PROGRAMMING IN JAVA

Lecture 5

Methods and Modular Programming in Java

Methods, Parameters, Return Values, Overloading, Scope, Static Methods, Varargs, Parameter Passing, and Recursion

Lecturer: Arshyn Altybay

Department of Computer Engineering,
Almaty Technological University, Almaty, Kazakhstan

1. Learning Objectives

By the end of this lecture, students should be able to:

- explain why methods are used to decompose programs;
- declare and call methods;
- distinguish parameters from arguments;
- create methods that return values and methods that return `void`;
- explain local-variable and method scope;
- use `static` methods in a class-level programming style;
- overload methods using different parameter lists;
- explain Java's pass-by-value rule;
- pass arrays to methods and return arrays from methods;
- create variable-argument methods;
- define and trace simple recursive methods;
- organize a program into small reusable modules.

2. Why Methods Matter

A method is a named block of code that performs a specific task. Methods help divide a large problem into smaller, understandable units.

Good method design improves:

- readability;

- reuse;
- testing;
- maintenance;
- separation of responsibilities.

> A useful method usually has one clear purpose and a name that describes that purpose.

3. Method Declaration and Method Call

A basic static method has the following structure:

```
public static returnType methodName(parameterList) {
    statements
}
```

A method is executed when it is called.

Example 1. Declaring and calling a method

```
1  public class MethodExample {
2
3      public static void greet() {
4          System.out.println("Welcome to Java.");
5      }
6
7      public static void main(String[] args) {
8          greet();
9          greet();
10     }
11 }
```

Expected output

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

4. Parameters and Arguments

A parameter is a variable declared in the method header. An argument is the actual value supplied when the method is called.

Example 2. Passing arguments to parameters

```java
public class ParameterExample {

    public static void greetStudent(String name) {
        System.out.println("Hello, " + name + "!");
    }

    public static void main(String[] args) {
        greetStudent("Aruzhan");
        greetStudent("Dias");
    }
}
```

Expected output

```
Hello, Aruzhan!
Hello, Dias!
```

5. Methods That Return Values

A non-`void` method returns a value compatible with its declared return type.

Example 3. Returning a calculated value

```
public class ReturnExample {

    public static double average(
            double a, double b) {

        return (a + b) / 2.0;
    }

    public static void main(String[] args) {
        double result = average(80, 90);
        System.out.println("Average = " + result);
    }
}
```

Expected output

```
Average = 85.0
```

Design rule: Prefer returning a value when the caller should decide what to do with the result. Printing directly inside every calculation method reduces reuse.

6. `void` Methods

A `void` method performs an action but does not return a result value.

Example 4. A `void` reporting method

```
1  public class VoidMethod {
2
3      public static void printLine() {
4          System.out.println("--------------------");
5      }
6
7      public static void main(String[] args) {
8          printLine();
9          System.out.println("Student Report");
10         printLine();
11     }
12 }
```

Expected output

```
--------------------
Student Report
--------------------
```

7. Method Scope and Local Variables

A local variable declared inside a method exists only within the block in which it is declared.

```
public static void example() {
    int x = 10;      // local variable
    System.out.println(x);
}
```

The variable `x` cannot be accessed from another method.

A parameter is also local to its method.

8. Static Methods

A `static` method belongs to the class rather than to a specific object.

In early procedural-style Java programs, `main()` can call other `static` methods directly.

Example 5. Several static utility methods

```
public class MathUtil {

    public static int square(int x) {
        return x * x;
    }

    public static int cube(int x) {
        return x * x * x;
    }

    public static void main(String[] args) {
        System.out.println(square(4));
        System.out.println(cube(3));
    }
}
```

Expected output

```
16
27
```

Instance methods are studied more deeply when classes and objects are introduced.

9. Method Overloading

Method overloading means using the same method name with different parameter lists.

Example 6. Overloaded `max` methods

```
public class OverloadingExample {

    public static int max(int a, int b) {
        return a > b ? a : b;
    }

    public static double max(
            double a, double b) {

        return a > b ? a : b;
    }

    public static void main(String[] args) {
        System.out.println(max(4, 7));
        System.out.println(max(3.5, 2.1));
    }
}
```

Expected output

```
7
3.5
```

The return type alone is not sufficient to distinguish overloaded methods.

10. Java Is Always Pass-by-Value

When a method is called, Java copies the value of each argument into the corresponding parameter.

For primitive types, the copied value is the primitive value itself.

Example 7. Primitive argument values are copied

```
1  public class PassByValuePrimitive {
2
3      public static void change(int x) {
4          x = 100;
5      }
6
7      public static void main(String[] args) {
8          int value = 10;
9          change(value);
10         System.out.println(value);
11     }
12 }
```

Expected output

```
10
```

For object and array variables, Java copies the reference value. Both reference values may then designate the same object.

11. Passing Arrays to Methods

An array variable contains a reference value. Passing an array copies that reference value, so the method can access the same array object.

Example 8. Modifying an array inside a method

```
public class ArrayParameter {

    public static void doubleValues(int[] data) {
        for (int i = 0; i < data.length; i++) {
            data[i] *= 2;
        }
    }

    public static void main(String[] args) {
        int[] values = {1, 2, 3};

        doubleValues(values);

        for (int value : values) {
            System.out.println(value);
        }
    }
}
```

Expected output

```
2
4
6
```

> **Precise wording:** Java does not "pass objects by reference." Java always passes by value. For an array or object variable, the value being copied is a reference value.

12. Returning Arrays

A method may create and return an array.

