SENG 201 – Object Oriented Programming

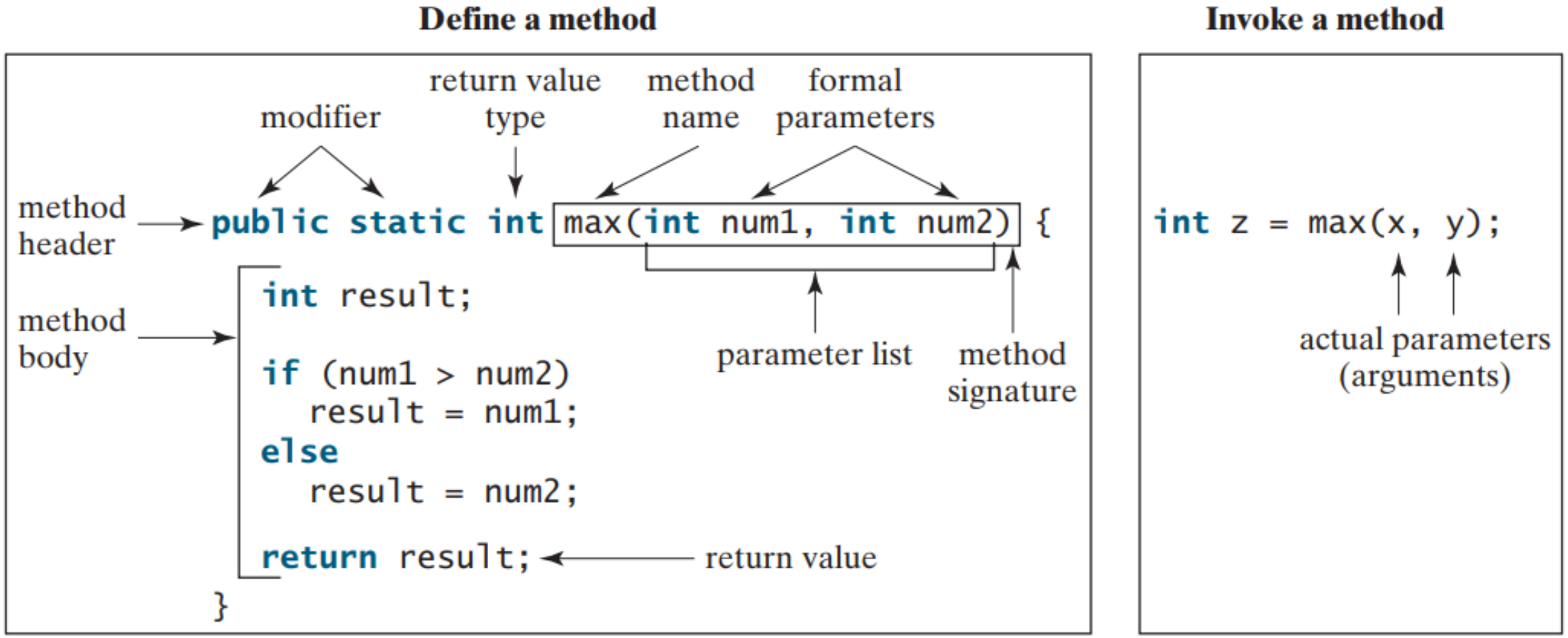
2025 – 2026 Fall Lectures – Week 5

Ömer Mintemur

Methods - Functions

- A method in Java is a block of code that performs a specific task, defined by its name, return type, and parameters. Methods help in code reusability and modularity.
- Syntax of a Method: Methods in Java follow a specific syntax:
  - Return type (e.g., void for no return)
  - Method name (e.g., calculateSum)
  - Parameter list enclosed in parentheses (can be empty or include parameters)
  - Method body enclosed in curly braces

Methods - Functions



Methods - Functions

Specify your return type

Write your functions after *main function*

public static is a must until we see Objects

```
public static void main(String[] args) {
    // TODO code application logic here
}

public static void function_name(int a, int b)
{

}
```

Methods - Functions

If a method does not return anything, you do not need to write a return statement at the end of the function

If a method returns something, both function type and return type must match.

```
public static void main(String[] args) {
    // TODO code application logic here
}

public static void first_function_name(int a, int b)
{
    int sum = a + b;
}
public static int second_function_name(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

Methods - Functions

- A method can `return`:
  - int
  - double
  - float
  - long
  - short
  - byte
  - char
  - boolean

Methods - Functions

- Defining a function is not enough, it has to be called from the `main` function

