Methods

CSE 114, Computer Science 1

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Opening Problem

Find multiple sums of integers:

    - from <u>1</u> to <u>10</u>,

    - from <u>20</u> to <u>30</u>,

    - from <u>35</u> to <u>45</u>,

    ...

Opening Problem

- Repeat code:

```
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);
```

Opening Problem Solution

- Use a method!

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

```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));
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Why write methods?

- To shorten your programs
  - avoid writing identical code twice or more
- To modularize your programs
  - fully tested methods can be trusted
- To make your programs more:
  - readable
  - reusable
  - testable
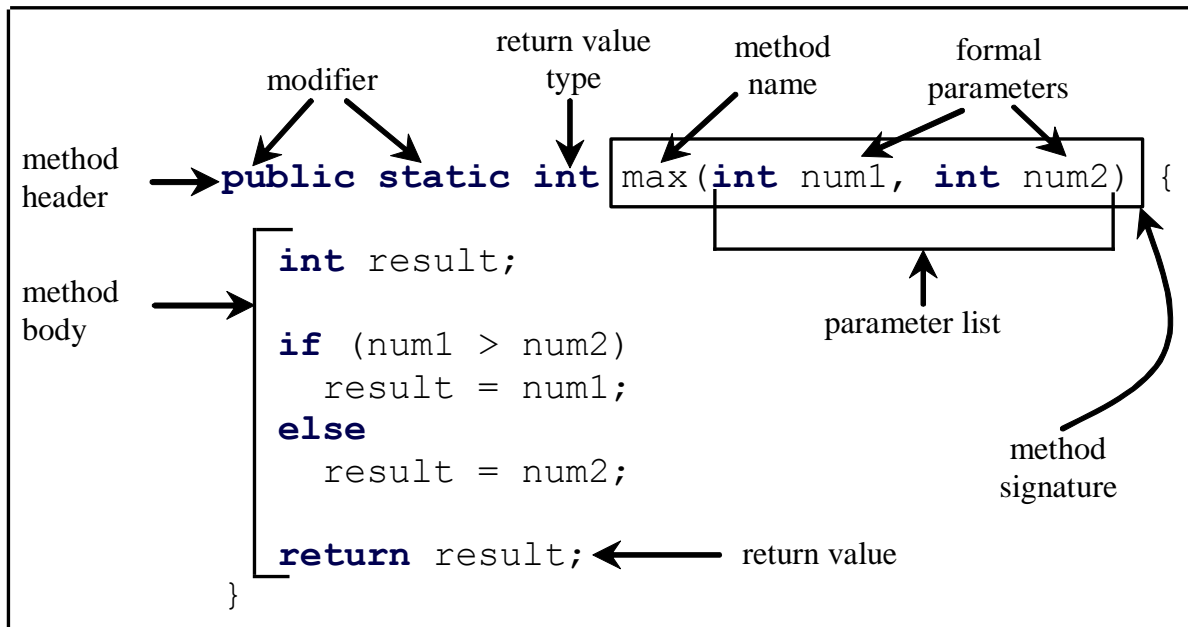 - debuggable
  - extensible
  - adaptable

Rule of thumb

- If you have to perform some operation in more than one place inside your program, make a new method to implement this operation and have other parts of the program use it.
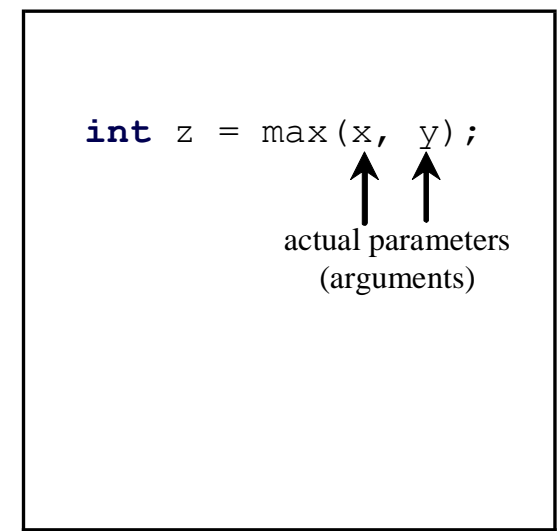
Defining Methods

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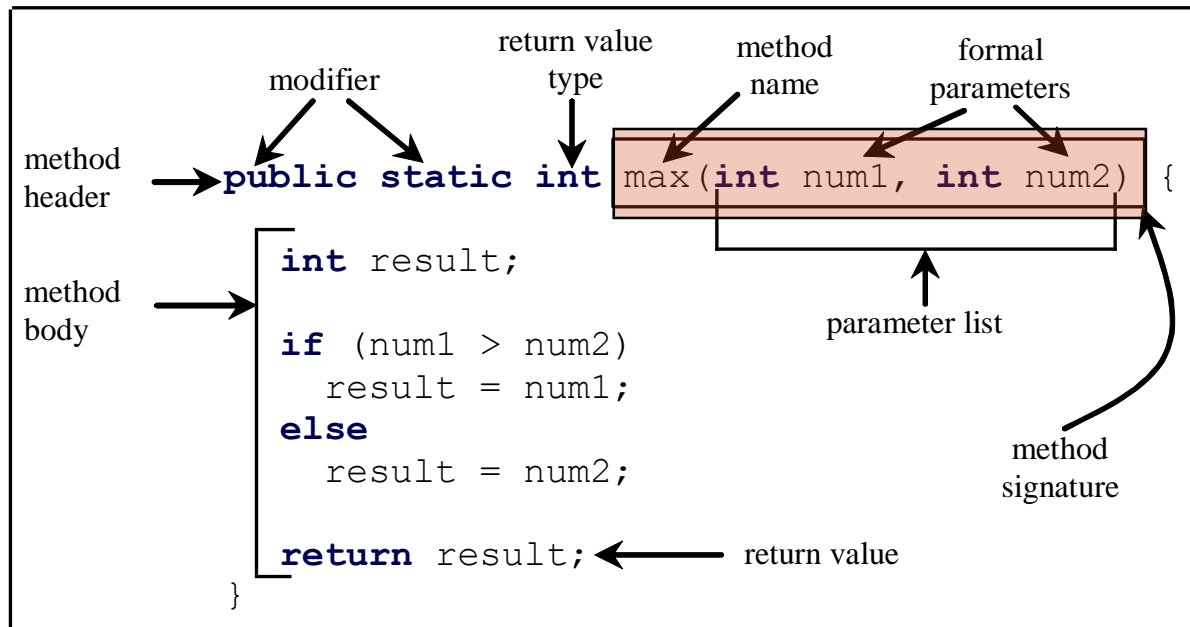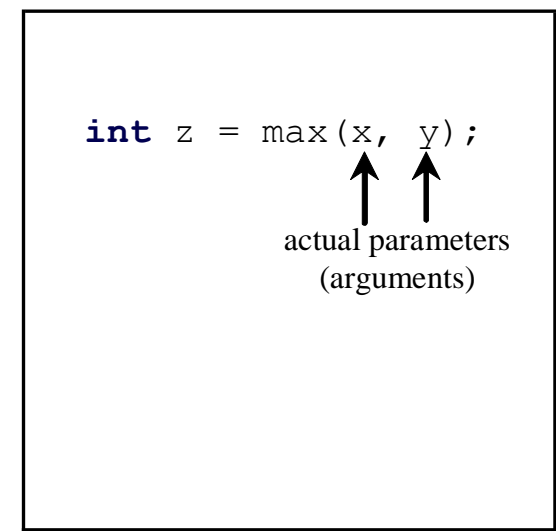
Define a method

Invoke a method

