Elementary Programming

Identifiers 标识符

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).

  ⁎ Must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.

  ⁎ Cannot be a reserved word. (See Appendix A for a list of reserved words).

  ⁎ Cannot be **`true`**, **`false`**, or **`null`**.

  ⁎ Can be of any length.

Naming Conventions 命名惯例

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase.
  - If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
  - For example, the variables **radius** and **area**, and the method **computeArea**.

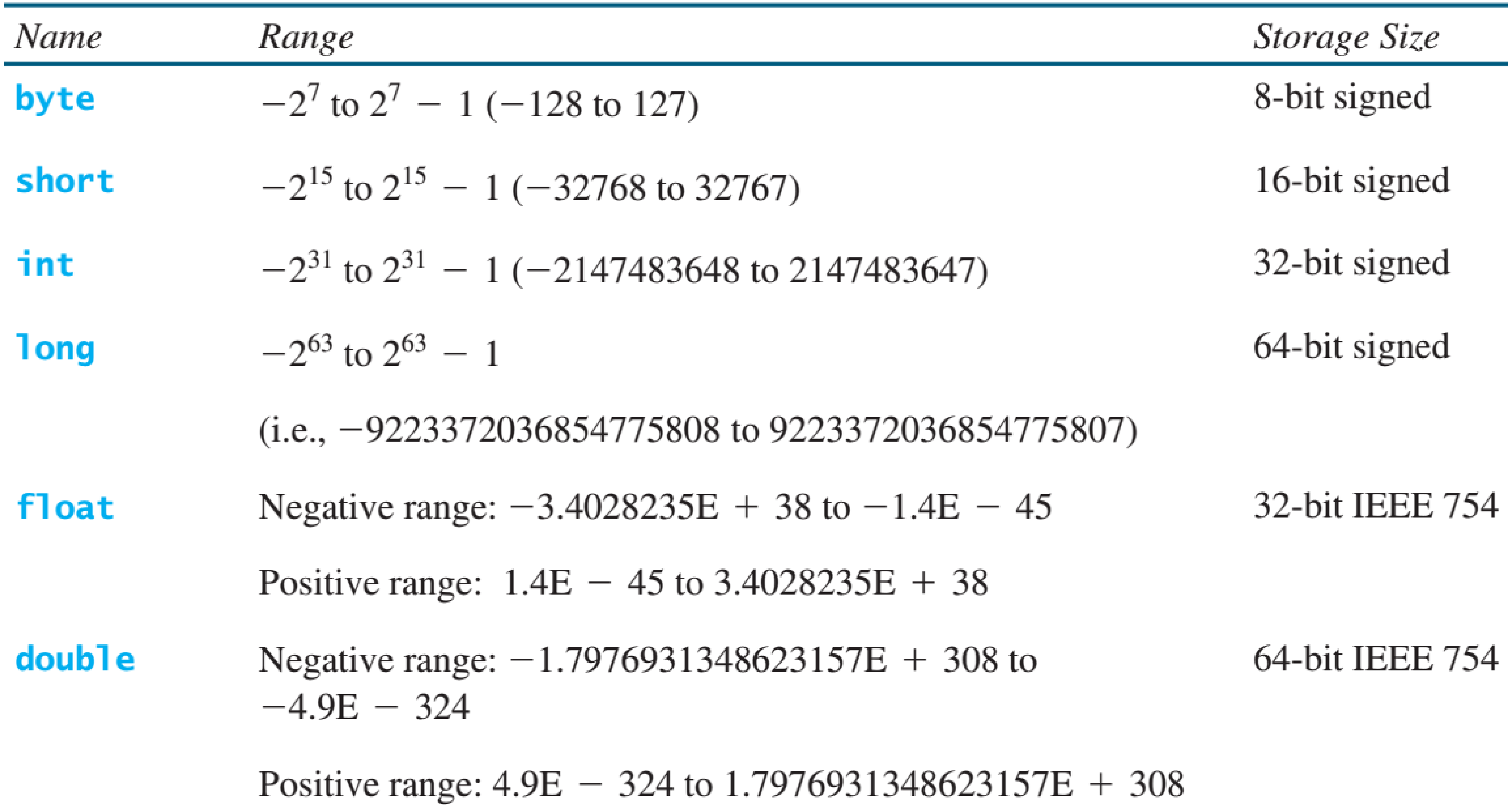
Class names:

- Capitalize the first letter of each word in the name.
- For example, the class name **`ComputeArea`**.

Constants:

- Capitalize all letters, and use underscores to connect words.
- For example, the constant **`PI`** `and` **`MAX_VALUE`**

Numerical Data Types 数值型数据

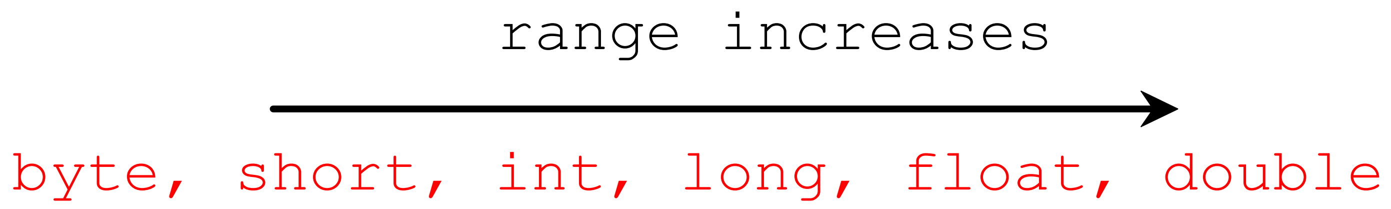


Name	*Range*	*Storage Size*
`byte`	-2^7 to $2^7 - 1$ (−128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (−32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (−2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., −9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: −3.4028235E + 38 to −1.4E − 45 Positive range: 1.4E − 45 to 3.4028235E + 38	32-bit IEEE 754
`double`	Negative range: −1.7976931348623157E + 308 to −4.9E − 324 Positive range: 4.9E − 324 to 1.7976931348623157E + 308	64-bit IEEE 754

Numeric Type Conversion Rules
数值类型之间的转换规则

* Java automatically converts the operand based on the following rules:

  - If one of the operands is double, the other is converted into double.
  - Otherwise, if one of the operands is float, the other is converted into float.
  - Otherwise, if one of the operands is long, the other is converted into long.
  - Otherwise, both operands are converted into int.



range increases

byte, short, int, long, float, double

Conversion between Strings and Numbers
字符串-数值之间转换

从字符串到数值

```
int intValue = Integer.parseInt(intString);
double doubleValue =
Double.parseDouble(doubleString);
```

从数值到字符串

```
String s = number + "";
```

Operator Precedence and Associativity
运算符的优先级和结合性

⁎ The expression in the parentheses is evaluated first. (Parentheses can be nested, the inner parentheses is executed first.)

⁎ When there are no parentheses, the operators are applied according to the precedence rule.

⁎ If operators with the same precedence are next to each other, their associativity determines the order of evaluation.

