COSC 111
Computer Programming I

Chapter 6 Methods

Dr. Abdallah Mohamed

Introduction

Today, we will learn how to properly organize our code. That is,

- we are going to discuss the **code organization**,
- NOT the program logic.

We will learn writing *modular code*

- **Modular programming** is focuses on *designing* the software with an emphasis on separating the functionality into **modules**, each containing code that executes only one aspect of the desired functionality.

Problem decomposition involves breaking down a large problem into subproblems which are easier to solve. Dividing problems into subproblems is called *divide and conquer*

Introduction

So far, we have been writing all our code in the main method. It is not advised to write code in a *monolithic* fashion.

Example Problem: Find the sum of integers from 1 to 10.

```
Public static void main(String[] args){
    int sum = 0;
    for (int i = 1; i <= 10; i++)
      sum += i;
    System.out.println("Sum from 1 to 10 is " + sum);
}
```

Opening Problem

Problem: Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.

```
Public static void main(String[] args){
    int sum = 0;
    for (int i = 1; i <= 10; i++)
        sum += i;
    System.out.println("Sum from 1 to 10 is " + sum);


    sum = 0;
    for (int i = 20; i <= 30; i++)
        sum += i;
    System.out.println("Sum from 20 to 30 is " + sum);


    sum = 0;
    for (int i = 35; i <= 45; i++)
        sum += i;
    System.out.println("Sum from 35 to 45 is " + sum);
}
```

Repeated code!

Solution – Use methods

Methods can be used to:

- **reduce redundant coding** and enable **code reuse**
- **modularize code** and improve the quality of the program.

```java
public static int sum(int i1, int i2) {
    int sum = 0;
    for (int i = i1; i <= i2; i++)
        sum += i;
    return sum;
}
```

Write it once,

(re)use it many times!

```java
public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1,10));
    System.out.println("Sum from 20 to 30 is "+ sum(20,30));
    System.out.println("Sum from 35 to 45 is "+ sum(35,45));
}
```

Method Structure

What is a method?

A method is a collection of statements that are grouped together to perform a specific **action**.

A method must have a name. Whenever we want to perform the method's action, we need to call (invoke) the method by its name.

When calling a method:

- It always performs an action
- *May* receive *input* data
  - Arguments
- *May* return *output* data
  - Returned value

Returned value

Arguments
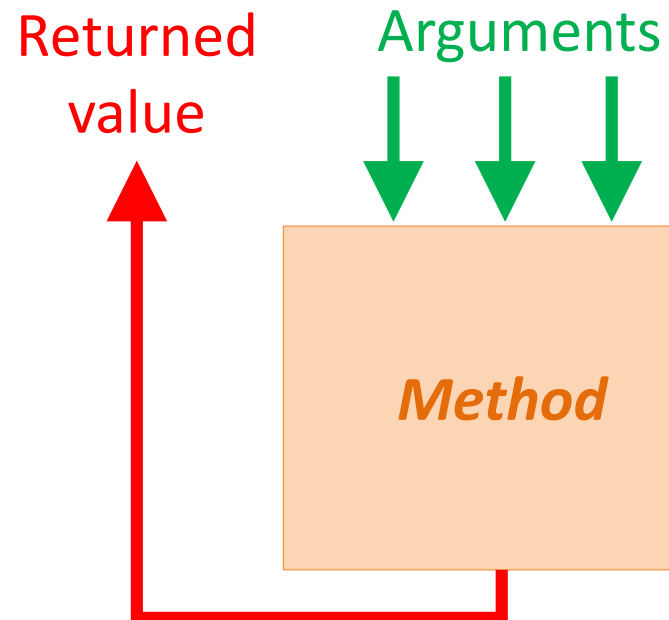
Method

Defining Methods

A method is a collection of statements that are grouped together to perform an operation. The syntax is as follows:

```
modifier returnType methodName(list of parameters) {
    // Method body;
}
```

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

    int result;

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

    return result;
}
```

Define a method

```
...
int z = max(3,4);
...
```

Invoke a method

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 8

Defining Methods

The **method header** specifies the *modifiers*, *return value type*, *method name*, and *parameters* of the method.

- A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform desired operations without returning a value. In this case, the returnValueType is the keyword void.

Modifiers

Return type

Method name

Method Header

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

Method Body

```
    int result;

