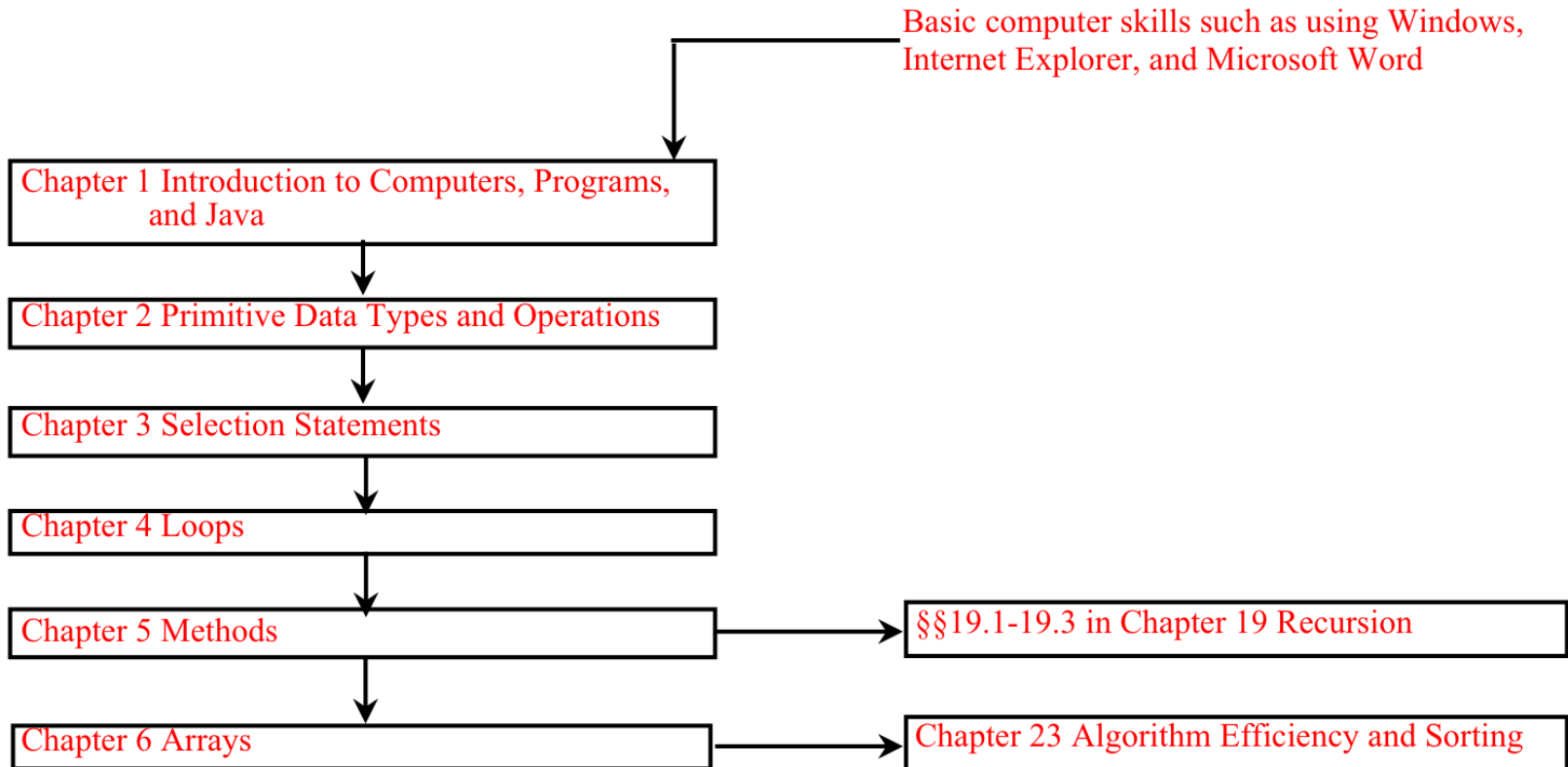


Chapter 5 Methods



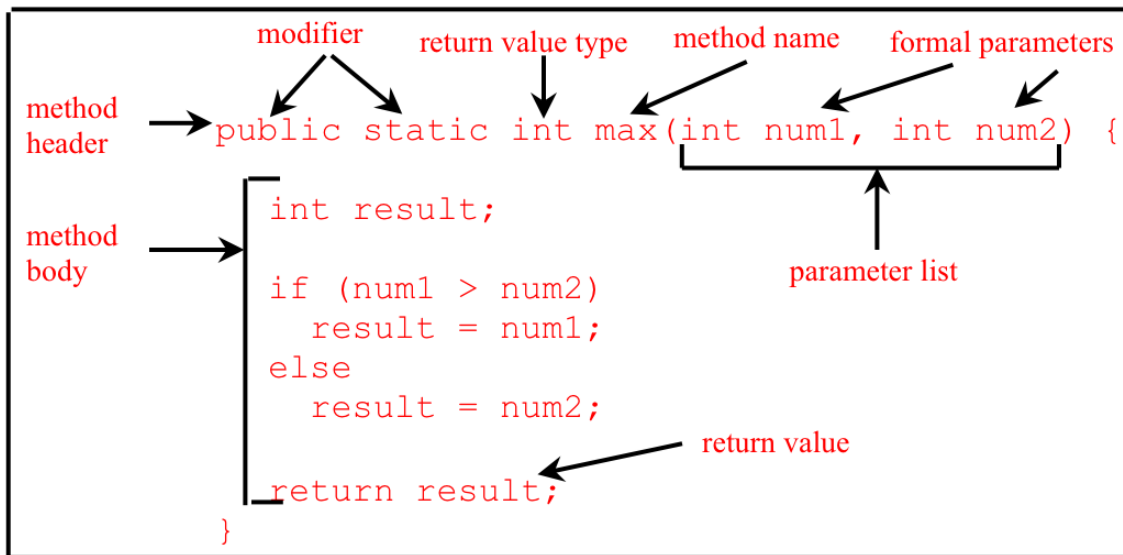
Objectives

- To declare methods, invoke methods, and pass arguments to a method (§5.2-5.4).
- To use method overloading and know ambiguous overloading (§5.5).
- To determine the scope of local variables (§5.6).
- To learn the concept of method abstraction (§5.7).
- To know how to use the methods in the Math class (§5.8).
- To design and implement methods using stepwise refinement (§5.10).
- To group classes into packages (§5.11 Optional).

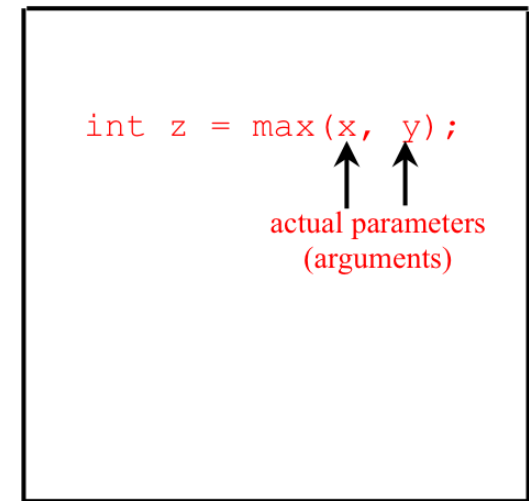
Introducing Methods

A method is a collection of statements that are grouped together to perform an operation.

Define a method



Invoke a method



Introducing Methods, cont.

- *Method signature* is the combination of the method name and the parameter list.
- The variables defined in the method header are known as *formal parameters*.
- When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.