Operator Precedence

- **var++, var--**
- **+, - (Unary plus and minus), ++var, --var**
- **(type) Casting**
- **! (Not)**
- ***, /, % (Multiplication, division, and remainder)**
- **+, - (Binary addition and subtraction)**
- **<, <=, >, >= (Relational operators)**
- **==, !=; (Equality)**
- **^ (Exclusive OR)**
- **&& (Conditional AND) Short-circuit AND**
- **|| (Conditional OR) Short-circuit OR**
- **=, +=, -=, *=, /=, %= (Assignment operator)**

Formatting Output

Use the printf statement.

```
System.out.printf(format, items);
```

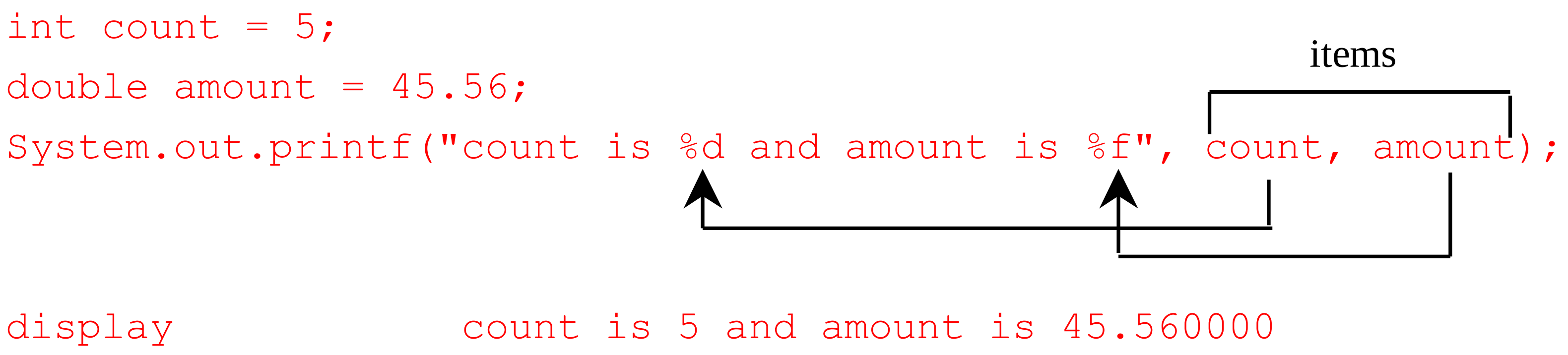
- Where format is a string that may consist of substrings and format specifiers.

- A format specifier specifies how an item should be displayed.

- An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s		1"



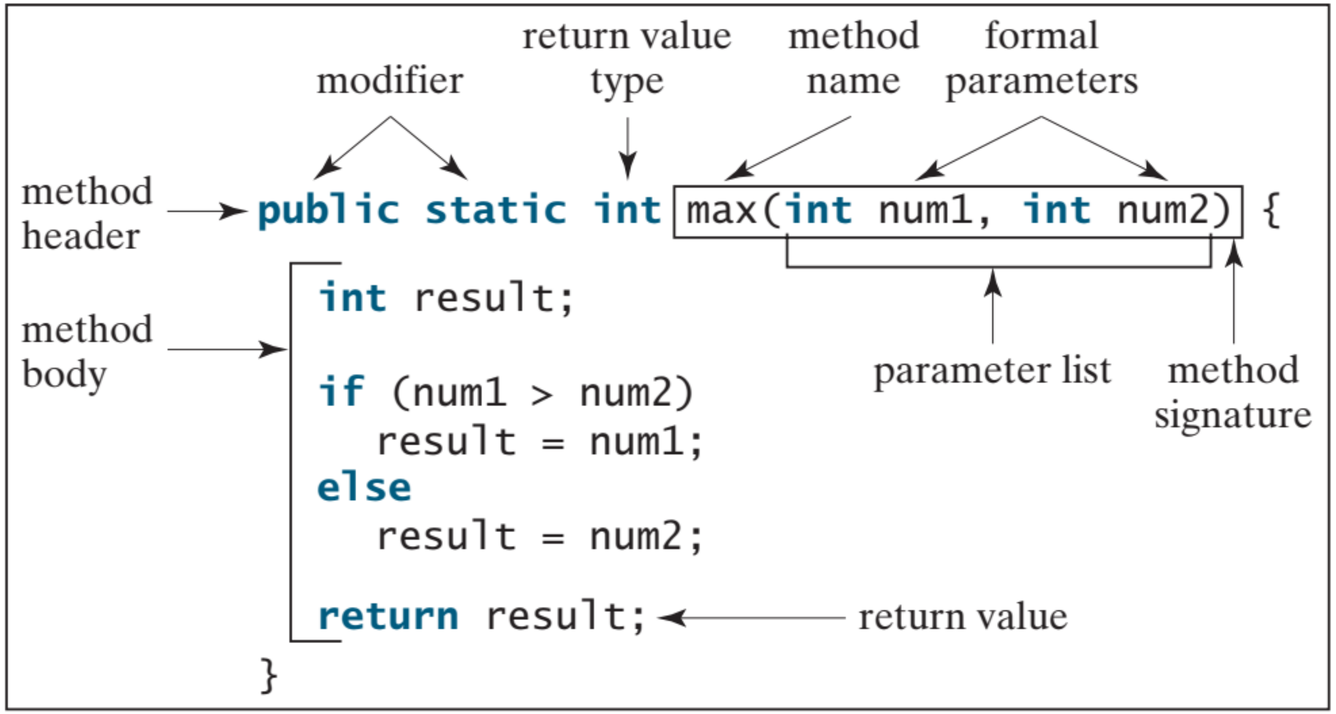
```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);


display          count is 5 and amount is 45.560000
```

Methods

Defining Methods 方法定义

- A method is a collection of statements that are grouped together to perform an operation.
- One of the benefits of methods is for reuse.

Define a method



modifier

return value type

method name

formal parameters

method header

method body

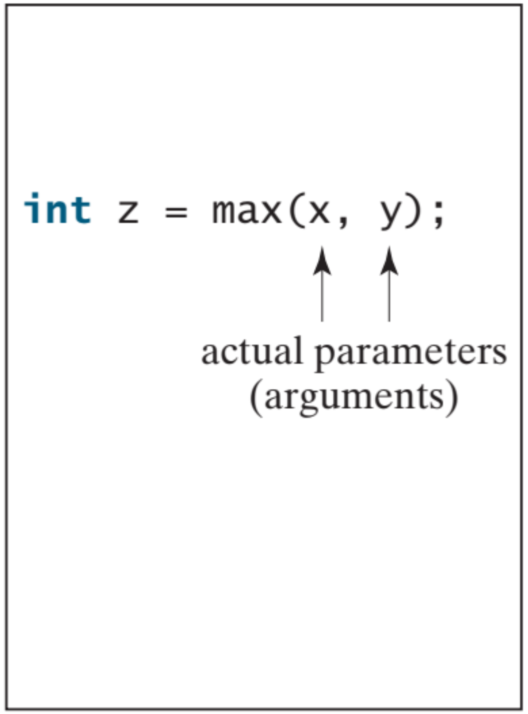
```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
      result = num1;
    else
      result = num2;

    return result;
}
```

parameter list

method signature

return value

Invoke a method



```
int z = max(x, y);
```

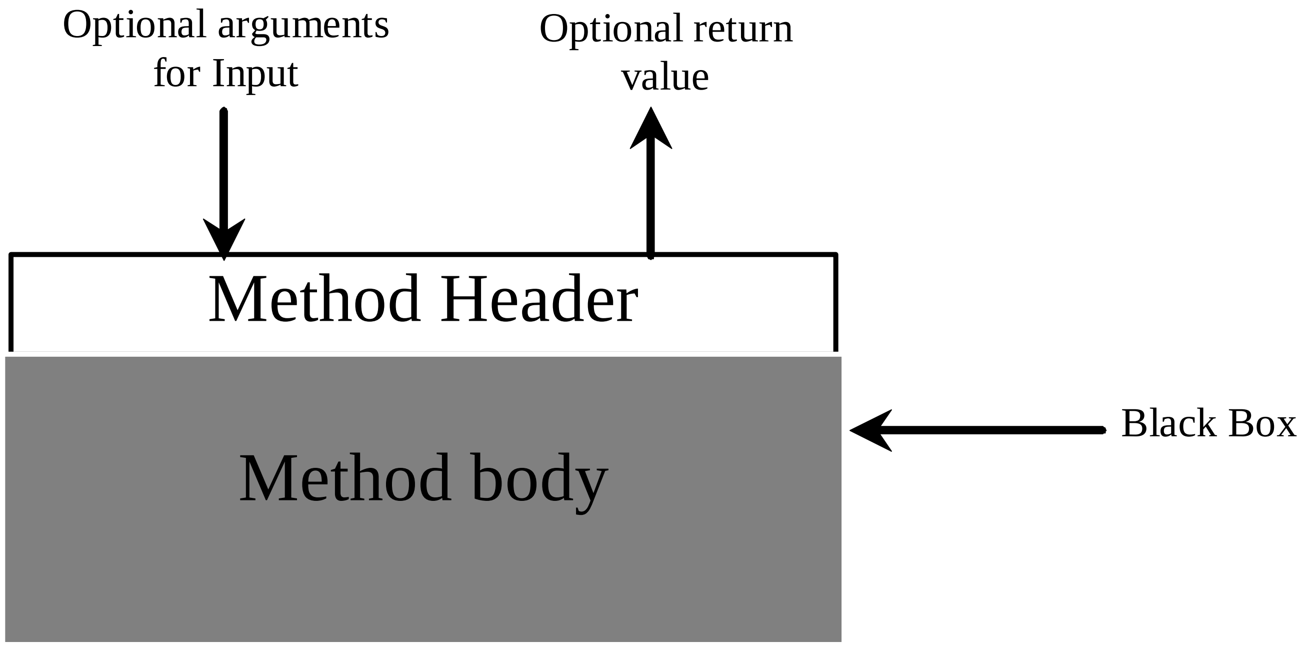
actual parameters (arguments)