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

    return result;
}
```
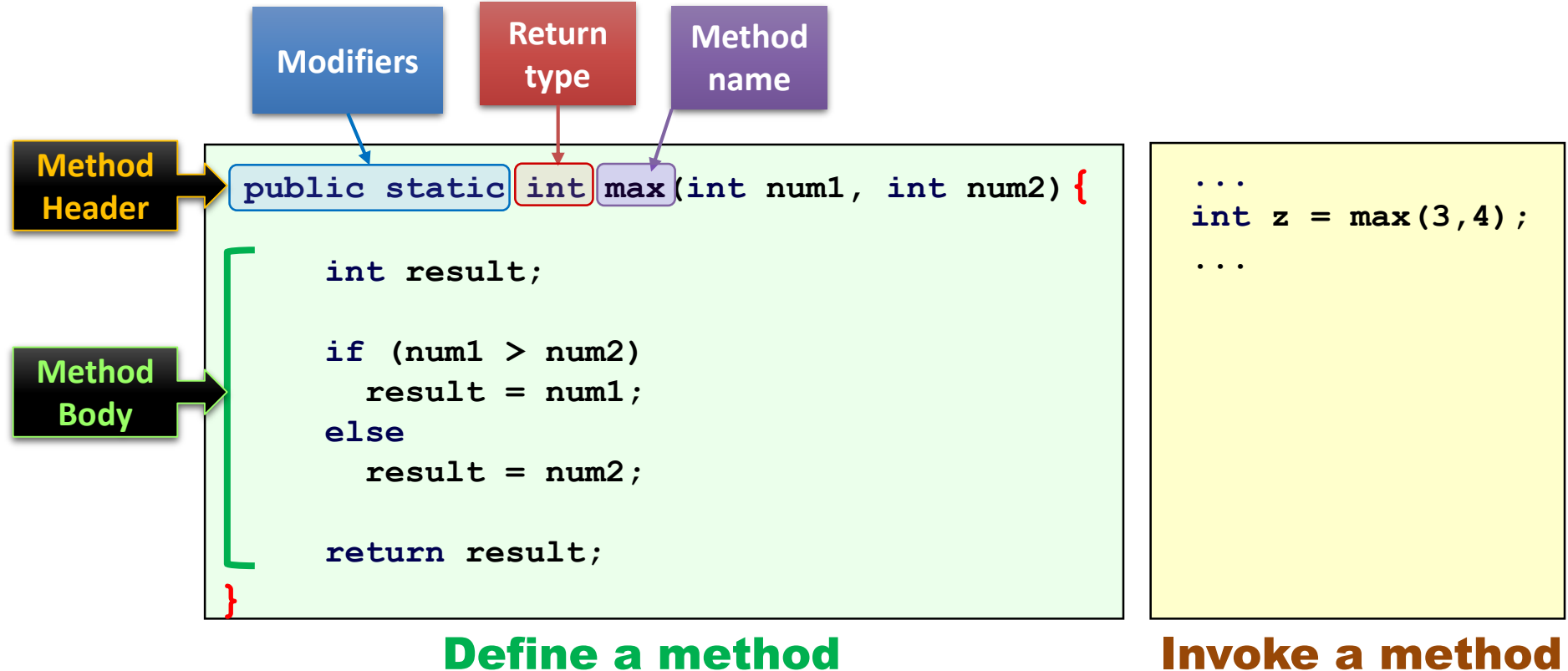
Define a method

```
...
int z = max(3,4);
...
```
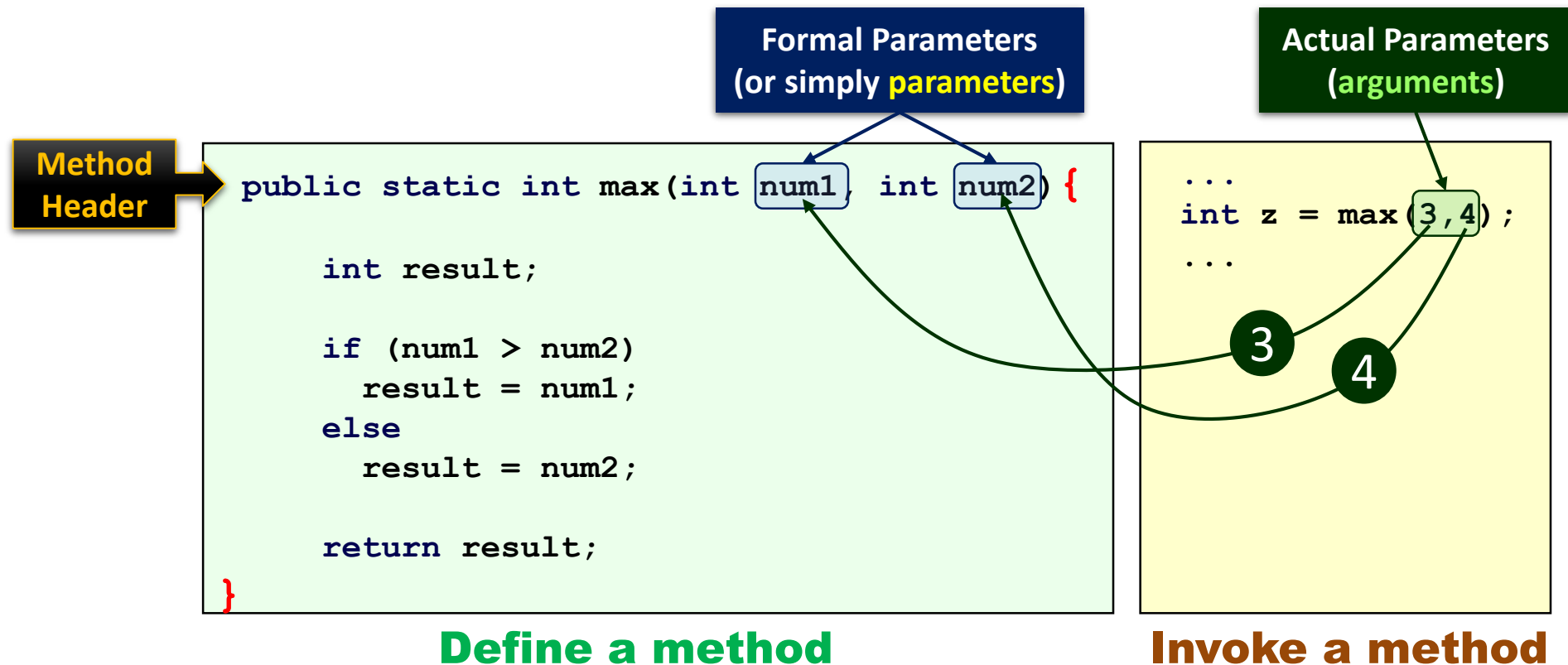
Invoke a method

Parameters and Arguments

The variables defined in the method header are known as ***formal parameters*** or simply ***parameters***.

A parameter is like a placeholder: when a method is invoked, you pass a value to the parameter. This value is referred to as an ***actual parameter, or simply arguments.***
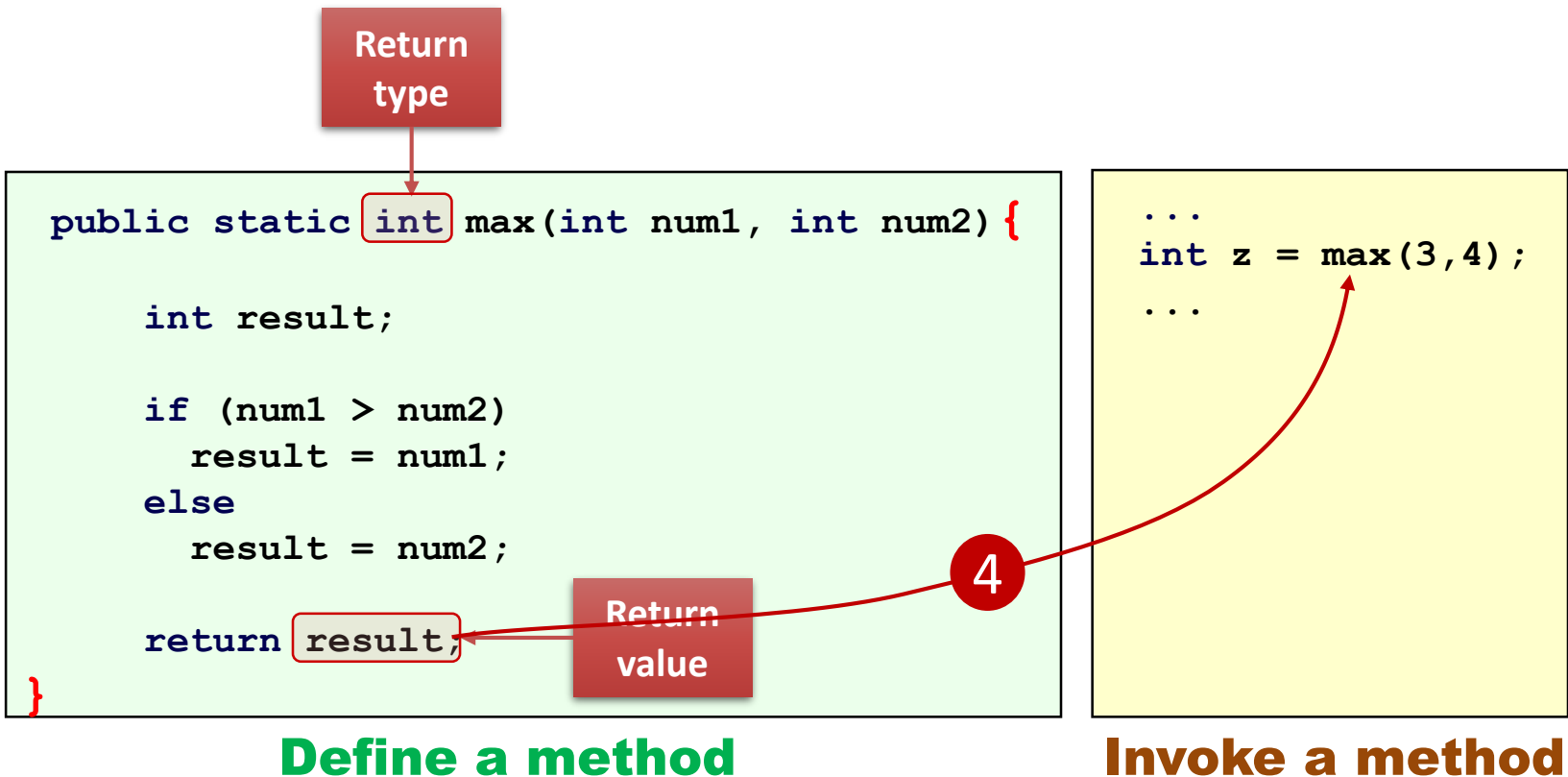
- you need to provide the arguments in the same order as their respective parameters in the method signature.

Formal Parameters (or simply parameters)

Actual Parameters (arguments)

Method Header

```java
public static int max(int num1, int num2) {

    int result;

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

    return result;
}
```

```java
...
int z = max(3,4);
...
```

3
4

Define a method

Invoke a method

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 10

Return Value Type

A method **<u>may</u>** return a value. The returnType is the data type of the value the method returns. If the method does not return a value, the returnType is the keyword **void**.

- For example, the returnValueType in the main method is void.

Return type

```java
public static int max(int num1, int num2) {

    int result;

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

    return result;
}
```

Return value

```java
...
int z = max(3,4);
...
```

4

Define a method

Invoke a method

Method Signature

The method name and the parameter list together constitute the *method signature.*

Method Signature

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

    int result;

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

    return result;
}
```

Define a method

```
...
int z = max(3,4);
...
```

Invoke a method

Calling Methods

Calling Methods

To execute the method, you have to *call* or *invoke* it.

Two ways to call a method:

- **If a method returns a value**, a call to the method **is usually treated as a value**.
  - Example1:

    int x = max(3, 4);

    calls **max(3, 4)** and assigns the result of the method to the variable **x**.
  - Example2:

    System.out.println(max(3, 4));

    prints the return value of the method call **max(3**, **4)**.

- **If a method returns void**, a call to the method must be **a statement**.
 - For example, the method **println** returns **void**. The following call is a statement:

    System.out.println(**"Welcome to Java!"**);

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 14

Calling Methods, cont.

When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when

- its **return** statement is executed, or
- when its method-ending **closing brace** is reached.

