FAKULTAS
ILMU
KOMPUTER

Methods

Dasar – Dasar Pemrograman 2

Dinial Utami Nurul Qomariah

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Credits

- Liang, Introduction to Java Programming, 11th Edition, Ch. 2
- Downey & Mayfield, Think Java: How to Think Like a Computer Scientist, Ch. 2
- Slide Kuliah Dasar-dasar Pemrograman 2 Semester Genap 2021/2022

Motivations

Find the sum of integers

1. from 1 to 10,
2. from 20 to 30, and
3. from 35 to 45, respectively.

Solution → Problem

```
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);
```

Solution → Problem

```
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);
```

Perhatikan bahwa bagian-bagian ini berisi langkah yang serupa.
Yang membedakan hanyalah nilai variabel perulangan i.

Solution without Problem

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

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));
}
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

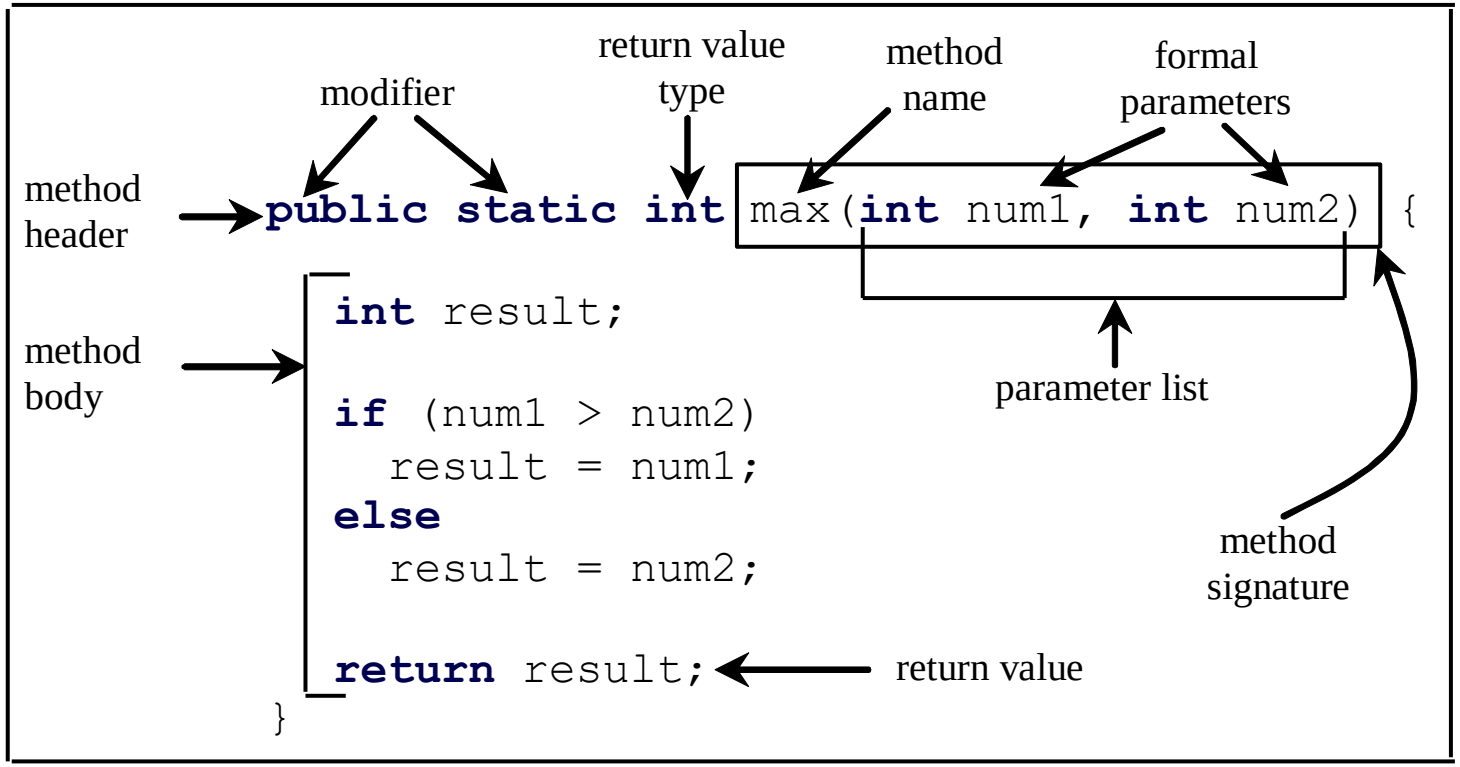
FAKULTAS ILMU KOMPUTER

Methods

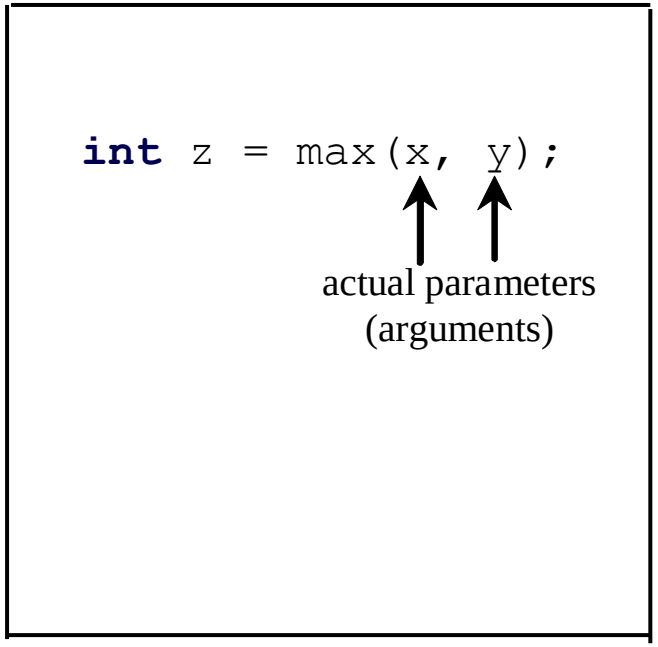
Method Component

