.
- Evaluation goes from left to right.

output = “test” + 1 + 2;



output = 1 + 2 + “test”;



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The DecimalFormat Class

- Use a DecimalFormat object to format the numerical output.

```
double num = 123.45789345;

DecimalFormat df = new DecimalFormat(“0.000”);
            //three decimal places
```

```
System.out.print(num);            → 123.45789345

System.out.print(df.format(num));
                                  → 123.458
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Standard Output

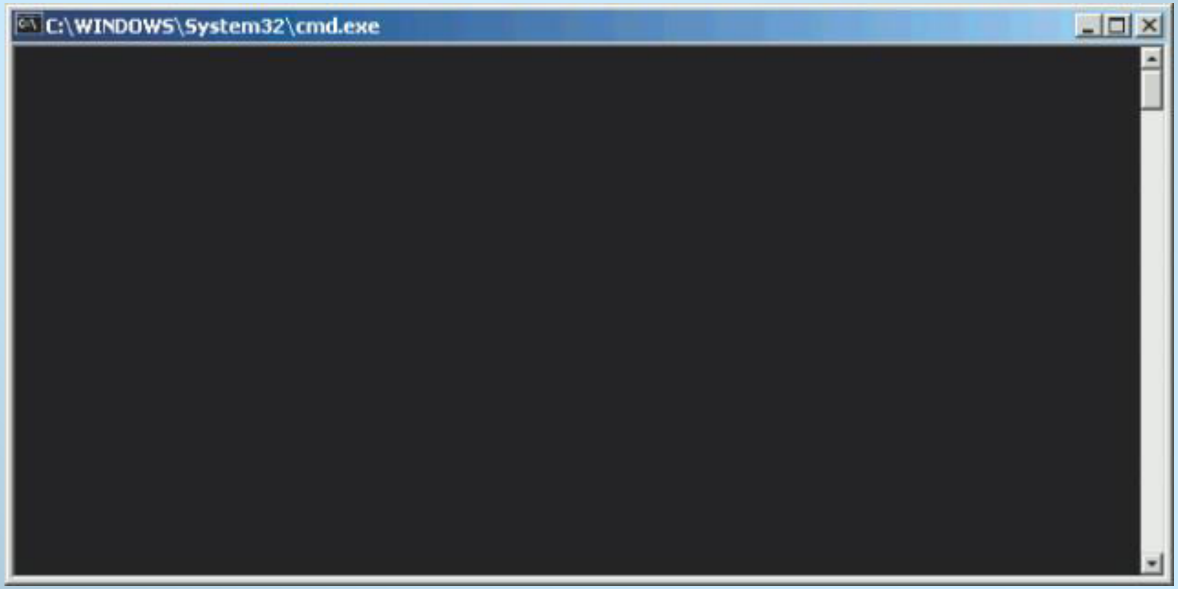
- The **showMessageDialog** method is intended for displaying short one-line messages, not for a general-purpose output mechanism.

- Using **System.out,** we can output multiple lines of text to the standard output window.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Standard Output Window

- A sample standard output window for displaying multiple lines of text.



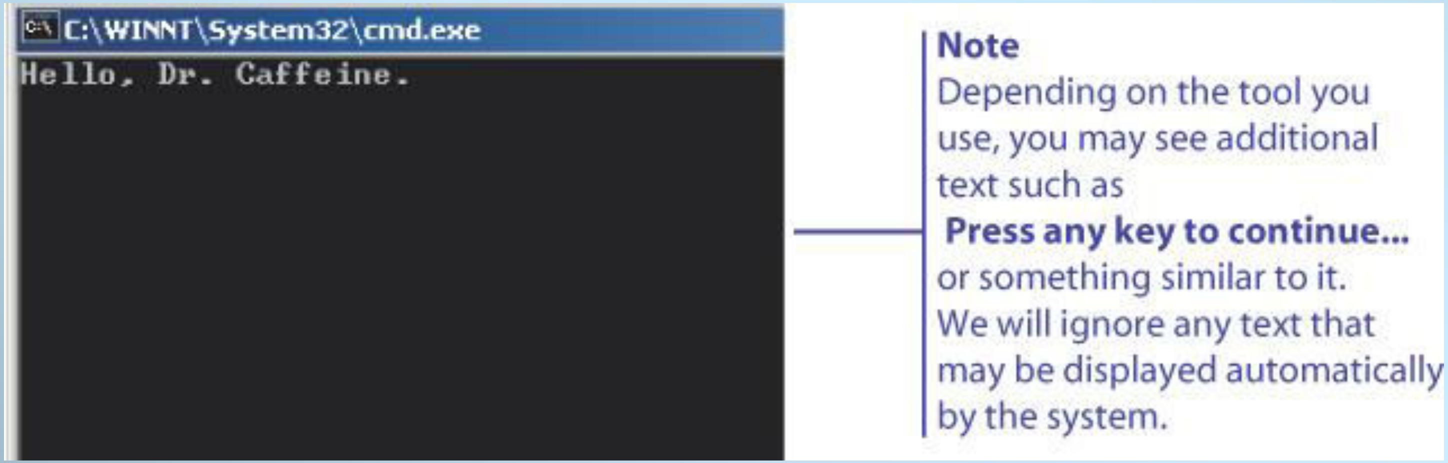
- The exact style of standard output window depends on the Java tool you use.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The print Method

- We use the **print** method to output a value to the standard output window.
- The **print** method will continue printing from the end of the currently displayed output.
- Example