Trace Call Stack

i is declared and initialized

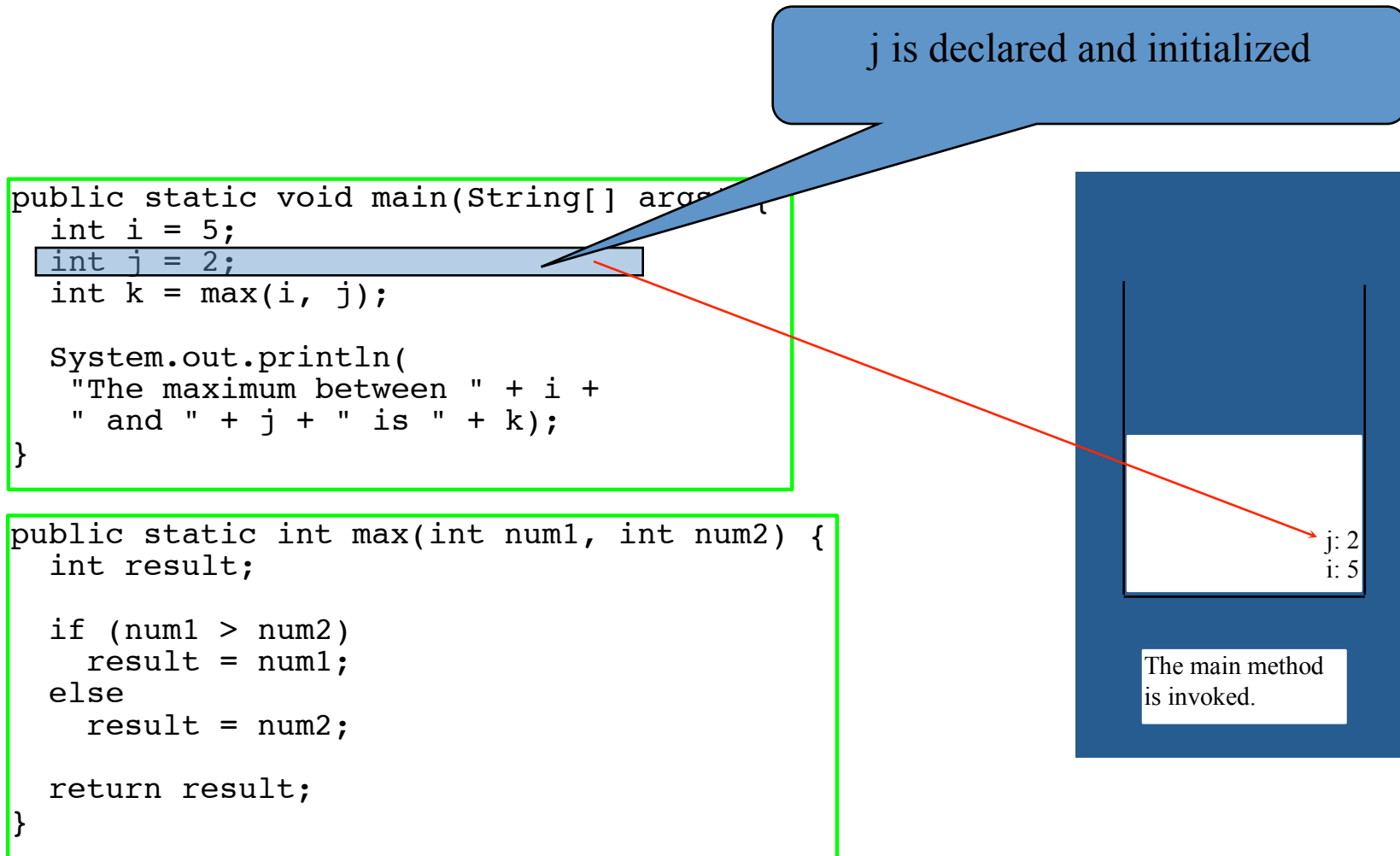
```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

i: 5

The main method
is invoked.

Trace Call Stack



Trace Call Stack

Declare k