```
                    modifier      return value     method          formal
                                      type           name         parameters

method ─────▶  public static int  max(int num1,  int num2)  {
header
                              int result;
method
body  ─────▶
                              if (num1 > num2)
                                result = num1;
                              else
                                result = num2;

                              return result;  ◀──── return value
                            }
```

return value type

method name

formal parameters

parameter list

method signature

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

actual parameters (arguments)

Method Signature

- *Method signature* is the combination of the method name and the parameter list.

Define a method

Invoke a method

```
              modifier          return value      method      formal
                                   type            name     parameters

method
header    →   public static int  max(int num1, int num2) {

method    →       int result;
body
                  if (num1 > num2)
                    result = num1;
                  else
                    result = num2;

                  return result;  ←  return value
                }
```

parameter list

method signature

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

actual parameters (arguments)
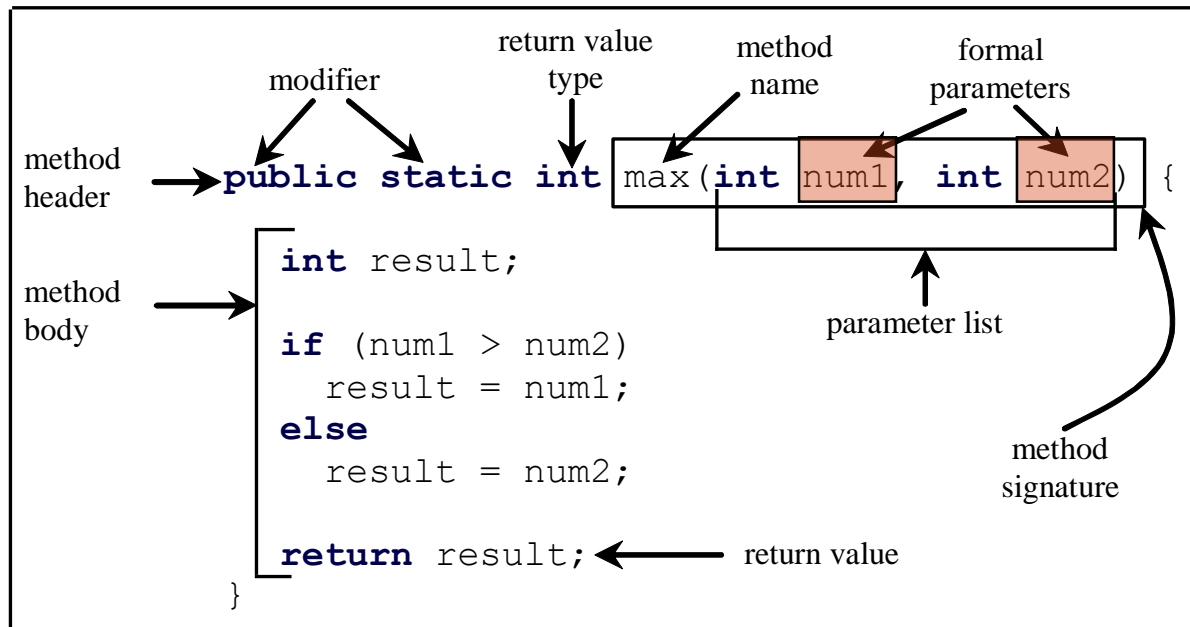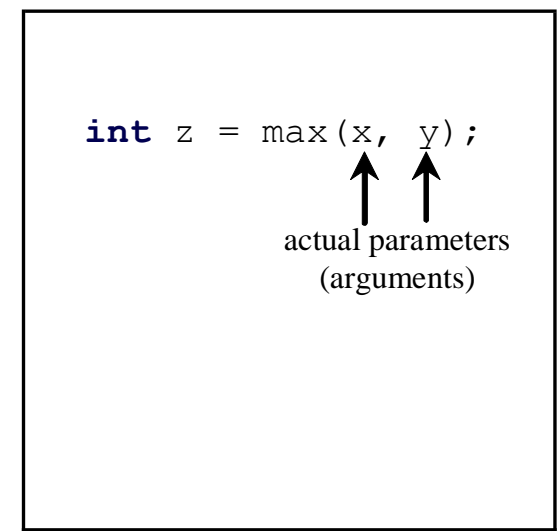
Formal Parameters

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

Define a method

