Methods

CSGE601021 Dasar-Dasar Pemrograman 2

Fakultas Ilmu Komputer Universitas Indonesia

Reference

Liang, Introduction to Java Programming, 11th Edition

Background

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

```
public class SumOfIntegers{
  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);
}
```

These code snippets do the same thing!

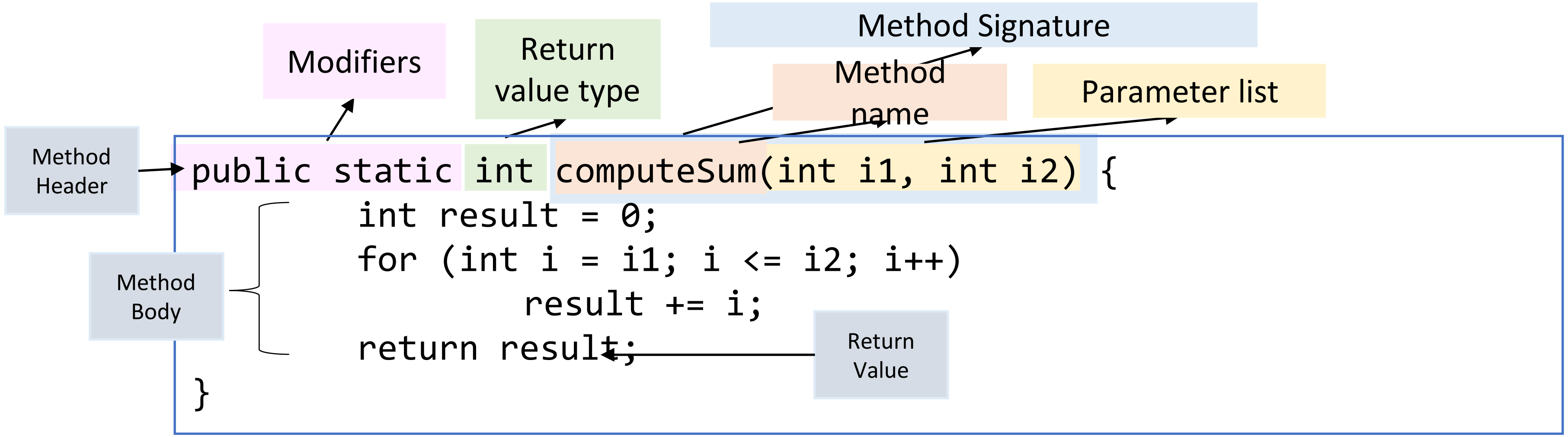
```
public class SumOfIntegers{
  public static void main(String[] args){

      System.out.println("Sum from 1 to 10 is " + computeSum(1,10));
      System.out.println("Sum from 20 to 30 is " + computeSum(20,30));
      System.out.println("Sum from 35 to 45 is " + computeSum(35,45));

  }

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

}
```

