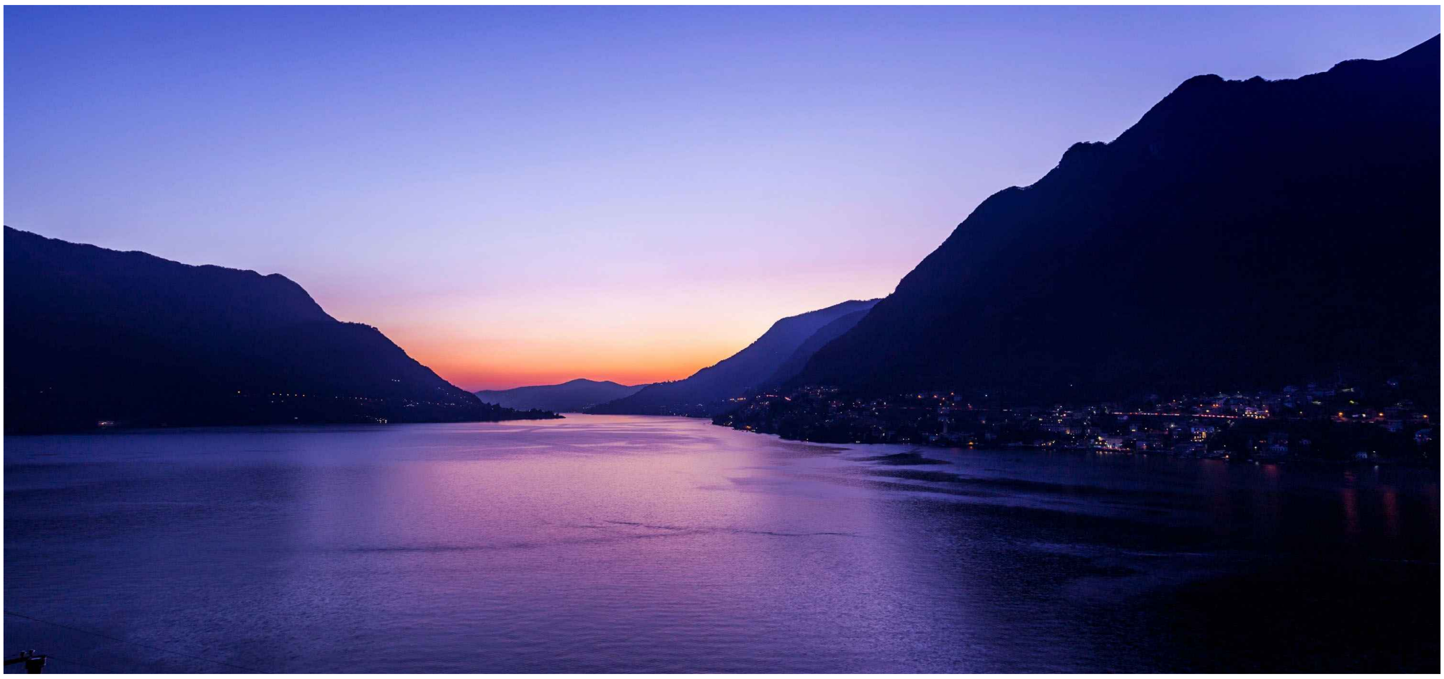


5-Loops I

Author: Vincent Lau

Learning Objectives

> Understand the reason for using FOR loop
>
> Understand the syntax of FOR loop

In Java, the `for` loop is used to repeatedly execute a block of code for a specified number of times. It is commonly used when you know the exact number of iterations needed.

Why do we need Loops

Imagine you are asked to print value each time after multiplying it by 2 and repeating it 10 times.

```java
public class Main {
  public static void main(String[] args) {
    int x = 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
  }
}
// Output
// 2 4 8 16 32 64 128 256 512 1024 2048
```

However, does this solution make sense in programming?

Programming is not doing repreated task by duplicating the codes. Loops can solve this requirement in a more elegant way. Basically, most programming languages have loops. Java has its own way of using loops.

Solution by FOR Loop

For-loop is one of the ways to make loops happen in Java.

Using looping statements, the above code of **10** lines can be **reduced to 3 lines, and with the same result**.

No worries, we will learn in this chapter step by step.

```java
public class Main {
  public static void main(String[] args) {
    int x = 2;
    for (int i = 0; i < 11; i++) {
      System.out.print(x + " ");
```

```
        x *= 2;
      }
    }
}
// Output
// 2 4 8 16 32 64 128 256 512 1024 2048
```

FOR Loop

Basic Syntax

```
for (initialization; condition; update) {
    // Code to be executed in each iteration
}
```

Here's an explanation of the different parts of the `for` loop:

- `initialization` : Initializes the loop counter or any other variables used in the loop. It is executed only once before the loop starts.
- `condition` : Specifies the condition that must be true for the loop to continue executing. If the condition evaluates to `false` , the loop terminates.
- `update` : Updates the loop counter or any other variables after each iteration. It is executed at the end of each iteration.