```
                                      return value      method              formal
              modifier                  type            name              parameters

method
header  ──►   public static int  max(int num1, int num2)  {

                  int result;
method
body                              
                  if (num1 > num2)
                      result = num1;
                  else
                      result = num2;                              method
                                                                  signature
                  return result;  ◄─── return value
              }
```
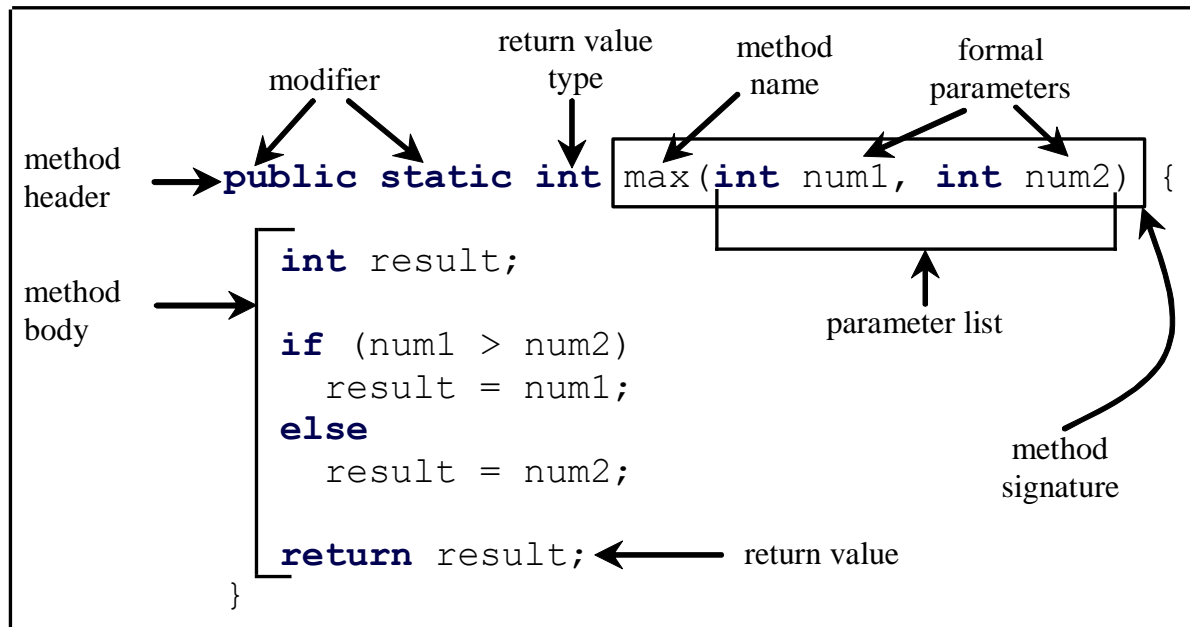
parameter list

Invoke a method

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

                          actual parameters
                            (arguments)
```
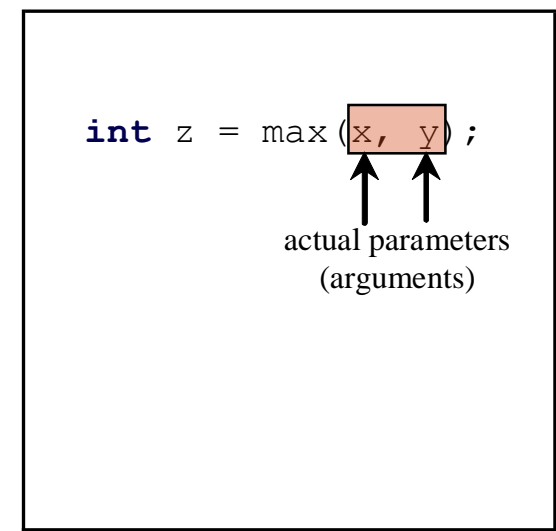
Actual Parameters

- When a method is invoked, you pass a value to the parameter: *actual parameter or argument.*
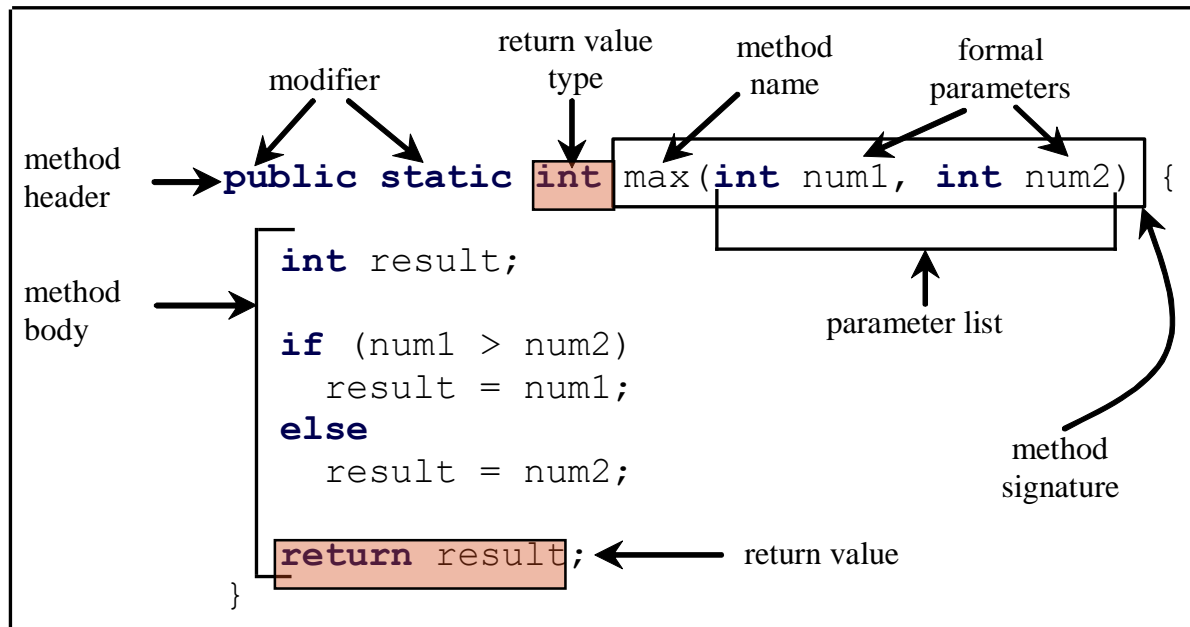
Define a method