```
System.out.print( “Hello, Dr. Caffeine.” );
```



```
Hello, Dr. Caffeine.
```

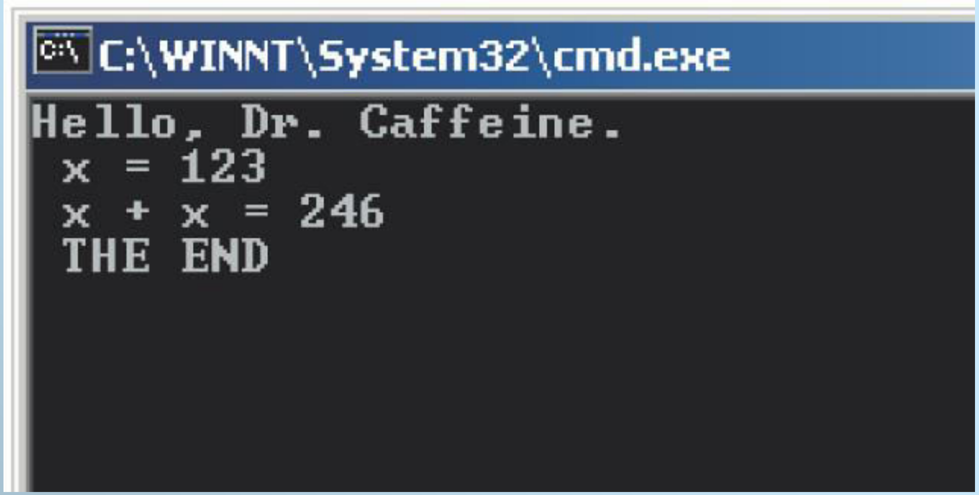
Note
Depending on the tool you use, you may see additional text such as **Press any key to continue...** or something similar to it. We will ignore any text that may be displayed automatically by the system.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The println Method

- We use **println** instead of **print** to skip a line.

```
int x = 123, y = x + x;
System.out.println( "Hello, Dr. Caffeine.“ );
System.out.print( " x = “ );
System.out.println( x );
System.out.print( " x + x = “ );
System.out.println( y );
System.out.println( " THE END“ );
```



```
Hello, Dr. Caffeine.
 x = 123
 x + x = 246
 THE END
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Standard Input

- The technique of using System.in to input data is called standard input.
- We can only input a single byte using System.in directly.
- To input primitive data values, we use the Scanner class (from Java 5.0).

```
Scanner scanner;

scanner = Scanner.create(System.in);

int num = scanner.nextInt();
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Common Scanner Methods:

Method	Example
nextByte( )	byte b = scanner.nextByte( );
nextDouble( )	double d = scanner.nextDouble( );
nextFloat( )	float f = scanner.nextFloat( );
nextInt( )	int i = scanner.nextInt( );
nextLong( )	long l = scanner.nextLong( );
nextShort( )	short s = scanner.nextShort( );
next()	String str = scanner.next();

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The Math class

- The **Math** class in the **java.lang** package contains class methods for commonly used mathematical functions.

