UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Dasar Dasar Pemrograman 2

Topik Pekan 2: Methods

Acuan: Introduction to Java Programming anda Data Structure, Bab 6

Sumber Slide: Liang

Dimodifikasi untuk Fasilkom UI oleh:

Muhammad Hilman
dan
Ade Azurat

Objectives

- To define methods with formal parameters (§6.2).
- To invoke methods with actual parameters (i.e., arguments) (§6.2).
- To define methods with a return value (§6.3).
- To define methods without a return value (§6.4).
- To pass arguments by value (§6.5).
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.6).
- To use method overloading and understand ambiguous overloading (§6.8).
- To determine the scope of variables (§6.9).
- To apply the concept of method abstraction in software development (§6.10).
- To design and implement methods using stepwise refinement (§6.10).

Defining Methods

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

Define a method

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

   int result;

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

   return result;
}
```

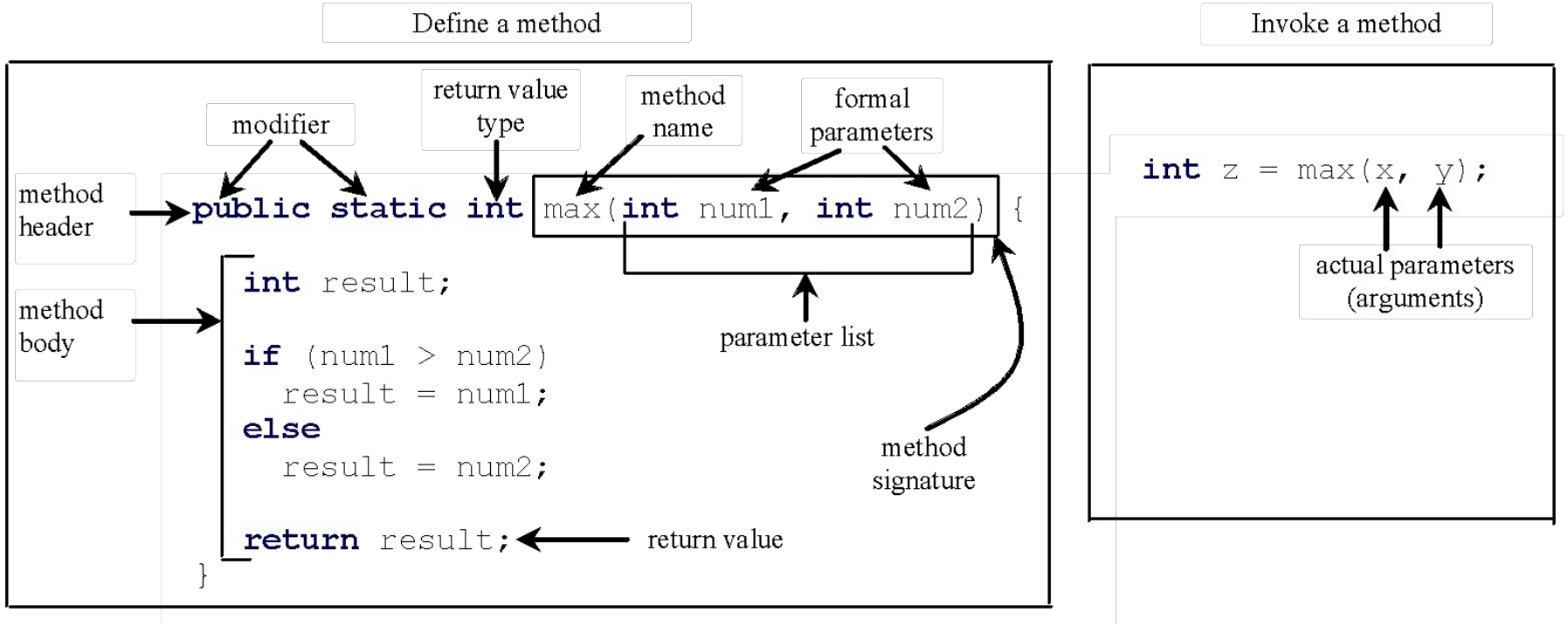
Invoke a method

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