Invoke a method

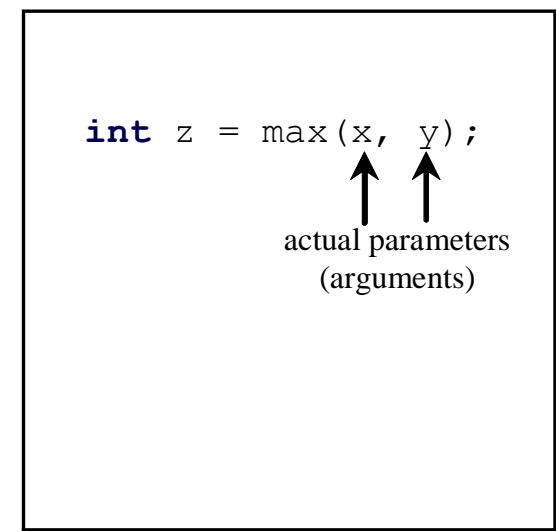modifier · return value type · method name · formal parameters

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

method          int result;
body

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

                return result;          ← return value
            }
```

parameter list

method signature

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

actual parameters (arguments)

Return Value Type

- ## A method may return a value.

- The *returnValueType* is the data type of the value the method returns.

If the method does not return a value, the returnValueType is the keyword void.

Define a method

Invoke a method

```
                    return value      method        formal
       modifier         type           name       parameters

method
header  →  public static int max(int num1, int num2) {

                                                parameter list
method
body  →    int result;

           if (num1 > num2)
             result = num1;
           else
             result = num2;
                                                method
                                                signature
           return result;  ←  return value
         }
```

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

          actual parameters
          (arguments)
```

Calling Methods

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;
```

Trace Method Invocation

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;
}
```

Trace Method Invocation

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;
}
```

14

Trace Method Invocation

invoke max(i, j)

```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;
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation

```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;
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation

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;
}
```

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;
}
```

Trace Method Invocation

result is now 5

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

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

  return result;
}
```

Trace Method Invocation

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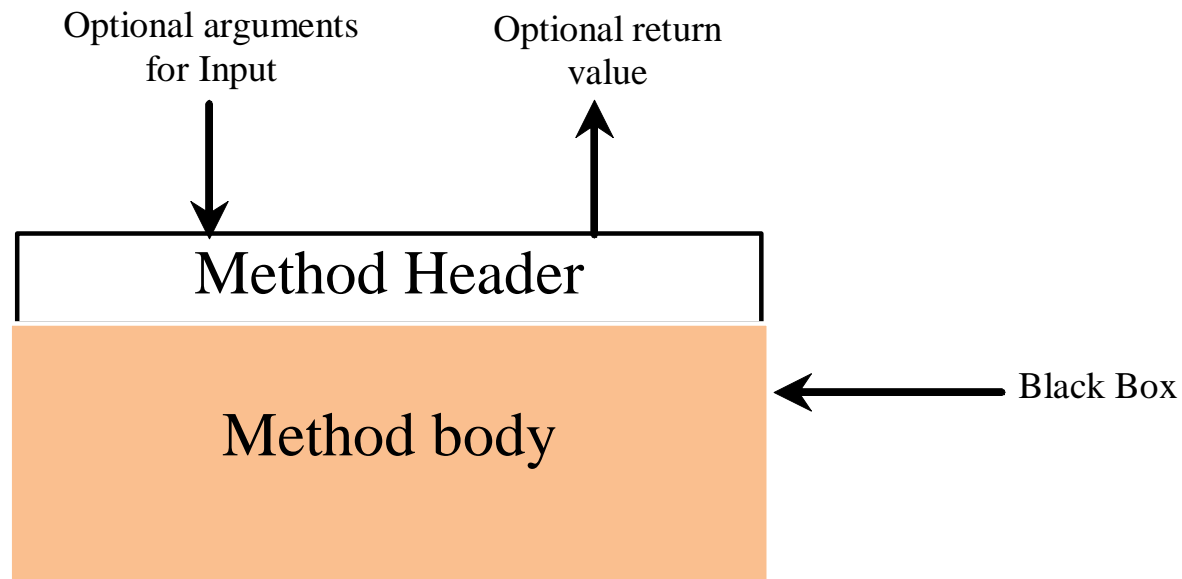
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

> return result, which is 5

Trace Method Invocation

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;
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation

Execute the print statement

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

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Benefits of Methods

- Write a method once and reuse it anywhere.

- Information hiding:
  - Hide the implementation from the user.

- Reduces complexity.

Method Abstraction

API = the method body is a black box that contains the detailed implementation for the method.

Optional arguments for Input → Method Header → Optional return value

Method Header

Method body ← Black Box

Javadoc

- Generate Javadoc for your project in Eclipse with:
    1. Project -> Generate Javadoc
    2. In the "Javadoc command" field - browse to find javadoc.exe
        - On the computers in the lab that is