Example: calling a method to return the largest of two int values
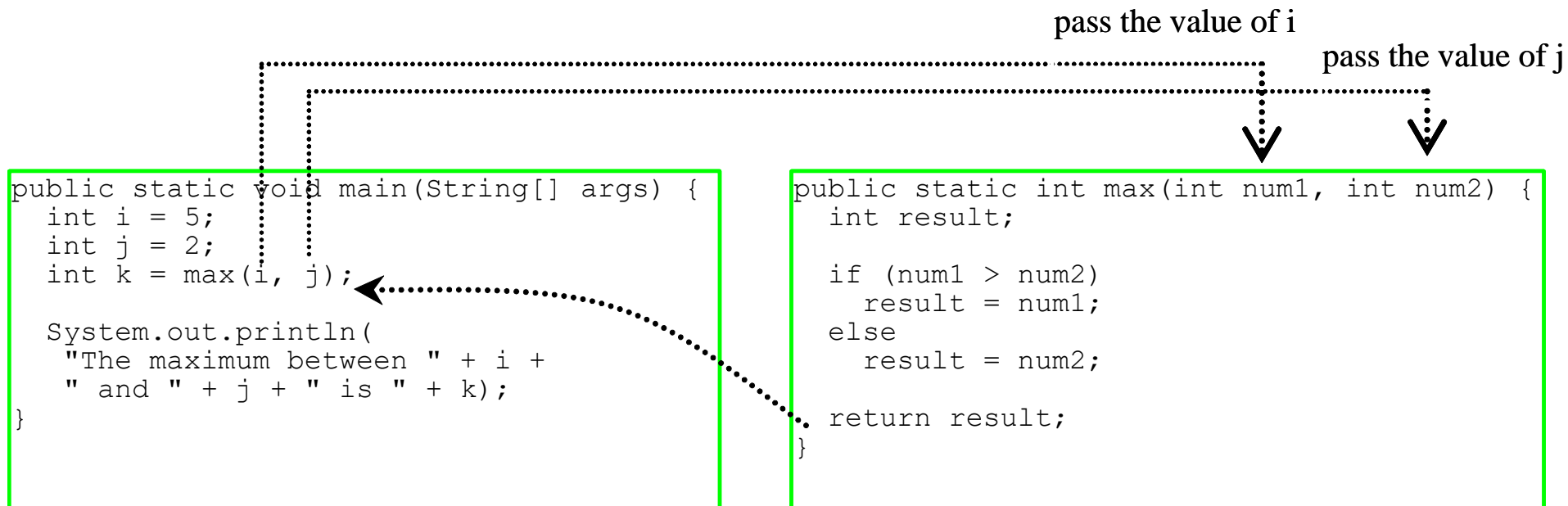
pass the value of i

pass the value of j

```
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 is now 5

```
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;
}
```

j is now 2

```
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;
}
```

Trace Method Invocation

invoke max(i, j)

```
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;
}
```

invoke max(i, j)
Pass the value of i to num1
Pass the value of j to num2

```
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;
}
```

19

Animation

declare variable result

```
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;
}
```

20

Trace Method Invocation

(num1 > num2) is true since num1 is 5 and num2 is 2

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

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

  return result;
}
```

21

result is now 5

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

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

  return result;
}
```

return result, which is 5

```
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) {
      result;

    num1 > num2)
    esult = num1;
   se
    result = num2;

  return result;
}
```

Animation

> return max(i, j) and assign the return value to 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;
}
```

Trace Method Invocation

Execute the print statement

```
public static void main(String     s) {
  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;
}
```

return vs. void

CAUTION

A **return statement** is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```java
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```
(a)

Should be →

```java
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```
(b)

To fix this problem, delete if (n < 0) in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

Methods returning values vs. void methods

Void method

```java
public class TestVoidMethod {
  public static void main(String[] args) {
        System.out.print("Grade is ");
        printGrade(78.5);

        System.out.print("Grade is ");
        printGrade(59.5);
  }
  public static void printGrade(double score){
        if (score >= 90.0) {
            System.out.println('A');
        } else if (score >= 80.0) {
            System.out.println('B');
        } else if (score >= 70.0) {
            System.out.println('C');
        } else if (score >= 60.0) {
            System.out.println('D');
        } else {
            System.out.println('F');
        }
  }
}
```

Method that returns data

```java
public class TestReturnGradeMethod {
  public static void main(String[] args) {
    System.out.println("Grade is "+getGrade(78.5));
    System.out.println("Grade is "+getGrade(59.5));
  }
  public static char getGrade(double score) {
    if (score >= 90.0)
        return 'A';
    else if (score >= 80.0)
        return 'B';
    else if (score >= 70.0)
        return 'C';
    else if (score >= 60.0)
        return 'D';
    else
        return 'F';
  }
}
```

return statement in void methods

return is **not needed for a void** method.

BUT it can be used for terminating the method.

For example

```java
public static void printGrade(double score) {
        if (score < 0 || score > 100) {
                System.out.println("Invalid score");
                return;
        }
        //other statements
}
```

Clicker Question

```
public static void nPrintln(String msg, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(msg);
}
```

What is the output if we invoke the method using

nPrintln(5, "ABC");

A. Error

B. Nothing

C. Print ABC five times

D. Print ABC six times

```
public static void nPrintln(String msg, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(msg);
}
```

What is the output if we invoke the method using

nPrintln();

A. Error

B. Nothing

C. Print ABC five times

D. Print ABC six times

```
public static void nPrintln(String msg, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(msg);
    return;
}
```

What is the output if we invoke the method using

nPrintln("ABC", 5);

A. Error

B. Nothing

C. Print ABC five times

D. Print ABC six times

Practice

Identify and correct the errors in the following program:

```java
public class Ex {
    public static void main(String[] args) {
        method1(3);
    }

    public static method1(int n, m) {
        n += m;
        method2(5.2);
    }

    public static int method2(int n) {
        if (n > 0)
            return 1;
        else if (n == 0)
            return 0;
        else if (n < 0)
            return -1;
    }
}
```

Wrong **number** arguments

No return type

m has no type

Wrong **type** of arguments

must have a return statement that the compiler knows it will run for sure.

Pass by Value

Pass by Value

When you invoke a method with an argument, the value of the argument is passed to the parameter. This is referred to as *pass-by-value.*

```java
public class Increment {
    public static void main(String[] args) {
        int x = 1;
        System.out.println("Before the call, x is " + x);
        increment(x);
        System.out.println("after the call, x is " + x);
    }

    public static void increment(int n) {
        n++;
        System.out.println("n inside the method is " + n);
    }
}
```

Output

```
Before the call, x is 1
n inside the method is 2
after the call, x is 1
```

Another example on pass by value:

```java
public class TestPassByValue {
    /** Main method */
    public static void main(String[] args) {
        int num1 = 1;
        int num2 = 2;
        System.out.println("Before invoking the swap method, num1 is " + num1 + " and num2 is " + num2);
        swap(num1, num2);
        System.out.println("After invoking the swap method, num1 is " + num1 + " and num2 is " + num2);
    }

    /** Swap two variables */
    public static void swap(int n1, int n2) {
        System.out.println("\tInside the swap method");
        System.out.println("\t\tBefore swapping, n1 is " + n1 + " and n2 is " + n2);
        // Swap n1 with n2
        int temp = n1;
        n1 = n2;
        n2 = temp;
        System.out.println("\t\tAfter swapping, n1 is " + n1 + " and n2 is " + n2);
    }
}
```

Output:

```
Before invoking the swap method, num1 is 1 and num2 is 2
        Inside the swap method
                Before swapping, n1 is 1 and n2 is 2
                After swapping, n1 is 2 and n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```

Clicker Question

What is the output?

```
public static void main(String[] args){
    int x = 0;
    set10(x++);
    System.out.println(x);
}

public static void set10(int n) {
    n = 10;
}
```

A. 0

B. 1

C. 10

D. 11

Overloading Methods

Overloading Methods

You can define several methods with **the same name** as long as their **signatures are different**. Java always calls the *most specific method*.

```java
public class TestMethodOverloading {
    /** Main method */
    public static void main(String[] args) {
        // Invoke the max method with int parameters
        System.out.println("The maximum of 3 and 4 is " + max(3, 4));
        // Invoke the max method with the double parameters
        System.out.println("The maximum of 3.0 and 5.4 is " + max(3.0, 5.4));
        // Invoke the max method with three double parameters
        System.out.println("The maximum of 3.0, 5.4, and 10.14 is " + max(3.0, 5.4, 10.14));
    }
    /** Method1: Return the max of two int values */
    public static int max(int num1, int num2) {
        if (num1 > num2) return num1;
        else return num2;
    }
    /** Method2: Find the max of two double values */
    public static double max(double num1, double num2) {
        if (num1 > num2) return num1;
        else return num2;
    }
    /** Method3: Return the max of three double values */
    public static double max(double num1, double num2, double num3) {
        return max(max(num1, num2), num3);
    }
}
```

Output

```
The maximum of 3 and 4 is 4
The maximum of 3.0 and 5.4 is 5.4
The maximum of 3.0, 5.4, and 10.14 is 10.14
```

Overloading Methods, cont.

Ambiguous Invocation *occurs* *when* there are **two or more possible matches** for the invocation of a method, but the compiler cannot determine the best match. Ambiguous invocation causes a compile error.