actual parameters (arguments)

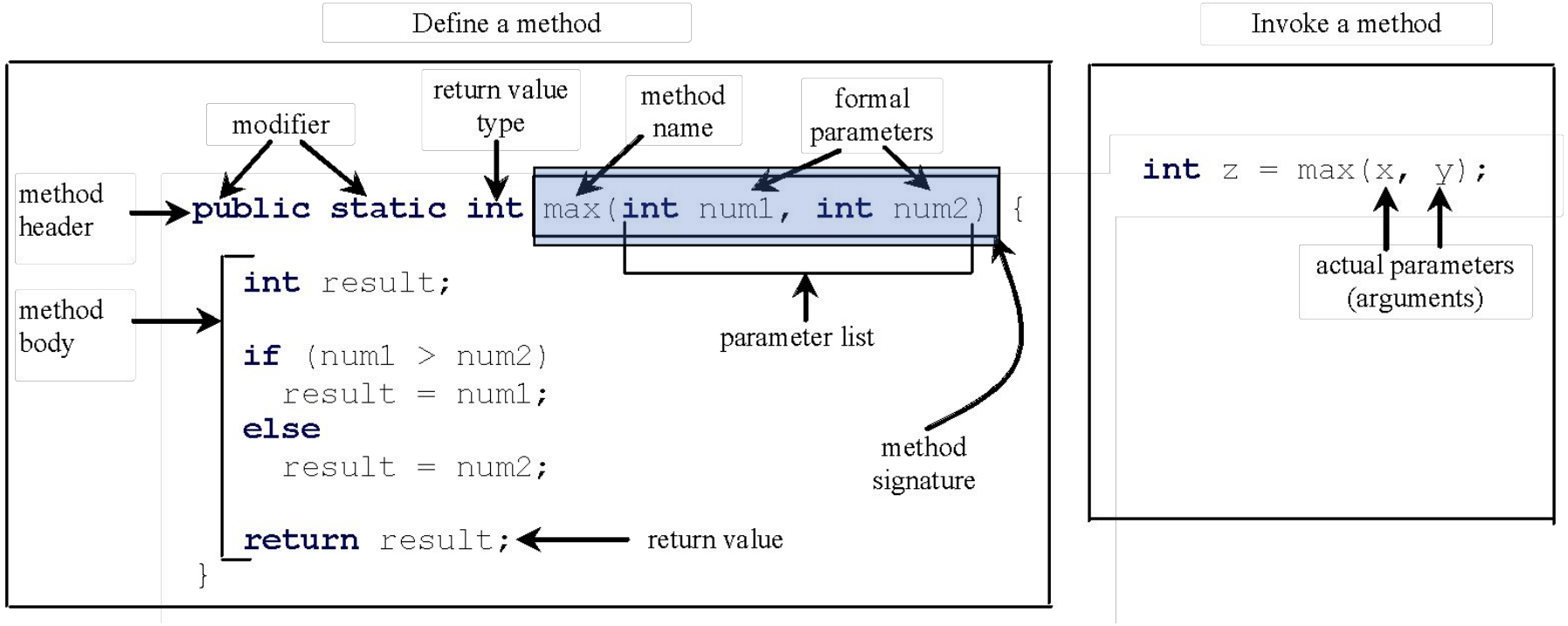
Defining Methods

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



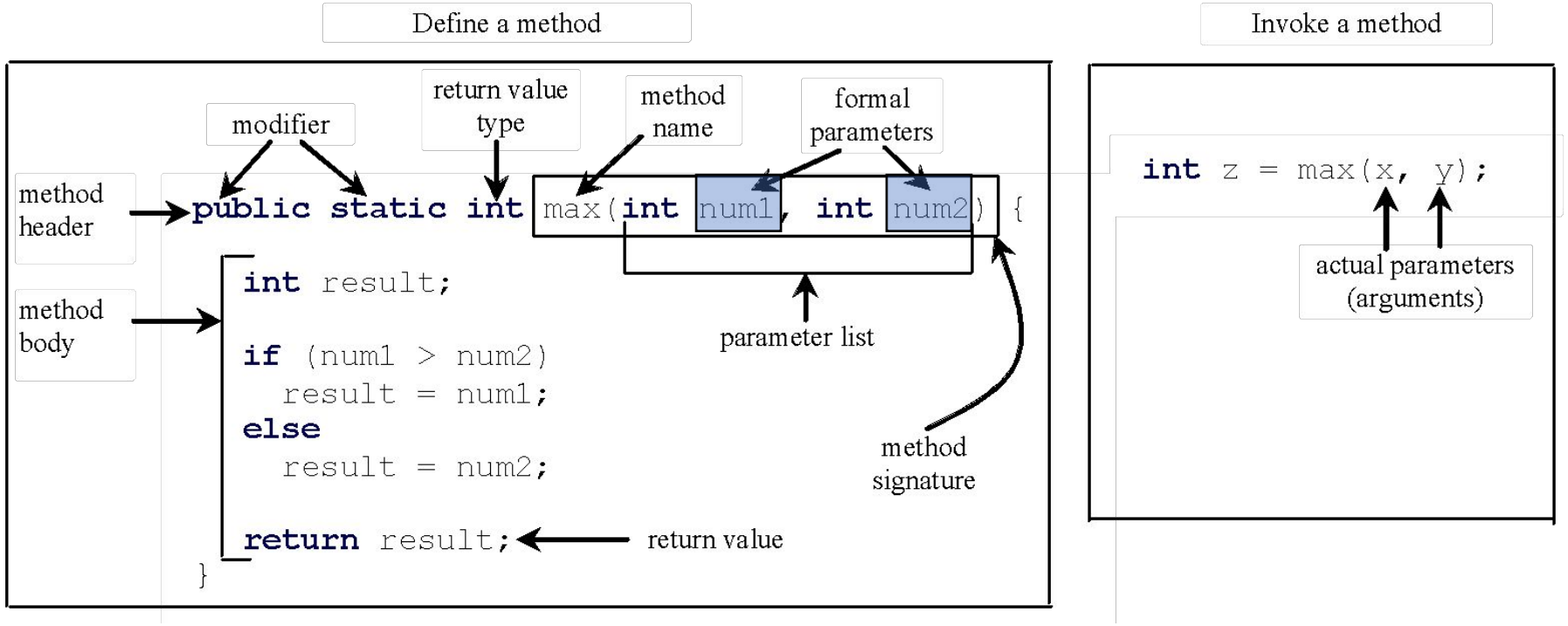
Method Signature

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



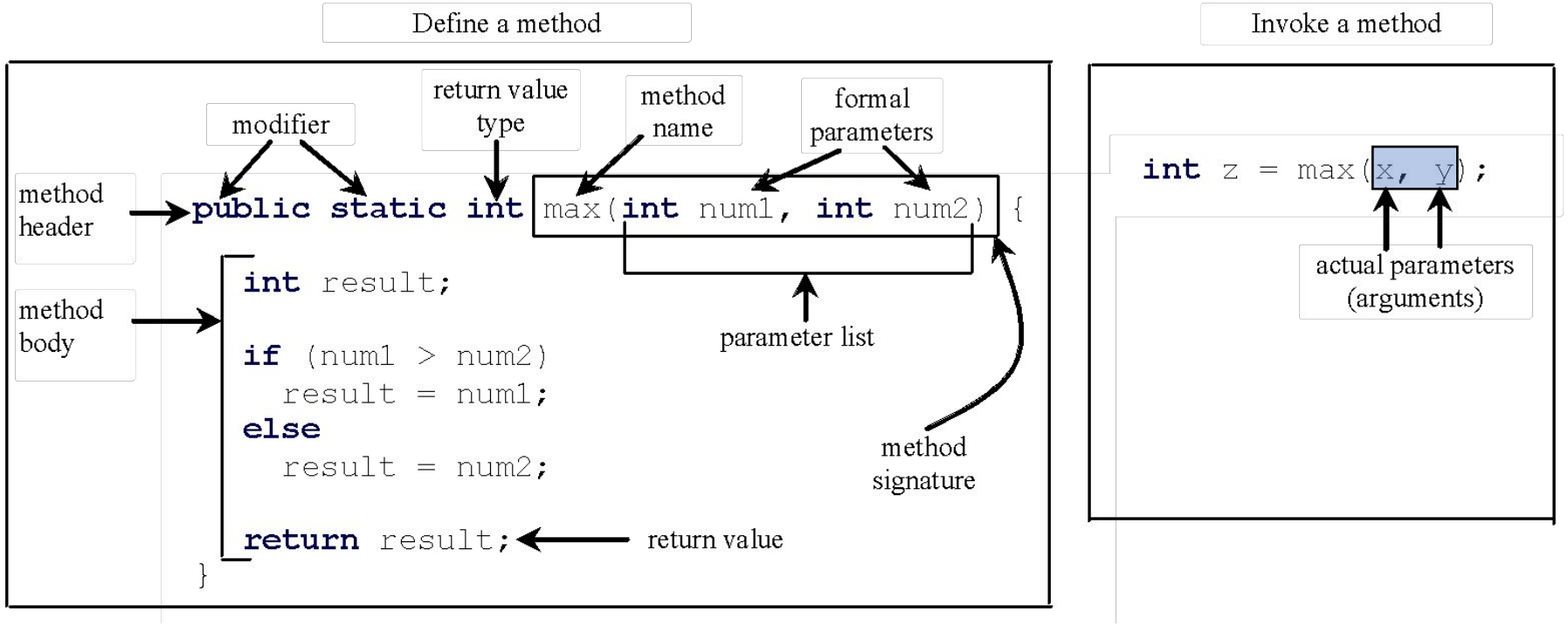
Formal Parameters

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



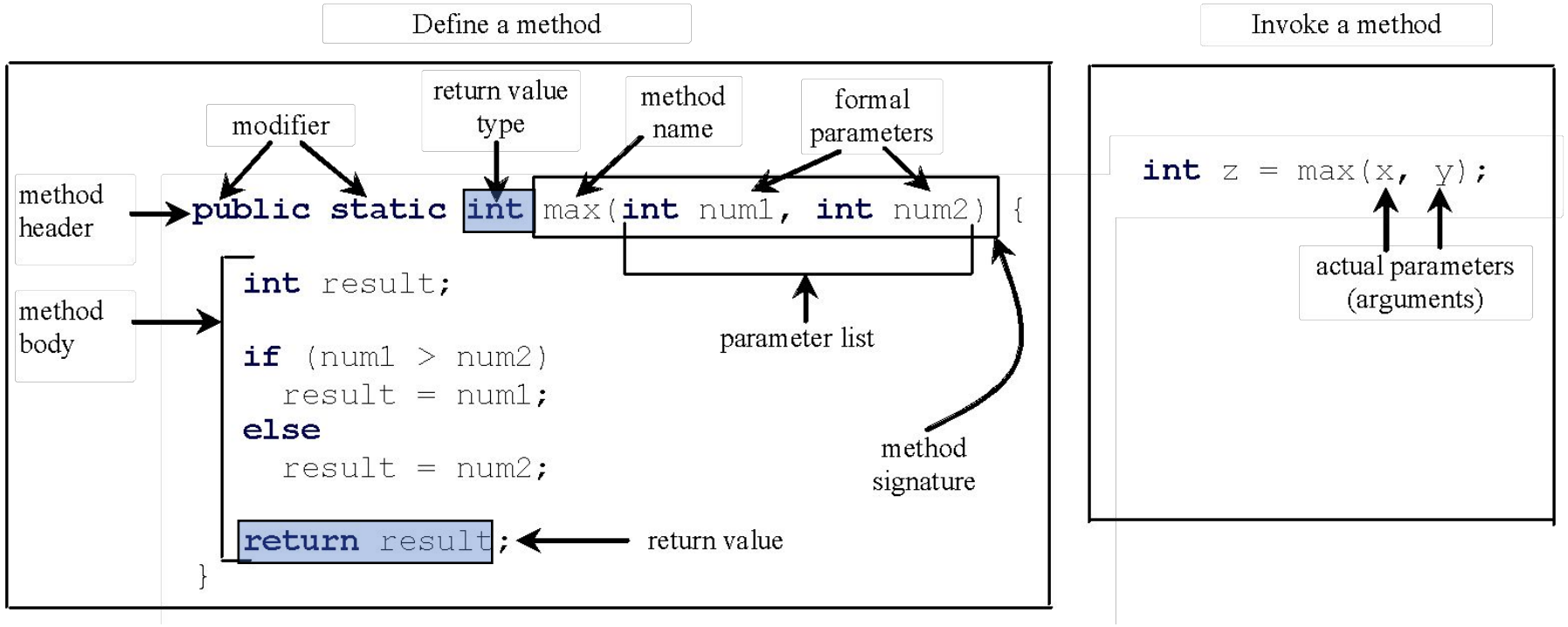
Actual Parameters

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



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



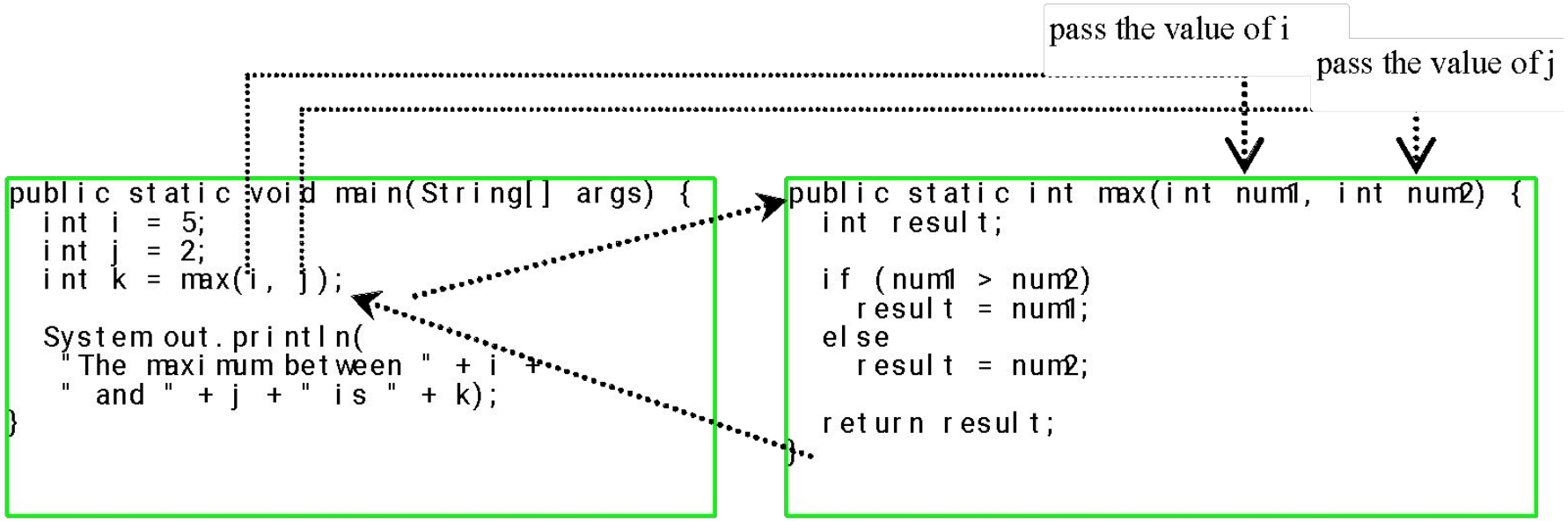
UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

- ❑ Terminology method modifiers
- ❑ Terminology method Signature
- ❑ Terminology method body vs method header
- ❑ Formal paramater vs actual parameter/argument
- ❑ Return value

Calling Methods



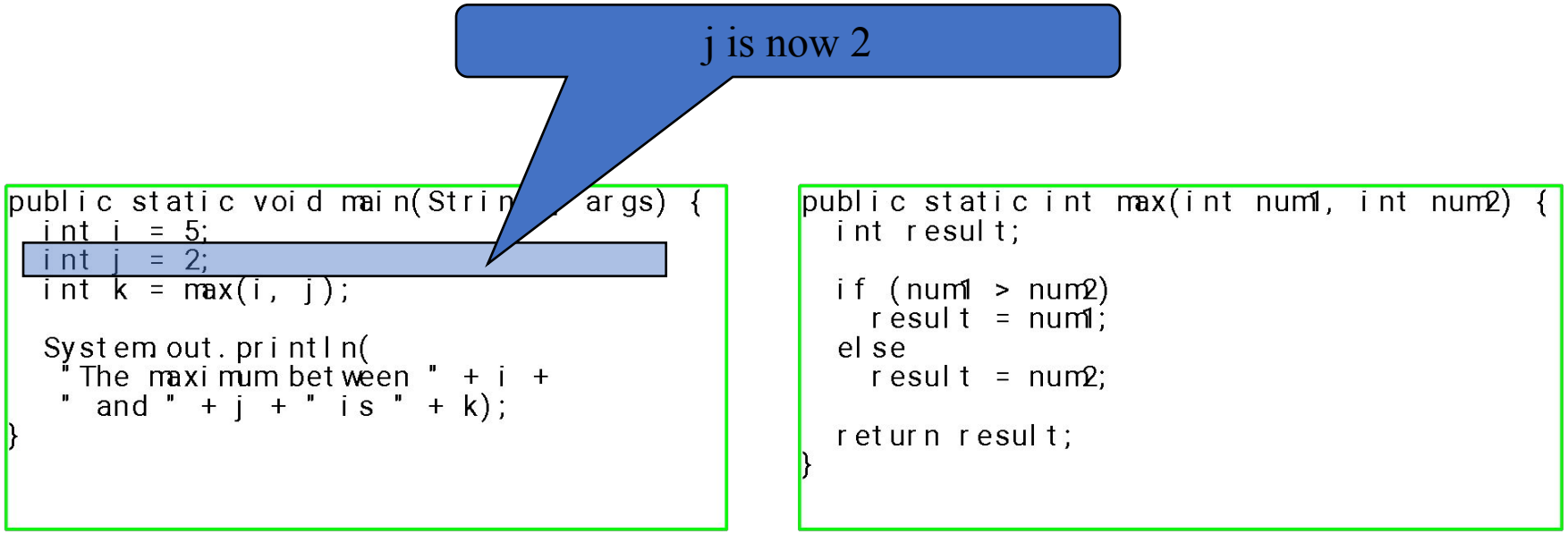
Trace Method Invocation

i is now 5

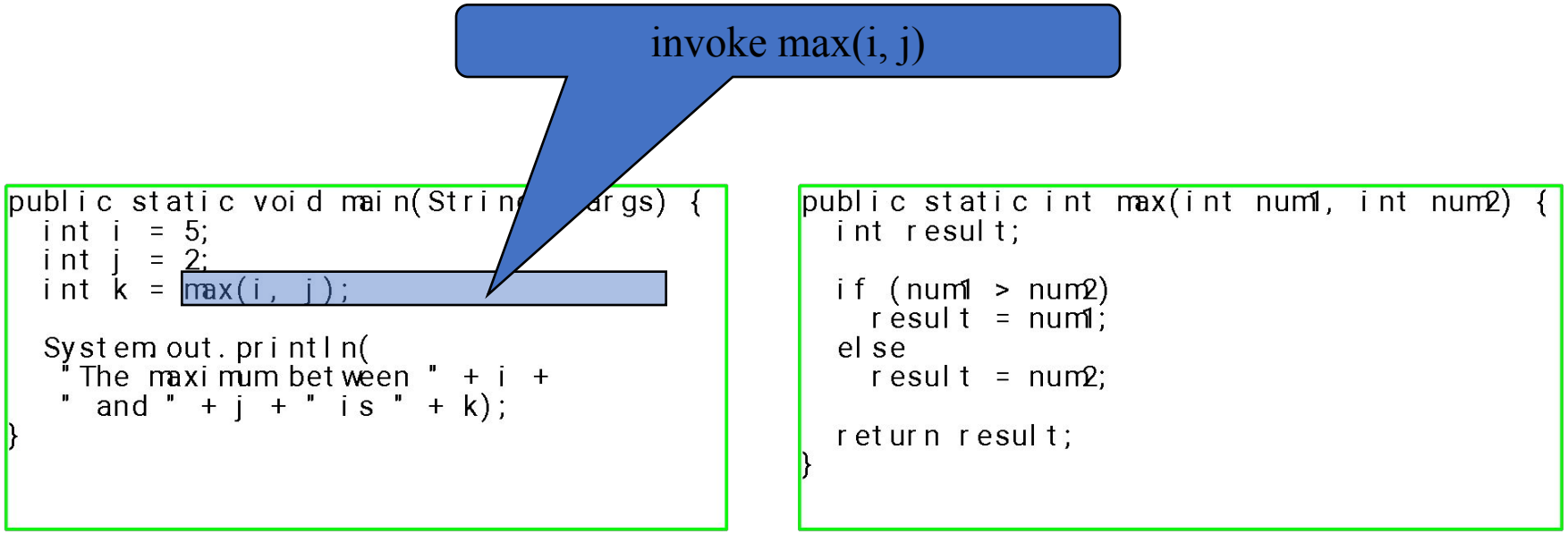
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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



Trace Method Invocation



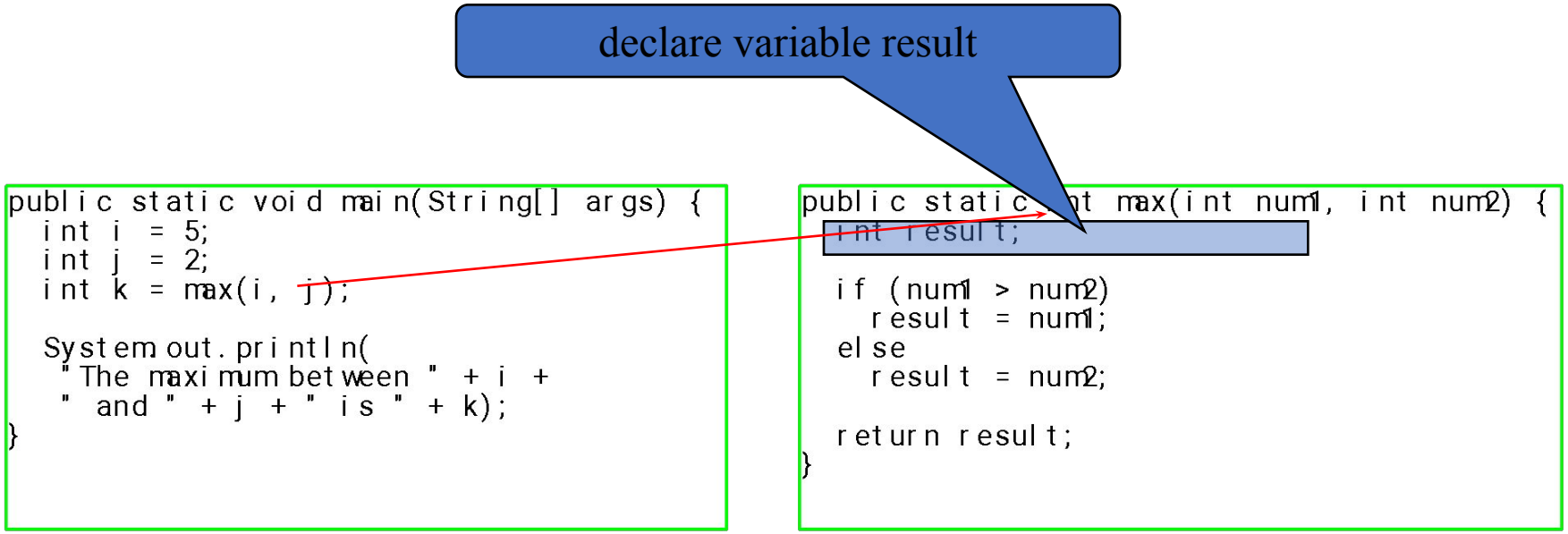
Trace Method Invocation

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;
}
```