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

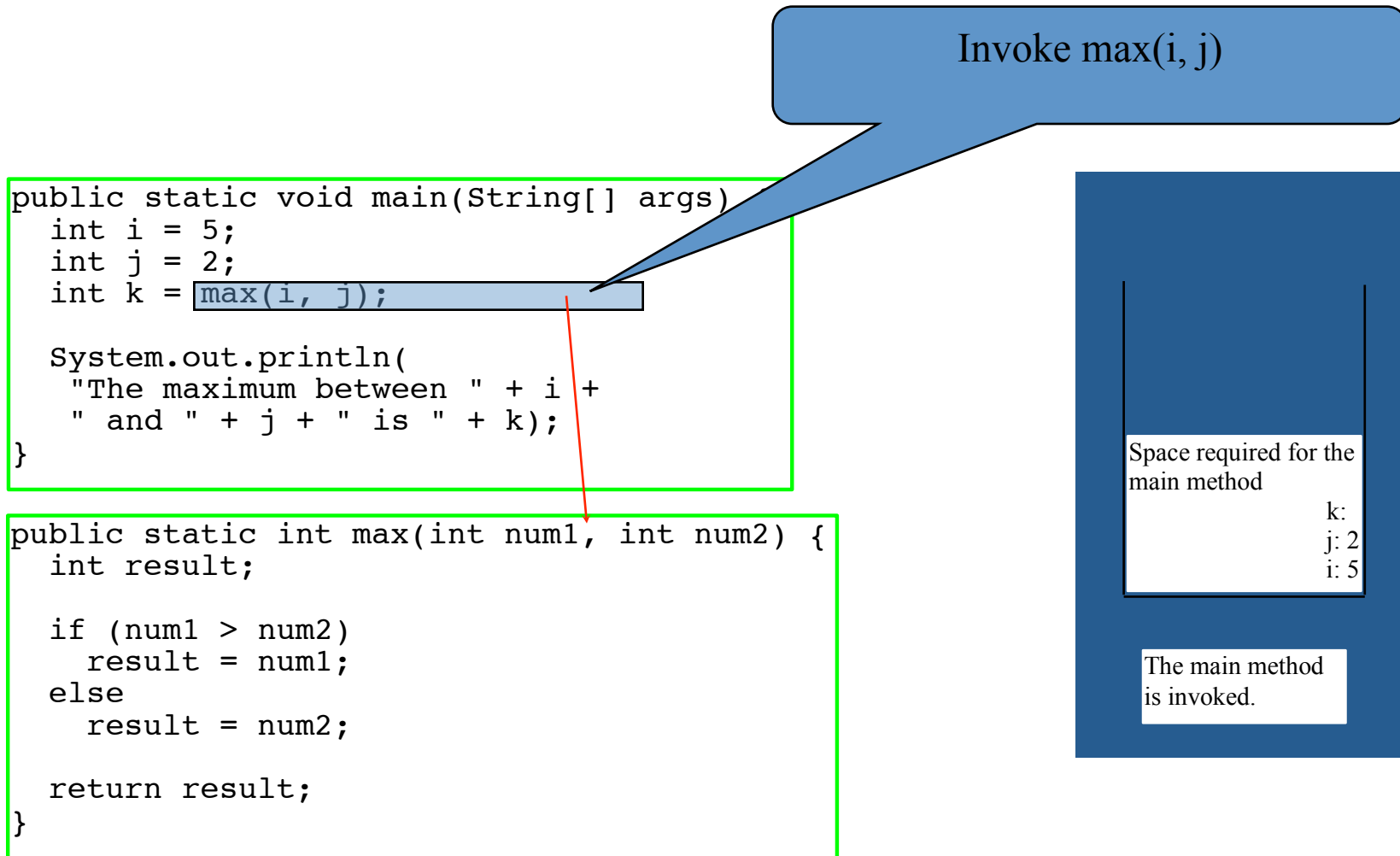
```
public static int max(int num1, int num2) {  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Space required for the
main method

k:
j: 2
i: 5

The main method
is invoked.

