Java Programs

Lecture 4

By
Dr. Radwa Rady & Dr. Ghada Fathy

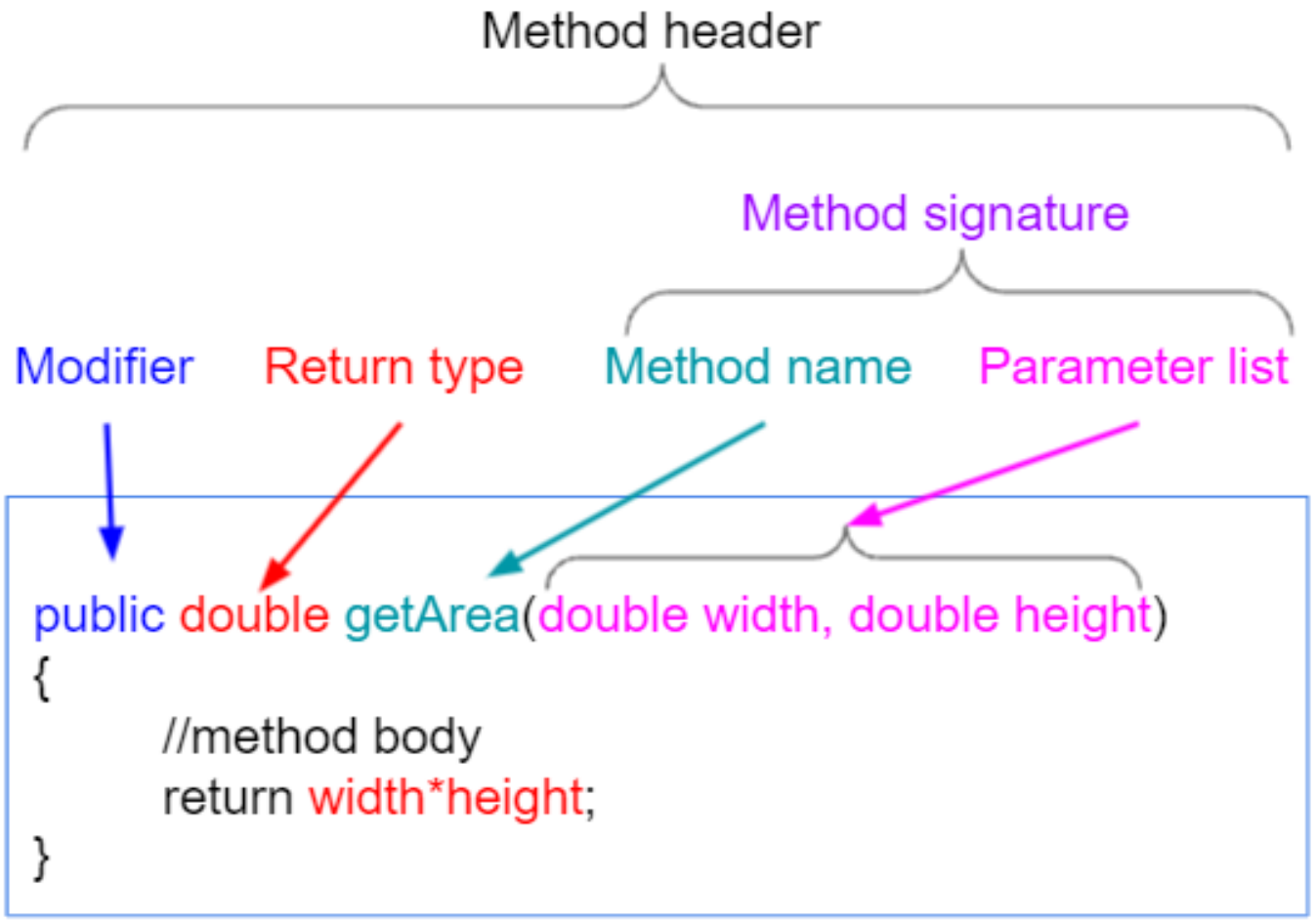
Objectives

- Methods:
  - ✓ Create Method
  - ✓ Calling Method
  - ✓ Method Parameters
  - ✓ Return Values
  - ✓ Variables Arguments
  - ✓ Overloading
  - ✓ Types
- Scope
- Recursion

Methods/Functions

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.
- A method must be declared within a class.

Create Method



Access Modifiers

- These control **who can access the method**.