Example 9. Returning an array

```
public class ReturnArray {

    public static int[] createRange(int n) {
        int[] result = new int[n];

        for (int i = 0; i < n; i++) {
            result[i] = i + 1;
        }

        return result;
    }

    public static void main(String[] args) {
        int[] values = createRange(5);

        for (int value : values) {
            System.out.print(value + " ");
        }
    }
}
```

Expected output

```
1 2 3 4 5
```

13. Variable-Length Argument Lists

A varargs parameter allows a method to receive zero or more values of one type.

```
public static int sum(int... values)
```

Inside the method, `values` behaves like an array.

Example 10. A varargs sum method

```
public class VarargsExample {

    public static int sum(int... values) {
        int total = 0;

        for (int value : values) {
            total += value;
        }

        return total;
    }

    public static void main(String[] args) {
        System.out.println(sum(1, 2, 3));
        System.out.println(sum(10, 20, 30, 40));
    }
}
```

Expected output

```
6

100
```

A varargs parameter must be the final parameter in the method parameter list.

14. Recursion

A recursive method calls itself. Every correct recursive design needs:

- a base case that stops recursion;
- a recursive step that moves the problem toward the base case.

Example 11. Recursive factorial

```
1  public class RecursiveFactorial {
2
3      public static long factorial(int n) {
4          if (n <= 1) {
5              return 1;
6          }
7
8          return n * factorial(n - 1);
9      }
10
11     public static void main(String[] args) {
12         System.out.println(factorial(5));
13     }
14 }
```

Expected output

```
120
```

> **Caution:** Recursion is elegant for some naturally recursive problems, but a recursive solution can consume more call-stack space than an iterative solution.

15. Modular Program Design

A large `main()` method is difficult to read. Break the program into methods that reflect the problem structure.

For example:

Possible decomposition of a student-result program

```
main()
 |
 +-- readData()
```

```
+-- calculateAverage()
+-- findMaximum()
+-- determineGrade()
+-- printReport()
```

This structure separates data acquisition, computation, decision logic, and presentation.

16. Comprehensive Example: Modular Score Analyzer

Example 12. Separating calculations into methods

```
public class ModularScoreAnalyzer {

    public static double average(int[] scores) {
        int total = 0;

        for (int score : scores) {
            total += score;
        }

        return (double) total / scores.length;
    }

    public static int maximum(int[] scores) {
        int max = scores[0];

        for (int score : scores) {
            if (score > max) {
                max = score;
            }
        }

        return max;
    }

    public static char grade(double average) {
        if (average >= 90) return 'A';
        if (average >= 80) return 'B';
        if (average >= 70) return 'C';
        if (average >= 60) return 'D';
        return 'F';
    }

```

Example 12 (continued)

```
    public static void printReport(
            String name, int[] scores) {

        double avg = average(scores);
        int max = maximum(scores);

        System.out.println("Student = " + name);
        System.out.println("Average = " + avg);
        System.out.println("Maximum = " + max);
        System.out.println("Grade = " + grade(avg));
    }

    public static void main(String[] args) {
        int[] scores = {85, 90, 78, 92, 88};
```

17. Common Errors and Good Practices

- Give methods names that describe actions or returned information.
- Keep one method focused on one task.
- Do not create unnecessary global/static mutable data.
- Remember that parameters are local to the method.
- Do not try to overload methods only by changing the return type.
- Remember that Java is always pass-by-value.
- Ensure recursive methods have a reachable base case.
- Prefer clear iteration when recursion adds no real benefit.

18. Lecture Summary

- Methods divide programs into reusable units.
- Parameters receive copied argument values.
- A method may return a value or be declared `void`.
- Local variables and parameters have limited scope.
- `static` methods belong to the class.
- Method overloading uses different parameter lists.
- Java always passes argument values by value.
- Arrays can be passed to and returned from methods.
- Varargs provide flexible argument counts.
- Recursion solves a problem by reducing it to smaller instances of itself.

19. Review Questions

1. What is a method?
2. Why are methods useful?
3. What is the difference between a parameter and an argument?
4. What does `void` mean?
5. What is a return value?
6. What is local-variable scope?
7. What is a static method?
8. What is method overloading?
9. Why can methods not be overloaded only by return type?
10. Is Java pass-by-value or pass-by-reference?
11. What value is copied when an array variable is passed to a method?
12. What is a varargs parameter?
13. Where must a varargs parameter appear in a parameter list?
14. What is recursion?
15. What are the two essential parts of a recursive solution?

20. Programming Exercises

1. Write a method that returns the square of an integer.
2. Write a method that returns the larger of two numbers.
3. Write a method that tests whether an integer is even.
4. Write a method that calculates the area of a circle.
5. Overload a method called `sum` for integers and doubles.
6. Write a method that returns the minimum array element.
7. Write a method that returns the average of an integer array.
8. Write a method that reverses an array in place.
9. Create a varargs method that calculates a product.
10. Write a recursive method for factorial.
11. Write a recursive method for the sum $1 + \cdots + n$.
12. Refactor a long console program into at least five smaller methods.

21. Homework

Develop a modular console application called `StudentResultSystem`. Use separate methods to:

- calculate total score;
- calculate average score;
- find minimum and maximum scores;
- determine the final letter grade;
- count passed assessments;
- print the final report.

At least one method must return an array or accept an array parameter, and the program must demonstrate clearly that Java uses pass-by-value.

22. Key Terms

Term	Meaning
method	named block of code that performs a task
parameter	variable declared in a method header
argument	value supplied in a method call
return value	value sent back to the caller
scope	region of source code in which a name is accessible
overload	use the same method name with different parameter lists
pass-by-value	calling rule in which argument values are copied to parameters
varargs	variable-length argument list represented as an array inside the method
recursion	technique in which a method calls itself
base case	condition that terminates recursive calls