```
double      num, x, y;

x = …;
y = …;

num = Math.sqrt(Math.max(x, y) + 12.4);
```

  - Table 3.6 in the textbook contains a list of class methods defined in the **Math** class.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

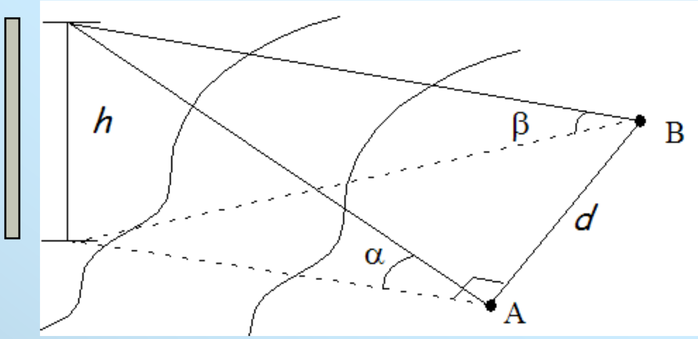
Some Math Class Methods

Method	Description
exp(a)	Natural number **e** raised to the power of **a**.
log(a)	Natural logarithm (base **e**) of **a**.
floor(a)	The largest whole number less than or equal to **a**.
max(a,b)	The larger of a and b.
pow(a,b)	The number **a** raised to the power of **b**.
sqrt(a)	The square root of **a**.
sin(a)	The sine of **a**. (Note: all trigonometric functions are computed in radians)

Table 3.8 page 113 in the textbook contains a list of class methods defined in the **Math** class.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Computing the Height of a Pole



$$h = \frac{d \sin\alpha \sin\beta}{\sqrt{\sin(\alpha+\beta)\sin(\alpha-\beta)}}$$

```
alphaRad = Math.toRadians(alpha);
betaRad  = Math.toRadians(beta);

height = ( distance * Math.sin(alphaRad) * Math.sin(betaRad) )
                       /
          Math.sqrt( Math.sin(alphaRad + betaRad) *
                                Math.sin(alphaRad - betaRad) );
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

