TECHIN

05 Metodai

Jaroslav Grablevski

Turinys

- Metodai grąžinantys reikšmę
- Metodai negrąžinantys reikšmės
- Uždavinių sprendimo pavyzdžiai
- Varargs
- Overloading

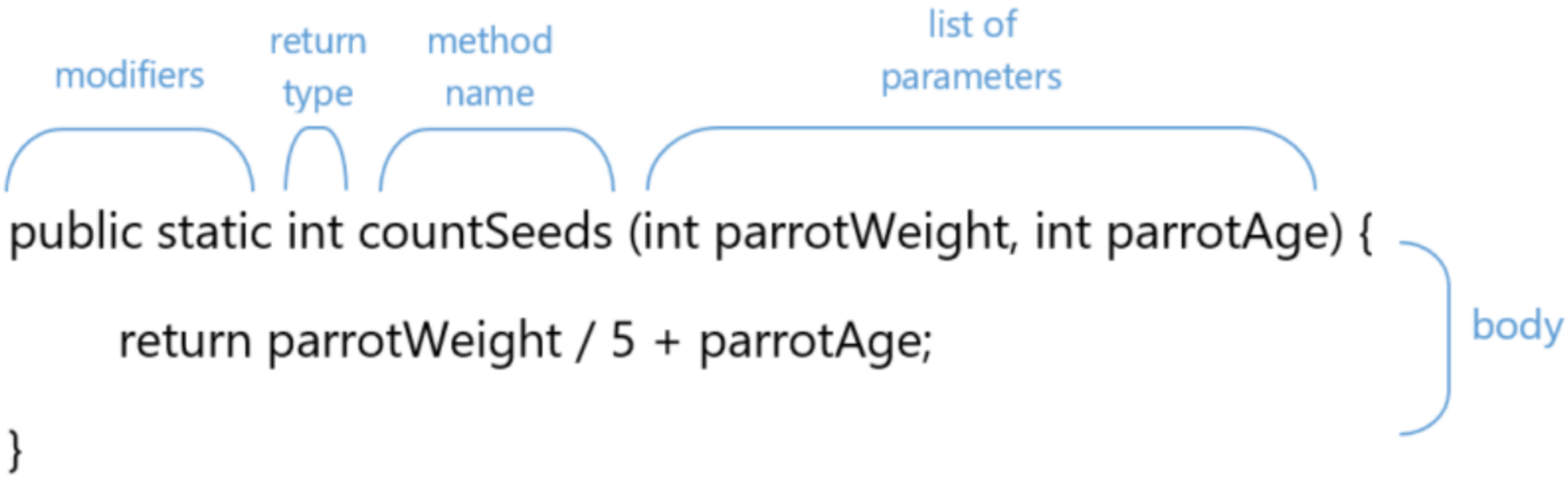
Metodai

```
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 = 11; i <= 20; i++)
    sum += i;
System.out.println("Sum from 11 to 20 is " + sum);

sum = 0;
for (int i = 21; i <= 30; i++)
    sum += i;
System.out.println("Sum from 21 to 30 is " + sum);
```