Benefits of Methods 好处

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity: modularize code and improve the quality of the program.

Method Abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.



Overloading Methods 方法重载

```
public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}


public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

public static double max(double num1, double num2,
 double num3) {
  return max(max(num1, num2), num3);
}
```

Ambiguous Invocation

* Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
* This is referred to as *ambiguous invocation*. Ambiguous invocation is a compile error.

Ambiguous Invocation

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```

compile error！

Instance method and Static method
实例方法和静态方法

∗ *instance method: a* method that is invoked from a specific string instance.

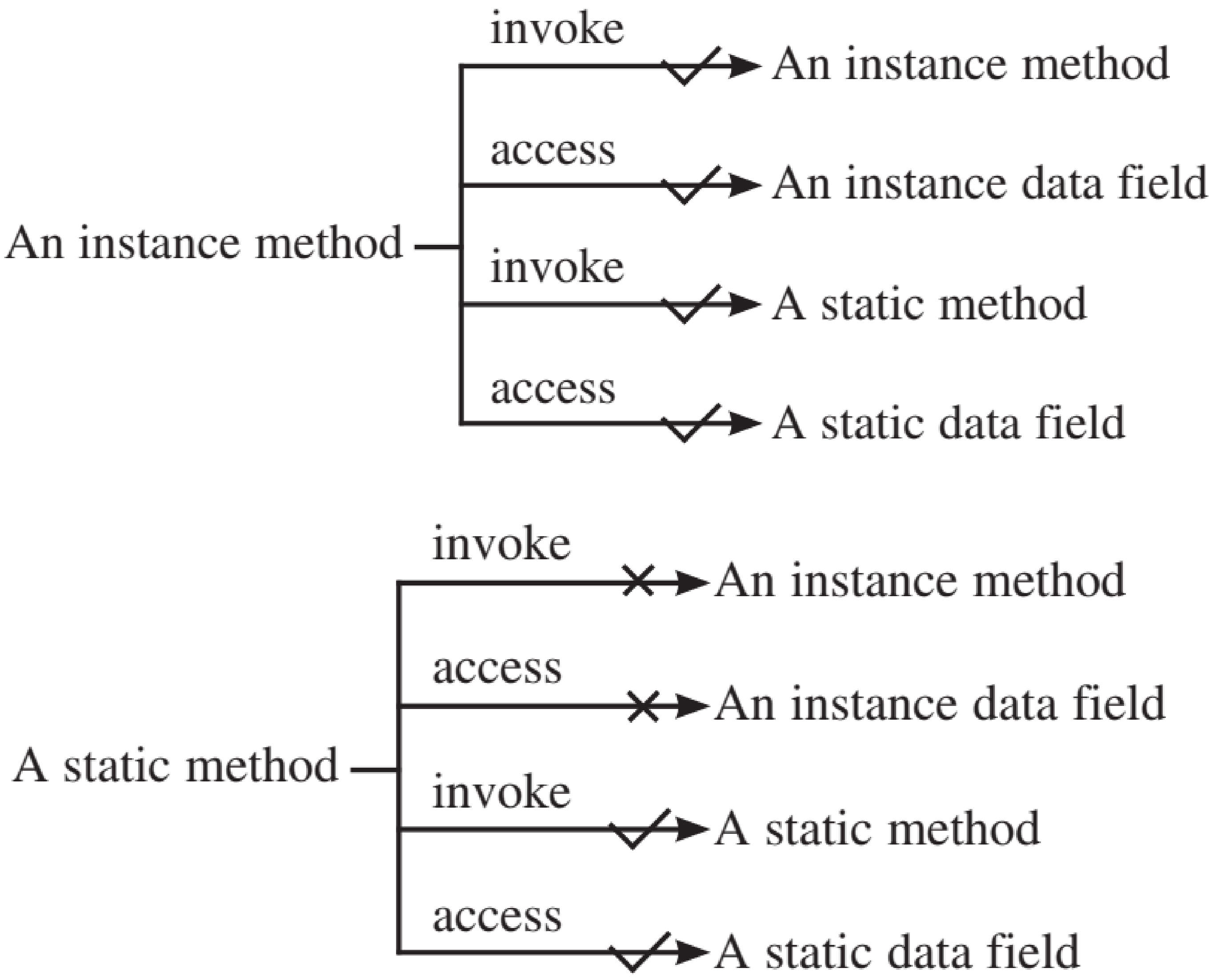
- Example

```
String message = "Welcome to Java";
int len = message.length();
```

∗ *static method: a* method that is invoked from a class.

- Example `double x = Math.pow(2, 3)`

Instance vs Static



Design Guide

* How to decide whether a variable or a method should be an instance one or a static one?

* Decision relies on: whether a variable or a method is dependent on a specific instance of the class？

- Yes: should be an instance variable or method.

  - Example: radius and getArea() is dependent on a specific circle.

- No: should be a static variable or method.

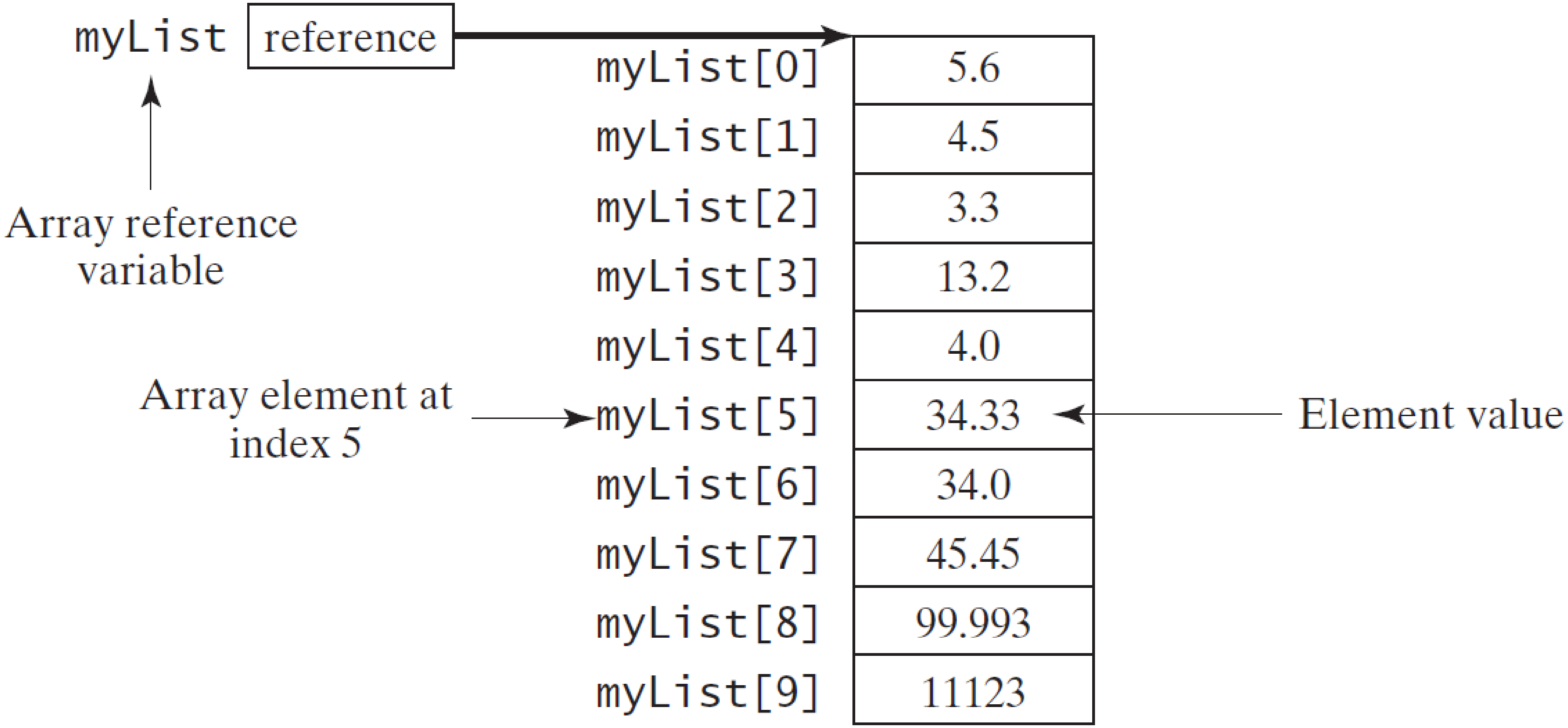
  - Example: Math.PI and Math.pow(a,b) in Math class.