For example:

```java
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;
    }
}
```

Clicker Question

Given two method headers,

```
public static double m(double x,double y) //first method

public static double m(int x,double y)    //second method
```

Which one is invoked by the statement:

```
double z = m(4, 5.2);
```

A. First method

B. Second method

C. Error

Given two method headers,

```
public static double m(double x, double y) //first method

public static double m(int x, double y)    //second method
```

Which one is invoked by the statement:

```
double z = m(4, 5);
```

A. First method

B. Second method

C. Error

Clicker Question

Given two method headers,

```
public static double m(double x,double y) //first method

public static double m(int x,double y)    //second method
```

Which one is invoked by the statement:

```
double z = m(4.0, 5);
```

A. First method

B. Second method

C. Error

Clicker Question

Given two method headers,

```
public static double m(double x, int y)  //first method

public static double m(int x, double y)  //second method
```

Which one is invoked by the statement:

```
double z = m(4, 5);
```

A. First method

B. Second method

C. Error

What is the output

```
public static void maim(String[] args){ m('a'); }

public static void m(int x)   { System.out.println("A:" + x); }
public static void m(double x){ System.out.println("B:" + x); }

public static void m(String x){ System.out.println("C:" + x); }
```

A. A:97

B. B:97.0

C. C:a

D. Error

The Scope of Variables

The Scope of Variables

The scope of a variable is the part of the program where the variable can be referenced.

- **A local variable**: a variable defined inside a method.
  - The scope of a local variable **starts from its declaration** and continues to the **end of the block that contains the variable**.
    - A **method parameter** is a local variable to that method. The scope of a method parameter covers the entire method.
    - A variable declared in the initial-action part of a **for**-loop header has its scope in the entire loop. However, a variable declared inside a **for**-loop body has its scope limited in the loop body from its declaration 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.

You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

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

It is wrong to declare i in two 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;
  }
}
```

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }

}
```

Invalid code!

Practice

Write three methods that count the number of letters, digits, and other characters in a string using the following header:

public static int countLetters(String s)

public static int countDigits(String s)

public static int countNonLetterOrDigit(String s)

Write a test program that prompts the user to enter a string and displays the number of letters, digits, and other characters in the string.

Clicker Question

What is the output?

```
for (int i = 0; i <= 10; i++);
    System.out.print(i);
```

A. nothing

B. error

C. 11
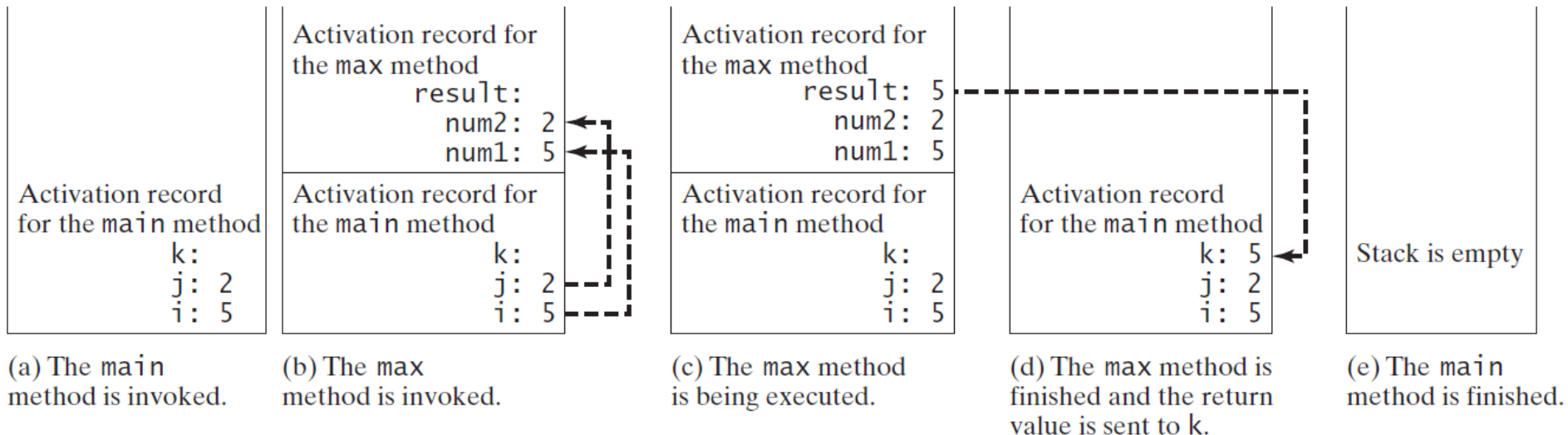
D. The numbers 0,1,2,3,…,9

E. The numbers 0,1,2,3,…,10

Call Stacks

Call Stacks

Each time a method is invoked, the system creates an *activation record* that **stores parameters and variables** for the method and places the activation record in an **area of memory** known as a call *stack*.

- **When a method calls another method, the caller's activation record is kept intact, and a new activation record is created for the new method called.**

When a method finishes its work and returns to its caller, its activation record is removed from the call stack.

(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.

Trace Call Stacks

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;
}
```

Activation record for
the main method

i: 5

The main method
is invoked.

animation

> j 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;
}
```

Activation record for the main method

j: 2
i: 5

The main method is invoked.

54

Trace Call Stacks

Declare k

```java
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);
}
```

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

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

  return result;
}
```

Activation record for
the main method

k:
j: 2
i: 5

The main method
is invoked.

Trace Call Stacks

Invoke max(i, j)

```
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;
}
```

Activation record for
the main method

k:
j: 2
i: 5

The main method
is invoked.

Trace Call Stacks

pass the values of i and j to num1 and num2

```
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;
}
```

Activation record
for the max method

num2: 2
num1: 5

Activation record for
the main method

k:
j: 2
i: 5

The max method is
invoked.

57

Trace Call Stacks

Declare result

```
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;
}
```

Activation record
for the max method

result:
num2: 2
num1: 5

Activation record for
the main method

k:
j: 2
i: 5

The max method is
invoked.

animation

(num1 > num2) is true

```
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;
}
```
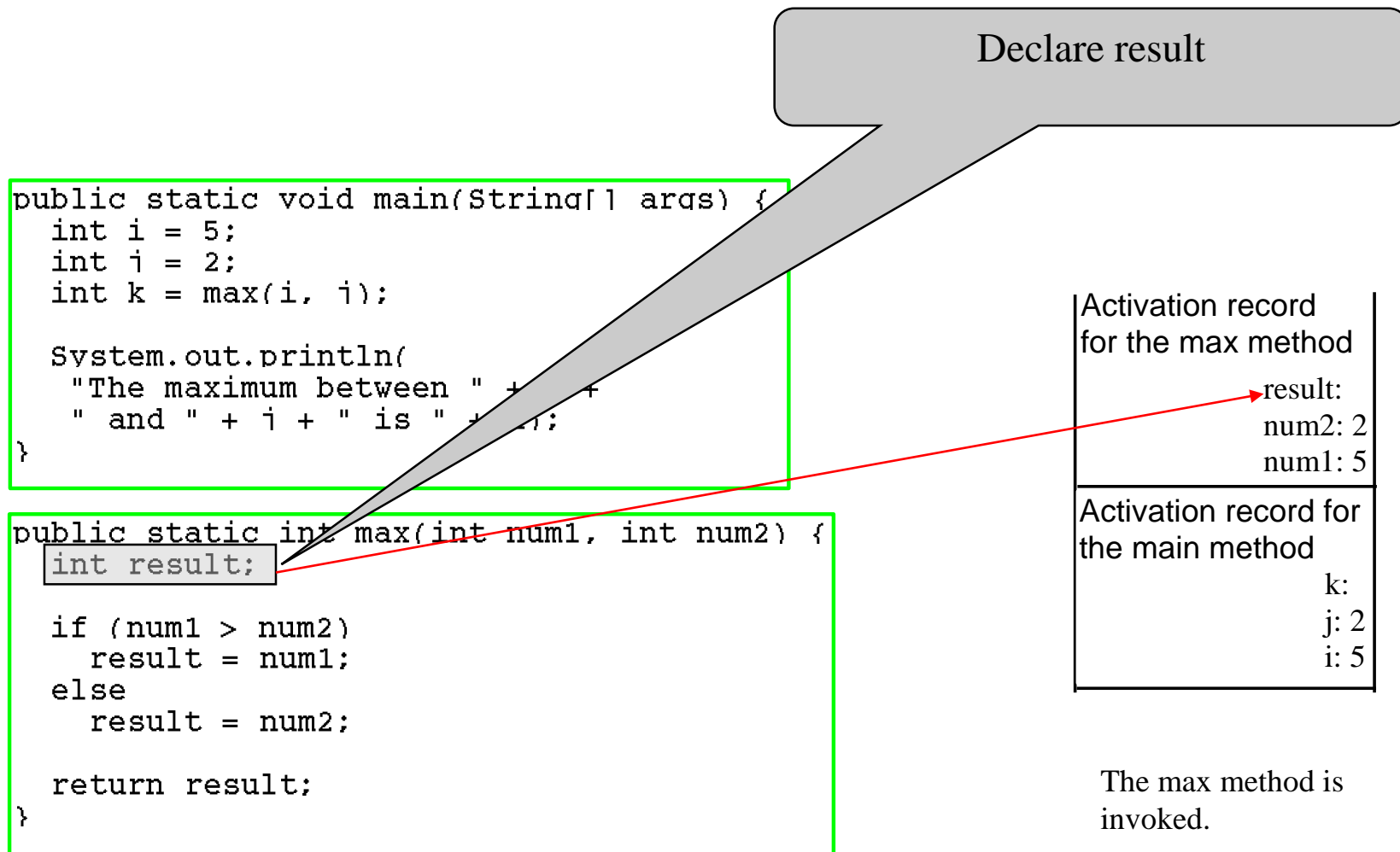
Activation record
for the max method

result:
num2: 2
num1: 5

Activation record for
the main method

k:
j: 2
i: 5

The max method is
invoked.

59

animation

Assign num1 to result