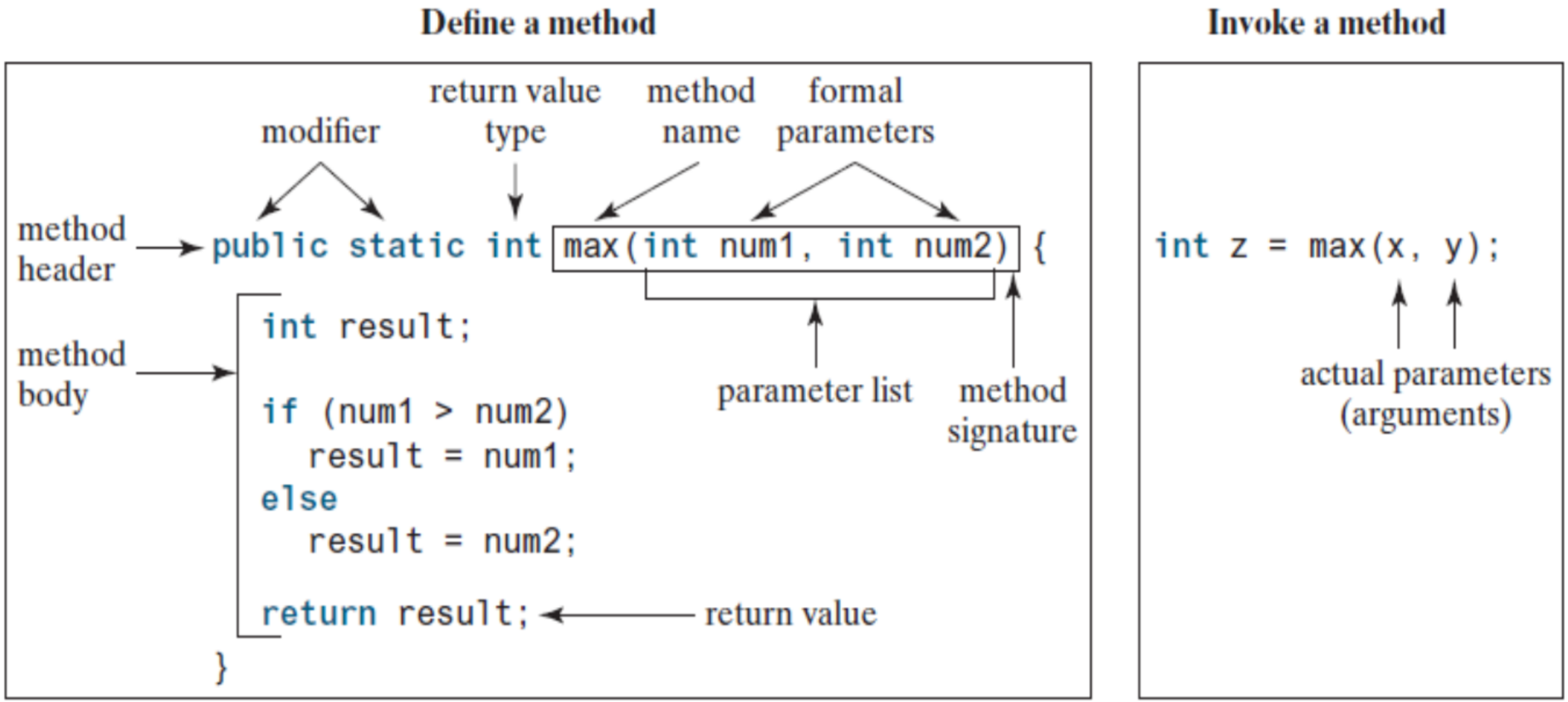
Metodai

```
public static int sum(int from, int to) {
    int result = 0;
    for (int i = from; i <= to; i++)
        result += i;

    return result;
}

public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 11 to 20 is " + sum(11, 20));
    System.out.println("Sum from 21 to 30 is " + sum(21, 30));
}
```

Metodai


modifiers
return type
method name
list of parameters
public static int countSeeds (int parrotWeight, int parrotAge) {
return parrotWeight / 5 + parrotAge;
}
body

Metodai


Define a method
Invoke a method
return value type
method name
formal parameters
modifier
method header
public static int max(int num1, int num2) {
int result;
method body
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
parameter list
method signature
return value
int z = max(x, y);
actual parameters (arguments)

Method modifiers (modifikatoriai)*

Access control modifiers (matomumo modifikatoriai):	
`public`	matomumo sritis neribojama (jei matoma jo klasė)
(*pagal nutylėjimą*)	matomi savo paketo klasėse
`protected`	matomi savo paketo ir išvestinėse klasėse
`private`	matomi savo klasėje

`static`	galima taikyti klasei (be objekto)
`abstract`	metodas neturi kūno
`final`	metodas negali būti užklotas (overridden)
`native`	parašytas ne javos kalba
`synchronized`	tuo pačiu metu metodą gali naudoti tik vienas procesas (thread)
`strictfp`	apribojimai skaičiavimams su slankiuoju kableliu

*plačiau bus kai nagrinėsime objektinį programavimą

Metodo vardas (method name)

- Must be a legal identifier
- By the convention:
  - should be a verb or a multi-word name that begins with a verb, followed by adjectives, nouns, etc
  - camelCase
- Examples:
  - `sum`
  - `getValue`
  - `calculateNumberOfOranges`
  - `findUserByName`
  - `printArray`

Parametrų sąrašas (argument list)

```java
//Parametrų sąrašas sudarytas iš kintamųjų tipų ir vardų poru,
//atskirtų kableliais.
public static void doSomething(String name, int age) {
    // do something
}

//Vienas int tipo parametras
public static void doSomething(int a) {
    // do something
}

//Parametrų sąrašas gali būti tuščias: ()
public static int getThree() {
    return 3;
}
```