        C:\Program Files\Java\jdk1.8.0\bin\javadoc.exe
        - On other computers it would be
        <path_to_jdk_directory>\bin\javadoc.exe
    3. Check the box next to the project/package/file for which you are creating the javadoc
    4. In the "Destination" field browse to find the desired destination (for example, the doc directory of the current project).
    5. Leave everything else as it is.
    6. Click "Finish" and open "index.html"
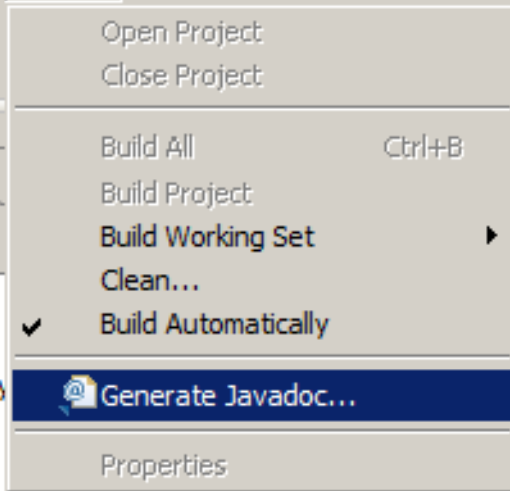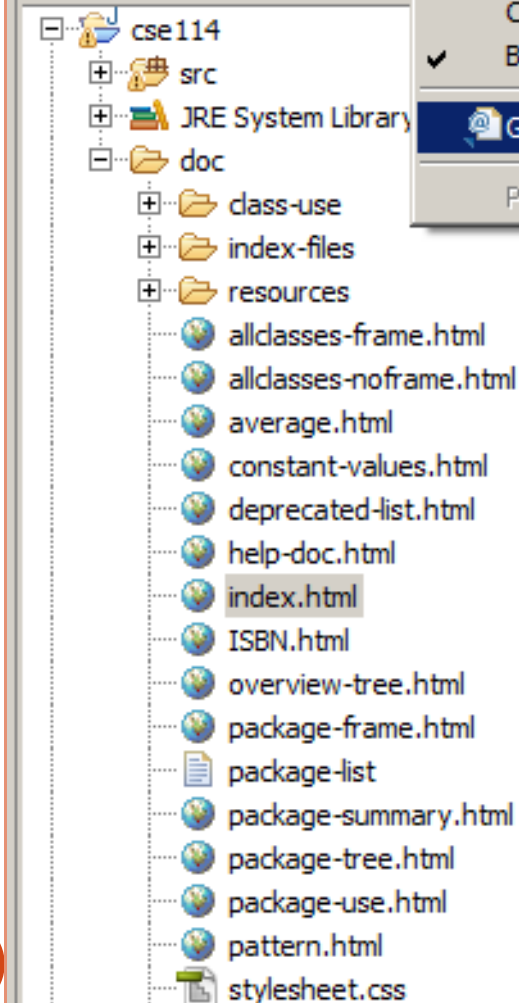
CAUTION: all execution paths

- A <u>return</u> statement is required for a <u>value-returning</u> method.

The method shown below 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;
}
```
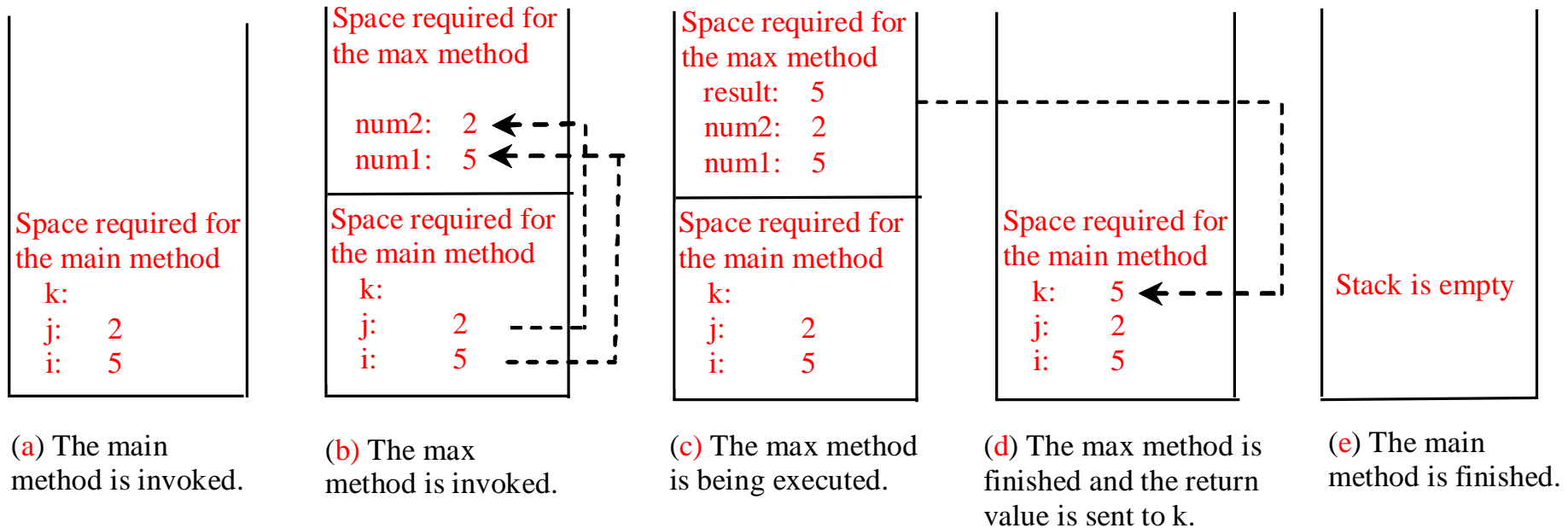(a)

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 <u>return</u> statement to be reached regardless of how the <u>if</u> statement is evaluated.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Call Stacks

Methods are executed using a stack data structure

Space required for
the max method

   num2:   2
   num1:   5

Space required for
the max method
   result: 5
   num2:   2
   num1:   5

Space required for
the main method
  k:
  j: 2
  i: 5

Space required for
the main method
  k:
  j: 2
  i: 5

Space required for
the main method
  k:
  j: 2
  i: 5

Space required for
the main method
  k: 5
  j: 2
  i: 5

Stack is empty

(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 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.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Call Stack

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;
}
```

j: 2
i: 5

The main method is invoked.

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

Invoke max(i, j)

```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);
}

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

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;
}
```

num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

Trace Call Stack

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;
}
```

result:
num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

Trace Call Stack

(num1 > num2) is true

```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;
}
```

result:
num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

Trace Call Stack

Assign num1 to result

```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;
}
```

Space required for the max method
    result: 5
    num2: 2
    num1: 5

Space required for the main method
    k:
    j: 2
    i: 5

The max method is invoked.

Trace Call Stack

Return result and assign it to 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);
}

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

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

  return result;
}
```

Space required for the max method

result: 5
num2: 2
num1: 5

Space required for the main method

k:5
j: 2
i: 5

The max method is invoked.