```
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;
}
```

Activation record for the max method
result: 5
num2: 2
num1: 5

Activation record for the main method
k:
j: 2
i: 5

The max method is invoked.

Trace Call Stacks

Return result and assign it to 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;
}
```

Activation record
for the max method

result: 5
num2: 2
num1: 5

Activation record for
the main method

k:5
j: 2
i: 5

The max method is
invoked.

animation
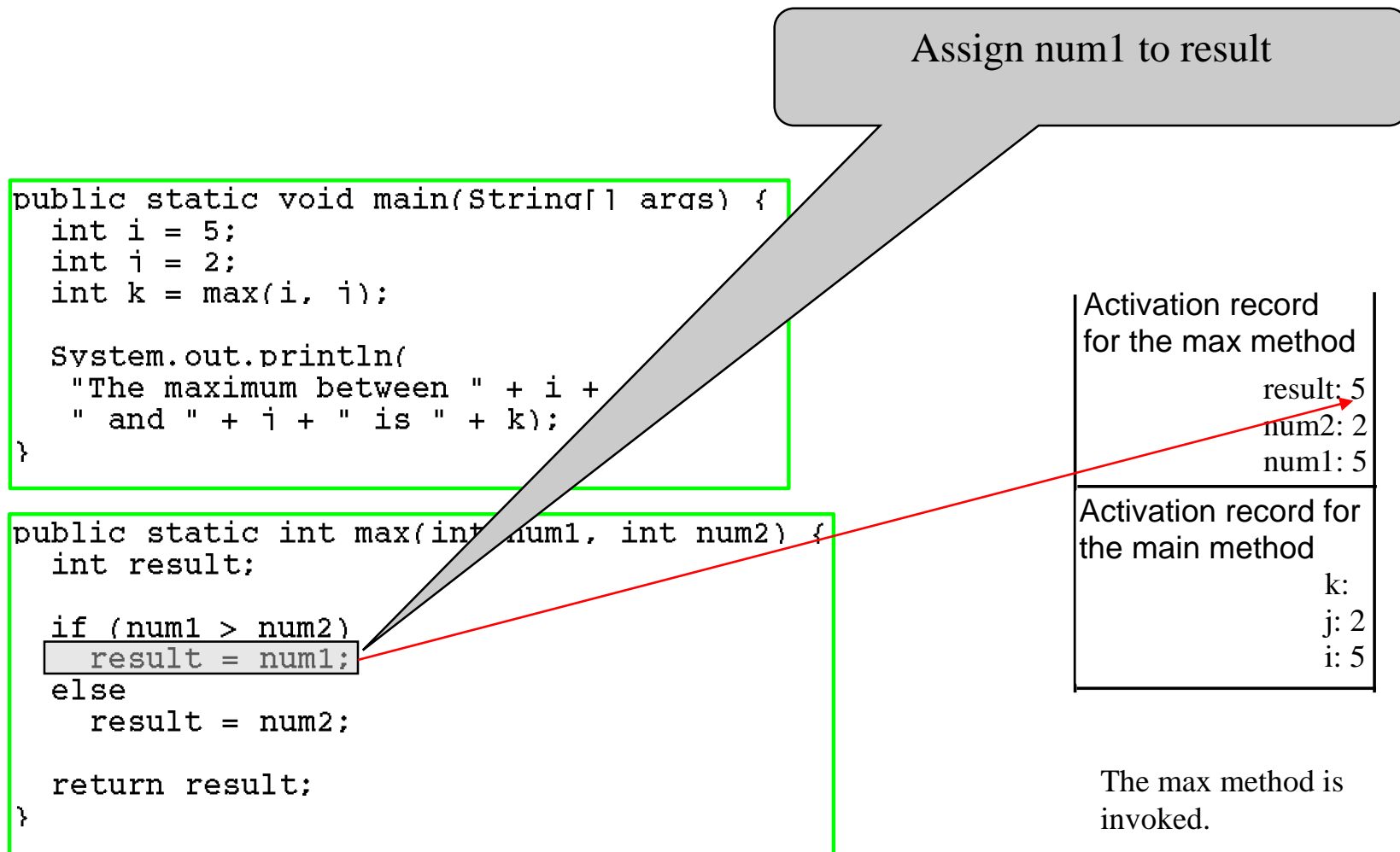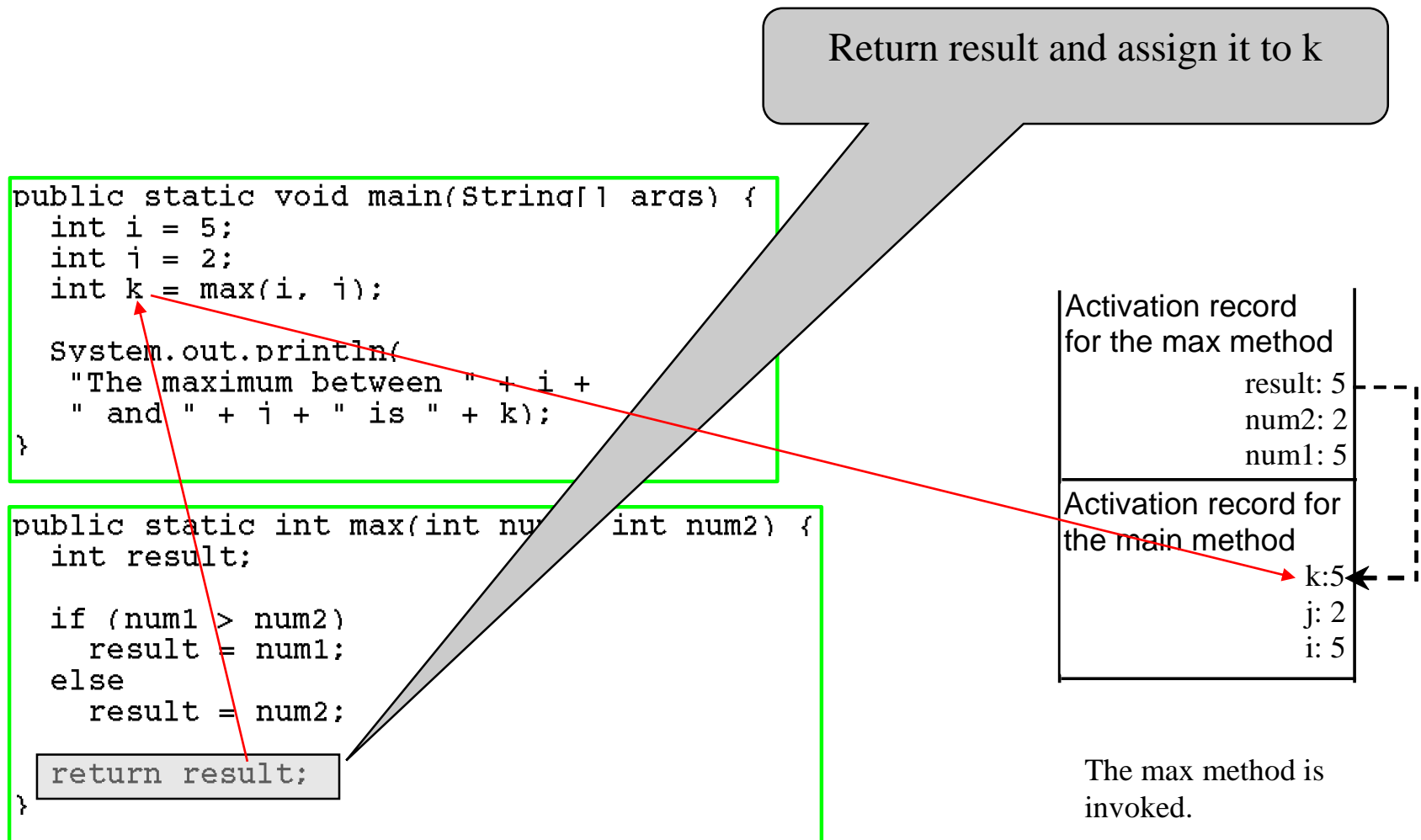
Execute print statement

```
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;
}
```