Method main

Method computeSum

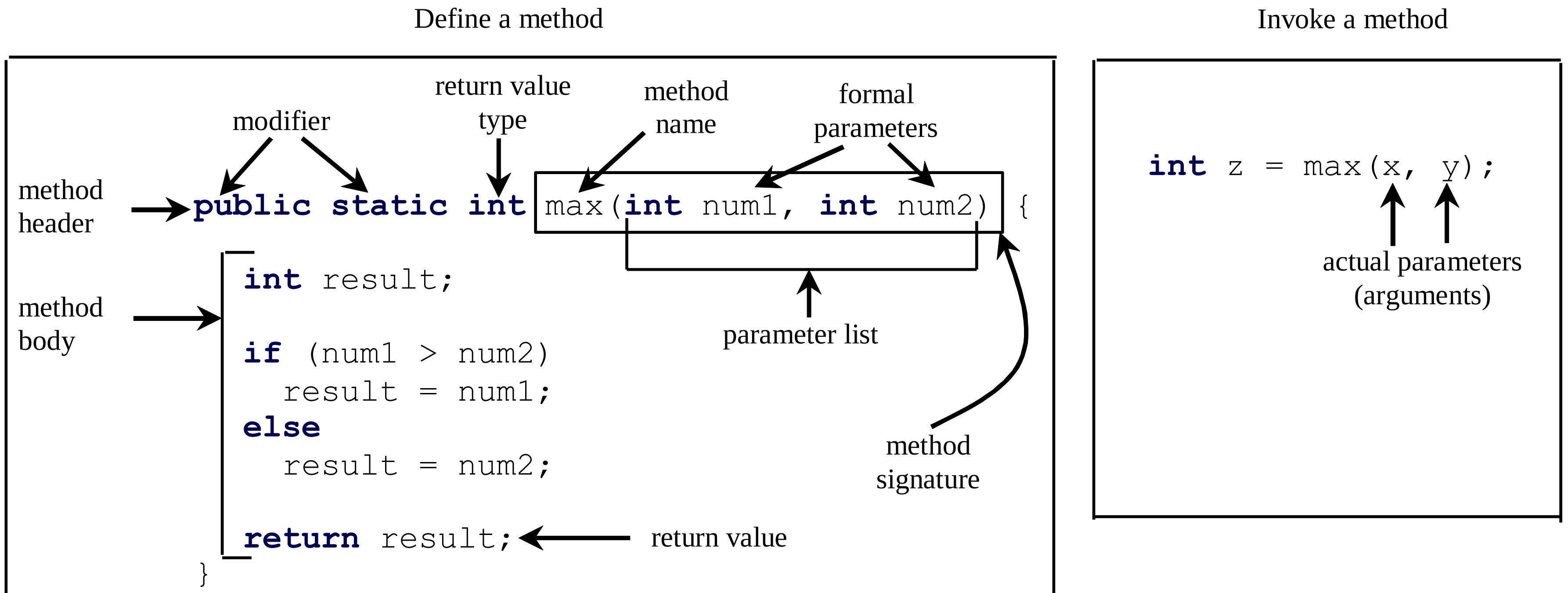
Methods

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



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

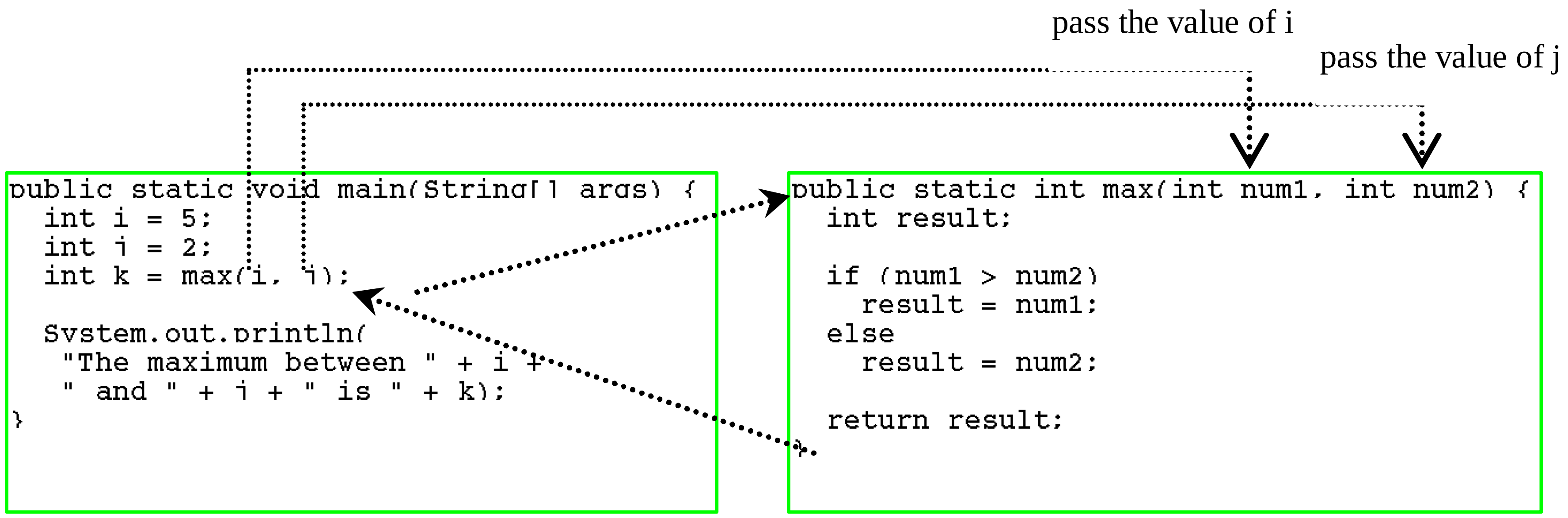
Method Components



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

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

    return result;
}
```

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

Calling Methods