Modifier	Access Level
public	Accessible from anywhere
private	Accessible only inside the same class
protected	Accessible in the same package and subclasses (children)
(default)	Accessible only inside the same package

```
public class Test {


    public void method1() {
        System.out.println("Public method");
    }

    private void method2() {
        System.out.println("Private method");
    }

    protected void method3() {
        System.out.println("Protected method");
    }

    void method4() {
        System.out.println("Default method");
    }


}
```

Non-Access Modifiers

Modifier	Meaning
static	Method belongs to the class, not an object
final	Method cannot be overridden
abstract	Method declared but not implemented (next lecture)

```
static void show() {
    System.out.println("Static method");
}

final void display() {
    System.out.println("Final method");
}
```

Static Vs Private

```java
class Test {

    static void show() {
        System.out.println("Static method");
    }

    public static void main(String[] args) {
        show();   // called without creating an object
    }
}
```

```java
class Test {

    private void show() {
        System.out.println("Private method");
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.show();   // allowed because it is inside the same class
    }
}
```

```java
class Test {

    private static void hello() {
        System.out.println("Hello");
    }

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

Calling Method

```
public class Test
{
    public static void main (String[] args)
    {
        myMethod();
    }

    static void myMethod()
    {
        System.out.println("I have been executed!");
    }
}
```

// Output
I have been executed!

Calling Method

A method can also be called multiple times.

```
public class Test
{
    public static void main (String[] args)
    {
        myMethod();
        myMethod();
        myMethod();
    }

    static void myMethod()
    {
        System.out.println("I have been executed!");
    }
}
```

```
// Output
I have been executed!
I have been executed!
I have been executed!
```

Method Parameters

Information can be passed to methods as parameter. Parameters act as variables inside the method.

- Parameters are specified after the method name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma.

```
public class Test
{
    public static void main (String[] args)
    {
        myMethod("Ali", 5);
        myMethod("Sara", 8);
        myMethod("Mohammed", 31);
    }

    static void myMethod(String fname, int age) {
        System.out.println(fname + " is " + age);
      }
}
```

```
// Output
Ali is 5
Sara is 8
Mohammed is 31
```

Return Values

If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method.

```
public class Test
{
    public static void main (String[] args)
    {
        System.out.println(myMethod(5, 3));
    }

    static int myMethod(int x, int y) {
        return x + y;
      }
}
```

```
// Output
8
```

Return Values

You can also store the result in a variable (recommended, as it is easier to read and maintain).

```
public class Test
{
    public static void main (String[] args)
    {
        int z = myMethod(5, 3);
        System.out.println(z);
    }

    static int myMethod(int x, int y) {
        return x + y;
      }
}
```

```
// Output
8
```

Example

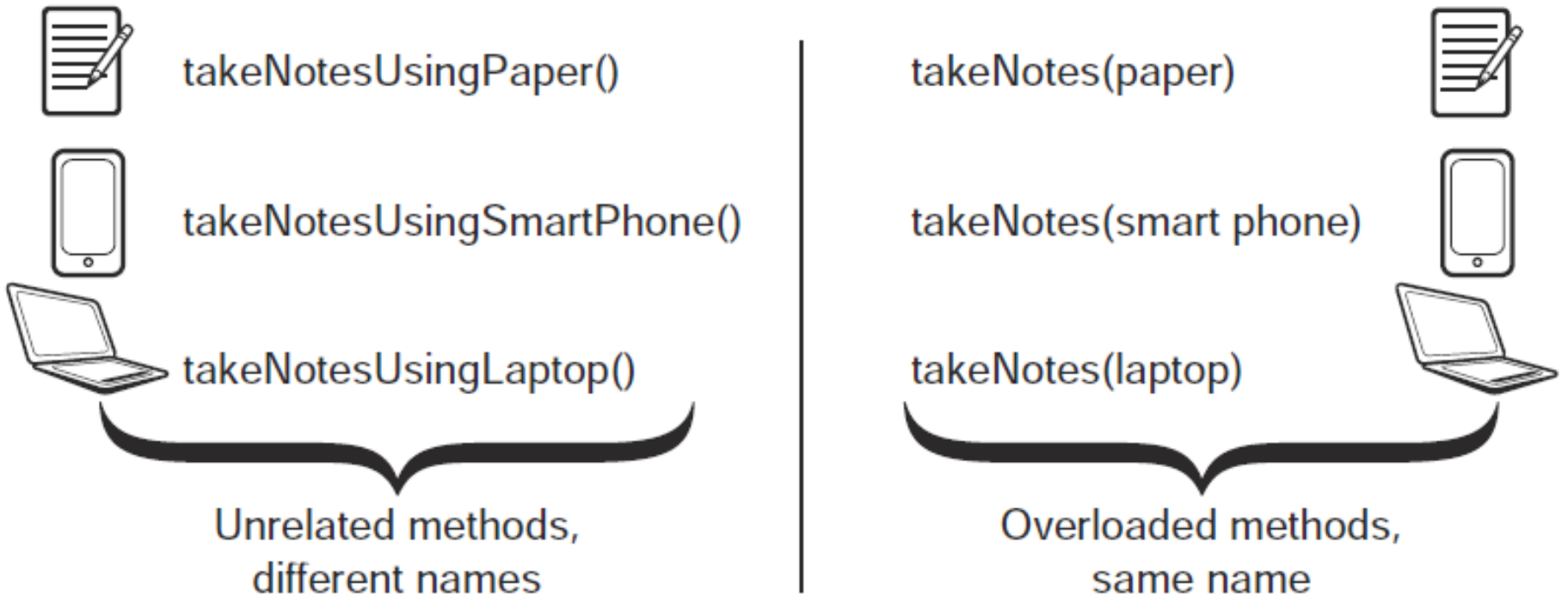
```
public class Test
{
    public static void main (String[] args)
    {
        checkAge(20); // Call the checkAge method and pass along an age of 20
    }