Metodai

```
public static void main(String[] args) {
    greet();
    greet();
    greet();
}

public static void greet() {
    System.out.println("Greetings from the world of methods!");
}
```

```
Greetings from the world of methods!
Greetings from the world of methods!
Greetings from the world of methods!
```

Metodai su parametrais

```java
public static void main(String[] args) {
    greet("Jonas");
    greet("Justina");
}

public static void greet(String name) {
    System.out.println("Hi " + name + ", greetings from the world of methods!");
}
```

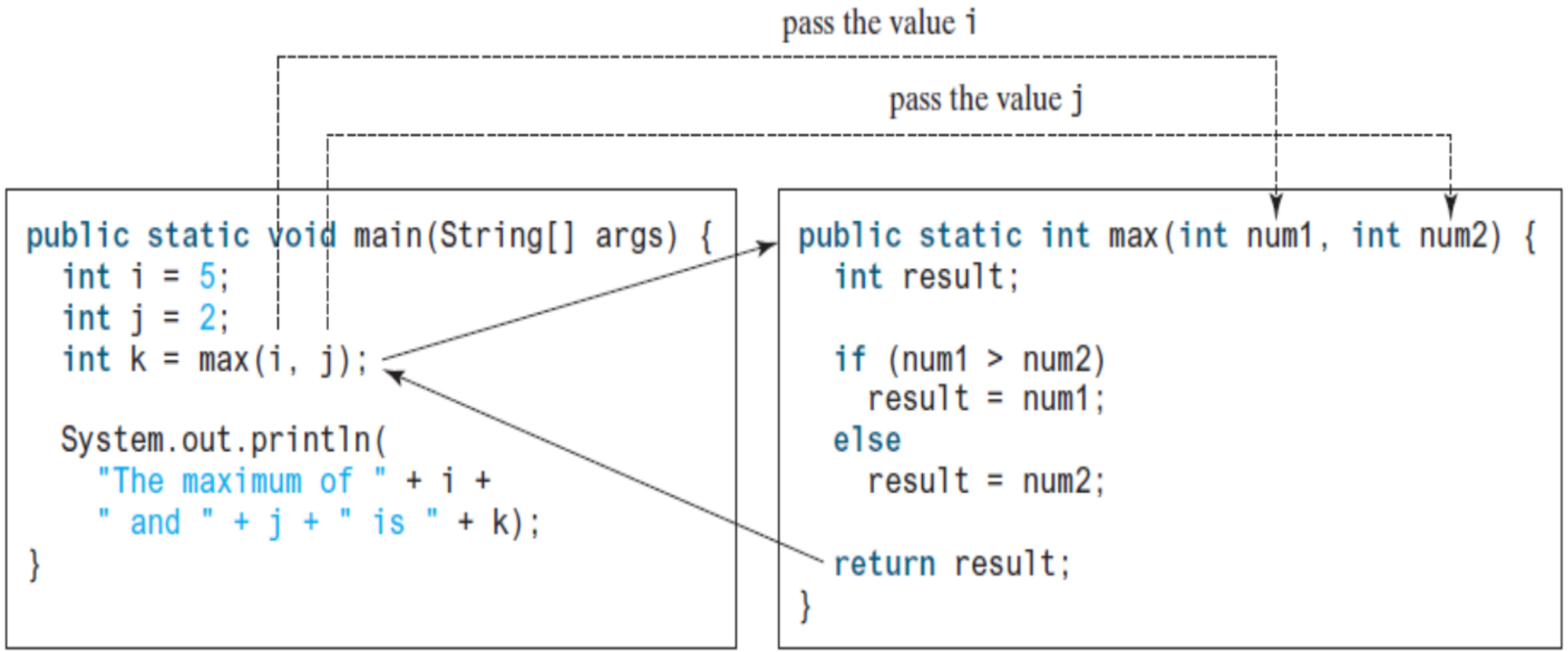
```
Hi Jonas, greetings from the world of methods!
Hi Justina, greetings from the world of methods!
```

Passing Parameters by Values

```
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 x) {
    x++;
    System.out.println("x inside the method is " + x);
}
```

```
Before the call, x is 1
x inside the method is 2
After the call, x is 1
```

Metodai - parametrai



Rezultato tipas (return value type)