Trace Method Invocation



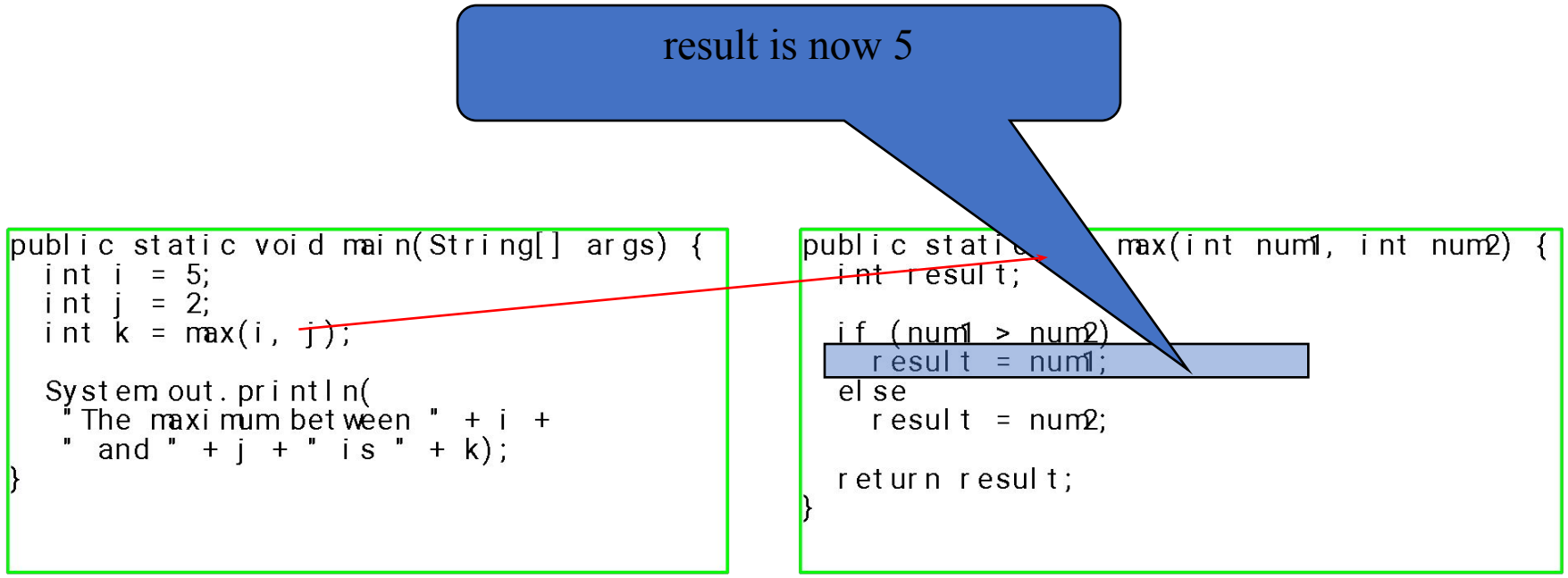
Trace Method Invocation

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

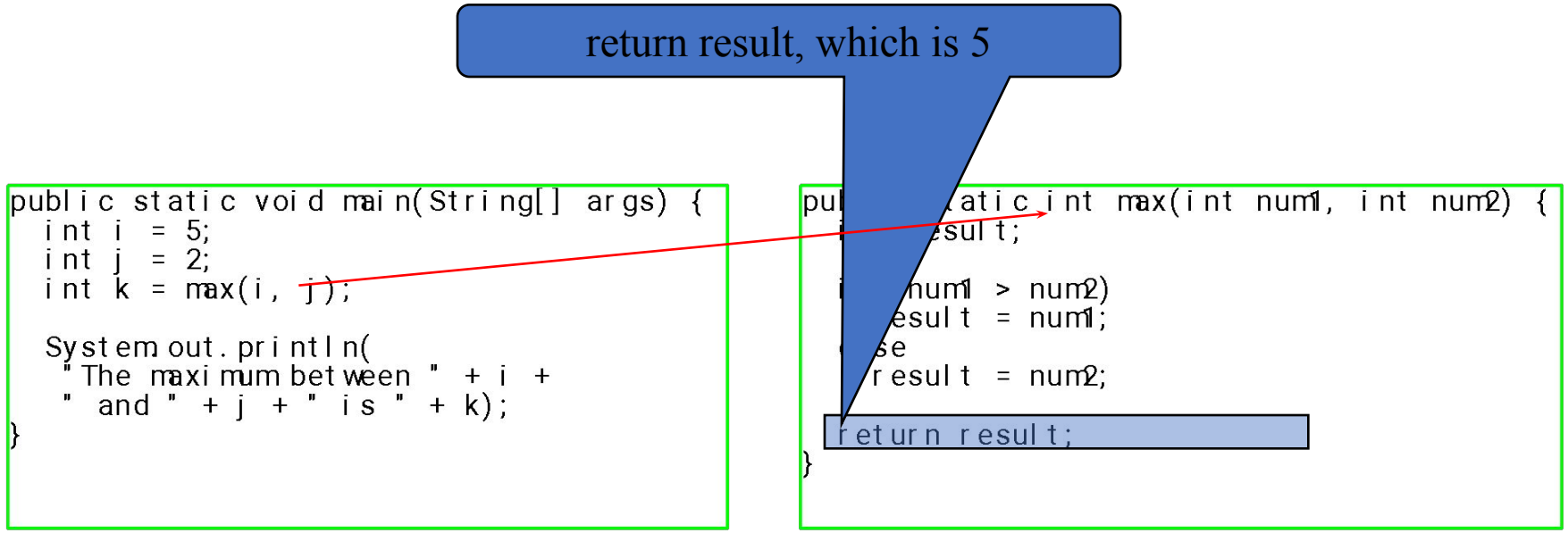
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

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

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

  return result;
}
```

Trace Method Invocation



Trace Method Invocation



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;
}
```

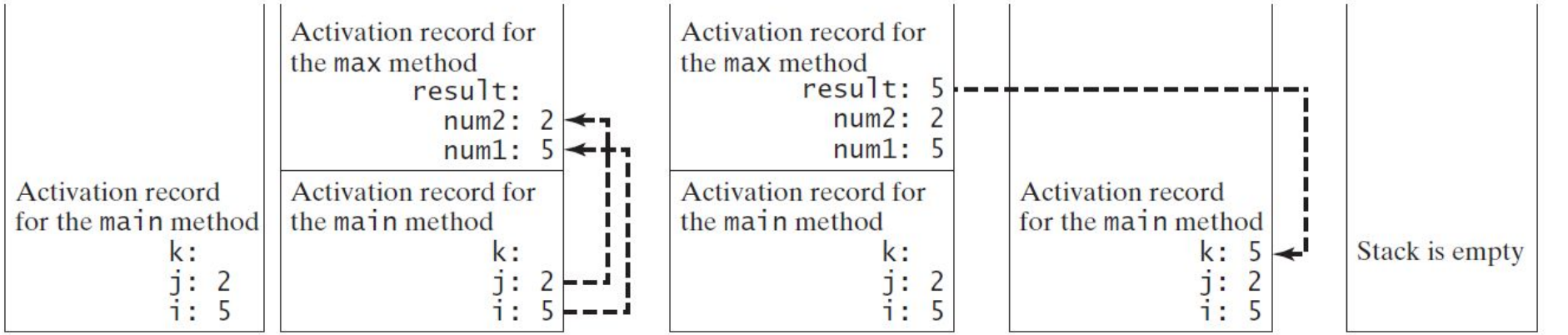
Trace Method Invocation

Execute the 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;
}
```

Call Stacks



(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`.

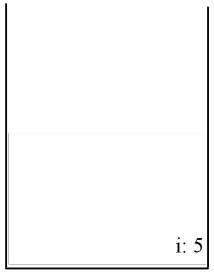
(e) The `main` method is finished.

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;
}
```



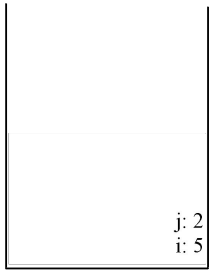
The main method is invoked.

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;
}
```



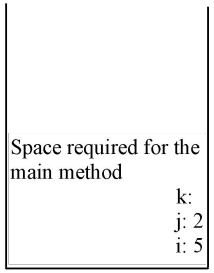
The main method is invoked.

Trace Call Stack

Declare k

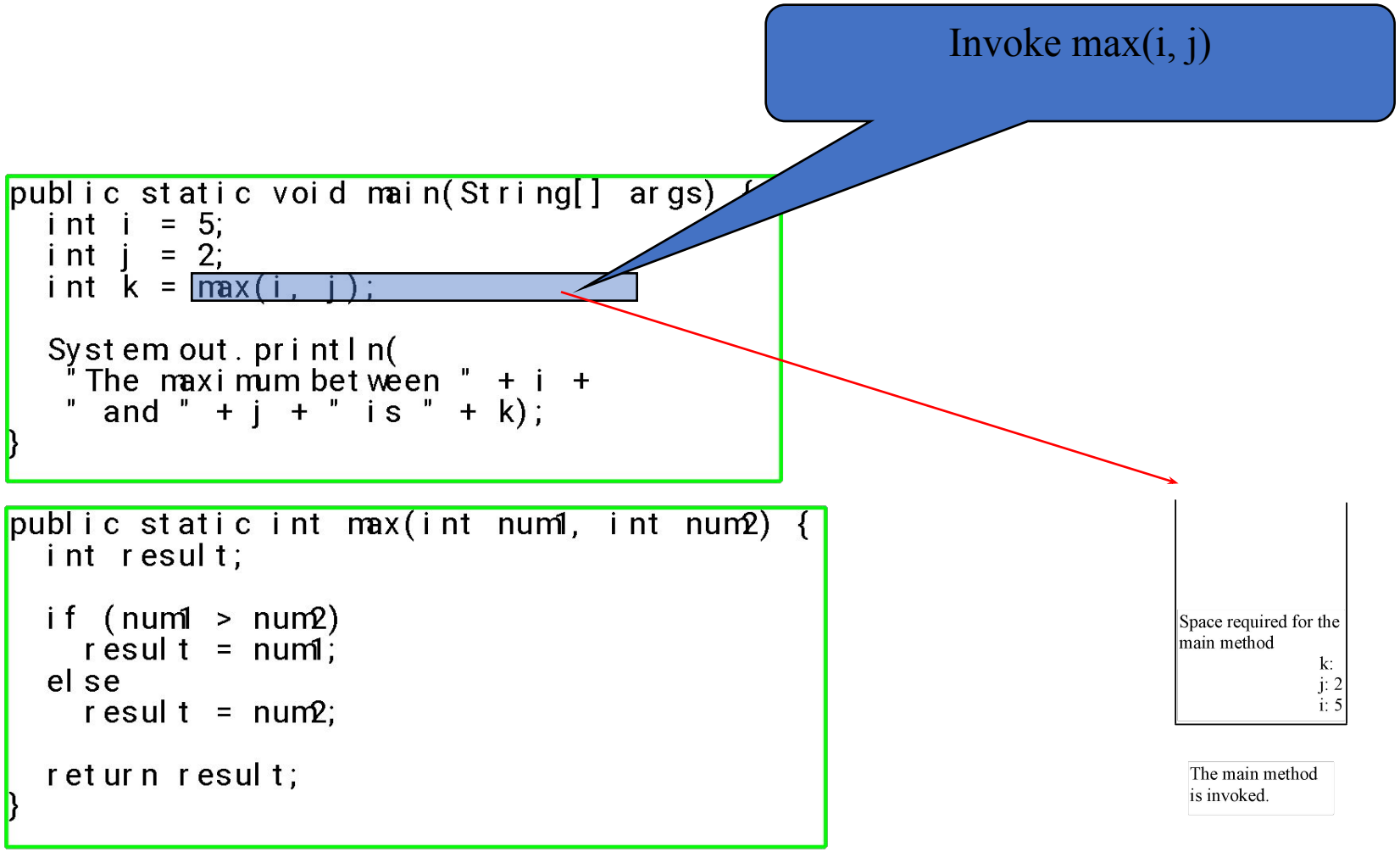
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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;
}
```



The main method is invoked.