Trace Call Stack



Trace Call Stack

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

pass the values of i and j to num1
and num2

num2: 2
num1: 5

Space required for the
main method

k:
j: 2
i: 5

The max method is
invoked.

Trace Call Stack

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

pass the values of i and j to num1
and num2

result:
num2: 2
num1: 5

Space required for the
main method

k:
j: 2
i: 5

The max method is
invoked.

Trace Call Stack

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

(num1 > num2) is true

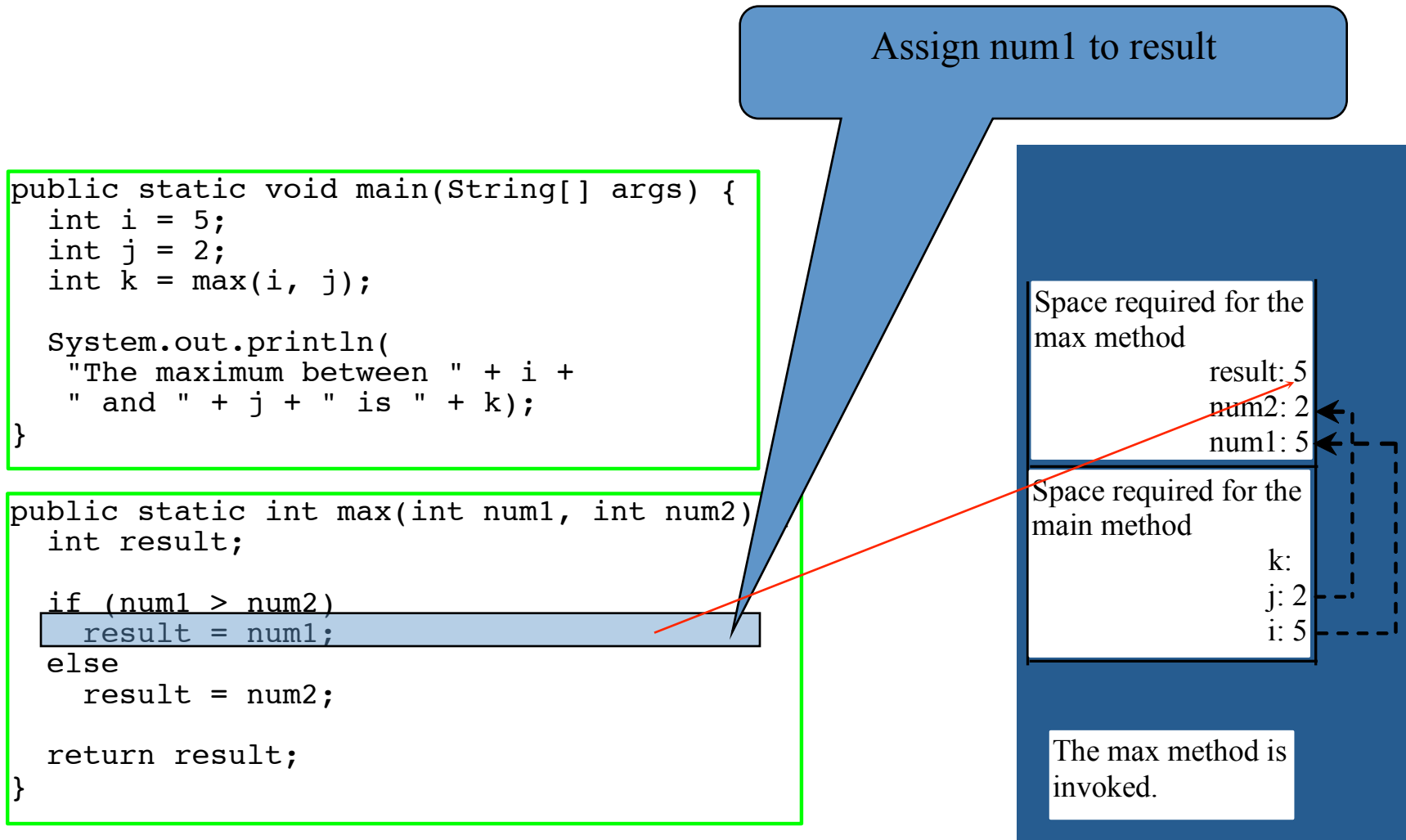
result:
num2: 2
num1: 5

Space required for the
main method

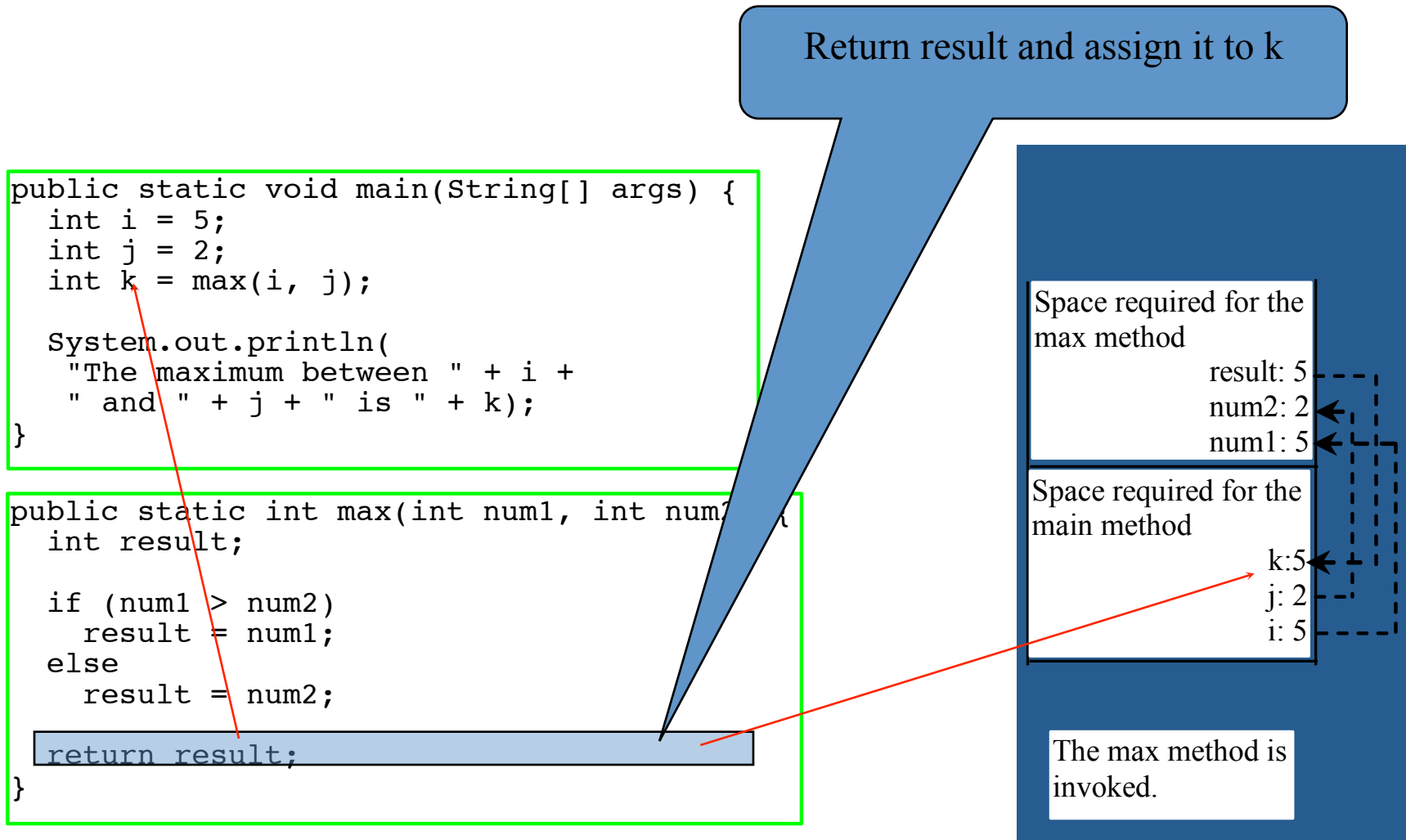
k:
j: 2
i: 5

The max method is
invoked.