    // Create a checkAge() method with an integer variable called age
    static void checkAge(int age) {

        // If age is less than 18, print "access denied"
        if (age < 18) {
        System.out.println("Access denied - You are not old enough!");

        // If age is greater than, or equal to, 18, print "access granted"
        } else {
        System.out.println("Access granted - You are old enough!");
        }
    }
}
```

// Outputs

"Access granted - You are old enough!"

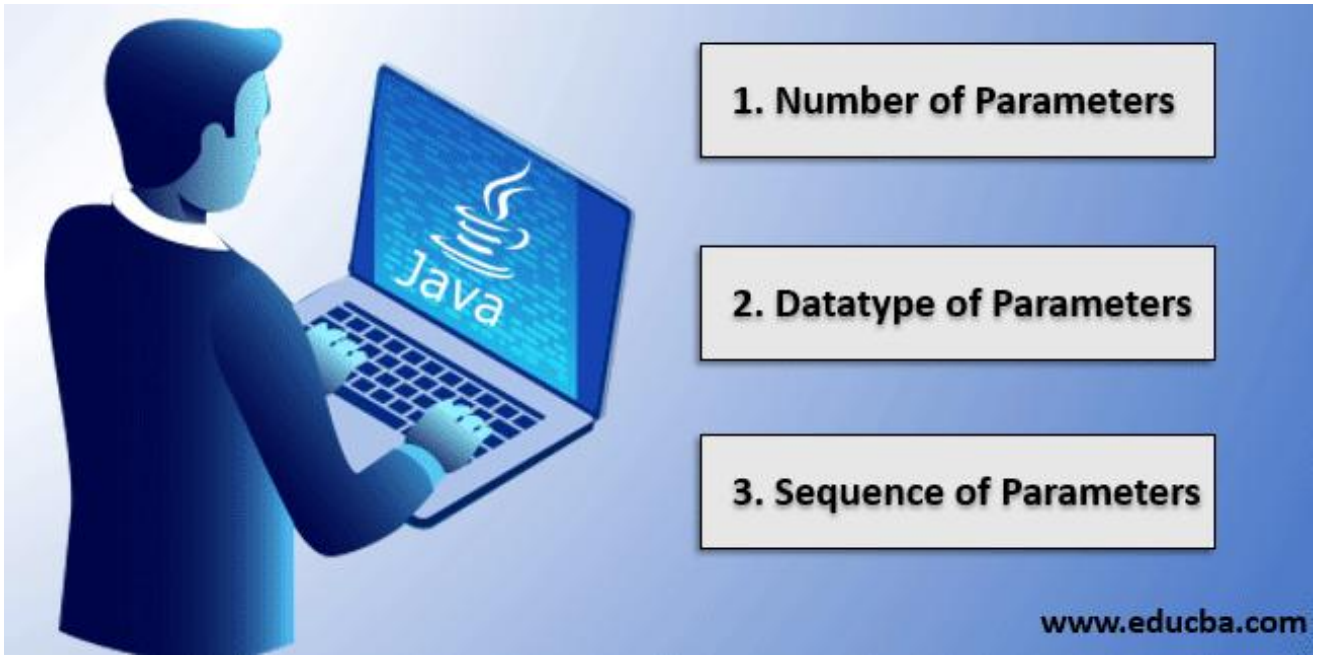
Method Overloading

With method overloading, multiple methods can have the same name with different parameters.



Method Overloading Types

```
void myMethod()

int myMethod(int x)

float myMethod(float x)

double myMethod(int x, double y)

double myMethod(double x, double y)
```



Example

Instead of defining two methods that should do the same thing, it is better to overload one. In the example below, we overload the plusMethod method to work for both int and double.