Trace Call Stack

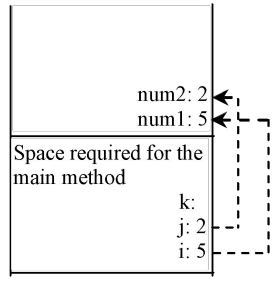


Trace Call Stack

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;
}
```



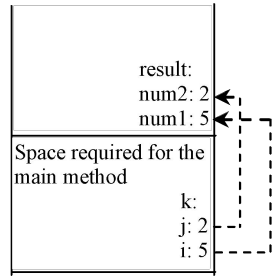
The max method is invoked.

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;
}
```

Declare result



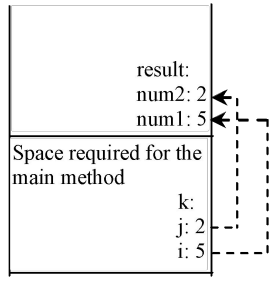
The max method is invoked.

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



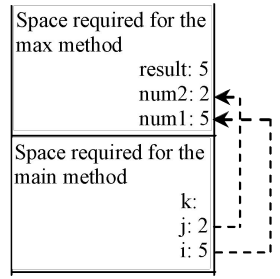
The max method is invoked.

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;
}
```

Assign num1 to result



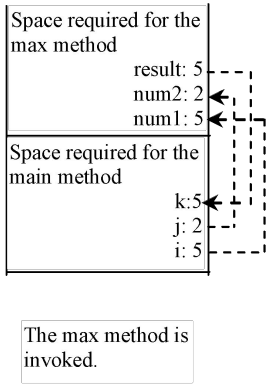
The max method is invoked.

Trace Call Stack

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;
}
```

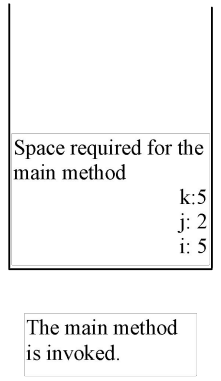


Trace Call Stack

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;
}
```



Reuse Methods from Other Classes

NOTE: One of the benefits of methods is for reuse. The max method can be invoked from any class besides TestMax. If you create a new class Test, you can invoke the max method using ClassName.methodName (e.g., TestMax.max).

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

- ❑ Calling Method
- ❑ Tracing method invocation
- ❑ Call stack
- ❑ Reuse method from other classes

void Method

This type of method does not return a value. The method performs some actions.

Passing Parameters

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

Suppose you invoke the method using

```
nPrintln("Welcome to Java", 5);
```

What is the output?

Suppose you invoke the method using

```
nPrintln("Computer Science", 15);
```

What is the output?

Can you invoke the method using

```
nPrintln(15, "Computer Science");
```

Modularizing Code

- ✔ Methods can be used to reduce redundant coding and enable code reuse.
- ✔ Methods can also be used to modularize code and improve the quality of the program.

Overloading Methods

Overloading the `max` Method

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

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;
  }
}
```

Scope of Local Variables

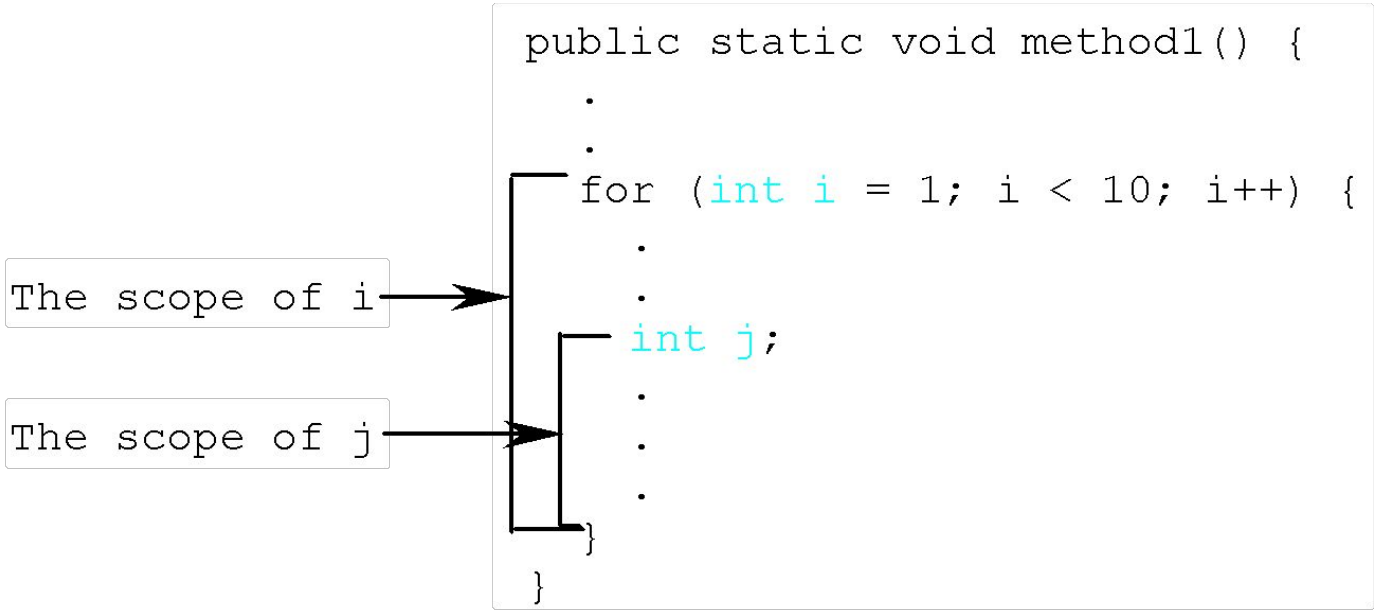
- **A local variable**: a variable defined inside a method.
- **Scope**: the part of the program where the variable can be referenced.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
- A local variable must be declared before it can be used.

Scope of Local Variables, cont.

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.

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.



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;
  }
}
```

Scope of Local Variables, cont.

```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Scope of Local Variables, cont.

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

- ❑ Void Method
- ❑ Overloading Method
- ❑ Ambiguous Invocation
- ❑ Scope of Local Variables

Recursion

Dasar-dasar Rekursif

- Definisi: Method rekursif adalah method yang dalam isi nya melakukan perintah pemanggilan method itu sendiri.
- Pola ini relatif umum pada matematik, misalnya pendefinisian Factorial.

$$n! = n * (n-1)!$$

- Terlihat definisi factorial (!) menggunakan factorial (!) lagi.
- Dalam beberapa kasus akan membuat program lebih *natural* dibuat.

Penyusunan Recursion

1. Tentukan ***base case***, kapan recursion nya berakhir.
   - Pada kasus factorial, basis nya adalah 0! = 1.
2. Tentukan ***recursive case***, parameter pemanggilan selanjutnya dan operasi lain bila dibutuhkan.
   a) Tentukan perubahan parameter yang dibutuhkan, misalnya apakah dikurangi atau ditambah parameter nya atau lainnya.
      - Pada kasus factorial, parameter-nya dikurangi.
   b) Tentukan operasi yang akan mengoperasikan
      - Pada kasus factorial, operasinya adalah dikalikan (*) yaitu: n * (n-1)!

Factorial: n! = n * (n-1)!

```
public class Factorial{
   public static int factorial(int bil){
         if (bil<=0 )
             return 1;
           return bil * factorial(bil-1);
   }
}
```

Faktor Persekutuan terBesar

```
public class FPB {
   public static int cariFPB(int bil1, int bil2){
      if (bil2==0)
         return bil1;
      return cariFPB(bil2, bil1%bil2);
   }
}
```

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

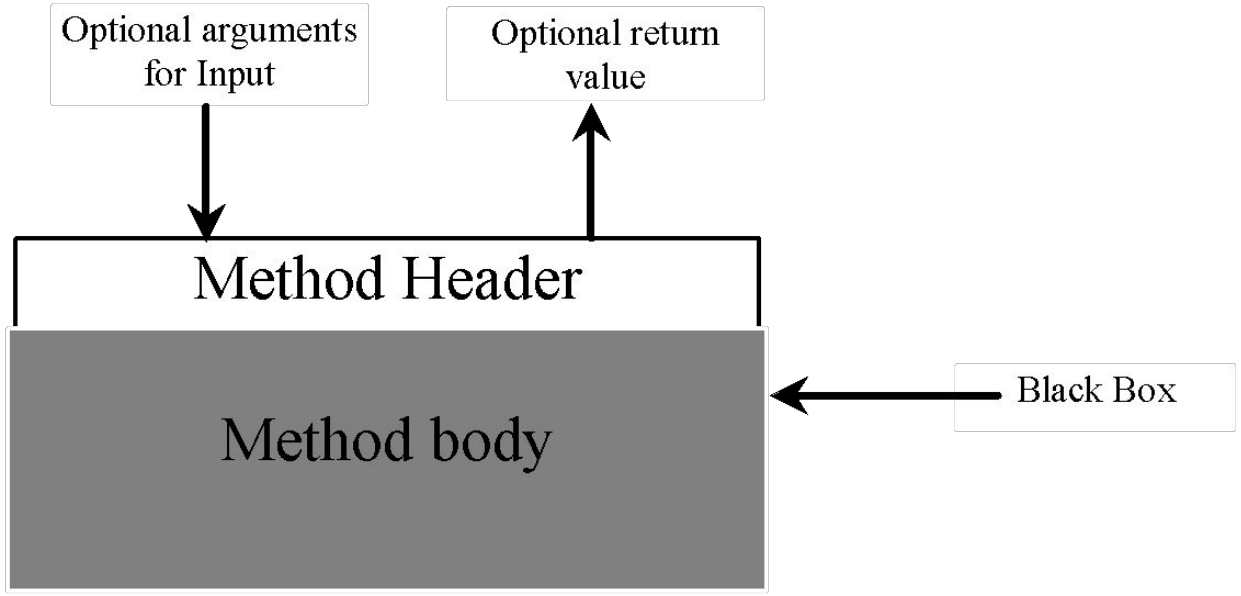
- Apakah rekursif?
- Better Problem Solving
- Top Down vs Bottom Up Refinement
- Benefit of abstraction, method modularity, Refinement

How to start coding?! (as a better problem solver)

- ✔ Beginning programmers often start by trying to work out the solution to every detail (*coding first, trial and error*).
- ✔ Although details are important in the final program, concern for detail in the early stages **may block the problem-solving process**.
- ✔ **To make problem solving flow as smoothly as possible**, we should using **method abstraction** to isolate details from design and only later implements the details.
- ✔ It also will help programmer to do ***Test Driven Development***

Method Abstraction

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



Benefits of Methods

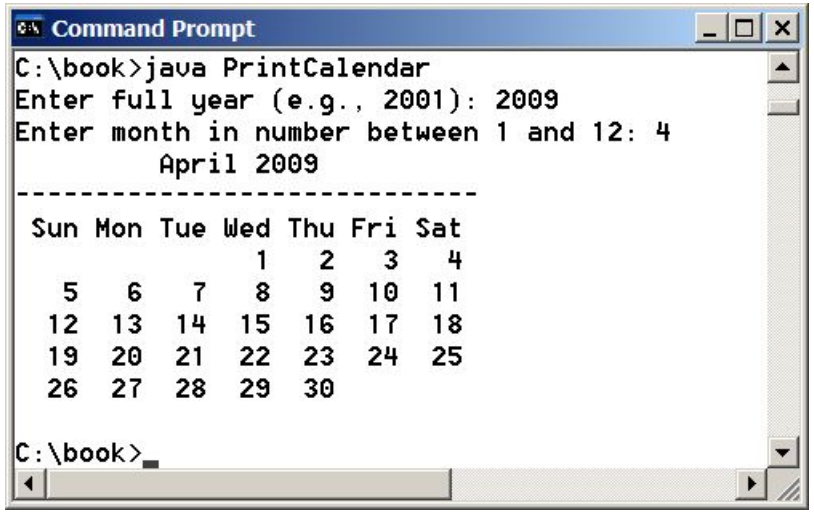
- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

Stepwise Refinement (Optional)

- 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.

