LOOPS. FOR loop

teacher: Майер С.Ф.
sfmayer@sfedu.ru

Introduction to Loops

Loops automate repetitive tasks — the third pillar of control structures (after sequence and selection).

- **Loop** = repeated execution of code block (loop body).
- Used when processing collections or repeating tasks.
- Java loop types:
  - **`for`** loop.
  - **`while`** loop.
  - **`do-while`** loop.

The for Loop — Syntax

```
for ([initializers]; [condition]; [iterator]) {
    // loop body
}
```

Components:

1. **Initializers:** set up loop counter (e.g., `int i = 0`).
2. **Condition:** checked before each iteration (e.g., `i < 10`).
3. **Iterator:** updates counter after each iteration (e.g., `i++`).

for **Loop — Example**

```
for (int i = 0; i < 10; i++) {
    System.out.println("Iteration: " + i);
}
```

Breakdown:

- `i = 0`: starts at zero.
- `i < 10`: runs while `i` is less than 10.
- `i++`: increments `i` by 1 after each loop.

Output: numbers 0 through 9.

Step-by-step execution table:

Iteration	`i` value	Output
1	0	"Iteration: 0"
2	1	"Iteration: 1"
...	...	...
10	9	"Iteration: 9"

Example

Task:

Write a Java program that prints the multiplication table for the number 5 (from 1 to 10).

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

```java
public class MultiplicationTable {
    public static void main(String[] args) {
        int number = 5;

        for (int i = 1; i <= 10; i++) {
            int result = number * i;
            System.out.println(number + " x " + i + " = " + result);
        }
    }
}
```

Explanation:

- int number = 5 : The number we're making a table for
- int i = 1; i <= 10; i++ : Loop from 1 to 10
- int result = number * i : Calculate 5 × current number
- System.out.println() : Print each line of the table

Example

Task:

Write a Java program that calculates the factorial of a given number **n = 6**. The factorial of a number (written as n!) is the product of all positive integers less than or equal to n.

Example: 5! = 5 × 4 × 3 × 2 × 1 = 120.

Print the calculation steps and the final result.

Calculating 6!...

1 x **1** = 1

1 x **2** = 2

2 x **3** = 6

6 x **4** = 24

24 x **5** = 120

120 x **6** = 720

The factorial of 6 is: 720

=

Calculating 6!...

6 x **5** = 30

30 x **4** = 120

120 x **3** = 360

360 x **2** = 720

720 x **1** = 720

The factorial of 6 is: 720

Solution 1

```java
public class FactorialCalculator {
    public static void main(String[] args) {
        int n = 6;
        long factorial = 1; // Use long to avoid overflow

        // Loop from 1 to n (ascending order works fine for factorial)
        for (int i = 1; i <= n; i++) {
            factorial *= i; // Same as: factorial = factorial * i
        }

        System.out.println("The factorial of " + n + " is: " + factorial);
    }
}
```

Example

The Vowel Counter

Task:

Write a Java program that takes a predefined word (e.g., "**implementation**") and uses a for loop to count how many vowels (**a, e, i, o, u**) it contains. Ignore case sensitivity. Print the final count.

```
Word: implementation
Number of vowels: 5
```

Solution

```
public class VowelCounter {
    public static void main(String[] args) {
        String word = "implementation";
        int vowelCount = 0;
        String lowerCaseWord = word.toLowerCase();

        for (int i = 0; i < lowerCaseWord.length(); i++) {
            char ch = lowerCaseWord.charAt(i);

            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                vowelCount++;
            }
        }

        System.out.println("Word: " + word);
        System.out.println("Number of vowels: " + vowelCount);
    }
}
```

The For-each Loop (Enhanced for)

Purpose: iterate over collections/arrays without manual index management.
Syntax:

```
for (Type element : collection) {
    // process element
}
```

Advantages:

- No need to know collection size.
- Cleaner, more readable code.
- Reduces risk of index errors.

collection → stream of elements → loop body

For-each — Example 1: String Characters

```
String str = "loops";
for (char c : str.toCharArray()) {
    System.out.print(c + " ");
}
// Output: l o o p s
```

Explanation:

- `str.toCharArray()` converts string to char array.
- Loop processes each character sequentially.

string → array → individual characters

For-each — Practical Example

Convert string to uppercase:

```
String str = "java";

for (char c : str.toCharArray()) {
        System.out.print(Character.toUpperCase(c) + " ");

}

// Output: J A V A
```

Tip: Use `Character` class methods for character manipulation.

input string → uppercase transformation → spaced output

When to Use For-each vs. Traditional for

Use For-each	Use Traditional for
No index needed	Need element index
Read/process elements	Traverse in reverse order
Code readability matters	Modify primitive array elements
Dynamic collections	Complex index-dependent condi tions

need index? → yes/no → choose loop type

Nested Loops — Concept

Definition:

- One loop inside another.
- Outer loop runs once → inner loop runs completely → repeat.
- Total iterations = outer × inner.

Common use cases:

- Multiplication tables.
- Matrix/grid processing.
- Combinations (e.g., coordinates).

Example

Rectangle of Symbols

Task:

Write a Java program that uses nested loops to print a rectangle with 4 rows and 8 columns, filled with the # symbol.

```
########
########
########
########
```

```java
public class RectanglePrinter {
    public static void main(String[] args) {
        int rows = 4;
        int columns = 8;

        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= columns; j++) {
                System.out.print("#");
            }
            System.out.println();
        }
    }
}
```

Explanation:

- **Outer loop (i)** : Controls the number of rows (rows = 4)
- **Inner loop (j)** : For each row, prints the # symbol columns times (columns = 8)
- System.out.print("#") : Prints the symbol without moving to a new line
- System.out.println() : After the inner loop finishes, moves to the next row

Example

Multiplication Table

Task:

Write a Java program that uses nested loops to print a multiplication table from 1 to 5.

```
1  2  3  4  5
2  4  6  8 10
3  6  9 12 15
4  8 12 16 20
5 10 15 20 25
```

Solution

```
public class MultiplicationTable {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                System.out.print((i * j) + "\t");
            }
            System.out.println();
        }
    }
}
```

Explanation:

- **Outer loop (**i**)** : Goes through rows (numbers from 1 to 5)
- **Inner loop (**j**)** : For each row, goes through columns (also from 1 to 5)
- System.out.print((i * j) + "\t") : Calculates and prints the product, \t adds a tab for alignment
- System.out.println() : After the inner loop finishes, moves to a new line

Nested Loops — Key Rules

Best practices:

1. Use different counter variables (e.g., `i`, `j`).
2. Keep inner loop logic simple.
3. Avoid excessive nesting (limit to 2–3 levels).
4. Add comments for clarity.

Pitfalls:

- Infinite loops (check conditions).
- Performance issues ($O(n^2)$ complexity).