Single-Dimensional Arrays

Introducing Arrays

Array is a data structure that represents a collection of the same types of data.

```
double[] myList = new double[10];
```



myList reference

Array reference variable

Array element at index 5

Element value

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4.0
myList[5]	34.33
myList[6]	34.0
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

Array Initializers 初始化

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

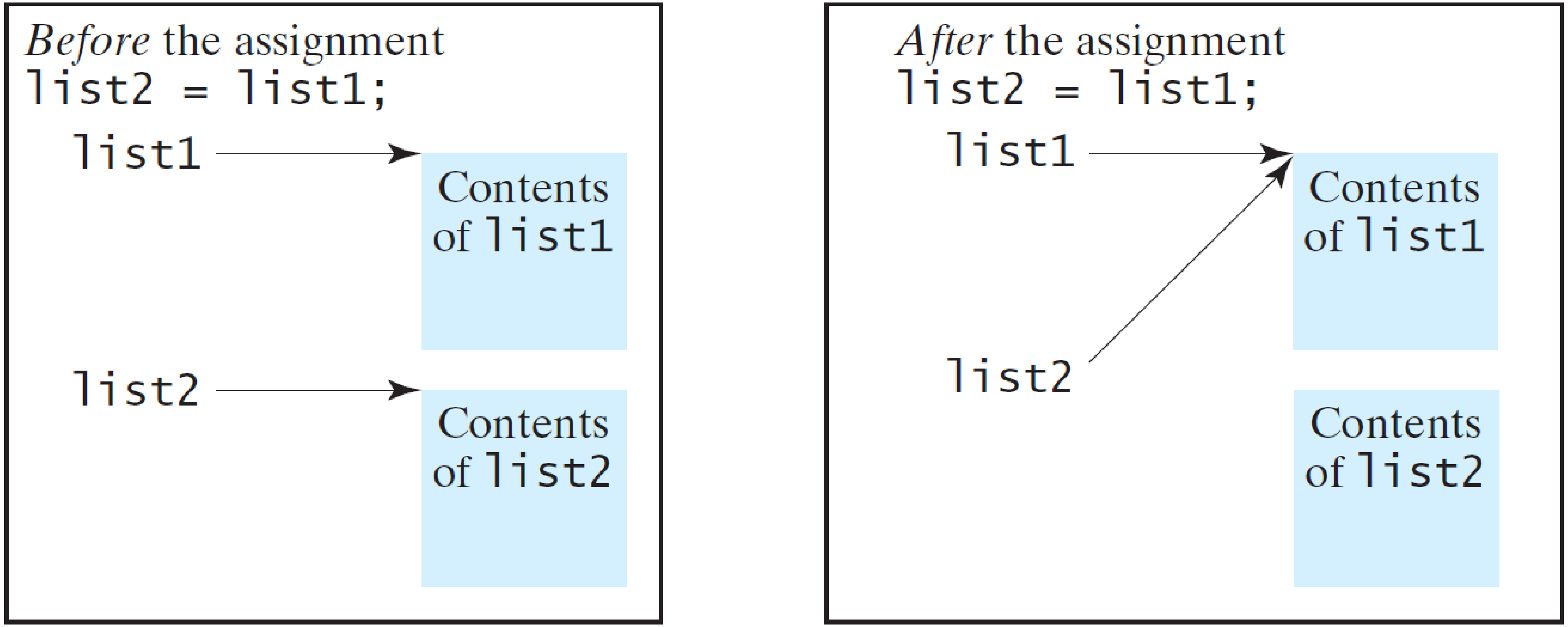
This shorthand syntax must be in one statement. The following is wrong: 这样就不行

```
double[] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

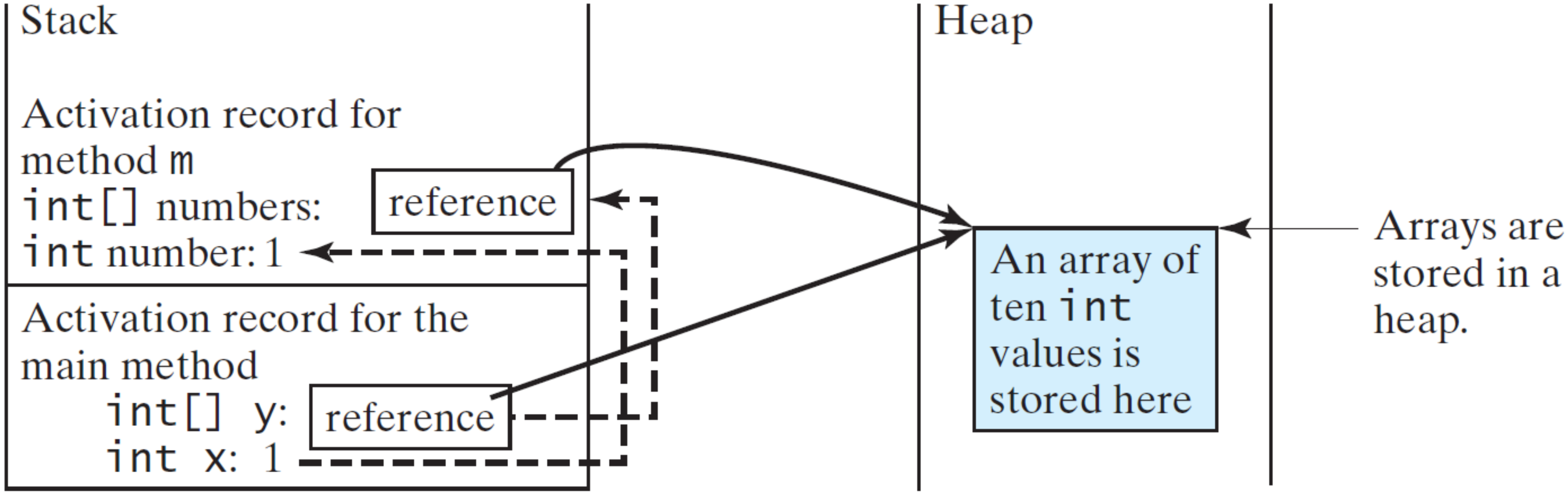
Copying Arrays 复制数组

- Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;



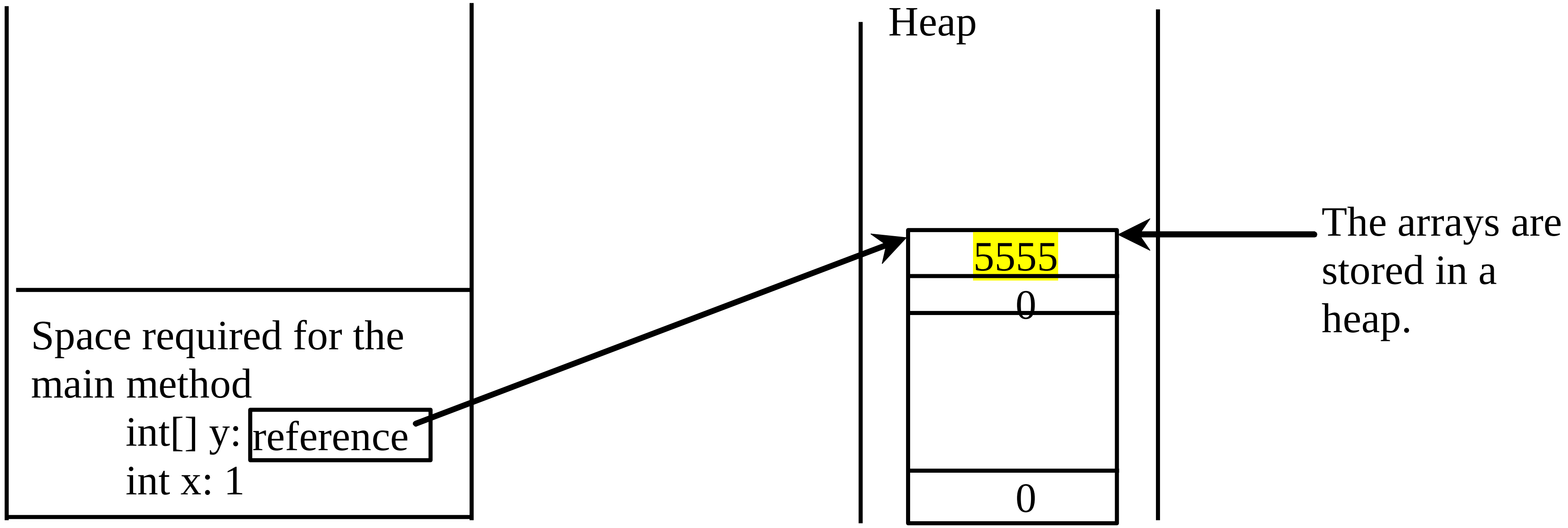
Call Stack


Stack
Activation record for method m
int[] numbers:
reference
int number: 1
Activation record for the main method
int[] y:
reference
int x: 1
Heap
An array of ten int values is stored here
Arrays are stored in a heap.

* When invoking m(x, y), the values of x and y are passed to number and numbers.

* Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Heap


Space required for the main method
int[] y: reference
int x: 1
Heap
5555
0
0
The arrays are stored in a heap.

The JVM stores the array in an area of memory, called *heap*, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

The Arrays.binarySearch Method

- Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(list, 11));
```