Trace Call Stack



Trace Call Stack



Trace Call Stack

Execute print statement

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

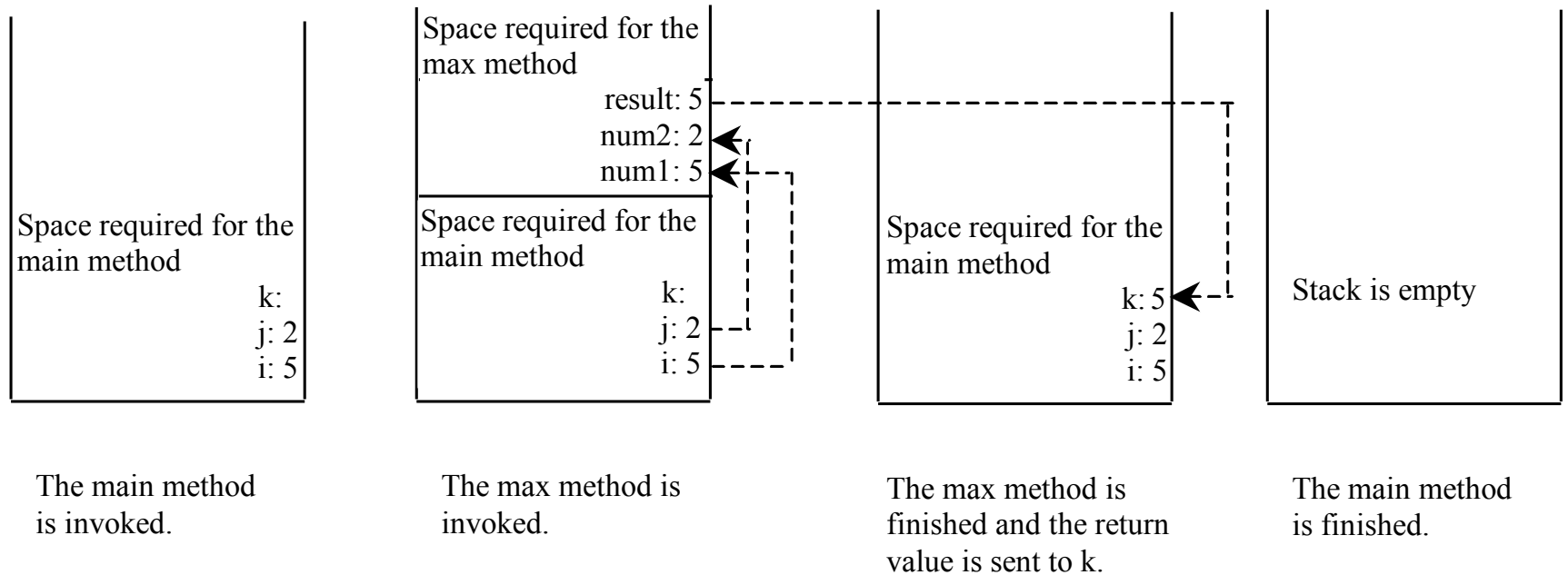
```
public static int max(int num1, int num2) {  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

Space required for the
main method

k:5
j: 2
i: 5

The main method
is invoked.

Call Stacks



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 compilation 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;  
    }  
}
```

Scope of Local Variables

A local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced.

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

Scope of Local Variables, cont.

A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```
public static void method1() {  
    .  
    .  
    for (int i = 1; i < 10; i++) {  
        .  
        .  
        int j;  
        .  
        .  
        .  
    }  
}
```

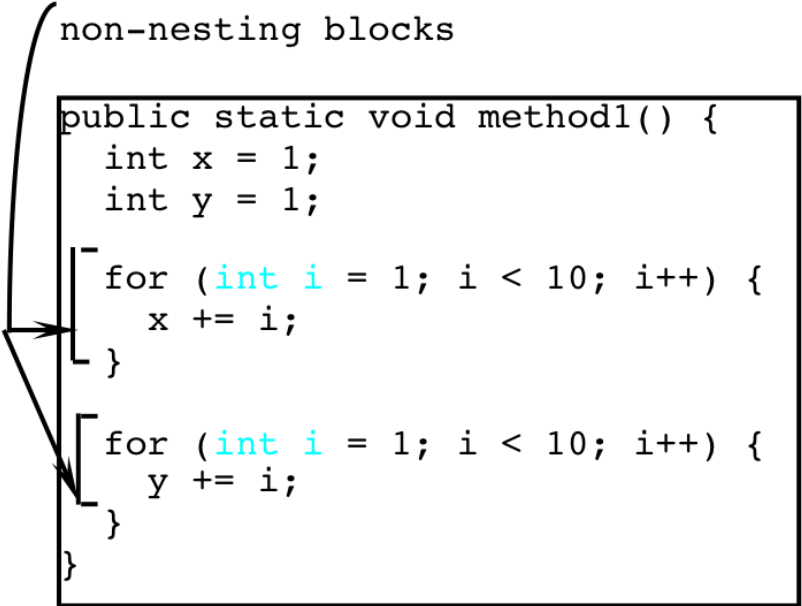
The scope of i →

The scope of j →

Scope of Local Variables, cont.

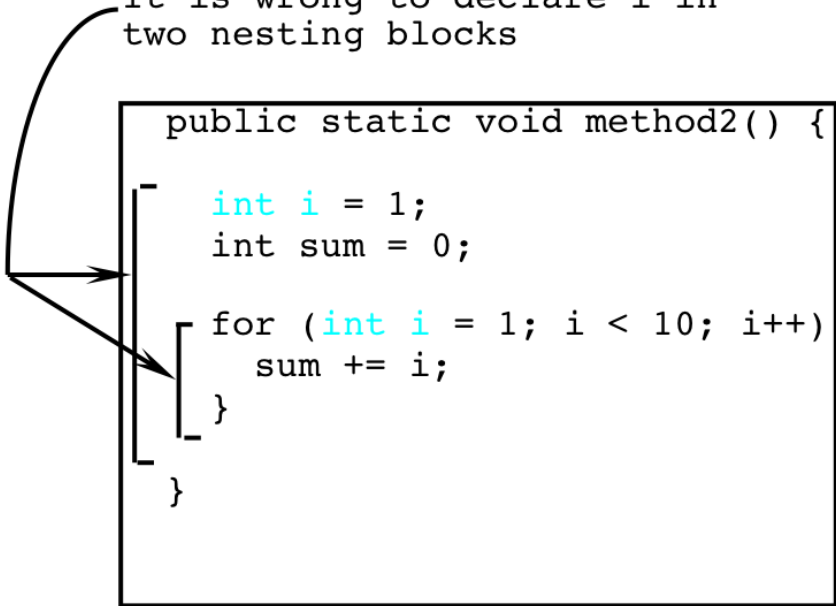
It is fine to declare `i` in two non-nesting blocks