```
public class Test {

    public static void main (String[] args) {
        int myNum1 = plusMethodInt(8, 5);
        double myNum2 = plusMethodDouble(4.3, 6.26);
        System.out.println("int: " + myNum1);
        System.out.println("double: " + myNum2);
    }

    static int plusMethodInt(int x, int y) {
        return x + y;
    }

    static double plusMethodDouble(double x, double y) {
        return x + y;
     }
}
```

```
public class Test {

    public static void main (String[] args) {
        int myNum1 = plusMethod(8, 5);
        double myNum2 = plusMethod(4.3, 6.26);
        System.out.println("int: " + myNum1);
        System.out.println("double: " + myNum2);
    }

    static int plusMethod(int x, int y) {
        return x + y;
    }

     static double plusMethod(double x, double y) {
        return x + y;
     }
}
```

// Output int: 13 double: 10.559999999999999

Methods Variables Arguments

JDK enables you to pass a variable number of arguments of the same type to a method. The parameter in the method is declared as follows:

typeName... parameterName

Example

```
static int sum(int... numbers)
```

- **int...** means any number of integers
- Inside the method, numbers behaves like an array

Example

```
public class Main {
    public static void main(String args[]) {
        // Call method with variable args
        printMax(34, 3, 3, 2, 56.5);
        printMax(4, 5, 5, 2);

    }

     public static void printMax( double... numbers) {
        if (numbers.length == 0) {
           System.out.println("No argument passed");
           return;
        }

        double result = numbers[0];

        for (int i = 1; i <  numbers.length; i++)
        if (numbers[i] >  result)
        result = numbers[i];

        System.out.println("The max value is " + result);
     }
}
```

// Outputs
The max value is 56.5
The max value is 5

Methods Variables Arguments

- Only one variable-length parameter may be specified in a method.

X void test(int... a, int... b)

- This parameter must be the last parameter.

X void display(int... marks, String name)

Scope

- In Java, variables are only accessible inside the region they are created. This is called **scope**.

- Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared.

- **Block Scope**: A block of code refers to all of the code between curly braces {}.

  Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared.

- A block of code may exist on its own or it can belong to an if, while or for statement. In the case of for statements, variables declared in the statement itself are also available inside the block's scope.

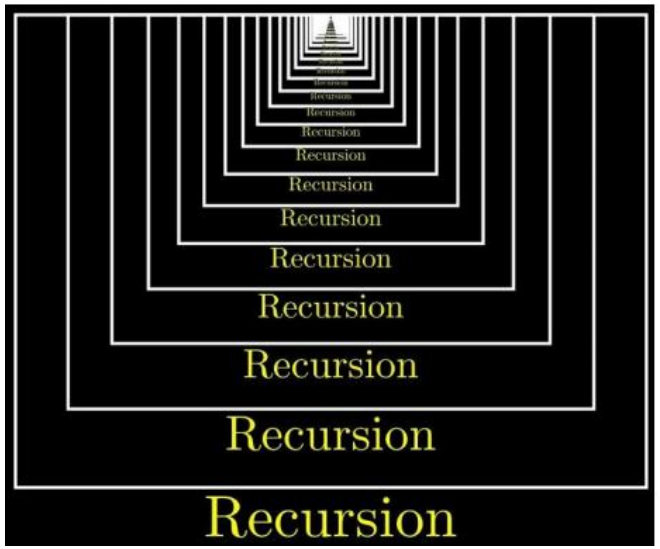
Example

```
public class Test
{
    public int sum (int a, int b){
     int z= a + b; // code here can use z
     return z;
    }
    public static void main (String[] args)
    { // This is a block
         // Code here CANNOT use z

        // Code here CANNOT use x

        int x = 100;

        // Code here can use x
        System.out.println(x);

    }  // The block ends here

        // Code here CANNOT use x
}
```

Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

```
10 + sum(9)
10 + ( 9 + sum(8) )
10 + ( 9 + ( 8 + sum(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```



- When the sum() function is called, it adds parameter k to the sum of all numbers smaller than k and returns the result. When k becomes 0, the function just returns 0.
- Since the function does not call itself when **k** is **0**, the program **stops** there and returns the result.

Example

```
public class Test
{
    public static void main (String[] args)
    {
        int result = sum(10);
        System.out.println(result);
    }

    public static int sum(int k)
    {
        if (k > 0) {
          return k + sum(k - 1);
        } else {
          return 0;
        }
    }
}
```

```
// Output
55
```

Thanks

References: https://www.w3schools.com