The GregorianCalendar Class

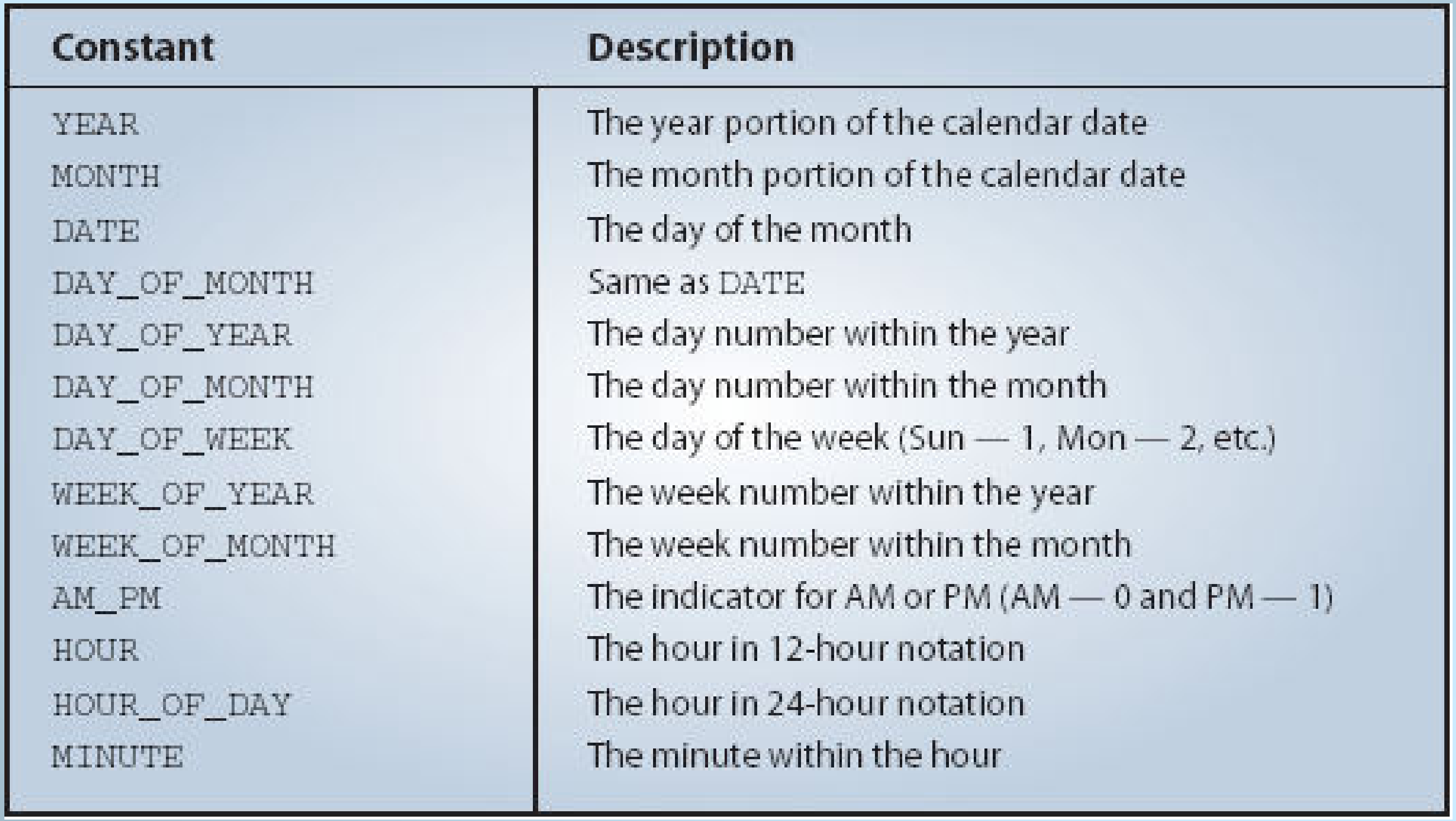
- Use a GregorianCalendar object to manipulate calendar information

```
GregorianCalendar today, independenceDay;

today    = new GregorianCalendar();

independenceDay
         = new GregorianCalendar(1776, 6, 4);
            //month 6 means July; 0 means January
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Retrieving Calendar Information

- This table shows the class constants for retrieving different pieces of calendar information from Date.



Constant	Description
YEAR	The year portion of the calendar date
MONTH	The month portion of the calendar date
DATE	The day of the month
DAY_OF_MONTH	Same as DATE
DAY_OF_YEAR	The day number within the year
DAY_OF_MONTH	The day number within the month
DAY_OF_WEEK	The day of the week (Sun — 1, Mon — 2, etc.)
WEEK_OF_YEAR	The week number within the year
WEEK_OF_MONTH	The week number within the month
AM_PM	The indicator for AM or PM (AM — 0 and PM — 1)
HOUR	The hour in 12-hour notation
HOUR_OF_DAY	The hour in 24-hour notation
MINUTE	The minute within the hour

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sample Calendar Retrieval

```
GregorianCalendar cal = new GregorianCalendar();
            //Assume today is Nov 9, 2003

System.out.print(“Today is ” +
      (cal.get(Calendar.MONTH)+1) + “/” +
      cal.get(Calendar.DATE) + “/” +
      cal.get(Calendar.YEAR));
```

Output

```
Today is 11/9/2003
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Problem Statement

- Problem statement:

  Write a loan calculator program that computes both monthly and total payments for a given loan amount, annual interest rate, and loan period.