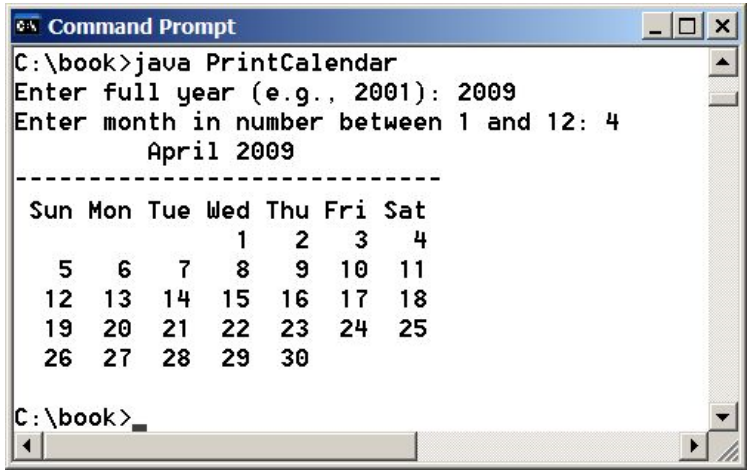
PrintCalender Case Study

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

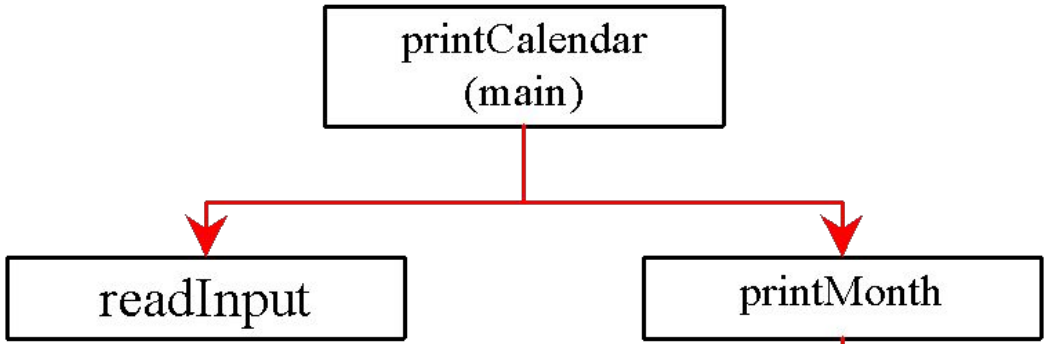


Design Diagram

printCalendar
(main)

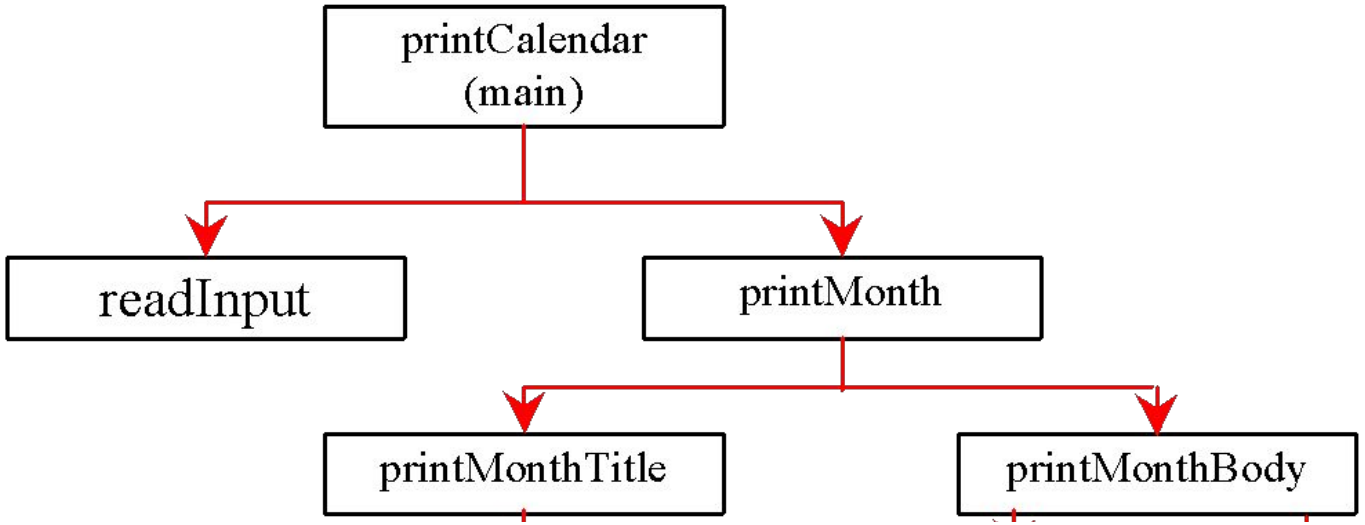


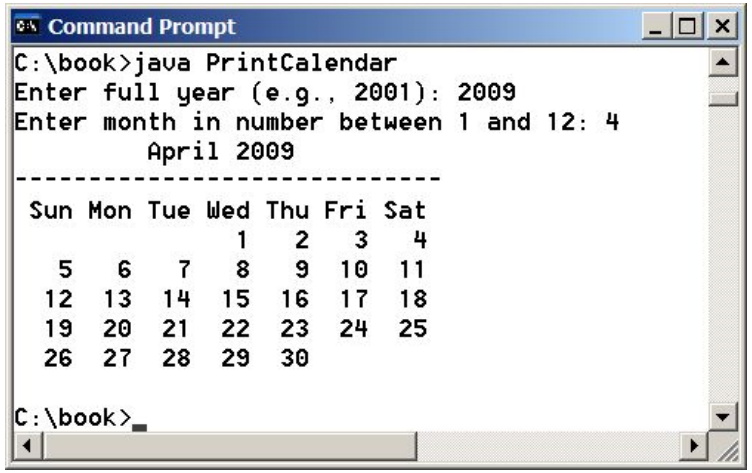
Design Diagram



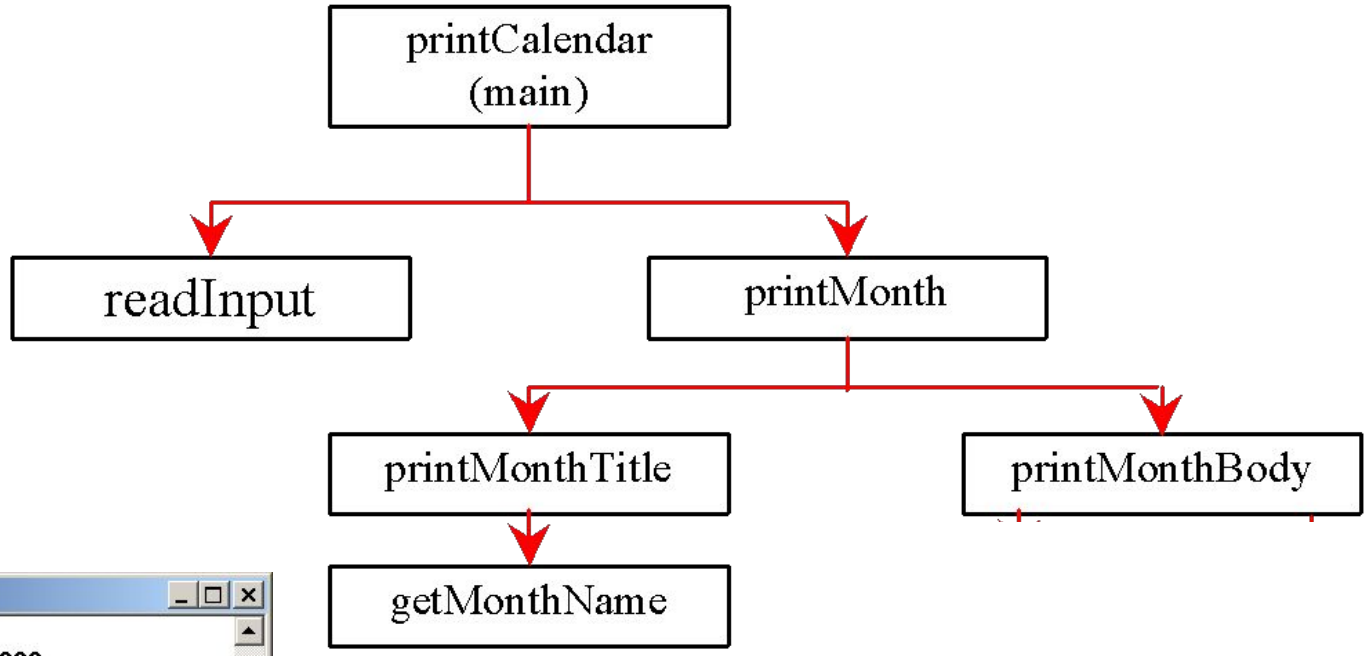
```
Command Prompt
C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
         April 2009
-----------------------------
 Sun Mon Tue Wed Thu Fri Sat
                   1   2   3   4
   5   6   7   8   9  10  11
  12  13  14  15  16  17  18
  19  20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

C:\book>_
```