Return is 4

```
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(chars, 't'));
```

Return is –4
(insertion point is
3,
so return is -3-1)

- The array must be pre-sorted in increasing order.

Pass Arguments to Invoke the Main Method

- The main method in class B is invoked by a method in A:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
  ...
  }
}
```

`args[0], args[1], ..., args[n]` corresponds to `arg0, arg1, ..., argn`

```
java TestMain arg0 arg1 arg2 ... argn
```

Multidimensional Arrays

Ragged Arrays

- Each row in a two-dimensional array is itself an array.
- So, the rows can have different lengths. Such an array is known as *a ragged array*.
- For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1

Ragged Arrays

- Each row in a two-dimensional array is itself an array.
- So, the rows can have different lengths. Such an array is known as *a ragged array*.
- For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1
```

Objects and Classes

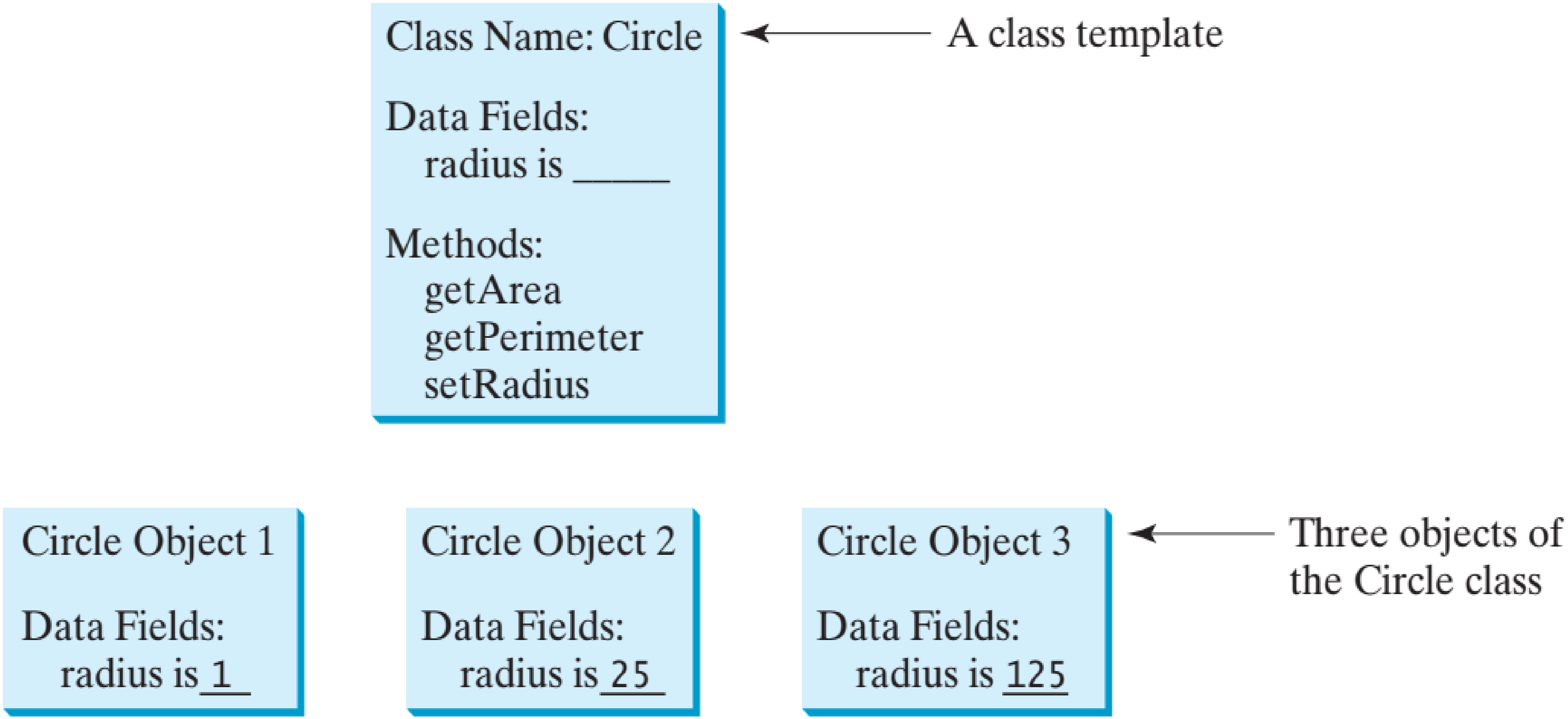
OO Programming Concepts

* Object-oriented programming (OOP) involves programming using objects. 面向对象编程

- An *object* represents an entity in the real world that can be distinctly identified.
  - For example, a student, a desk, a circle, a button.
  - An object has a unique identity, state, and behaviors.
  - The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values.
  - The *behavior* of an object is defined by a set of methods.

Objects and Classes 对象和类

- *Classes* are constructs that define objects of the same type.
- data fields: variables   behaviors: methods
- special methods: constructors, used to construct objects.