- Metodo antraštėje turime nurodyti vieną rezultato tipą (pvz. *double, boolean, String, int[], Person*)
- Metodo apraše (kūne) grąžinamą rezultatą nurodome raktiniu žodžiu *return*
- Jei metodas nieko negražina, turime nurodyti atributą *void*

Metodai – negrąžinantys reikšmės

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

    sum(2, 2);
    int c = 3, d = 2;
    sum(c, d);

}

public static void sum(int a, int b) {
    System.out.println("Sum: " + (a + b));
}
```

```
Sum: 4
Sum: 5
```

Explicit return

```
public static void simpleMethod() {
    if (Math.random() > 0.5) {
        System.out.println("Didesnis skaičius");
        return; // explicit return
    } else {
        System.out.println("Mažesnis skaičius");
    }
    System.out.println("Pabaiga");
    //rezultato tipas yra "void", todėl galime
    //nerašyti raktinio žodžio "return"
}
```

Metodai – grąžinantys reikšmę

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

    System.out.println("Sum: " + sum(2, 2));
    int c = 3, d = 2;
    int sum = sum(c, d);
    System.out.println("Sum: " + sum);

}

public static int sum(int a, int b) {
    return a + b;
}
```

```
Sum: 4
Sum: 5
```

Metodai – grąžinantys reikšmę

- If a method returns a value, the method's body must contain the return keyword.
- Method may have multiple returns.
- Each state can return only a single value

```
public static int divideBy2(int number, int times) {
    if (times <= 0) {
        return number;
    }
    for (int i = 0; i < times; i++) {
        number /= 2;
    }
    return number;
}
```

Metodai – grąžinantys reikšmę

```
public static String returningMethod(){
    if(Math.random()>0.5){
        return "Išėjom ankščiau";
    }
    //return 100; - netinka, nes rezultato tipas kitoks
    return "Metodo pabaiga";
}
```

Pavyzdys (1)

- Duotas sveikųjų skaičių intervalas [m, n].
- Parašykite programą, kuri suskaičiuotų kiek tame intervale yra skaičių, kurie dalinasi iš juos sudarančių skaitmenų sumos.
- Metodas – grąžinantis vieno skaičiaus skaitmenų sumą.

Sprendimas (1) Pagrindinis metodas main

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

    Scanner rd = new Scanner(System.in);

    int kiek = 0;

    // duomenu ivedimas
    System.out.print("Iveskite m: ");
    int m = rd.nextInt();
    System.out.print("Iveskite n: ");
    int n = rd.nextInt();

    for(int i=m; i<=n; i++){
        if(i % skaitmenuSuma(i) == 0)
            kiek++;
    }

    System.out.println("Skaiciu kiekis: " + kiek);

}
```

Sprendimas (2) Metodas skaitmenuSuma

```
public static int skaitmenuSuma(int sk){
     int suma = 0;
     int psk;
     while(sk > 0){
          psk = sk % 10;
          suma = suma + psk;
          sk = sk / 10;
     }
     return suma;
}
```

Pavyzdys su boolean išraiška

```java
public static boolean isEven(int number) {
    boolean isEven;
    if (number % 2 == 0) {
        isEven = true;
    } else {
        isEven = false;
    }
    return isEven;
}
```

=

```java
public static boolean isEven(int number) {
    return number % 2 == 0;
}
```

Variable length argument list (varArgs)

- Trys taškai už parametro tipo reiškia, kad argumentų skaičius gali būti bet koks.
- Metode šie argumentai naudojami kaip masyvo elementai.

```
public static int addUpNumbers(int... values){
    int sum = 0;
    for (int v: values){
        sum +=v;
    }
    return sum;
}

public static void main(String[] args) {
    System.out.println(addUpNumbers(1,2,3));
    System.out.println(addUpNumbers(1,2,3,4,5,6,7));
    System.out.println(addUpNumbers());
    System.out.println(addUpNumbers(new int[]{1,2,3})); //masyvas
}
```

Overloading

- If methods have the same name, but a different number or type of parameters, they are overloaded.