```
public static void main(String[] args) {
    // TODO code application logic here

    first_function_name(15, 15);
    second_function_name(15, 15);
}

public static void first_function_name(int a, int b)
{
    int sum = a + b;
}
public static int second_function_name(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

Methods - Functions

```java
public static void main(String[] args) {
    // TODO code application logic here

    int x = 10;
    int y = 12;
    first_function_name(x, y);
    int result = second_function_name(x, y);

    System.out.println(result);
}

public static void first_function_name(int a, int b)
{
    int sum = a + b;
}
public static int second_function_name(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here

    int x = 10;
    int y = 12;
    first_function_name(x, y);

    System.out.println(second_function_name(x, y));
}

public static void first_function_name(int a, int b)
{
    int sum = a + b;
    System.out.println(sum);
}
public static int second_function_name(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

Methods - Functions

- In Java, all primitive types (e.g. `int, double`) are passed by value, meaning a copy of the variable is passed to the function.
- Changes made to the parameter inside the function do not affect the original variable.

```java
public static void main(String[] args) {
    // TODO code application logic here

    int x = 10;
    int y = 12;
    System.out.println("Outside of the function x = " + x + " y = " + x);
    change_numbers(x, y);
    System.out.println("Outside of the function x = " + x + " y = " + x);
}

public static void change_numbers(int a, int b)
{
    a = 5;
    b = 8;
    System.out.println("Inside of the function x = " + a + " y = " + b);
}
```

Methods - Functions

- You can not return more than one variable from a function in Java

```
public static void main(String[] args) {
    // TODO code application logic here

    int x = 10;
    int y = 12;
    x,y = change_numbers(x, y);
}

public static int return_numbers(int a, int b)
{
    a = 5;
    b = 8;
    return a,b;
}
```

Methods - Functions

- If you have an ***array***, you can manipulate it in a function. Its effect will continue in the `main` function. You do not have to use `return` keyword

```
public static void main(String[] args) {
    // TODO code application logic here

    int[] x = {10,11,12};
    System.out.println(x[2]);
    return_numbers(x);
    System.out.println(x[2]);

}

public static void return_numbers(int[] a)
{
    a[2] = 20;
}
```

```
run:
12
20
BUILD SUCCESSFUL (total time: 0 seconds)
```

Methods - Functions

- Function Overloading
  - Function overloading allows you to define multiple functions with the same name but different parameter lists

```
run:
10
10.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    int x = 5;
    int y = 5;
    System.out.println(add_numbers(x,y));
    double a = 5;
    double b = 5;
    System.out.println(add_numbers(a,b));
}
public static int add_numbers(int a, int b)
{
    return a + b;
}
public static double add_numbers(double a, double b)
{
    return a + b;
}
```

Methods - Functions

- Sometimes there are two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
- ***Overloaded methods must have different parameter lists.***

```
public static void main(String[] args) {
    // TODO code application logic here
    System.out.println(add_numbers(1,2));
    System.out.println(add_numbers(1,2));
}
public static double add_numbers(int a, double b)
{
    return (a + b);
}
public static double add_numbers(double a, int b)
{
    return (a + b);
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    System.out.println(add_numbers(1,2f));
    System.out.println(add_numbers(1f,2));
}
public static double add_numbers(int a, float b)
{
    return (a + b);
}
public static float add_numbers(float a, int b)
{
    return (a + b);
}
```

Methods – Functions – An Example

```
public static void main(String[] args) {
    // TODO code application logic here
    int numbers = 100;
    int number_per_lines=10;
    System.out.println("First " + numbers + " prime numbers are: ");
    int count = 0;

    while (numbers > 0)
    {
        if(isPrime(numbers))
        {

            count++;
                if ((count % number_per_lines) == 0)
                {
                    System.out.print(numbers + " \n");
                }
                else
                {
                    System.out.print(numbers + " ");
                }
        }

    numbers--;
    }
}
```

Met

(*Estimate* π) π can be computed using the following series:

$$m(i) = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^{i+1}}{2i - 1}\right)$$

Write a method that returns `m(i)` for a given `i` and write a test program that displays the following table:

i	m(i)
1	4.0000
101	3.1515
201	3.1466
301	3.1449
401	3.1441
501	3.1436
601	3.1433
701	3.1430
801	3.1428
901	3.1427