Trace Call Stack

Execute print statement

```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);
}

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.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Call-by-value

- Method formal arguments are *copies of the original data*.

- Consequence?
  - methods <u>cannot</u> assign („=") new values to primitive type formal arguments and <u>affect the original passed variables</u>.

- Why?
 - changing argument values changes the copy, not the original.

Call-by-value

The values of num1 and num2 are
passed to n1 and n2. Executing swap
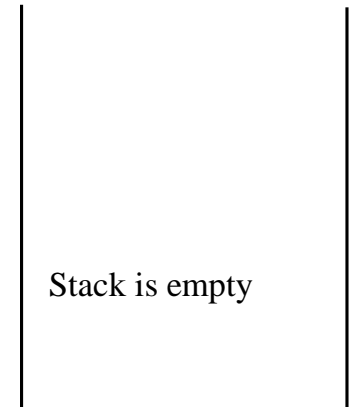does not affect num1 and num2.

Space required for the swap method	Space required for the swap method		
	temp:		
	n2: 2		
	n1: 1		

Space required for the
main method
num2: 2
num1: 1

Space required for the
main method
num2: 2
num1: 1

Space required for the
main method
num2: 2
num1: 1

Stack is empty
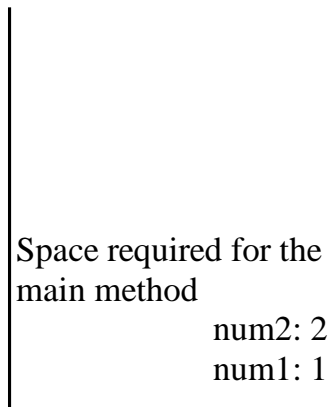
The main method
is invoked
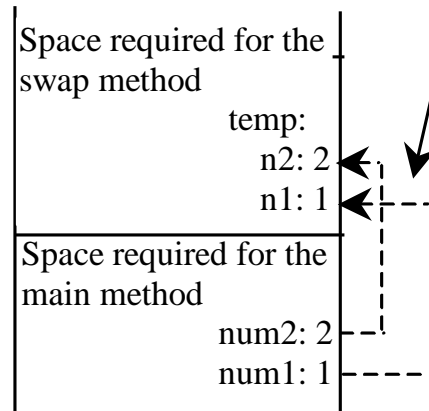
The swap method
is invoked
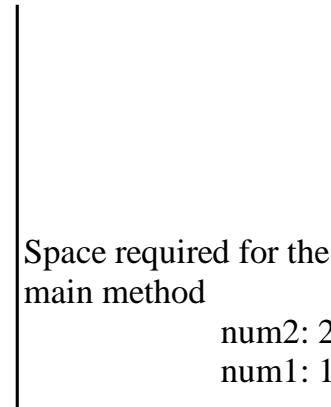
The swap method
is finished

The main method
is finished

Overloading

- Method overloading is the ability to create multiple methods of the same name with different implementations.

```java
// Overload the name max for different invocations
public static int max(int x, int y){
        return (x>y) ? x : y;
}

public static double max(double x, double y){
        return (x>y) ? x : y;
}

public static void main(String[] args) {
        System.out.println(max(1,2)); // will call max(int,int)
        System.out.println(max(3.5,4.7)); // will call max(double,double)
}
```

Overloading & 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.

Overloading & Ambiguous Invocation

```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;
  }
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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.
  - In Java, the scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
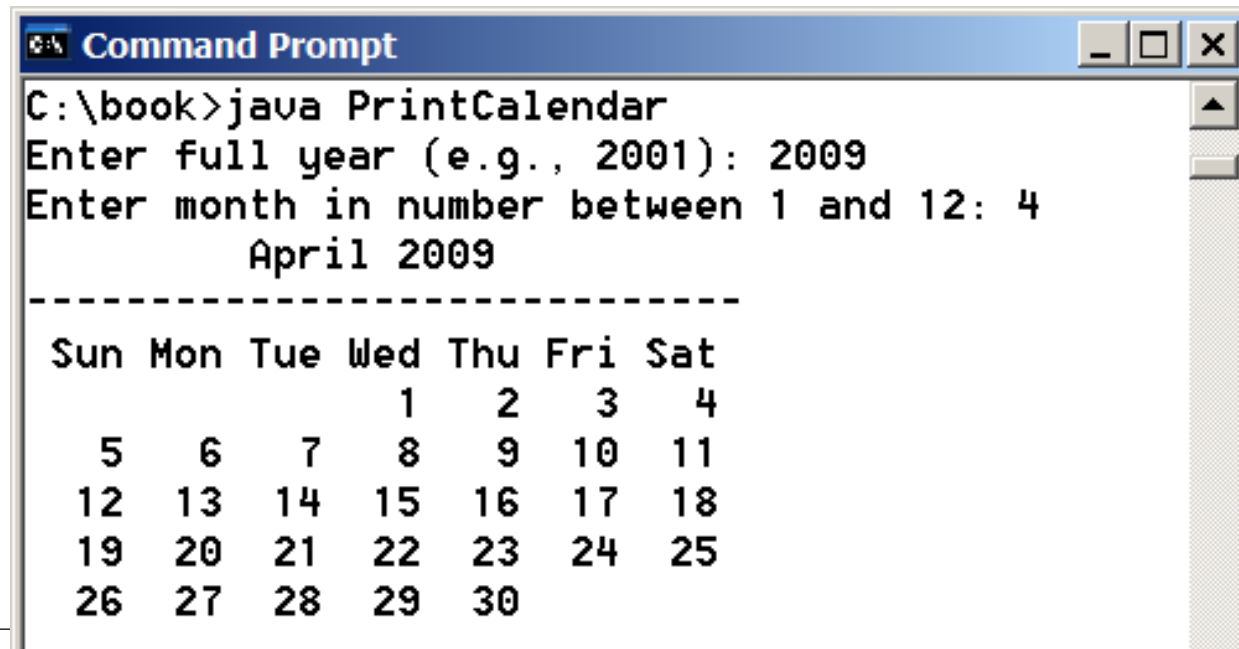
Scope of Local Variables