Class Name: Circle

Data Fields:
radius is _____

Methods:
getArea
getPerimeter
setRadius

A class template

Circle Object 1

Data Fields:
radius is 1

Circle Object 2

Data Fields:
radius is 25

Circle Object 3

Data Fields:
radius is 125

Three objects of the Circle class

Constructors 构造器

* Constructors are a special kind of methods that are invoked to construct objects.

- *no-arg constructor*: with no parameters.
- must have the same name as the class itself.
- do not have a return type.
- are invoked using the ***new*** operator, to create and initialize objects.

```
Circle() {
}

Circle(double newRadius) {
  radius = newRadius;
}
```

Default Constructor 默认构造器

* A class may be defined without constructors.

- In this case, a no-arg constructor with an empty body is implicitly defined in the class.

- This constructor, called *a default constructor*, is provided automatically only if no constructors are explicitly defined in the class.

Default Value for a Data Field
数据字段的默认值

* The default value of a data field is:
  - reference type: null
  - numeric type: 0
  - boolean type: false
  - char type: '\u0000'

```
public class Student {
  String name; // default value null
  int age; // default value 0
  boolean isScienceMajor; // default value false
  char gender; // default value '\u0000'
}
```

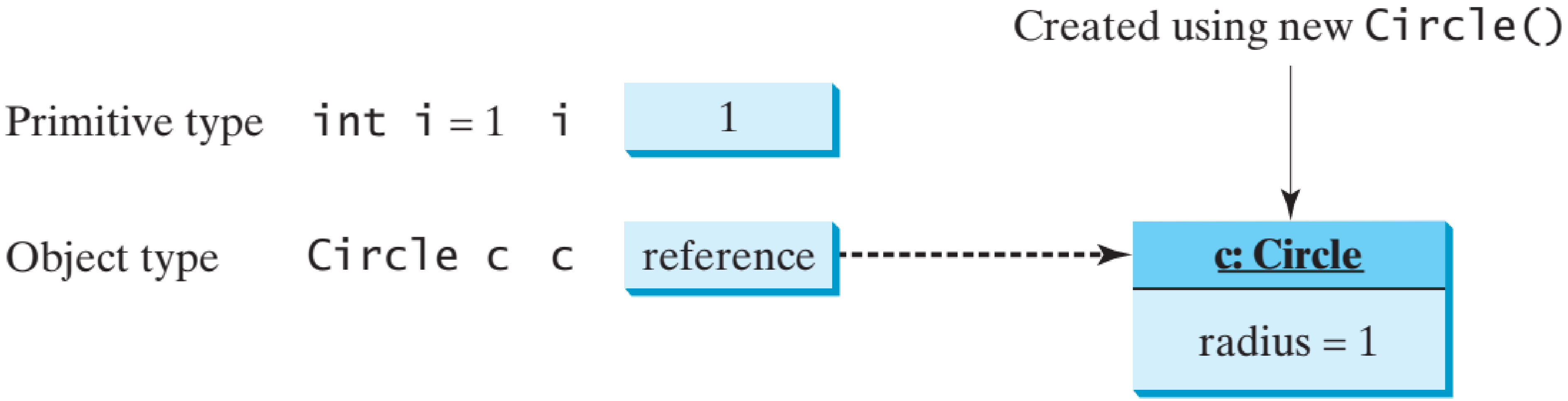
How about variables inside methods ?
方法内部的变量呢?

⋆ Java assigns no default value to a local variable inside a method.

```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

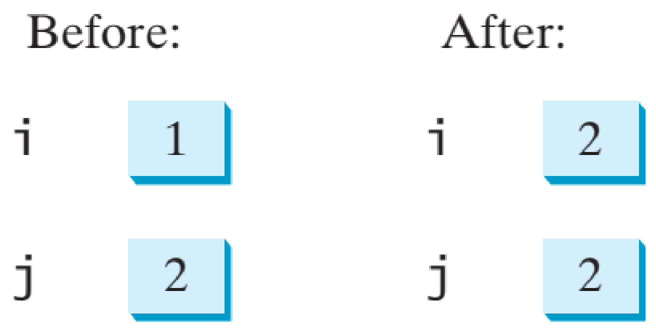
Compile error: variable not initialized

Differences between Variables of Primitive Data Types and Object Types 原始数据类型和对象类型的区别



Copying Variables of Primitive Data Types and Object Types

Primitive type assignment `i = j`



Before:

i 1

j 2

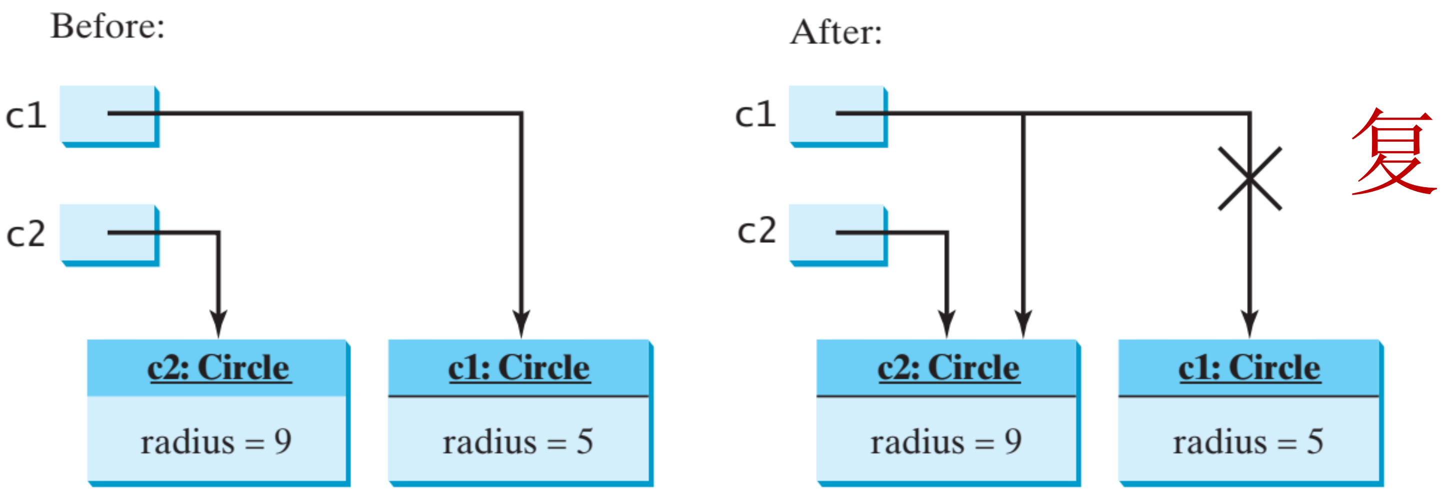
After:

i 2

j 2

复制原始数据类型

Object type assignment `c1 = c2`



Before:

c1

c2

c2: Circle

radius = 9

c1: Circle

radius = 5

After:

c1

c2

c2: Circle

radius = 9

c1: Circle

radius = 5

复制对象类型

Garbage Collection 垃圾收集

⁎ As shown, after the assignment statement c1 = c2, c1 points to the same object referenced by c2.

- The object previously referenced by c1 is no longer referenced.
- This object is known as garbage.
- Garbage is automatically collected by JVM.

TIP:

⁎ If you know that an object is no longer needed, you can explicitly assign *null* to a reference variable for the object.

⁎ The JVM will automatically collect the space if the object is not referenced by any variable．

Visibility Modifiers and Accessor/Mutator Methods

* By default, the class, variable, or method can be accessed by any class in the same package.

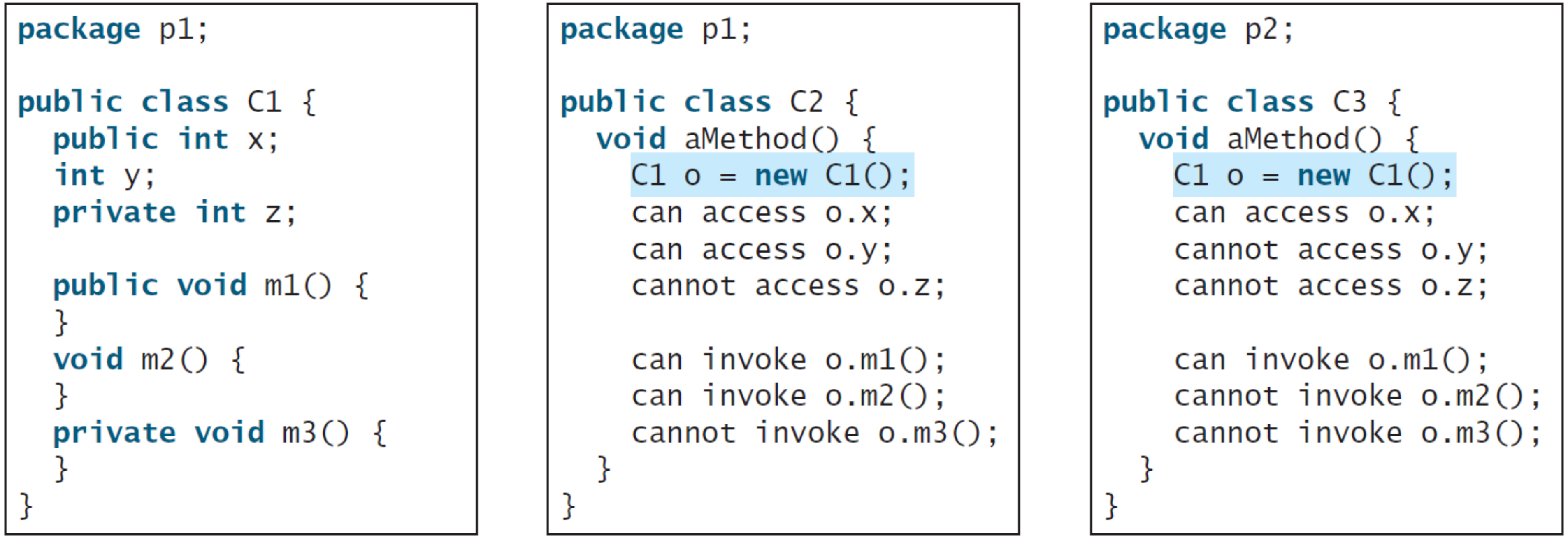
❑ `public`

  * The class, data, or method is visible to any class in any package.

❑ `private`

  * The data or methods can be accessed only by the declaring class.

  * The ***get*** and ***set*** methods are used to read and modify private properties.

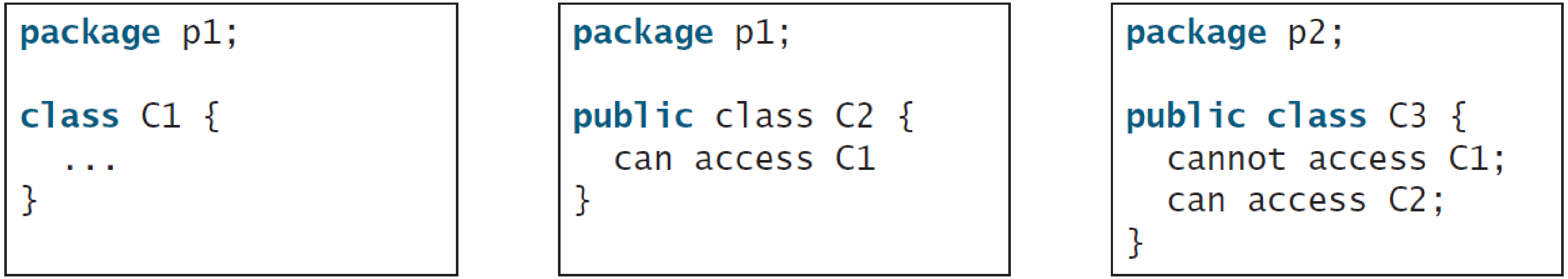