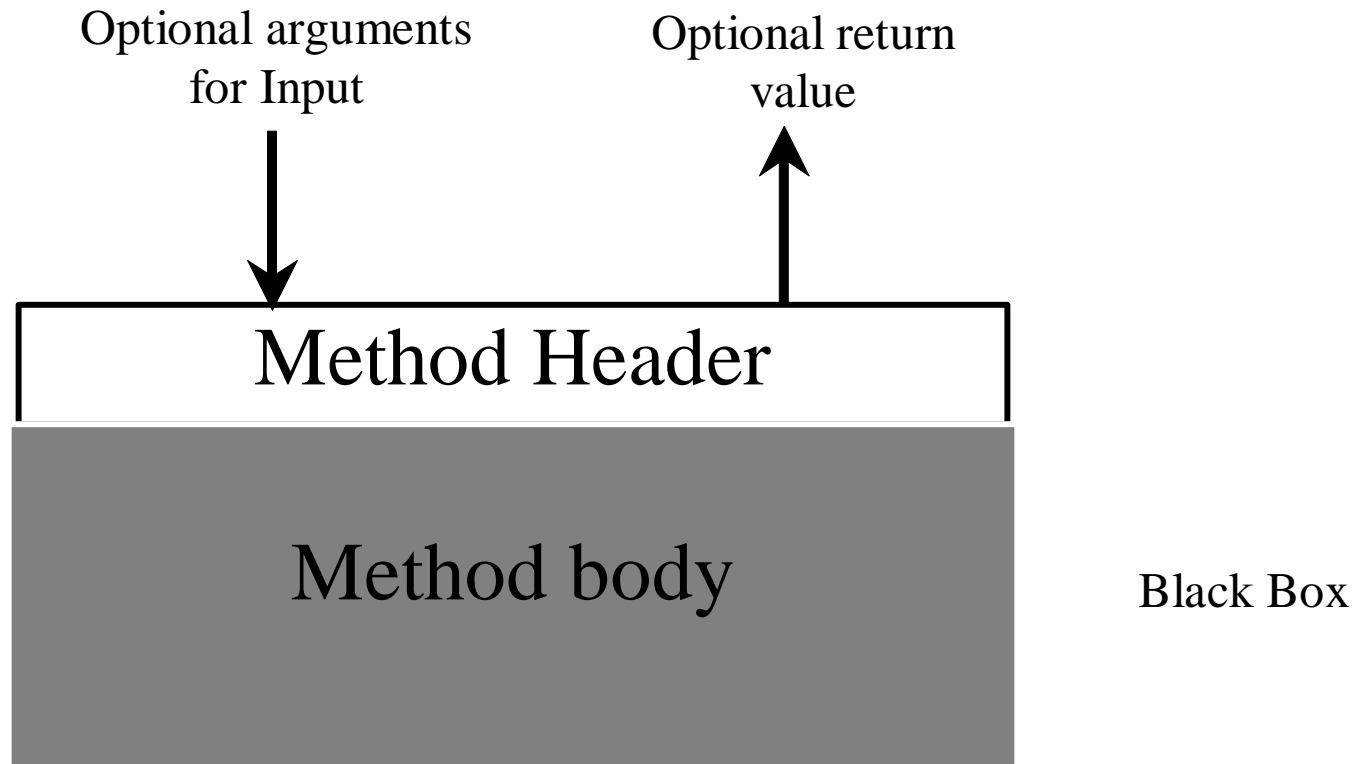Activation record for
the main method

k:5
j: 2
i: 5

The main method
is invoked.

62

Stepwise Refinement

Method Abstraction

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

- The details of the implementation are **encapsulated** in the method and **hidden** from whoever invokes the method.

Optional arguments for Input

Optional return value

Method Header

Method body

Black Box

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 64

Stepwise Refinement

The concept of method abstraction can be applied to the process of developing programs.

When writing a large program, you can use the "***divide and conquer***" strategy, also known as ***stepwise refinement***, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

Stepwise Refinement – Example

Example: Suppose you write a program that displays the calendar for a given month of the year. The program prompts the user to enter the year and the month, then displays the entire calendar for the month, as shown in the following sample run.

```
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
            April 2009
-------------------------------
Sun Mon Tue Wed Thu Fri Sat
                  1   2   3   4
  5   6   7   8   9  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30
```

How would you get started on such a program?

- We shall use an approach called Top-Down Design.
  - the problem is first broken into subproblems, each of which is further broken into subproblems, and so on.

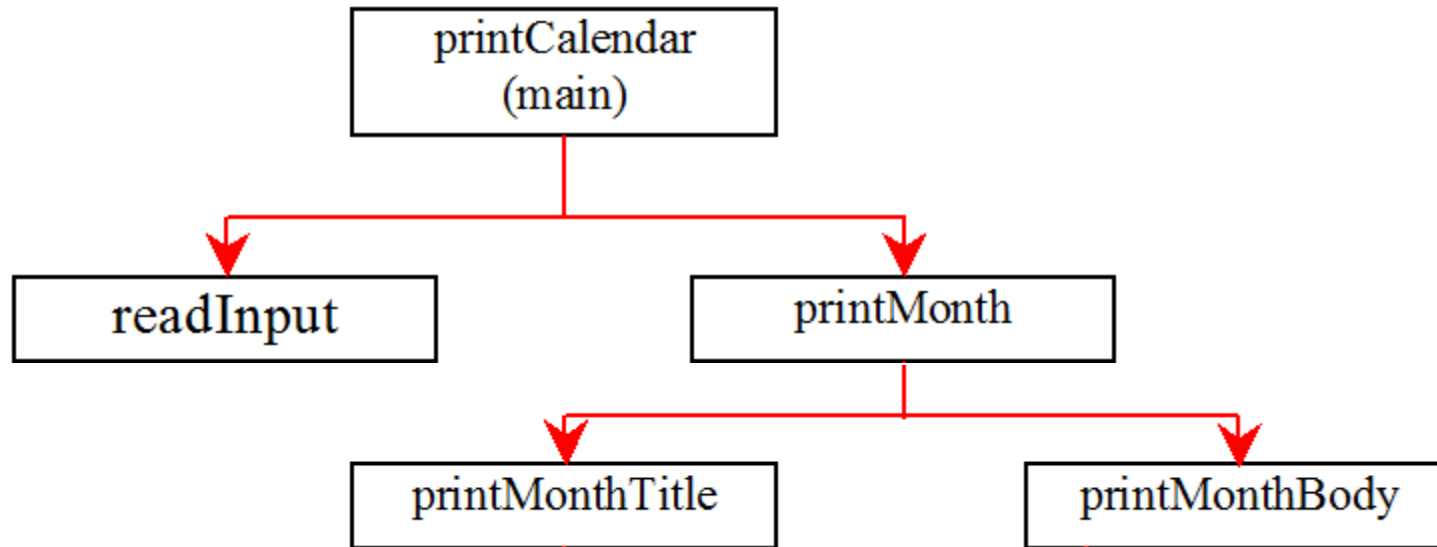# Stepwise Refinement – Example, cont.
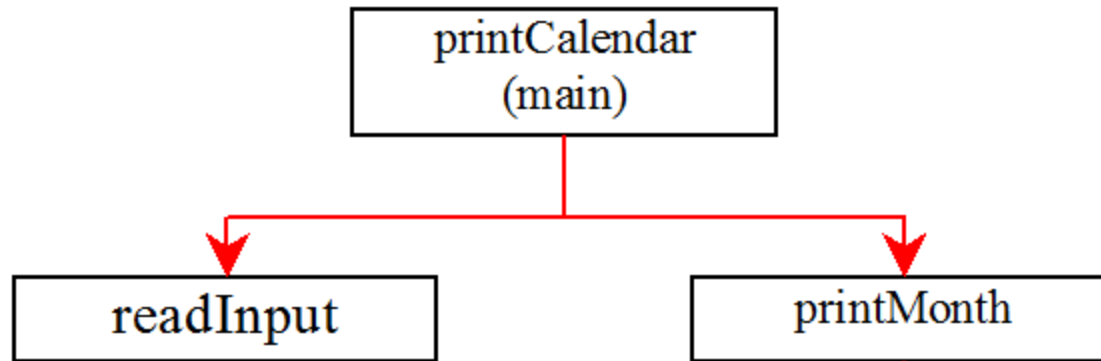
The **printCalendar** problem is broken into
- Get input from the user (**readInput**), and
- Print the calendar for the month (**printMonth**).