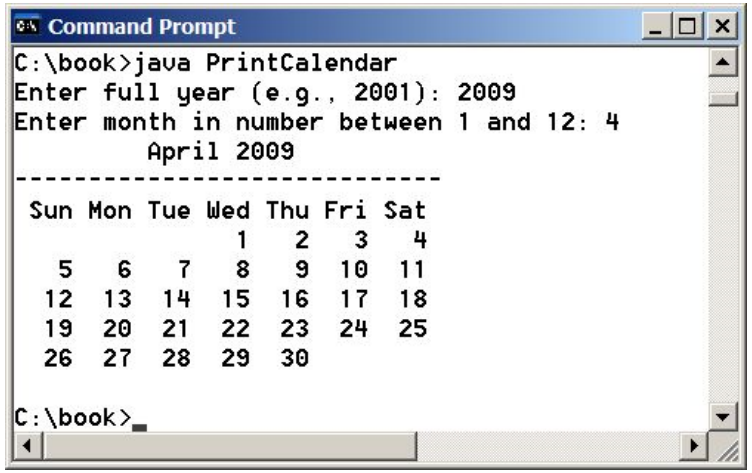
Design Diagram



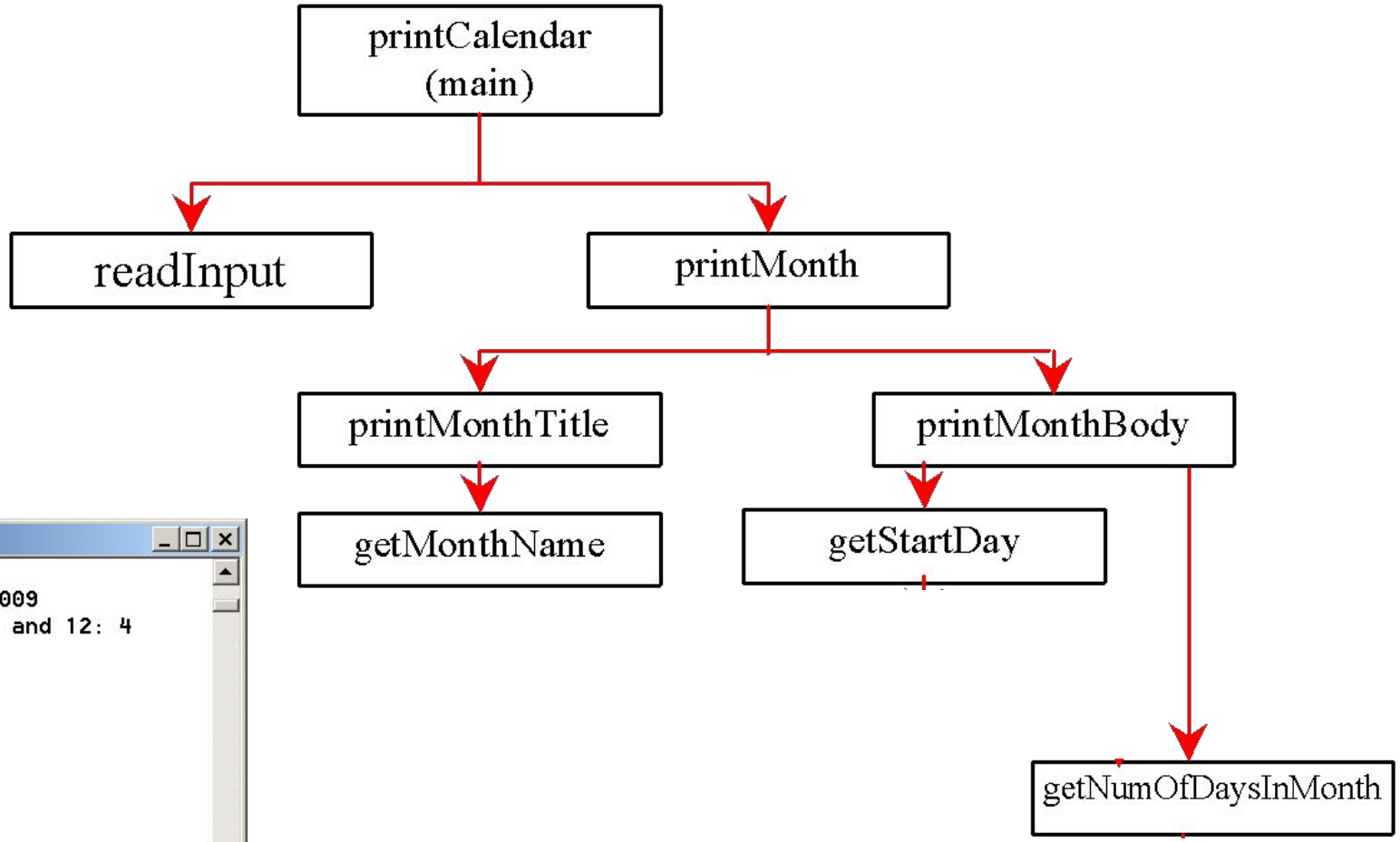


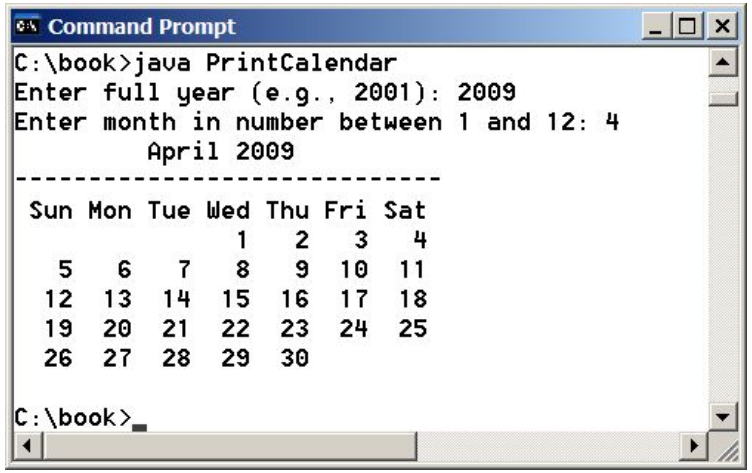
Design Diagram



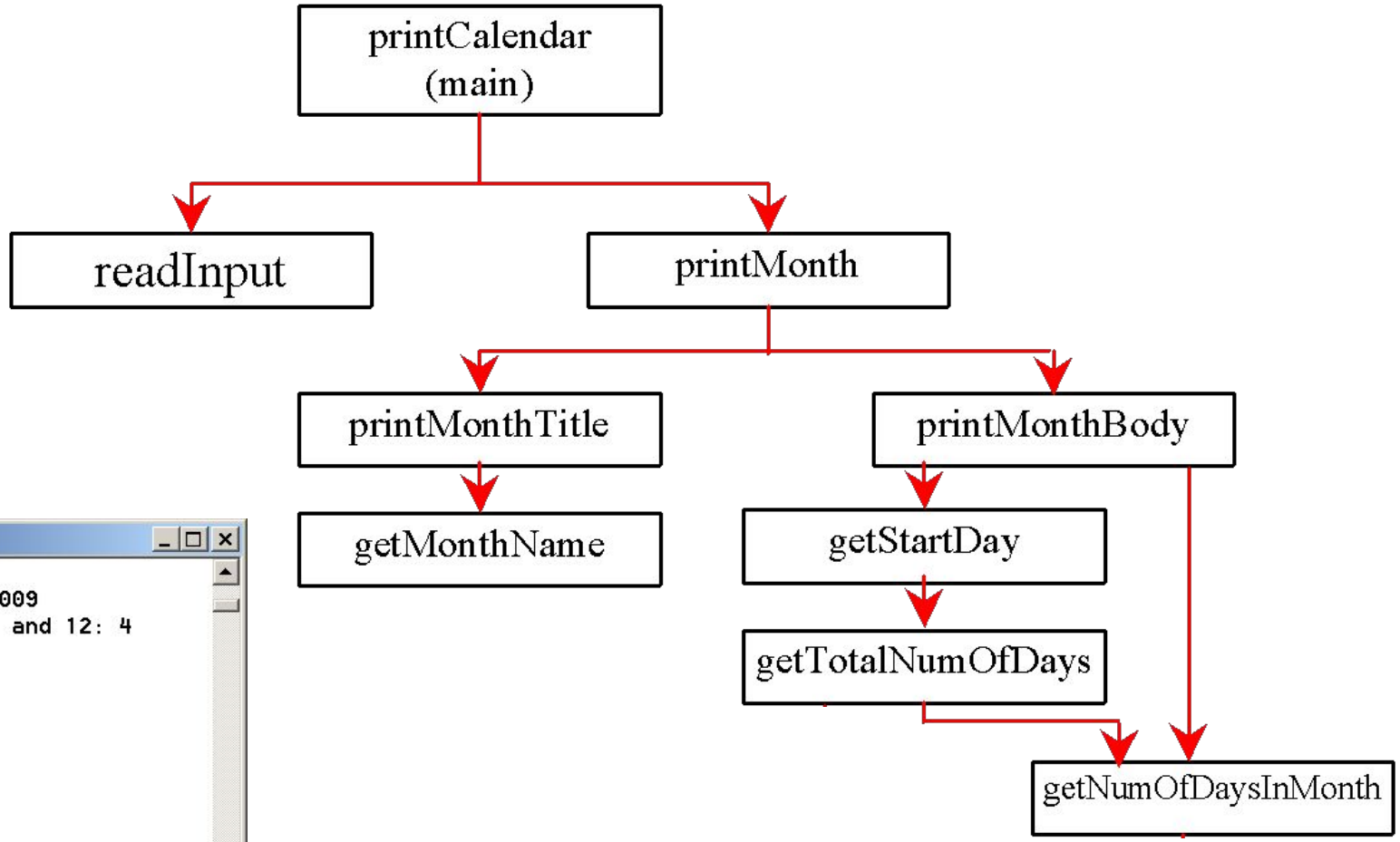


Design Diagram



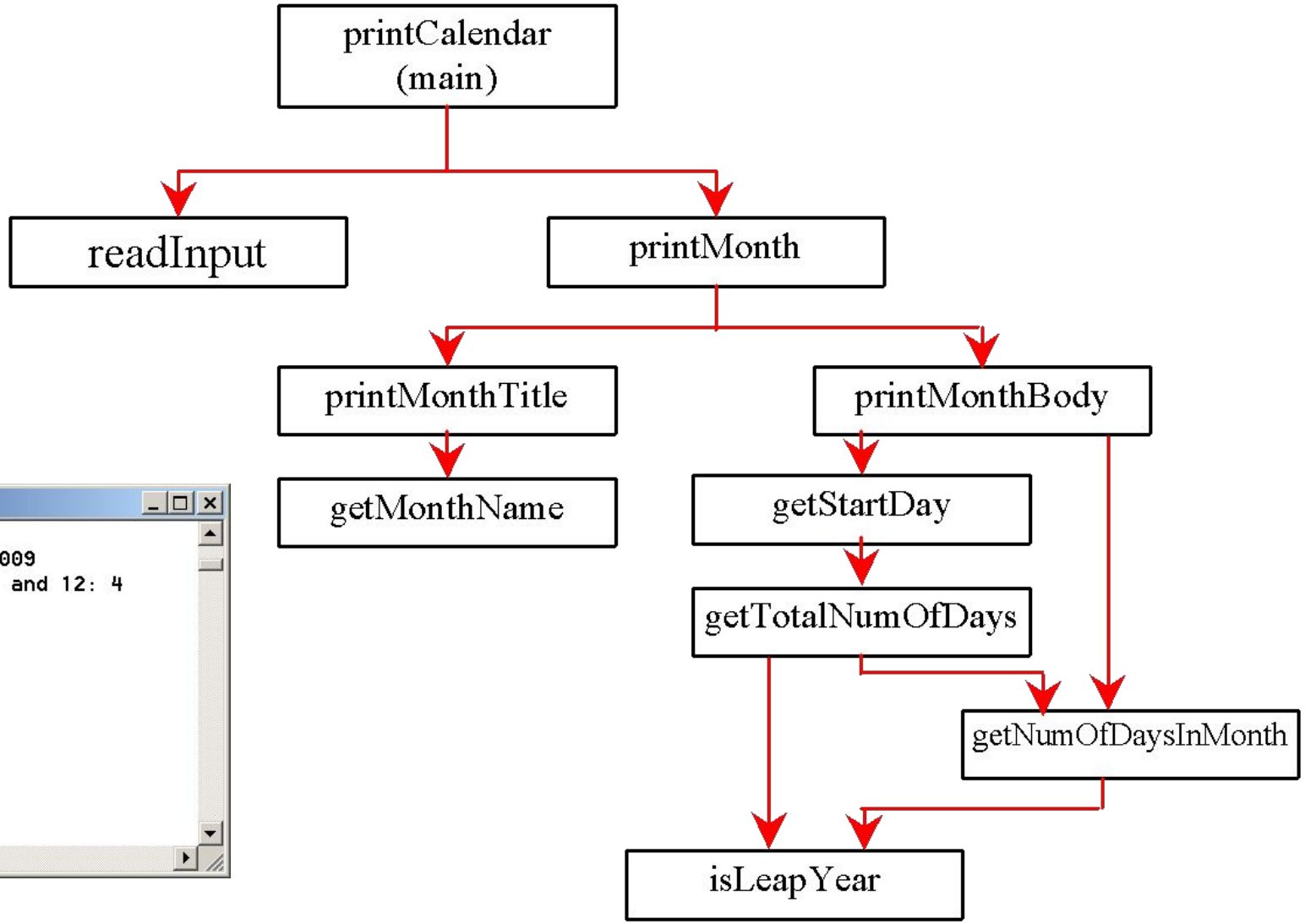


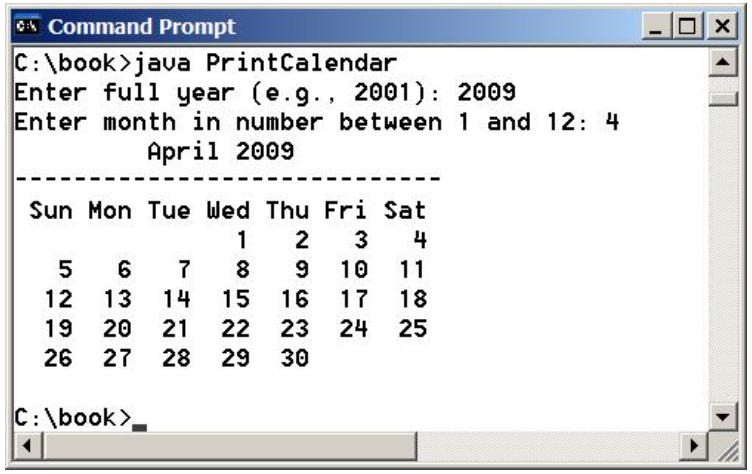
Design Diagram




```
Command Prompt
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

C:\book>
```

Design Diagram





Implementation: Top-Down

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

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 it.
- ✔ 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.
- ✔ Sometimes, they can be used together.

Benefits of Stepwise Refinement

- ✔ Simpler Program
- ✔ Reusing Methods
- ✔ Easier Developing, Debugging, and Testing
- ✔ Better Facilitating Teamwork

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

- Abstraction: Method header vs. Method body
- Better Problem Solving
- Top Down vs Bottom Up Refinement
- Benefit of abstraction, method modularity, Refinement

Why Testing is Necessary

Other requirements:

- You will have a general working knowledge of architecture.
- You are able to execute a mid-complexity task.
- As a Software Engineer, you must have the ability to do Unit Testing.