```
public static void print(String stringToPrint) {
    System.out.println(stringToPrint);
}
public static void print(String stringToPrint, int times) {
    for (int i = 0; i < times; i++) {
        System.out.println(stringToPrint);
    }
}
public static void print(int val) {
    System.out.println(val);
}
```

Overloading and casting

```java
public static void print(short a) {
    System.out.println("short arg: " + a);
}
public static void print(long a) {
    System.out.println("long arg: " + a);
}
public static void print(double a) {
    System.out.println("double arg: " + a);
}
public static void main(String[] args) {
    print(100);
}
```

```
long arg: 100
```

Overloading and casting

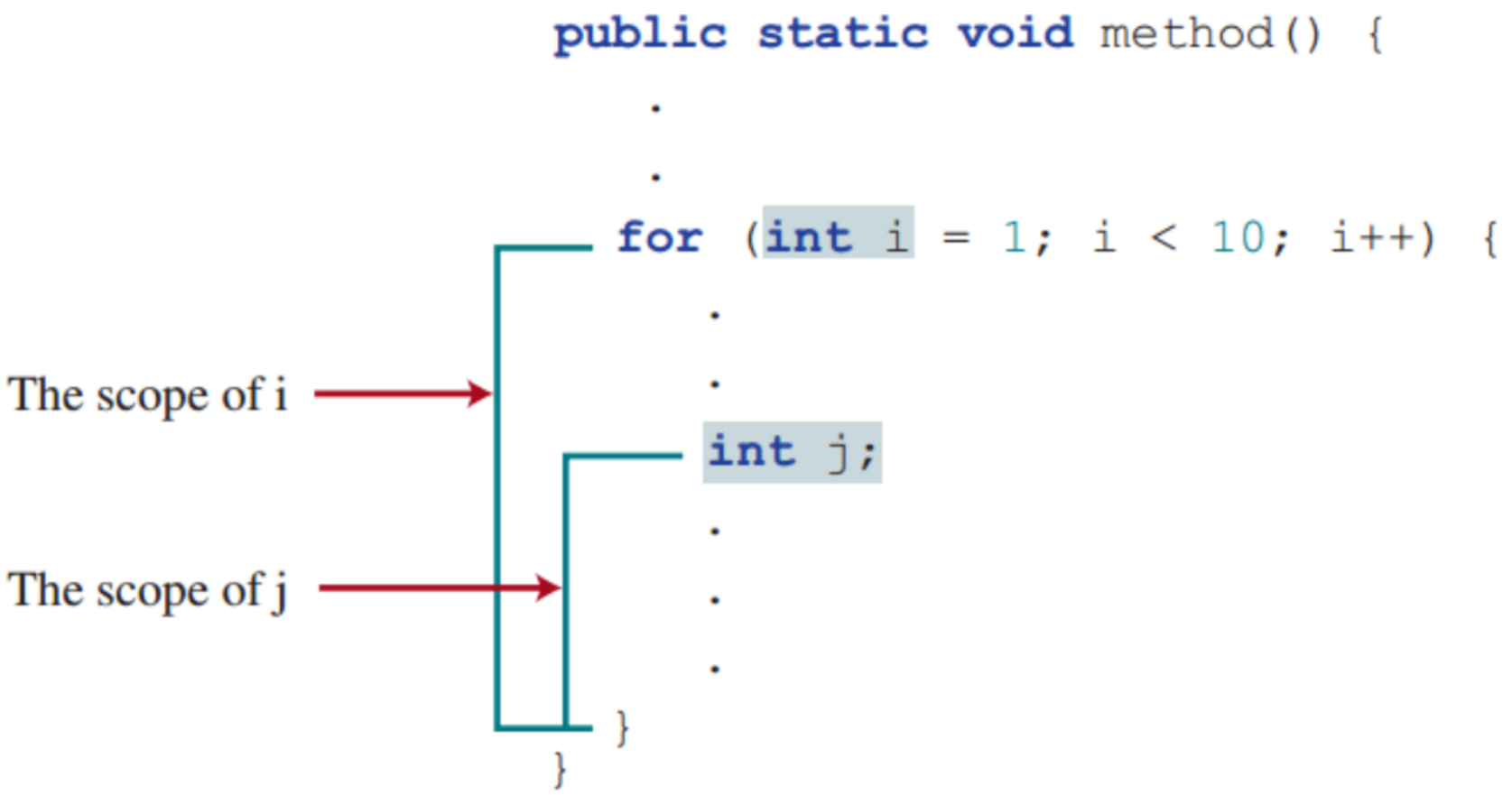
```
public static double max(int num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
public static double max(double num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
public static void main(String[] args) {