```
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;
}
```

Example: Void Method

```
import java.util.Scanner;

public class TestMethod{
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        cekNilai(input.nextInt());
    }

    public static void cekNilai(int angka){
        if(angka % 2 == 0)
            System.out.println("Angka genap");
        else
            System.out.println("Angka ganjil");
    }
}
```

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

Exercise

1. Write a Java method to compute the average of three numbers.
2. Write a Java method to count all vowels in a string.
3. Write a Java method to compute the sum of the digits in an integer.

Exercise: Average of three numbers

```
public static double average(double x, double y, double z)
{
   return (x + y + z) / 3;
}
```

Exercise: count all vowels in a string.

```
public static int count_Vowels(String str) {
    int count = 0;
    for (int i = 0; i < str.length(); i++) {
        if(str.charAt(i) == 'a' || str.charAt(i) == 'e' ||
           str.charAt(i) == 'i’ || str.charAt(i) == 'o' ||
           str.charAt(i) == 'u’) {
            count++;
        }
    }
    return count;
}
```

Exercise: compute the sum of the digits in an integer.

```
public static int sumDigits(long n) {
    int result = 0;

    while(n > 0) {
      result += n % 10;
      n /= 10;
    }

    return result;
}
```

Exercise: Patterned Table

Create a method genTable(N) to generate the table above with N as the row limit

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
...
N 2N 3N 4N 5N 6N 7N 8N 9N
```

Exercise: Patterned Table

```
public static void genTable(int n){
    for(int i = 1; i <= n; i++){
        for(int j = i; j <= 9*i; j+=i){
            System.out.print(j + "  ");
        }
        System.out.println("");
    }
}
```

Passing Parameters

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

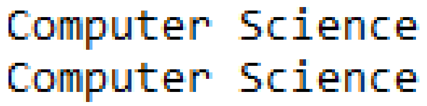
Suppose you invoke the method using `nPrintln(“Computer Science”, 15);`
What is the output?

Can you invoke the method using `nPrintln(15, “Computer Science”);`

Passing Parameters

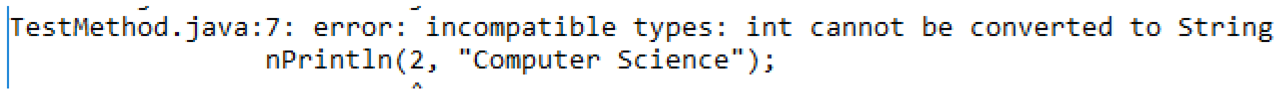
```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Suppose you invoke the method using `nPrintln(“Computer Science”, 2);`
What is the output?



```
Computer Science
Computer Science
```

Can you invoke the method using `nPrintln(2, “Computer Science”);`



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

Passing by Value

A copy of the passed-in variable is copied into the argument of the method. Any changes to the argument do not affect the original one.

Passing by Value

```
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);
  }
}
```

Java is Strictly Pass by Value!

What’s the output?

```
**
 * Impossible Swap function in Java
 * @author www.codejava.net
 */
public class Swap {
    public static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
        System.out.println("x(1) = " + x);
        System.out.println("y(1) = " + y);
    }
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        swap(x, y);
        System.out.println("x(2) = " + x);
        System.out.println("y(2) = " + y);
    }
}
```

Overloading Methods

- Two ways to overload a method:
  - Change the **number** of arguments
  - Change the argument’s **data type**

Overloading Method (1)

```
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;
  }
}
```

Change data type of the arguments

Overloading Method (2)

```
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;
  }
}
```

Can we overload a method by changing return type of the method?

Overloading Method (2)

```
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

Exercise: Overloading Method

- Create overloaded methods times(X, Y) taking two parameters in that:
- When both X and Y are integers, return their multiplication.
    - times(3, 4) returns 12
- When one argument is an int and the other is a String, return the String repeated int times.
    - times(3, “fasilkom”) returns fasilkomfasilkomfasilkom

Exercise: Overloading Method

```
public static int times(int x, int y){
     return x * y;
}

public static String times(int x, String str){
     String result = "";
     for(int i = 0; i < x; i++){
          result += str;
     }
     return result;
}
```

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

```
// 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;
  }
}
```

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```