Less

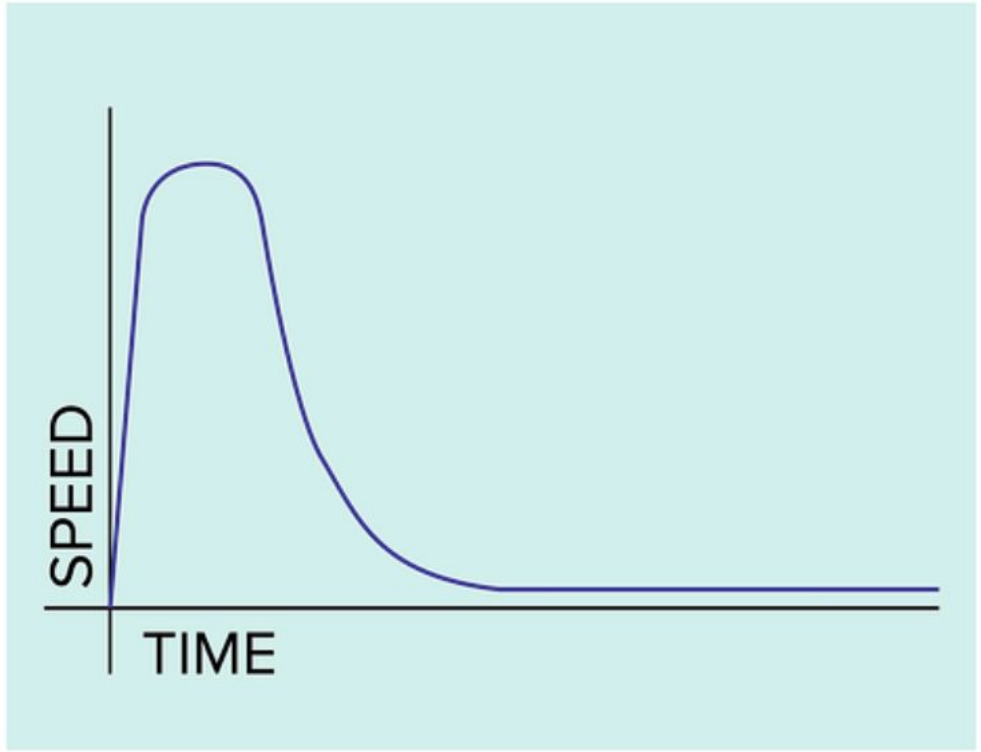
Junior

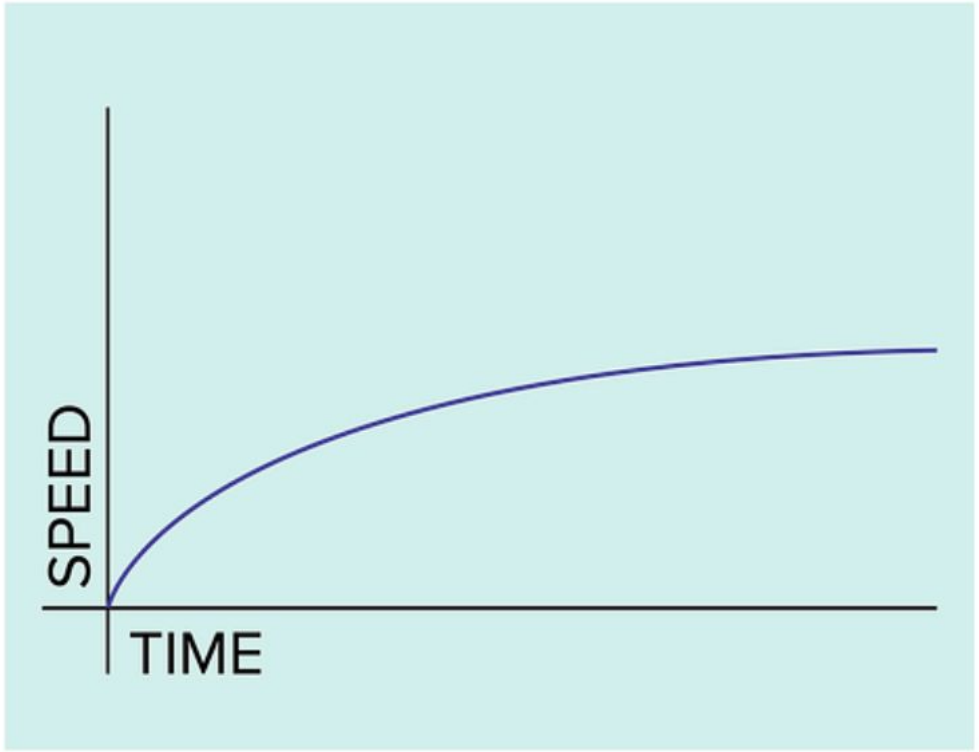
Staff

The Motivation of Test Driven Development

Motivation

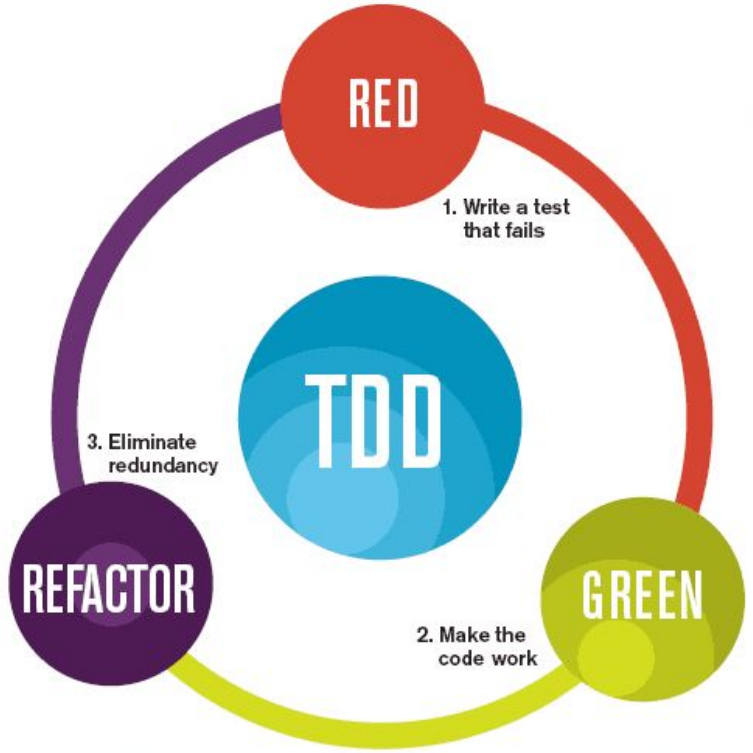
- Developing software in a continuous work, not just one time
- These graphs show performance speed over time.
- Which one you prefer?





What is TDD?

- Test-Driven Development (TDD) is a technique for building software that guides software development by writing tests.
- It was developed by Kent Beck in the late 1990's as part of Extreme Programming.

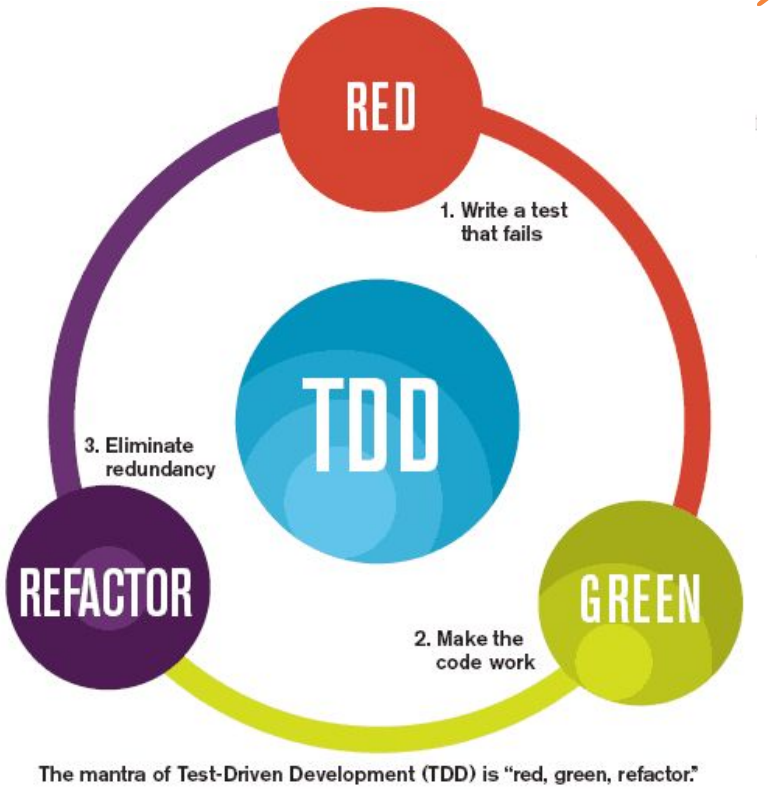


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Image Source:
https://www.nimblework.com/agile/test-driven-development-tdd/

What is TDD?

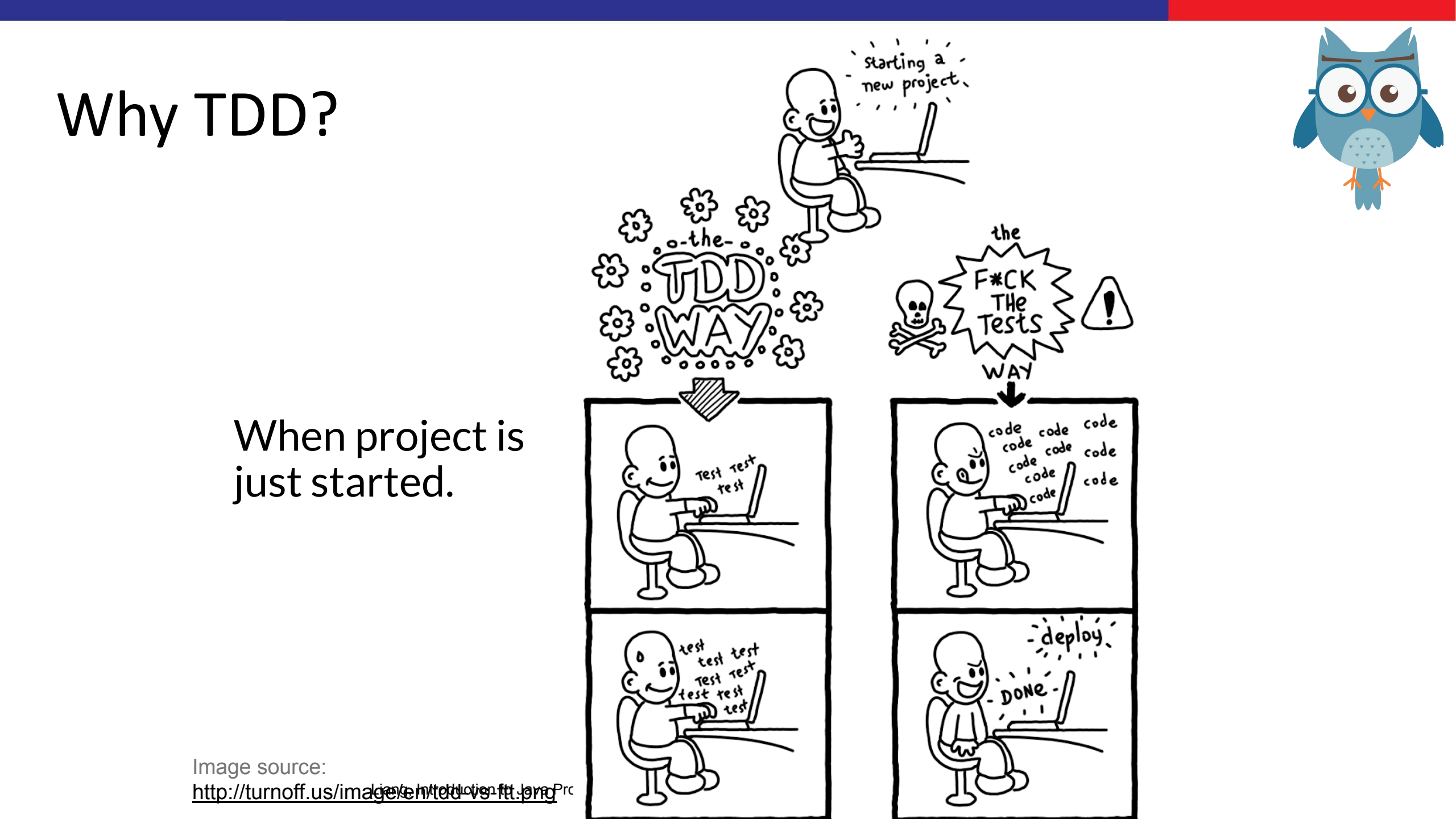
- In essence you follow three simple steps repeatedly:
  - (Red) **Write a test** for the next bit of functionality you want to add.
  - (Green) **Write the functional code** until the test passes.
  - (Refactor) **Refactor** both new and old code to make it well structured.



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Image Source:
https://www.nimblework.com/agile/test-driven-development-tdd/

Why TDD?



When project is just started.

Image source:
http://turnoff.us/image/en/tdd-vs-ftt.png

When project continues,

with addition of features and bugs fixing…..