    max(1, 2);  //compile error
}
```

Scope

The scope of a declaration is the portion of the program that can refer to the declared entity by its name. Such an entity is said to be “in scope” for that portion of the program.

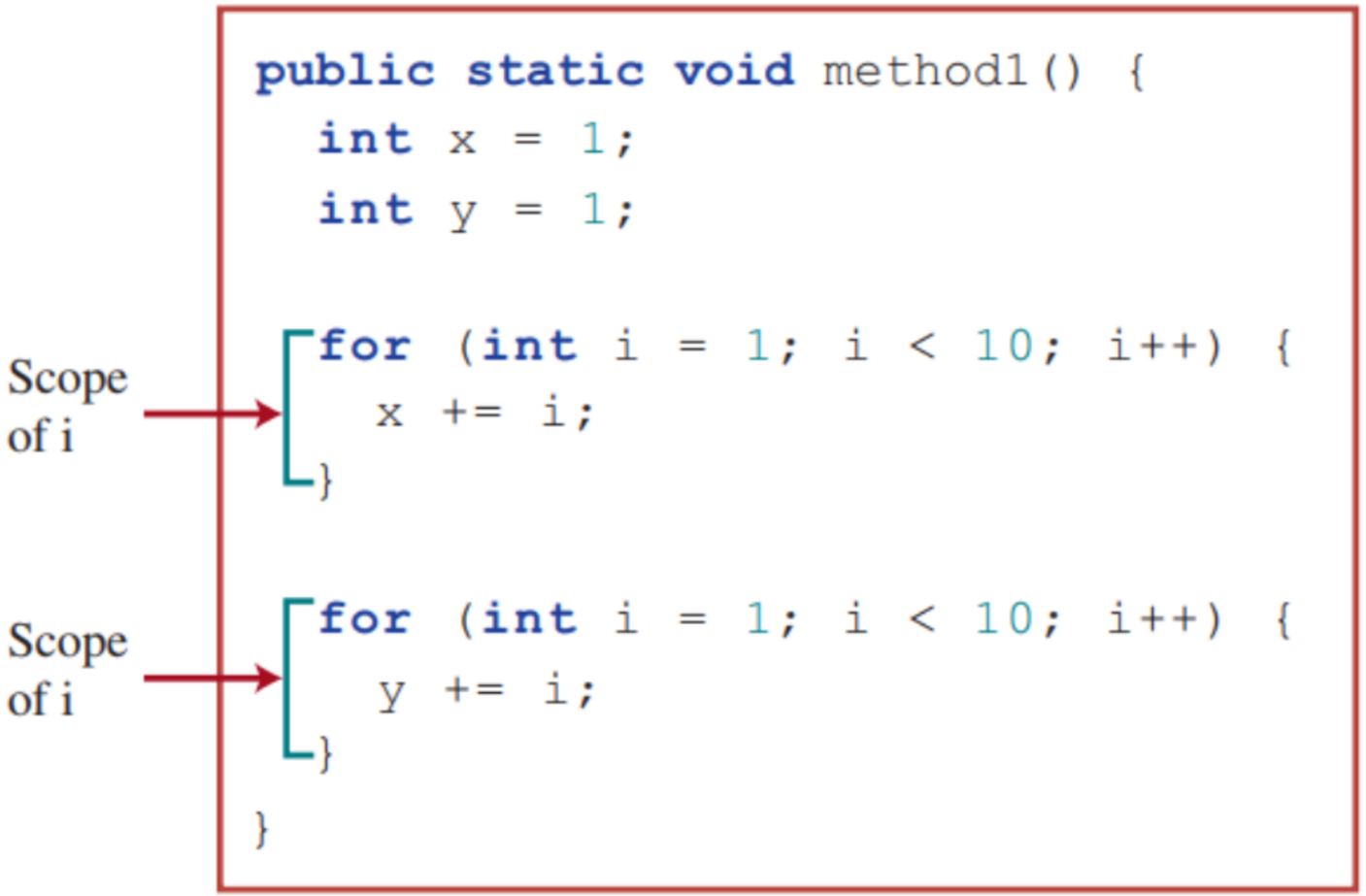
- The scope of a parameter declaration is the body of the method in which the declaration appears.
- The scope of a local-variable declaration is from the point at which the declaration appears to the end of that block.
- The scope of a local-variable declaration in a for statement’s header is the body
  of the for statement and the other expressions in the header.
- A method or field’s scope is the entire body of the class

Scope


public static void method() {
.
.
for (int i = 1; i < 10; i++) {
.
.
int j;
.
.
.
}
}
The scope of i
The scope of j

Scope

- It is fine to declare **i** in two nonnested blocks



Scope

- It is wrong to declare `i` in two nested blocks