- A nested block cannot redefine a local variable:

```
public static void correctMethod() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        int x = 0; // error
        x += i;
    }
}
```
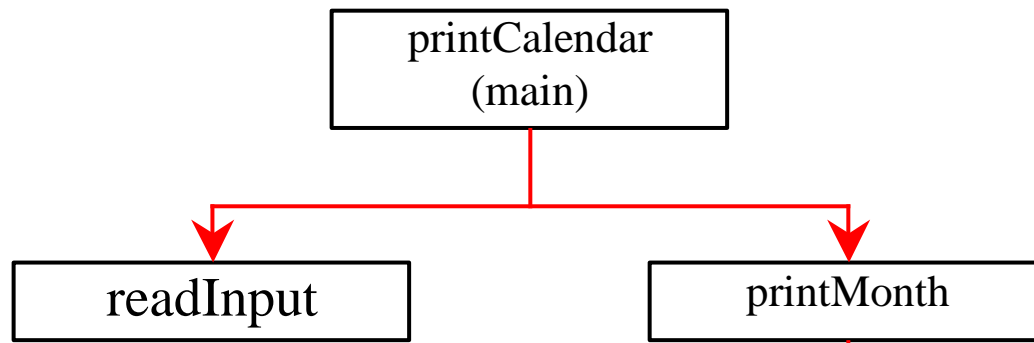
Stepwise Refinement

- 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.
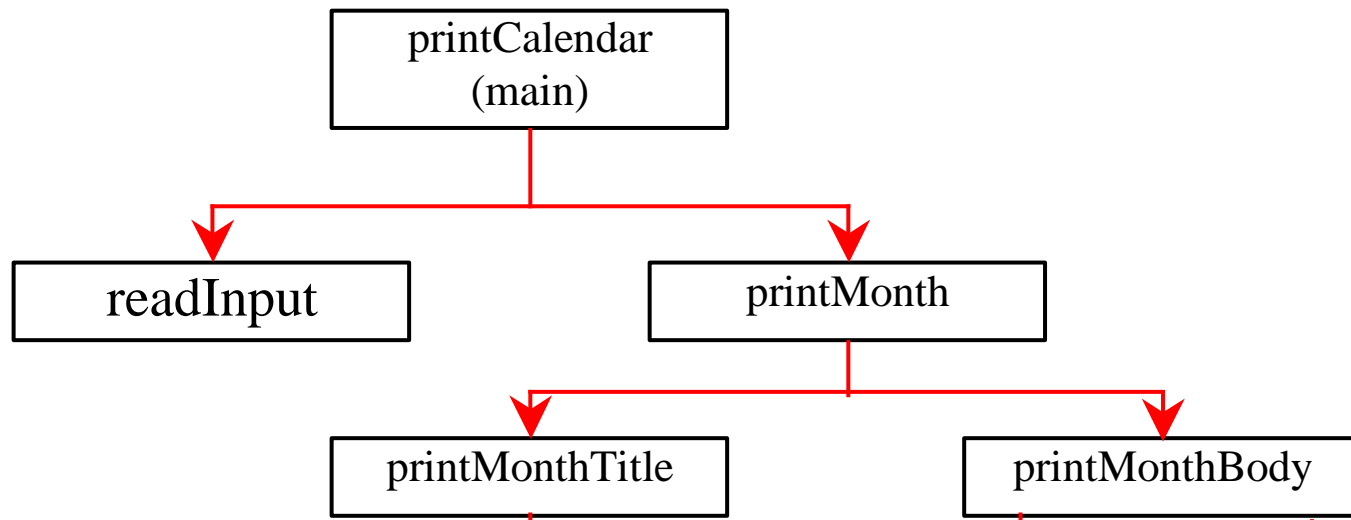- For example, consider a PrintCalendar program:


```
C:\book>java PrintCalendar
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
```
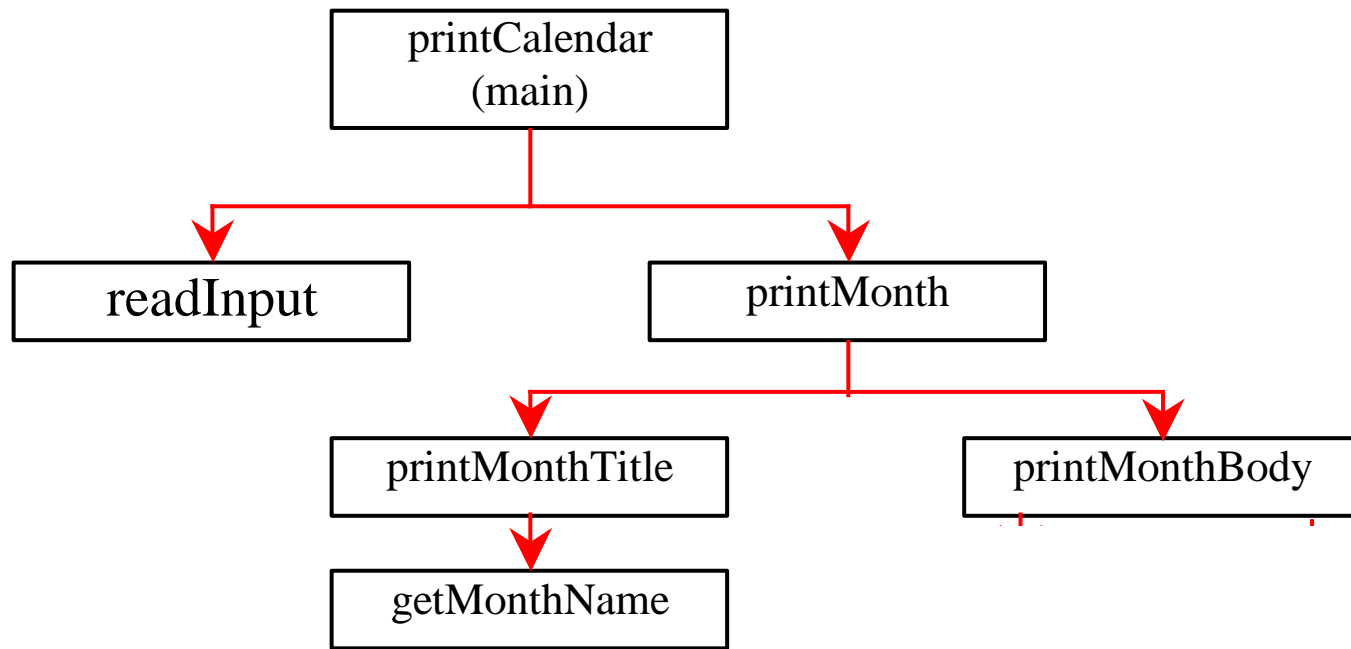
Design Diagram

```
          ┌──────────────────┐
          │  printCalendar   │
          │     (main)       │
          └──────────────────┘
                    │
        ┌───────────┴───────────┐
        ▼                       ▼
┌──────────────┐      ┌──────────────┐
│  readInput   │      │  printMonth  │
└──────────────┘      └──────────────┘
```