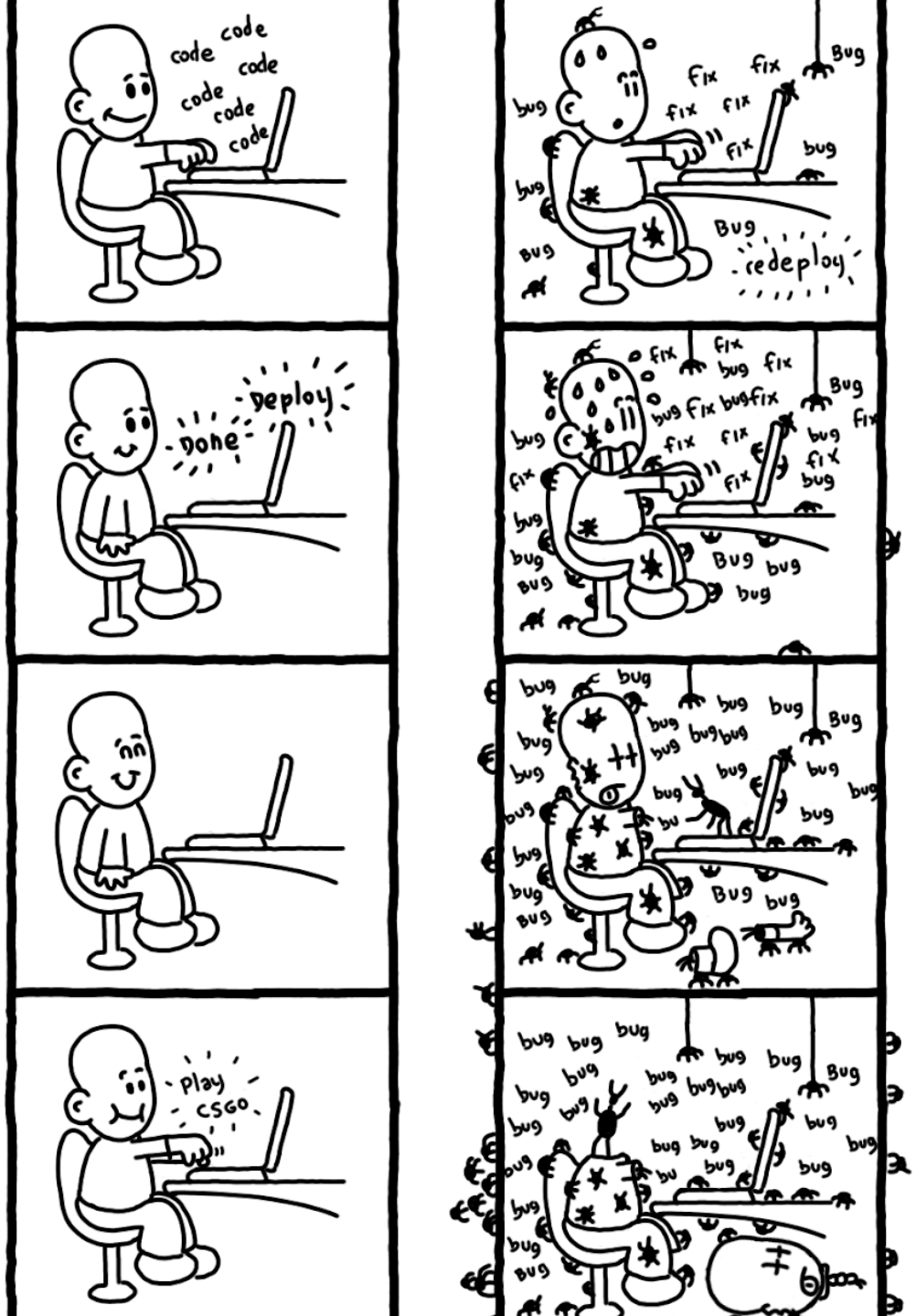
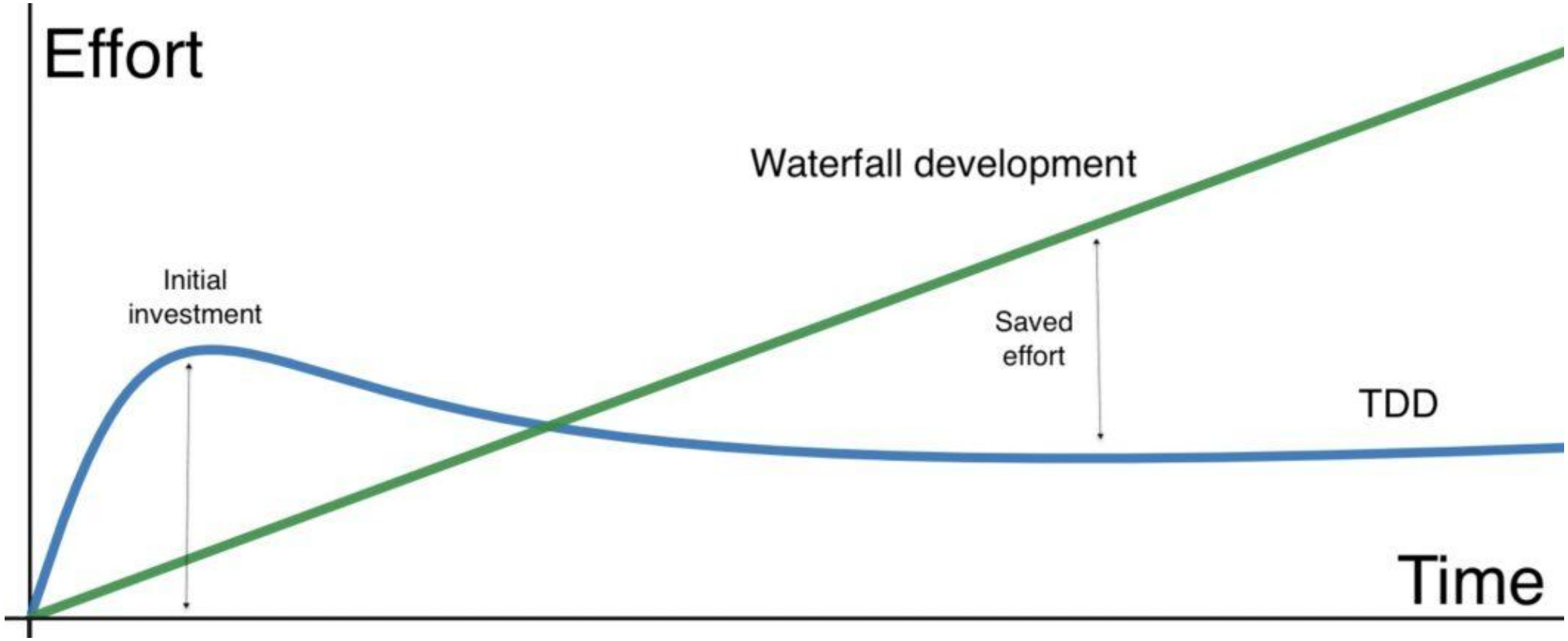


Image source:
http://turnoff.us/image/en/tdd-vs-ftt.png

TDD in continues development

- ✔ Developing software is a continues development.
- ✔ Regularly we add additional improvement to the software.
- ✔ The improvement should be done *fast forever*, every time we need it.
  - Go Fast Forever
  - Clean Code
  - Refactoring
  - Confidence
  - Tests / TDD

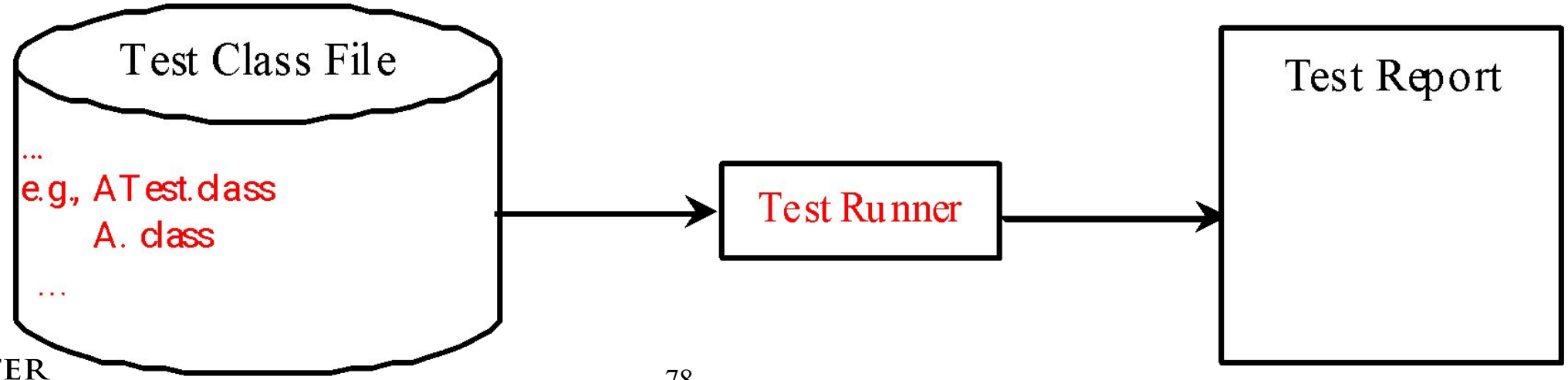
Why TDD?


Effort
Waterfall development
Initial
investment
Saved
effort
TDD
Time

How to do testing in Java

JUnit Basics

- *JUnit* is the de facto framework for testing Java programs.
- JUnit is a third-party open source library packed in a jar file.
- The jar file contains a tool called *test runner*, which is used to run test programs.
- Suppose you have a class named A.
- To test this class, you write a test class named ATest.
- This test class, called a *test class*, contains the methods you write for testing class A.
- The test runner executes ATest to generate a test report



Obtaining and Running JUnit

You will see how JUnit works from an example. To create the example, first you need to download JUnit from http://sourceforge.net/projects/junit/files/.
Download this file to c:\book\lib and add it to the classpath environment variable as follows:

set classpath=.;%classpath%;c:\book\lib\junit-4.10.jar

To test if this environment variable is set correctly, open a new command window, and type the following command:

java org.junit.runner.JUnitCore

Configuring junit in VSCode

- Download JUnit from http://sourceforge.net/projects/junit/files/.
- put it in the lib directory of your vscode workspace
- by default testrunner plugin has been installed in vscode
- if you have a source code with @Test annotation, the plugin will automatically recognize it.

```
2   import static org.junit.Assert.*;
3   import org.junit.Test;
4
    Run Test | Debug Test | × | no references found for MaxArrayTest
5   public class MaxArrayTest {
6
7
        1 reference
8       @Test
        Run Test | Debug Test | ✓
9       public void testFindMaxPositif(){
```

Unit Test

Membuat unit testing tidak lah sulit, kita hanya perlu menuliskan apa yang diharapkan dari sebuah fungsi yang kita buat.

Misalkan kita perlu membuat program menghitung FPB (Faktor Persekutuan Terbesar).

Apa yang dilakukan?

```
@Test
public void checkFPB_case0(){
        assertEquals(21, FPB.cariFPB(63, 42));
}
```

A JUnit Test Class

To use JUnit, create a test class. By convention, if the class to be tested is named A, the test class should be named ATest. A simple template of a test class may look like this:

```
import org.junit.*;
import static org.junit.Assert.*;
public class ATest {
  @Test
  public void m1() {
    // Write a test method
  }
  @Test
  public void m2() {
    // Write another test method
  }
  @Before
  public void setUp() throws Exception {
    // Common objects used by test methods may be set up here
  }
}
```

Example: MaxArray

Simple Example to calculate the maximum element of an integer array

```
public class MaxArray {
   public static int findMax(int arr[]){
      int max=0;
      for(int i=1;i<arr.length;i++){
         if(max<arr[i])
            max=arr[i];
      }
   return max;
   }
}
```

Test the MaxArray

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MaxArrayTest {

    @Test
    public void testFindMaxPositif(){
        assertEquals("Gagal Pemeriksaan Positif",
                     4, MaxArray.findMax(new int[]{1,3,4,2}));
    }

    @Test
    public void testFindMaxNegatif(){
        assertEquals("Gagal Pemeriksaan Negatif",
                     -1, MaxArray.findMax(new int[]{-12,-1,-3,-4,-2}));
    }
}
```

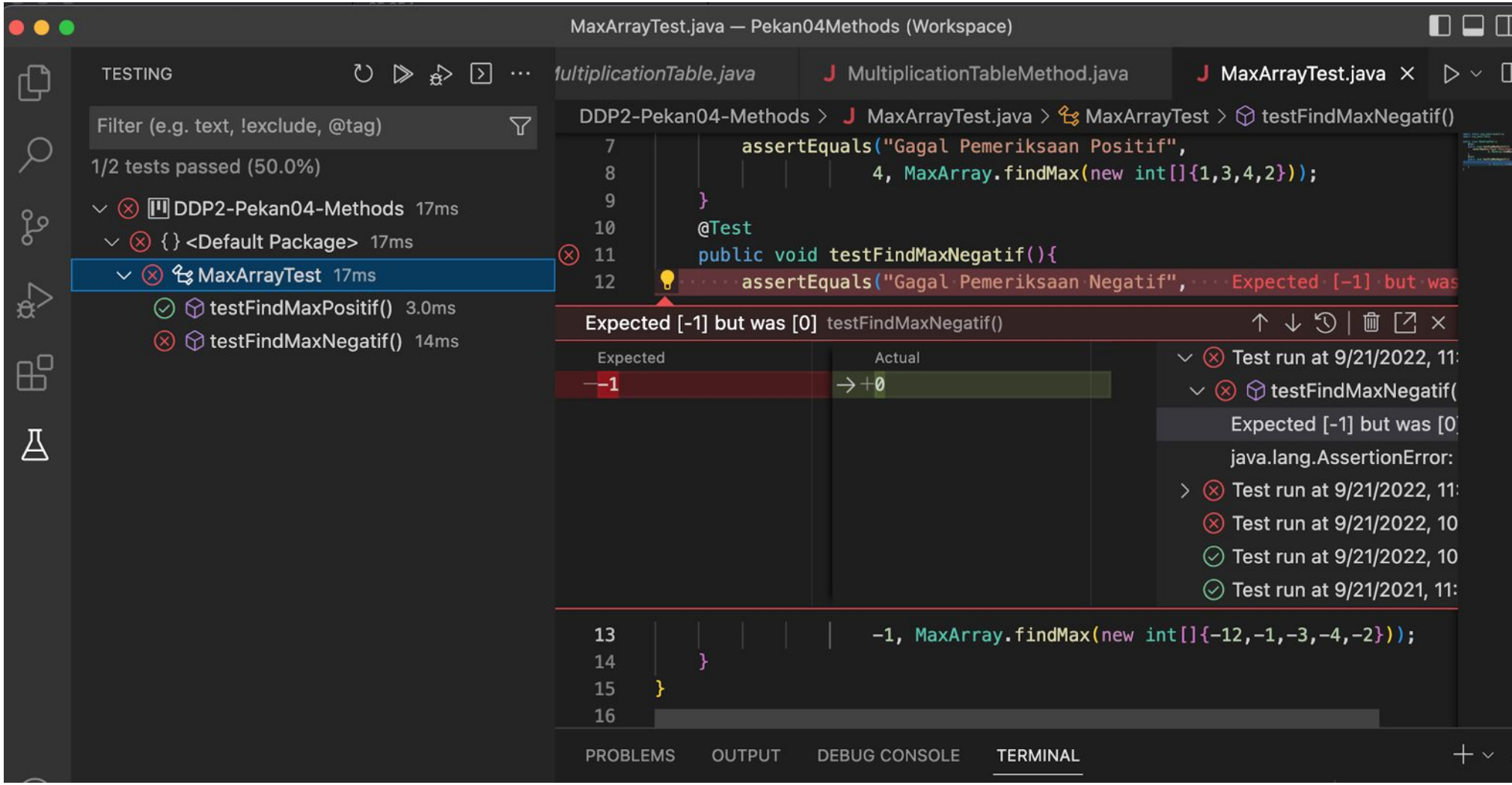
Run the Test
(later when you need to apply CI/CD in server)

To run the test from the console, use the following command:

java org.junit.runner.JUnitCore mytest.ATest

Run the Test in VSCode

To run the test in **VScode**, the click the run test on the ***codelens*** information:



UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS
ILMU
KOMPUTER

Selamat Berlatih!

Perhatikan lagi List Objective yang perlu dikuasai pekan ini.
Baca buku acuan dan berlatih!
Bila masih belum yakin tanyakan ke dosen, tutor atau Kak Burhan.

Semangat !