At this stage, you should be concerned with what the subproblems will achieve, not with how to get input and print the calendar for the month.
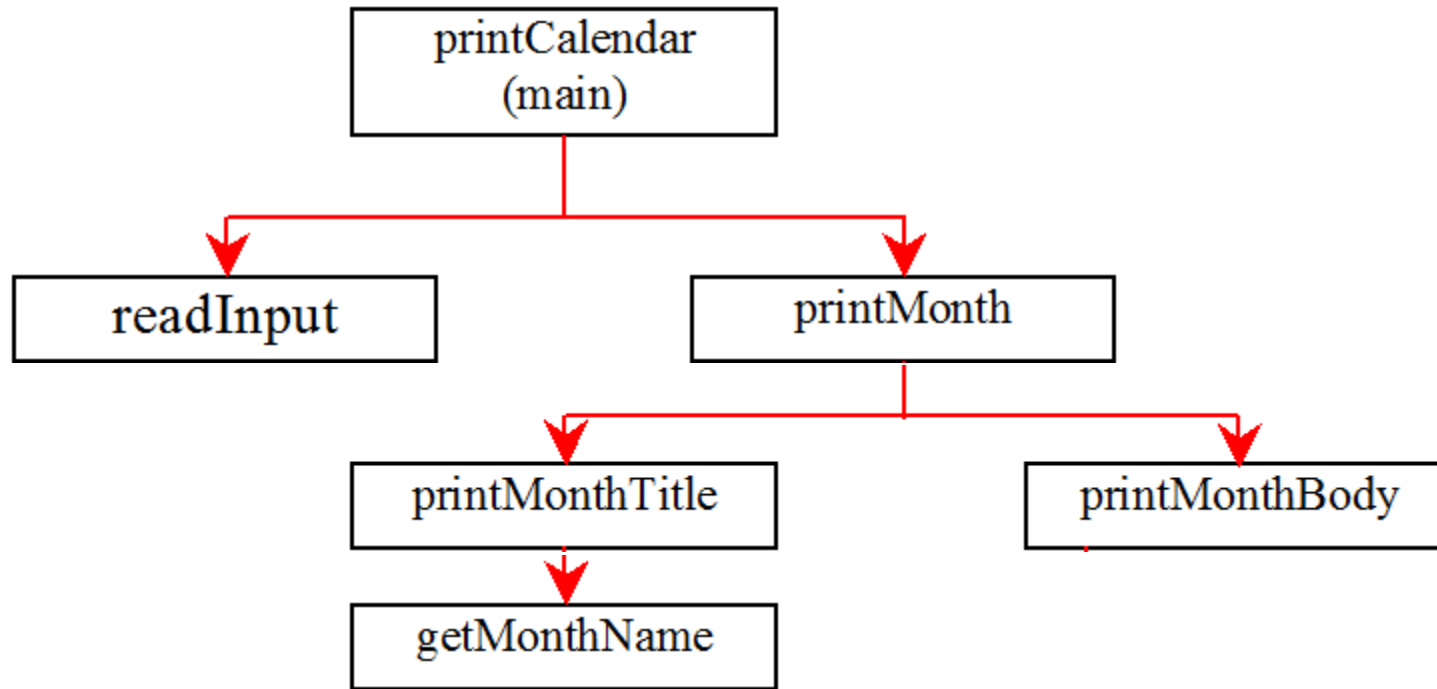
Stepwise Refinement – Example, cont.

You can use **Scanner** to read input for the year and the month.
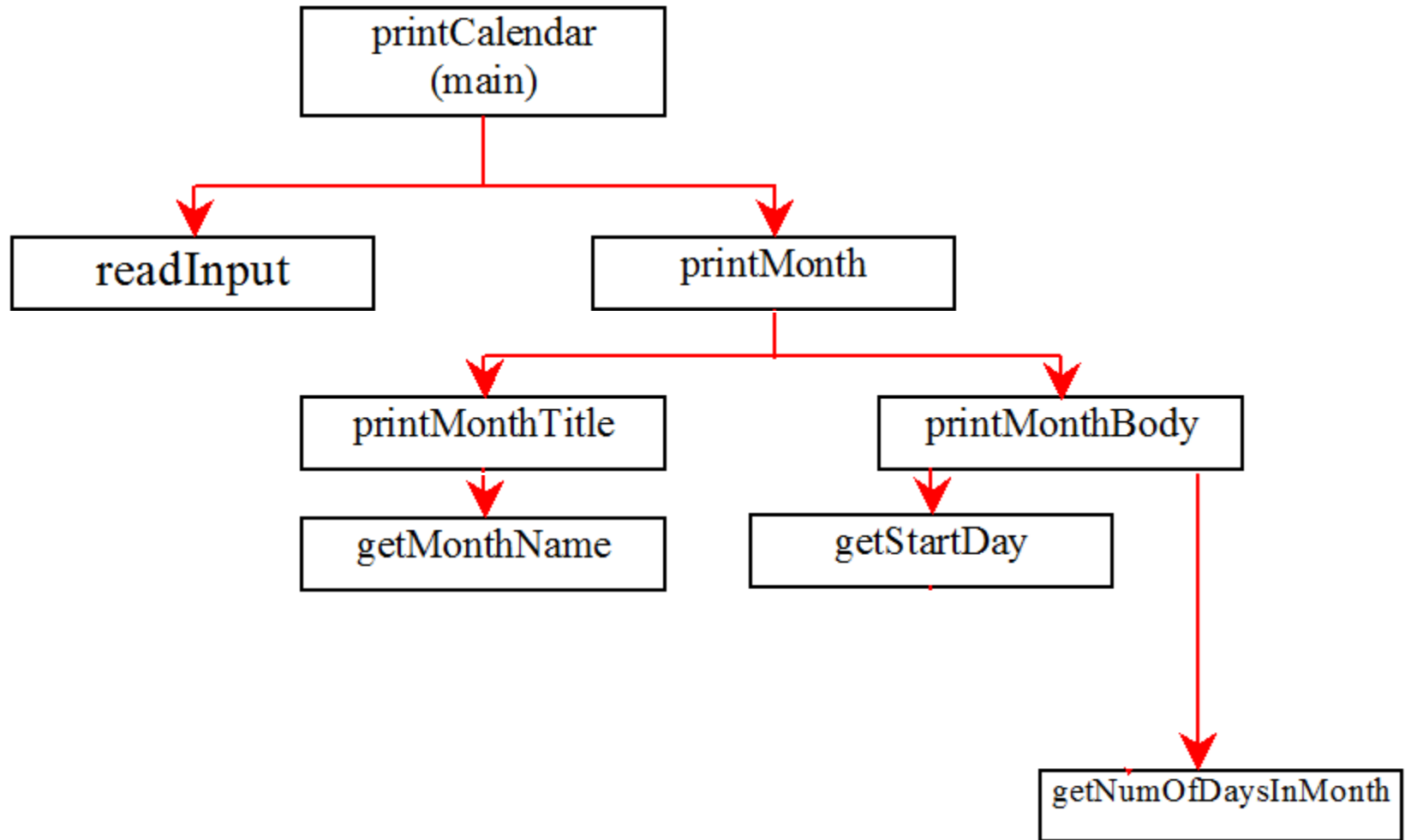
The **printMonth** problem is broken into
- print the month title (**printMonthTitle**) and
- print the month body (**printMonthBody**)

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 68

Stepwise Refinement – Example, cont.

The month title consists of three lines: month and year, a dashed line, and the names of the seven days of the week. You need to get the month name (e.g., January) from the numeric month (e.g., 1). This is accomplished in **getMonthName**

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 69

Stepwise Refinement – Example, cont.