Design Diagram

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Design Diagram

Design Diagram

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
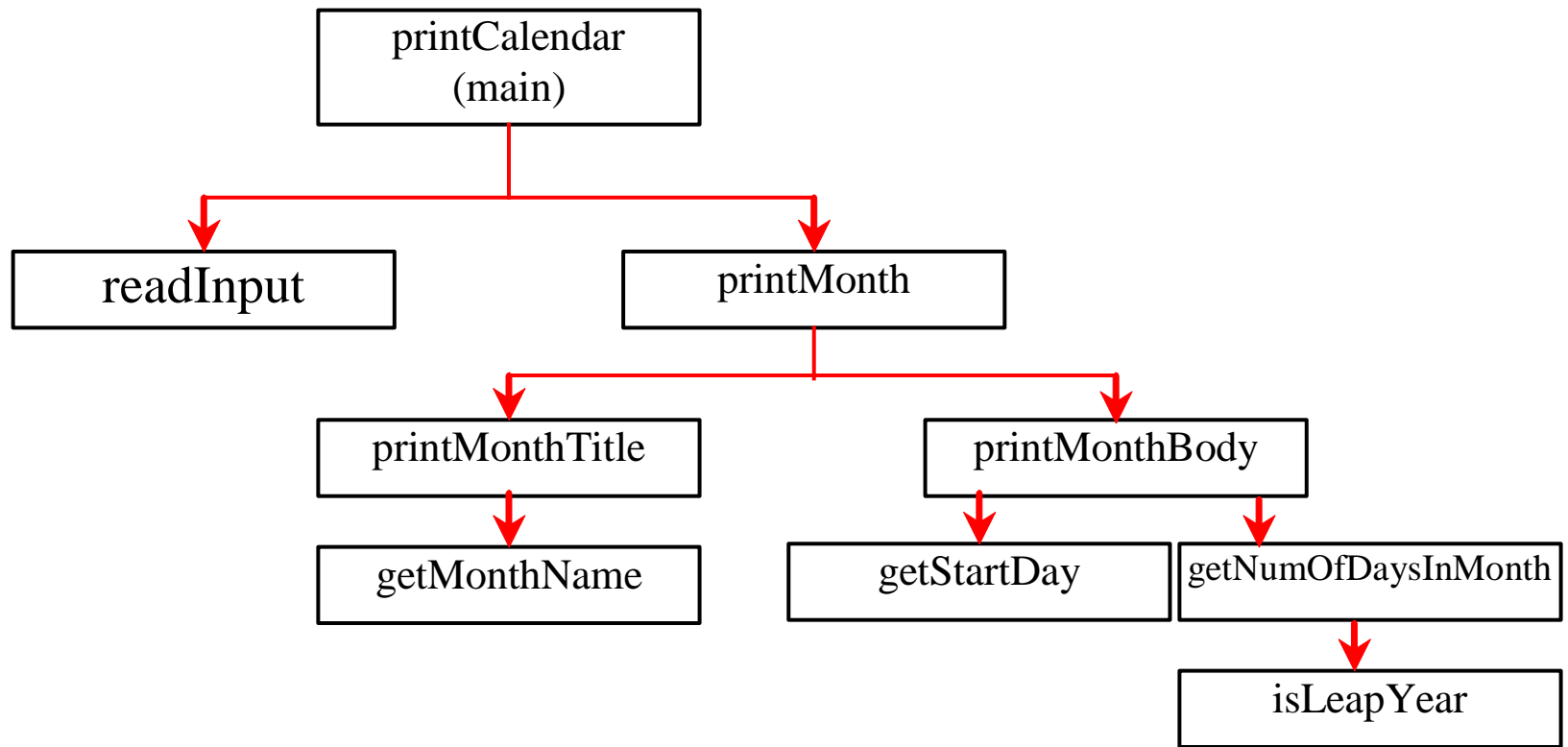
Implementation: Top-Down

- The ***top-down*** approach is to implement one method in the structure chart at a time from the top to the bottom
  - A *stub* is a simple but incomplete version of a method. Stubs can be used for the methods waiting to be implemented.

    ```
    /** A stub for getStartDay may look like this */
    public static int getStartDay(int year, int month) {
            return 1; // A dummy value
    }
    ```

  - The use of stubs enables you to test invoking the method from a caller.

- Implement the main method first and then use a stub for the printMonth method.
  - Then implement the complete methods.

Implementation: Bottom-Up

- ***Bottom-up approach*** is to implement one method in the structure chart at a time from the bottom to the top.
  - For each method implemented, write a test program to test only that method

- Both top-down and bottom-up methods are fine.
  - Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy.
  - Most of the time, they are used together

Benefits of Stepwise Refinement

- Simpler Program

- Reusing Methods

- Easier Developing, Debugging, and Testing

- Better Facilitating Teamwork