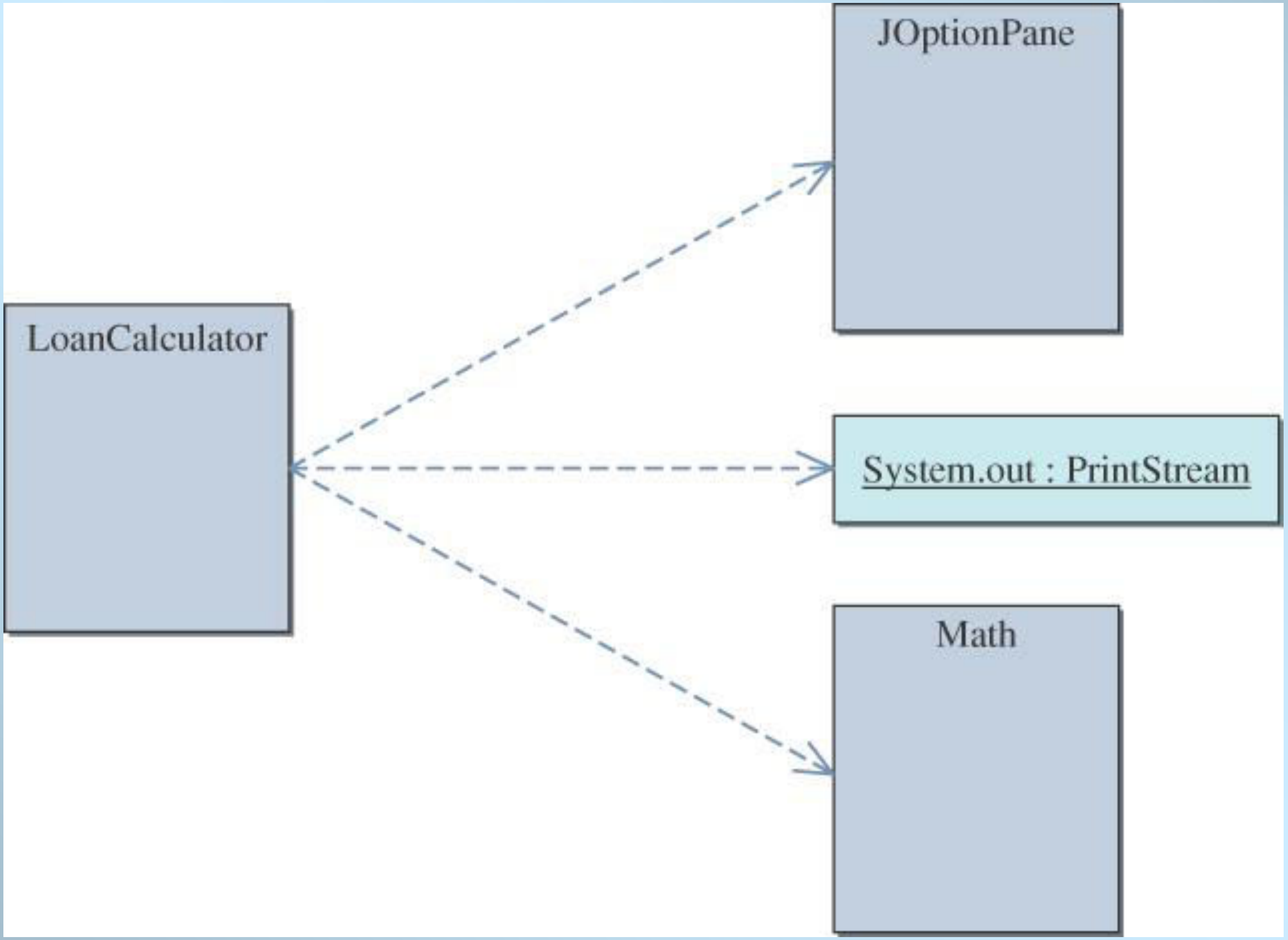
©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Overall Plan

- Tasks:
  - Get three input values: **loanAmount**, **interestRate**, and **loanPeriod**.
  - Compute the monthly and total payments.
  - Output the results.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Required Classes



©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Development Steps

- We will develop this program in four steps:

1. Start with code to accept three input values.
2. Add code to output the results.
3. Add code to compute the monthly and total payments.
4. Update or modify code and tie up any loose ends.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 1 Design

- Call the **showInputDialog** method to accept three input values:
  - loan amount,
  - annual interest rate,
  - loan period.
- Data types are

Input	Format	Data Type
loan amount	dollars and cents	double
annual interest rate	in percent (e.g.,12.5)	double
loan period	in years	int