```
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

* The private modifier restricts access to within a class,
* the default modifier restricts access to within a package,
* and the public modifier enables unrestricted access.



```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

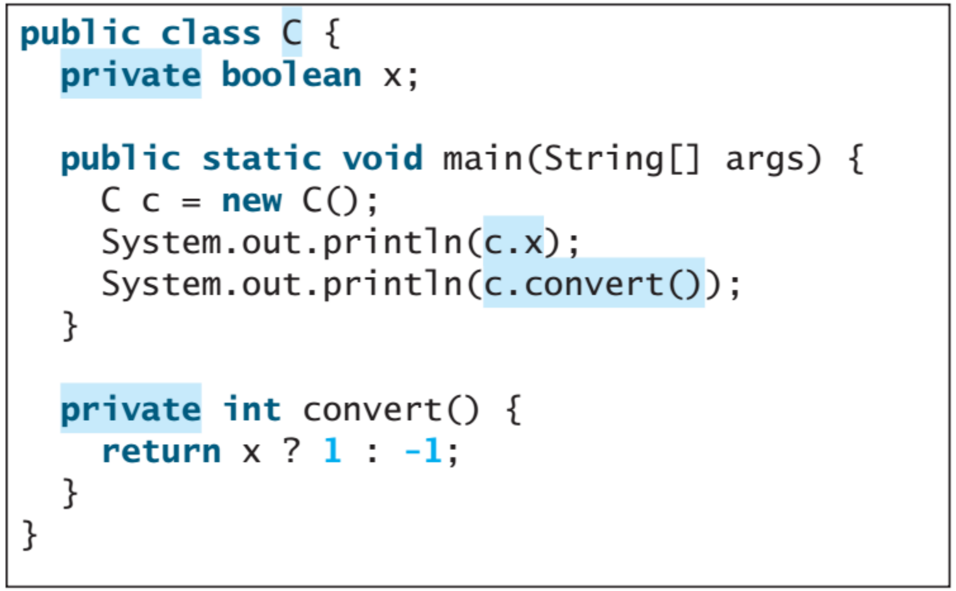
```
package p2;

public class C3 {
  cannot access C1;
  can access C2;
}
```

* A nonpublic class has package-access

NOTE

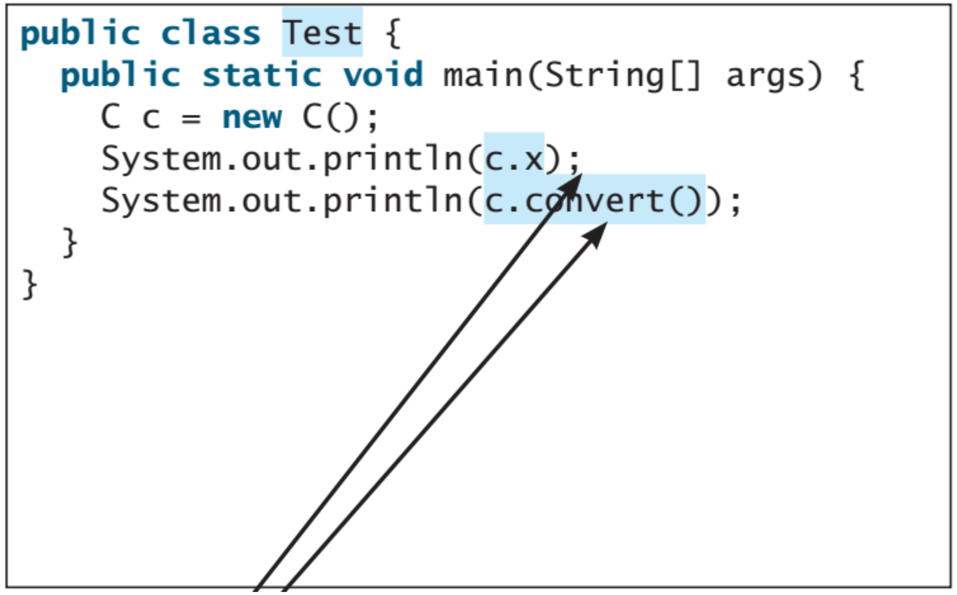
- An object cannot access its private members, as shown in (b).

- It is OK, however, if the object is declared in its own class,