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

Define a method



Invoke a method



UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Method

1

```
//Method tanpa return tanpa parameter
   public static void Halo(){
      System.out.println("Halo Apa Kabar");
   }
```

2

```
// method dengan return tanpa parameter
   public static double Hitung(){
      double Luas = 0.0;
      double Jari = 10.0;
      Luas = Math.PI * Jari * Jari;
      System.out.println("Luas Area Lingkaran : " +
Luas);
      return Luas; }
```

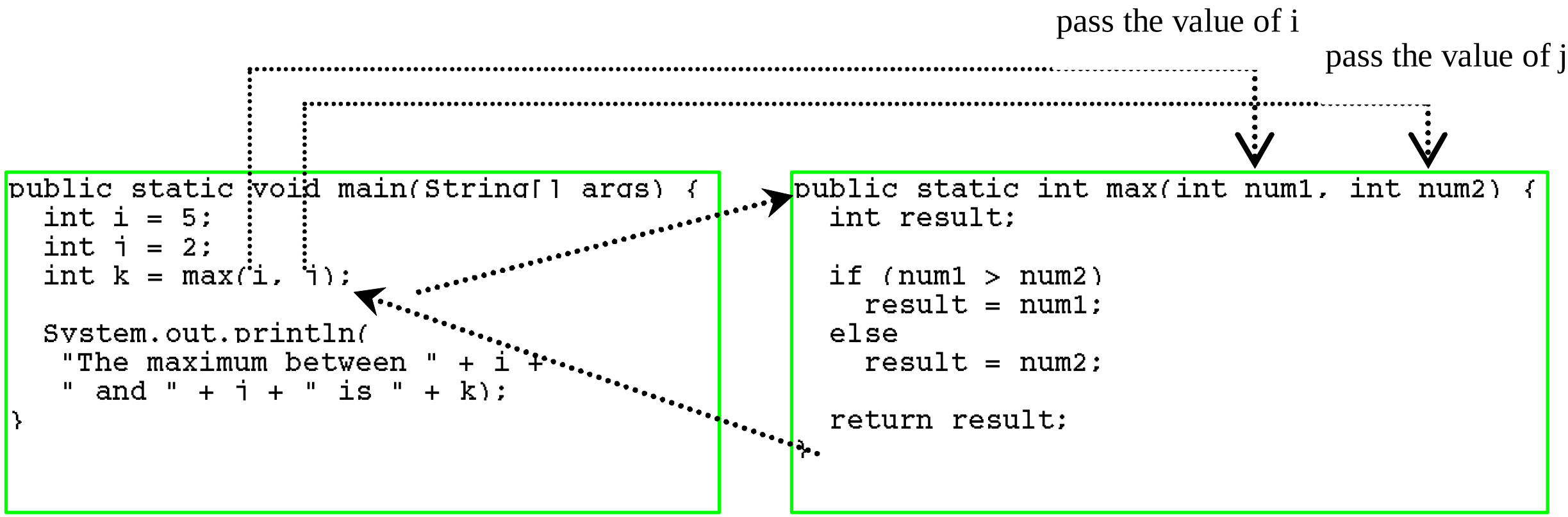
3

```
// Method tanpa return dengan parameter
   public static void IsiNama(String nama){
      System.out.println("Nama : " + nama);
   }
```

4

```
// method dengan return dengan parameter
   public static double Hitung2(double Jari2){
      double Luas2 = 0.0;
      Luas2 = Math.PI * Jari2 * Jari2;
      System.out.println("Luas Area Lingkaran : " +
Luas2);
      return Luas2;  }
```

Calling Method



Method Void

- Method Void → tidak mengembalikan Nilai
- Void (hampa), → kosong, sehingga tidak perlu membuat sebuah nilai return didalam method void.