```
public static void method1() {  
    int x = 1;  
    int y = 1;  
    [ for (int i = 1; i < 10; i++) {  
        x += i;  
    }  
    [ for (int i = 1; i < 10; i++) {  
        y += i;  
    }  
}
```



It is wrong to declare `i` in two nesting blocks

```
public static void method2() {  
    [ int i = 1;  
      int sum = 0;  
      [ for (int i = 1; i < 10; i++)  
        sum += i;  
      ]  
    ]  
}
```



The Math Class

- Class constants:
 - PI
 - E
- Class methods:
 - Trigonometric Methods
 - Exponent Methods
 - Rounding Methods
 - min, max, abs, and random Methods

Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Radians

`toRadians(90)`

Examples:

```
Math.sin(0) returns 0.0
```

```
Math.sin(Math.PI / 6)  
returns 0.5
```

```
Math.sin(Math.PI / 2)  
returns 1.0
```

```
Math.cos(0) returns 1.0
```

```
Math.cos(Math.PI / 6)  
returns 0.866
```

```
Math.cos(Math.PI / 2)  
returns 0
```

Exponent Methods

- `exp(double a)`
Returns e raised to the power of a .
- `log(double a)`
Returns the natural logarithm of a .
- `log10(double a)`
Returns the 10-based logarithm of a .
- `pow(double a, double b)`
Returns a raised to the power of b .
- `sqrt(double a)`
Returns the square root of a .

Examples:

```
Math.exp(1) returns 2.71
```

```
Math.log(2.71) returns 1.0
```

```
Math.pow(2, 3) returns 8.0
```

```
Math.pow(3, 2) returns 9.0
```

```
Math.pow(3.5, 2.5) returns  
22.91765
```

```
Math.sqrt(4) returns 2.0
```

```
Math.sqrt(10.5) returns 3.24
```

Rounding Methods

- `double ceil(double x)`
x rounded up to its nearest integer. This integer is returned as a double value.
- `double floor(double x)`
x is rounded down to its nearest integer. This integer is returned as a double value.
- `double rint(double x)`
x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- `int round(float x)`
Return `(int)Math.floor(x+0.5)`.
- `long round(double x)`
Return `(long)Math.floor(x+0.5)`.

The random Method

Generates a random double value greater than or equal to 0.0 and less than 1.0 ($0 \leq \text{Math.random()} < 1.0$).

Examples:

`(int) (Math.random() * 10)` \longrightarrow Returns a random integer between 0 and 9.

`50 + (int) (Math.random() * 50)` \longrightarrow Returns a random integer between 50 and 99.

In general,

`a + Math.random() * b` \longrightarrow Returns a random number between a and a + b, excluding a + b.

Case Study: Generating Random Characters, cont.

As discussed in Section 2.9.4, all numeric operators can be applied to the char operands. The char operand is cast into a number if the other operand is a number or a character. So, the preceding expression can be simplified as follows:

$$'a' + \text{Math.random()} * ('z' - 'a' + 1)$$

So a random lowercase letter is

$$(\text{char})('a' + \text{Math.random()} * ('z' - 'a' + 1))$$

The RandomCharacter Class

```
// RandomCharacter.java: Generate random characters
public class RandomCharacter {
    /** Generate a random character between ch1 and ch2 */
    public static char getRandomCharacter(char ch1, char ch2) {
        return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
    }

    /** Generate a random lowercase letter */
    public static char getRandomLowerCaseLetter() {
        return getRandomCharacter('a', 'z');
    }

    /** Generate a random uppercase letter */
    public static char getRandomUpperCaseLetter() {
        return getRandomCharacter('A', 'Z');
    }

    /** Generate a random digit character */
    public static char getRandomDigitCharacter() {
        return getRandomCharacter('0', '9');
    }