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 1 Code

Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

Directory: Chapter3/Step1

Source File: Ch3LoanCalculator.java

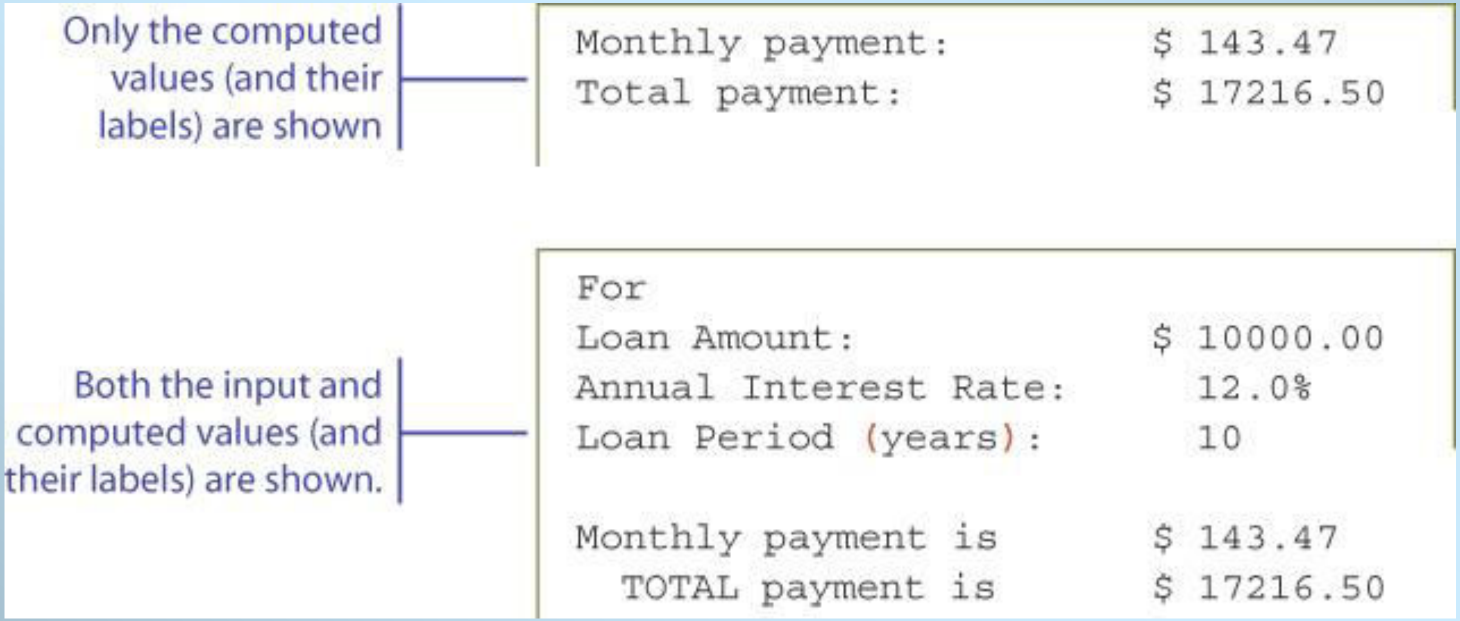
Step 1 Test

- In the testing phase, we run the program multiple times and verify that
  - we can enter three input values
  - we see the entered values echo-printed correctly on the standard output window

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Design

- We will consider the display format for out.
- Two possibilities are (among many others)



Only the computed values (and their labels) are shown

```
Monthly payment:            $ 143.47
Total payment:              $ 17216.50
```