In order to print the month body, you need to know:
- which day of the week is the first day of the month (**getStartDay**) and
- how many days the month has (**getNumberOfDaysInMonth**)

Stepwise Refinement – Example, cont.


```
        printCalendar
           (main)

   readInput        printMonth

           printMonthTitle        printMonthBody

                getMonthName        getStartDay

                              getTotalNumOfDays

                                        getNumOfDaysInMonth
```
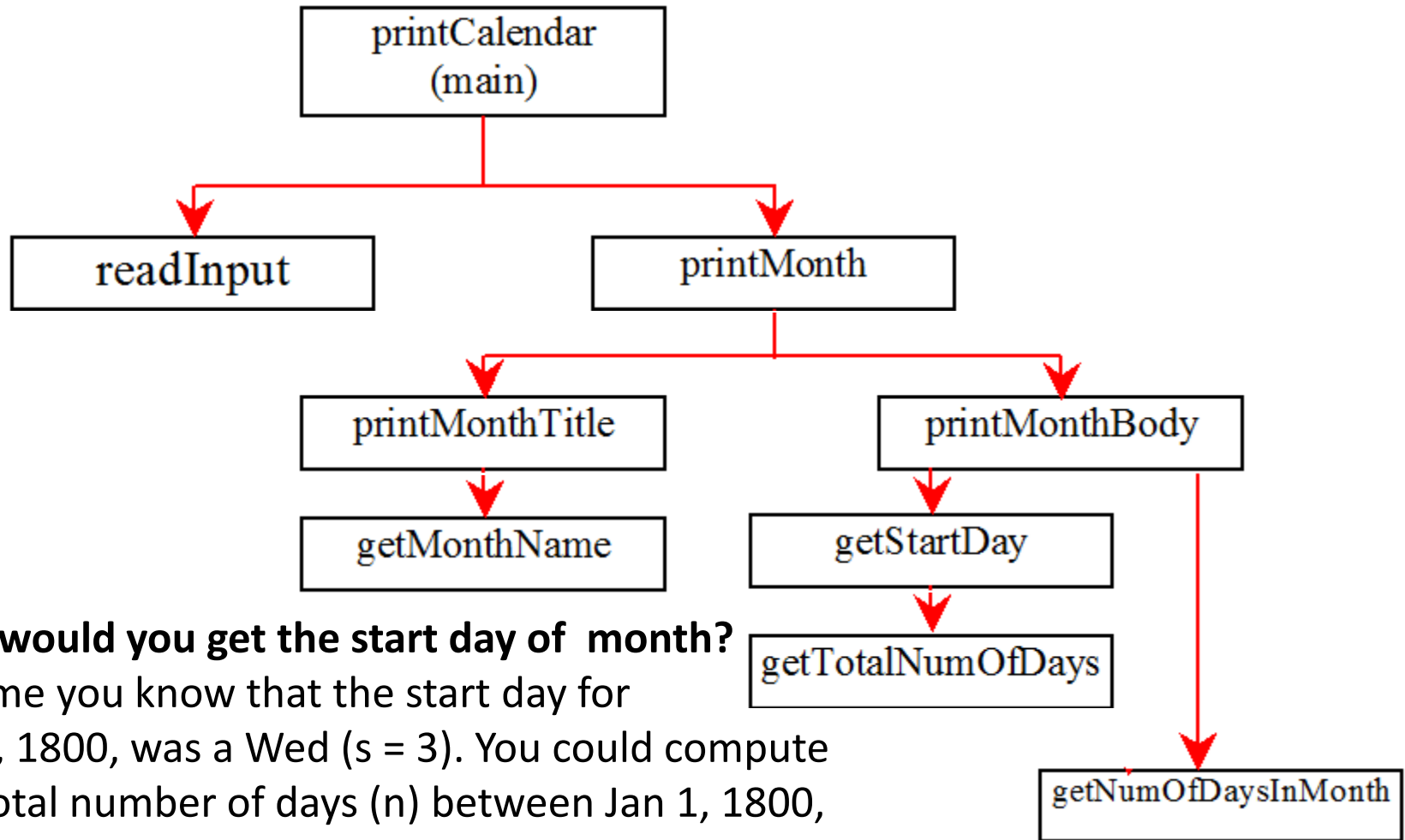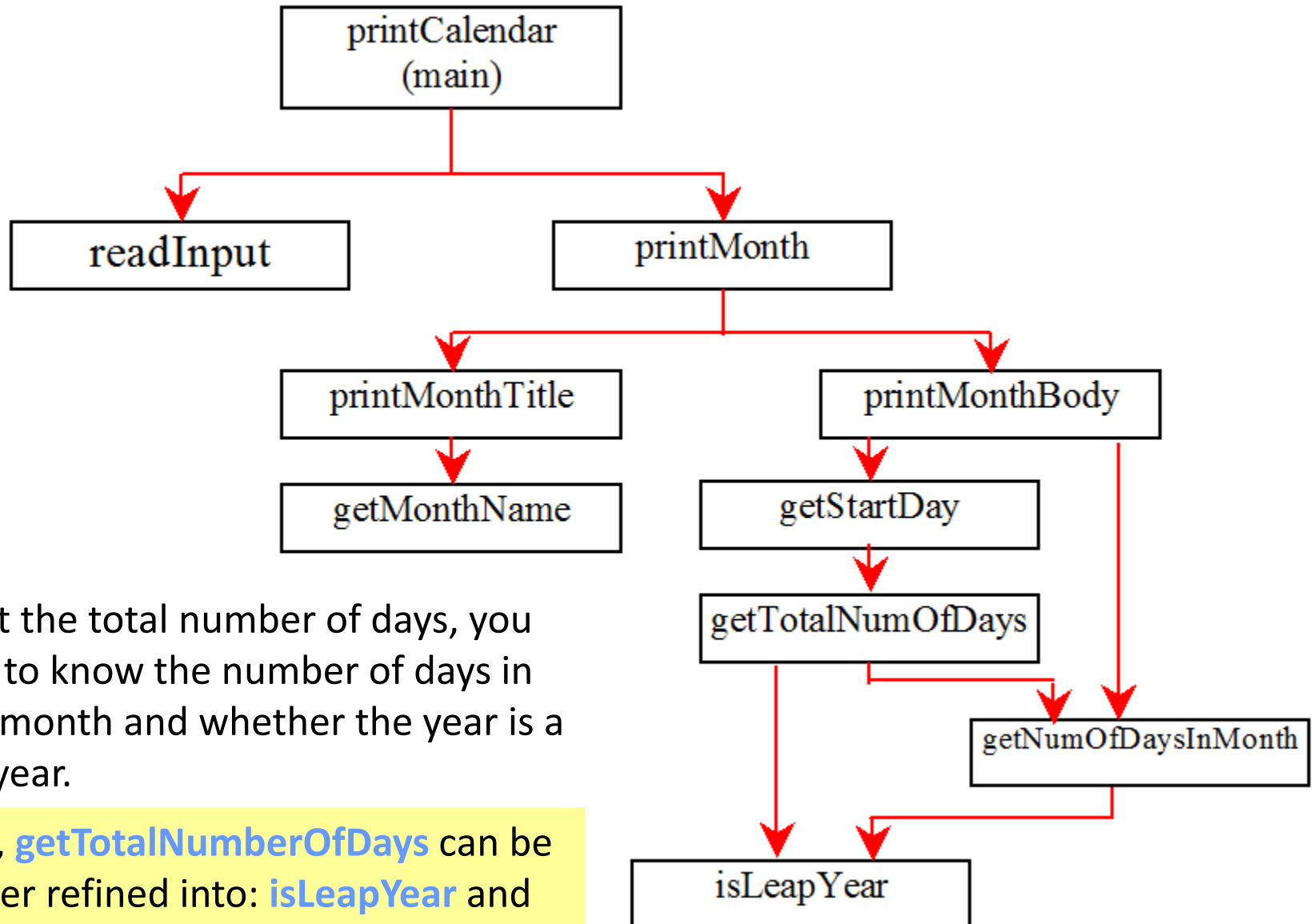
How would you get the start day of month?
Assume you know that the start day for Jan 1, 1800, was a Wed (s = 3). You could compute the total number of days (n) between Jan 1, 1800, and the first date of the calendar month. The start day for the calendar month is (n + s) % 7.

Thus, the **getStartDay** problem needs to **getTotalNumberOfDays**

Stepwise Refinement – Example, cont.

To get the total number of days, you need to know the number of days in each month and whether the year is a leap year.

Thus, **getTotalNumberOfDays** can be further refined into: **isLeapYear** and **getNumberOfDaysInMonth**

Given the previous design, your program may begin like this:

```java
public class PrintCalendar {
  public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      // Prompt the user to enter month/year
      System.out.print("Enter full year (e.g., 2012): ");
      int year = input.nextInt();
      System.out.print("Enter month as a number between 1 and 12: ");
      int month = input.nextInt();
      // Print calendar for the month of the year
      printMonth(year, month);
  }
  /** Stubs for different methods may look like this */
  public static void printMonth(int year, int month) {}
  public static void printMonthTitle(int year, int month) {}
  public static void printMonthBody(int year, int month) {}
  public static String getMonthName(int month) {return "January"; /*dummy value*/}
  public static int getStartDay(int year, int month) {return 1; /*dummy value*/}
  public static int getTotalNumberOfDays(int year,int month){return 10000;/*dummy value*/}
  public static int getNumberOfDaysInMonth(int year,int month) {return 31;/*dummy value*/}
  public static Boolean isLeapYear(int year) {return true; /* A dummy value*/}
}
```

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc.

COSC 111. Page 73

Stepwise Refinement – Example, cont.

The next step is to work on the implementation details for each method (and adjust your design if necessary).

- Section 6.11.3 in the textbook includes the details for each method as well as the complete program.

Benefits of Stepwise Refinement

Simpler Program

Reusing Methods

Easier Developing, Debugging, and Testing

Better Facilitating Teamwork