    /** Generate a random character */
    public static char getRandomCharacter() {
        return getRandomCharacter('\u0000', '\uFFFF');
    }
}
```

Package

There are three reasons for using packages:

1. *To avoid naming conflicts.* When you develop reusable classes to be shared by other programmers, naming conflicts often occur. To prevent this, put your classes into packages so that they can be referenced through package names.
2. *To distribute software conveniently.* Packages group related classes so that they can be easily distributed.
3. *To protect classes.* Packages provide protection so that the protected members of the classes are accessible to the classes in the same package, but not to the external classes.

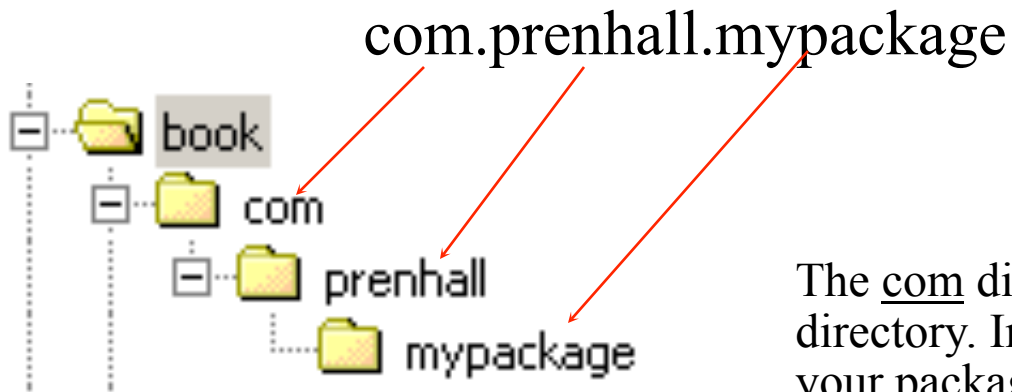
Package-Naming Conventions

Packages are hierarchical, and you can have packages within packages. For example, java.lang.Math indicates that Math is a class in the package lang and that lang is a package in the package java. Levels of nesting can be used to ensure the uniqueness of package names.

Choosing a unique name is important because your package may be used on the Internet by other programs. Java designers recommend that you use your Internet domain name in reverse order as a package prefix. Since Internet domain names are unique, this prevents naming conflicts. Suppose you want to create a package named mypackage on a host machine with the Internet domain name prenhall.com. To follow the naming convention, you would name the entire package com.prenhall.mypackage. By convention, package names are all in lowercase.

Package Directories

Java expects one-to-one mapping of the package name and the file system directory structure. For the package named `com.prenhall.mypackage`, you must create a directory, as shown in the figure. In other words, a package is actually a directory that contains the bytecode of the classes.



The com directory does not have to be the root directory. In order for Java to know where your package is in the file system, you must modify the environment variable classpath so that it points to the directory in which your package resides.

Setting classpath Environment

The com directory does not have to be the root directory. In order for Java to know where your package is in the file system, you must modify the environment variable classpath so that it points to the directory in which your package resides.

Suppose the com directory is under c:\book. The following line adds c:\book into the classpath:

```
classpath=.;c:\book;
```

The period (.) indicating the current directory is always in classpath. The directory c:\book is in classpath so that you can use the package com.prenhall.mypackage in the program.



Putting Classes into Packages

Every class in Java belongs to a package. The class is added to the package when it is compiled. All the classes that you have used so far in this book were placed in the current directory (a default package) when the Java source programs were compiled. To put a class in a specific package, you need to add the following line as the first noncomment and nonblank statement in the program:

```
package packagename;
```


Listing 5.8 Putting Classes into Packages

Problem

This example creates a class named Format and places it in the package com.prenhall.mypackage. The Format class contains the format(number, numOfDecimalDigits) method that returns a new number with the specified number of digits after the decimal point. For example, format(10.3422345, 2) returns 10.34, and format(-0.343434, 3) returns -0.343.

Solution

1. Create Format.java as follows and save it into c:\book\com\prenhall\mypackage.

```
// Format.java: Format number.
package com.prenhall.mypackage;

public class Format {
    public static double format(
        double number, int numOfDecimalDigits) {
        return Math.round(number * Math.pow(10, numOfDecimalDigits)) /
            Math.pow(10, numOfDecimalDigits);
    }
}
```

2. Compile Format.java. Make sure Format.class is in c:\book\com\prenhall\mypackage.

Using Classes from Packages

There are two ways to use classes from a package.

- One way is to use the fully qualified name of the class. For example, the fully qualified name for `JOptionPane` is `javax.swing.JOptionPane`. For `Format` in the preceding example, it is `com.prenhall.mypackage.Format`. This is convenient if the class is used a few times in the program.
- The other way is to use the import statement. For example, to import all the classes in the `javax.swing` package, you can use

```
import javax.swing.*;
```

An import that uses a `*` is called an import on demand declaration. You can also import a specific class. For example, this statement imports `javax.swing.JOptionPane`:

```
import javax.swing.JOptionPane;
```

The information for the classes in an imported package is not read in at compile time or runtime unless the class is used in the program. The import statement simply tells the compiler where to locate the classes.

Listing 5.9 Using Packages

Problem

This example shows a program that uses the Format class in the com.prenhall.mypackage.mypackage package.

Solution

1. Create TestFormatClass.java as follows and save it into c:\book. The following code gives the solution to the problem.

```
// TestFormatClass.java: Demonstrate using the Format class
import com.prenhall.mypackage.Format;

public class TestFormatClass {
    /** Main method */
    public static void main(String[] args) {
        System.out.println(Format.format(10.3422345, 2));
        System.out.println(Format.format(-0.343434, 3));
    }
}
```