Both the input and computed values (and their labels) are shown.

```
For
Loan Amount:                $ 10000.00
Annual Interest Rate:         12.0%
Loan Period (years):          10

Monthly payment is          $ 143.47
  TOTAL payment is          $ 17216.50
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 2 Code

Directory: Chapter3/Step2

Source File: Ch3LoanCalculator.java

Step 2 Test

- We run the program numerous times with different types of input values and check the output display format.
- Adjust the formatting as appropriate

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 3 Design

- The formula to compute the geometric progression is the one we can use to compute the monthly payment.
- The formula requires the loan period in months and interest rate as monthly interest rate.
- So we must convert the annual interest rate (input value) to a monthly interest rate (per the formula), and the loan period to the number of monthly payments.

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

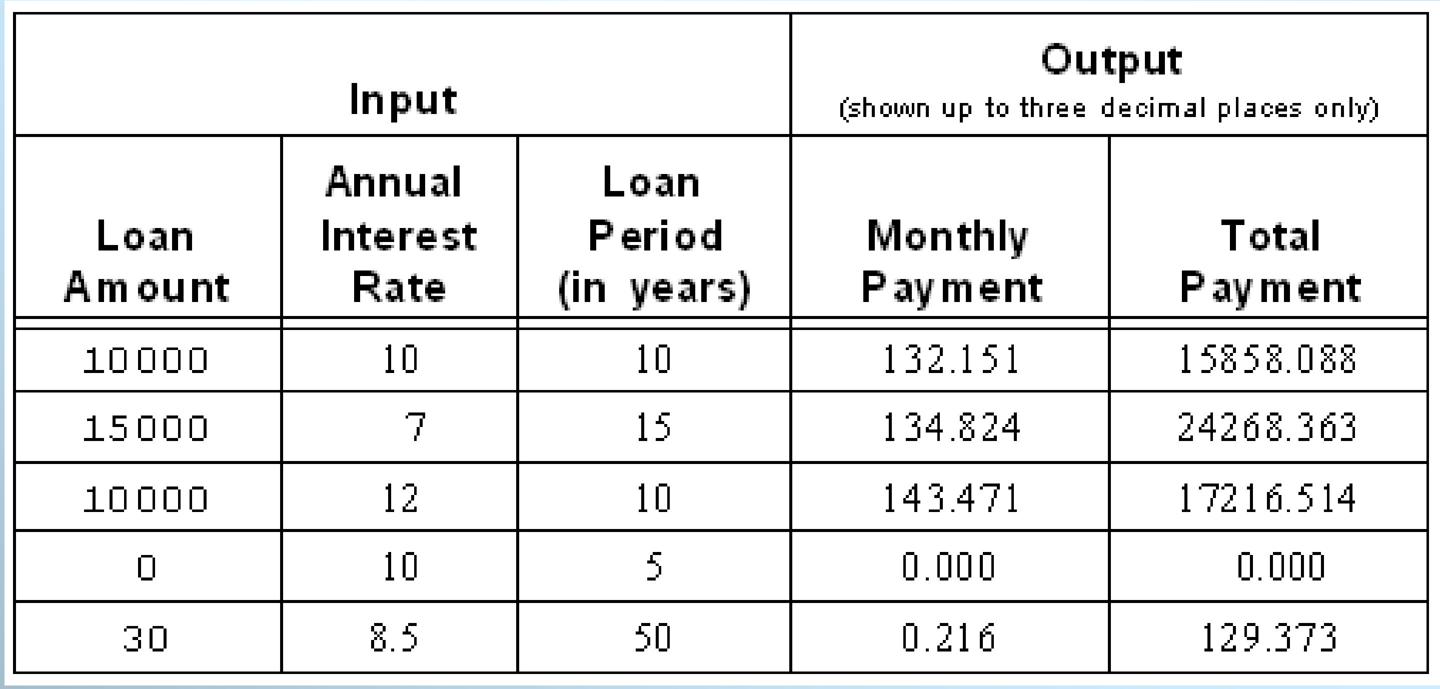
Step 3 Code

Directory: Chapter3/Step3

Source File: Ch3LoanCalculator.java

Step 3 Test

- We run the program numerous times with different types of input values and check the results.



Input			Output (shown up to three decimal places only)	
Loan Amount	Annual Interest Rate	Loan Period (in years)	Monthly Payment	Total Payment
10000	10	10	132.151	15858.088
15000	7	15	134.824	24268.363
10000	12	10	143.471	17216.514
0	10	5	0.000	0.000
30	8.5	50	0.216	129.373

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Step 4: Finalize

- We will add a program description
- We will format the monthly and total payments to two decimal places using DecimalFormat.

Directory: Chapter3/Step4

Source File: Ch3LoanCalculator.java

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.