Kosong/
Return 0;

```
public static void main(String[] args){
    Halo();
    IsiNama(“Maemunah");
    Hitung();
    Hitung2(100.0);
}
//Method tanpa return tanpa parameter
public static void Halo(){
    System.out.println("Halo Apa Kabar");
}
// Method tanpa return dengan parameter
public static void IsiNama(String nama){
    System.out.println("Nama : " + nama);
}
```

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");

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?

```
Welcome to Java
Welcome to Java
Welcome to Java
```

Suppose you invoke the method using
nPrintln("Computer Science", 15);
What is the output?

```
Computer Science
Computer Science
```

Can you invoke the method using
nPrintln(15, "Computer Science");

```
TestMethod.java:7: error: incompatible types: int cannot be converted to String
                nPrintln(2, "Computer Science");
                         ^
```

Call Stacks

Call Stack adalah urutan pemanggilam method di dalam Java.

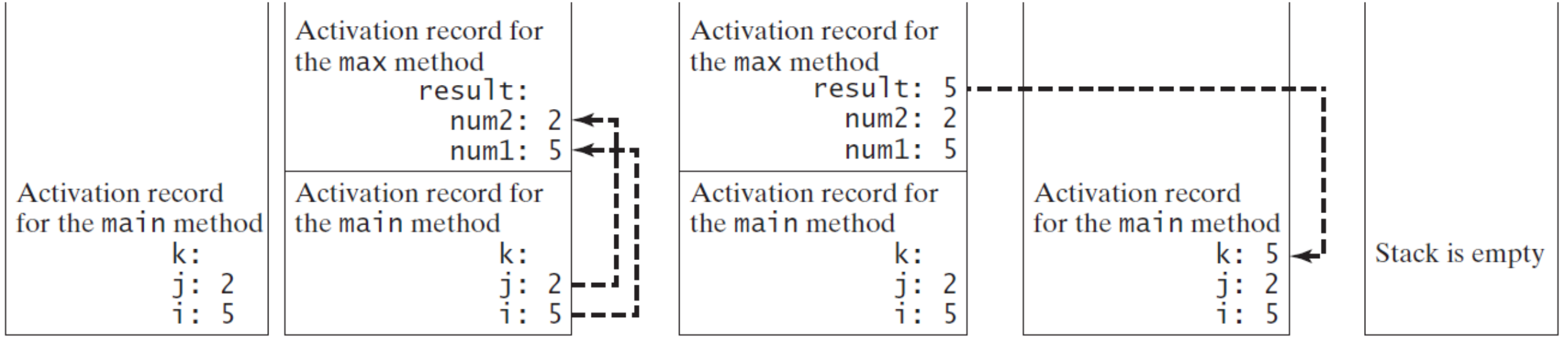
Pertama kali tentu saja method main.

Kemudian dari method main ini memanggil method lain lalu method lain tersebut memanggil method yang lain lagi,

begitu seterusnya sehingga pemanggilan methodnya bertumpuk.

Setelah method selesai dieksekusi, activation record untuk method tersebut akan dibuang dari call stack.

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.

Overloading Methods

- Overloading methods enable you to define the methods with the **same name** as long as their **parameter lists are different**.
- Two ways to overload a method:
  - Change the number of arguments
  - Change the argument's data type

Overloading Methods
Change data type arguments

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

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

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

Overloading Methods
Change return type

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

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

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

Overloading Methods

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

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

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

We can't only change the return type of the method

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.

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

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

```
// 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

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

Method Composition

Suatu method besar dapat dipecah menjadi method-method kecil. Dengan demikian, kita
dapat menggunakan suatu method sebagai bagian (komposisi) dari method lain.

```
public static double circleArea
        (double xc, double yc, double xp, double yp) {
    double radius = distance(xc, yc, xp, yp);
    double area = calculateArea(radius);
    return area;
}
```

Implementasi method circleArea memanfaatkan method distance() dan method calculateArea().