```
public class C {
  private boolean x;

  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }

  private int convert() {
    return x ? 1 : -1;
  }
}
```

(a) This is okay because object c is used inside the class C



```
public class Test {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }
}
```

(b) This is wrong because x and convert are private in class C

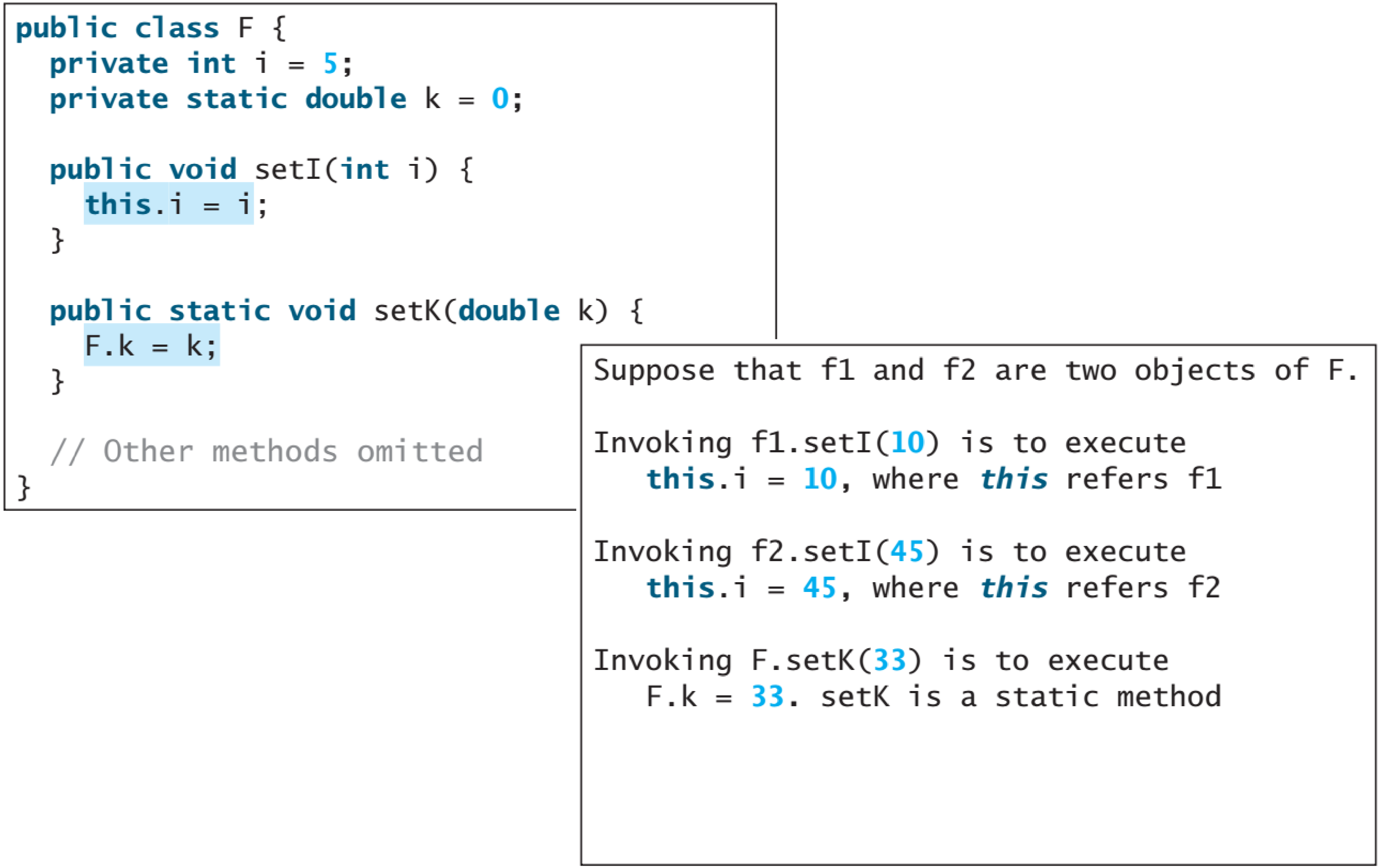
The this Keyword

❑The this keyword is the name of a reference that refers to an object itself.

❑ One common use of the this keyword is reference a class’s *hidden data fields*.

❑Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

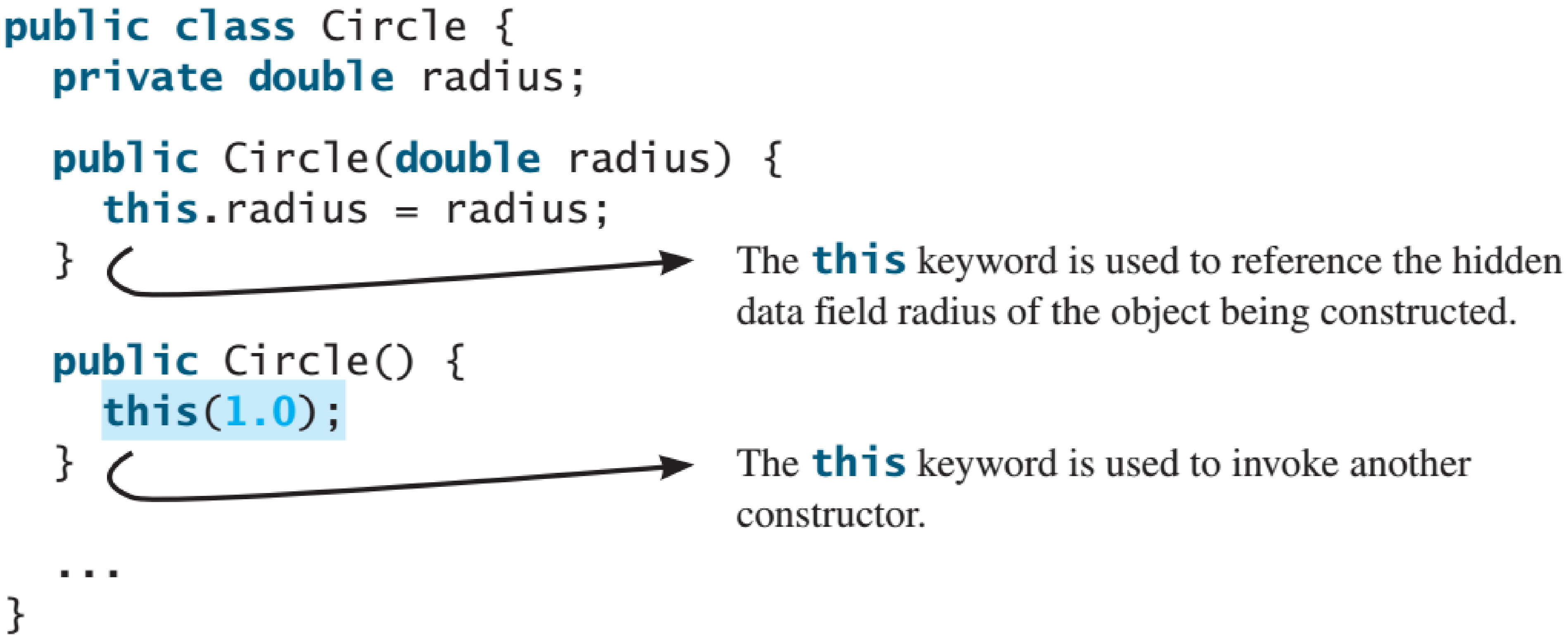
Reference the Hidden Data Fields



```java
public class F {
  private int i = 5;
  private static double k = 0;

  public void setI(int i) {
    this.i = i;
  }

  public static void setK(double k) {
    F.k = k;
  }

  // Other methods omitted
}
```

```
Suppose that f1 and f2 are two objects of F.

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2

Invoking F.setK(33) is to execute
   F.k = 33. setK is a static method
```

Calling Overloaded Constructor



```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  ...
}
```

The **this** keyword is used to reference the hidden data field radius of the object being constructed.

The **this** keyword is used to invoke another constructor.