Now, let's look at examples that showcase the usage of the `for` loop:

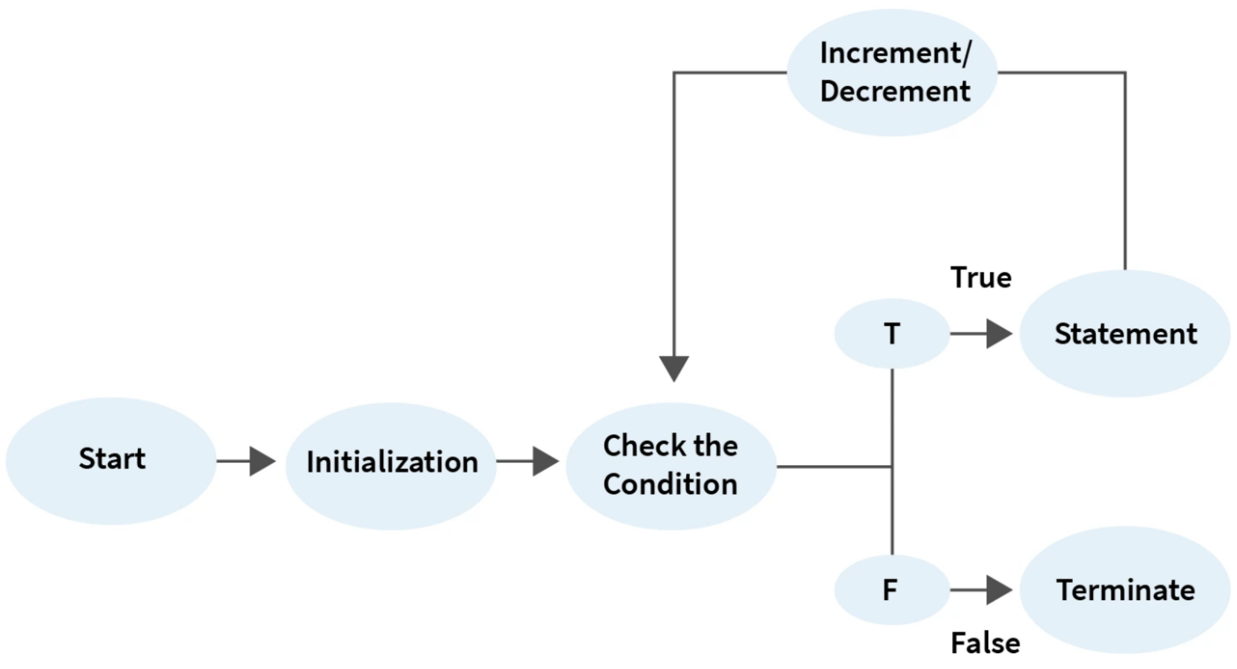
Example 1: Printing Numbers from 1 to 5

```
int maxNum = 5;
/* loop will run 5 times until test condition becomes false */
for (int i = 1; i <= maxNum; i++) {
    System.out.println(i);
}
```

```
// Output
1
2
```

```
3
4
5
```

Here, initially i is 1, which is less than maxNum so it is printed. Then i is incremented and printed until it is greater than maxNum.



Iteration	Variable	Condition i<=maxNum	Action and Update Expression
1st	i=1, maxNum=5	true	Print 1 and Increment i
2nd	i=2, maxNum=5	true	Print 2 and Increment i
3rd	i=3, maxNum=5	true	Print 3 and Increment i
4th	i=4, maxNum=5	true	Print 4 and Increment i
5th	i=5, maxNum=5	true	Print 5 and Increment i
6th	i=6, maxNum=5	false	Terminate Loop

Example 2: Summing Numbers from 1 to 10

```java
int sum = 0;

for (int i = 1; i <= 10; i++) {
    sum += i;
}

System.out.println("Sum: " + sum);
```

In this example, the `for` loop is used to calculate the sum of numbers from 1 to 10. The `sum` variable is initialized to 0, and `i` is incremented from 1 to 10. The sum of the numbers will be stored in the `sum` variable and printed at the end.

Example 3: Counting Backward

```java
for (int i = 10; i >= 1; i--) {
    System.out.println(i);
}
```

In this example, the `for` loop starts with `i` initialized to 10, continues as long as `i` is greater than or equal to 1, and decrements `i` by 1 in each iteration. The loop will print the numbers from 10 to 1 in descending order.

Example 4:

Infinite Loop

An infinite loop is a looping construct that executes the loop body infinitely until the user manually stops the execution. It is also called an **endless loop** or **indefinite loop**. It doesn't terminate at all.

Example 1: Logical mistakes

```java
for(int i = 0; i >= 0; i++) { /
    // code
}
```

Example 2: No Condition

```java
for (int i = 0; ; i++) {
  System.out.println("Hello World");
}

for( ; ; ){
    // code
}
```

Empty Loop

An empty loop is a loop that doesn’ t have a body or has an empty body. It has applications too. It is used to delay execution by introducing a costly loop that does nothing.

It is different from the infinite loop as in infinite loop condition is true throughout and the loop may have a body, whereas the empty loop runs for a large but finite number of times and it doesn’ t have a body.

Example 1: Empty body

```
for(int i = 0; i <= 1000; i++){
    // empty
}
```

Nested for Loops

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        System.out.println("i: " + i + ", j: " + j);
    }
}
```

In this example, there are two nested `for` loops. The outer loop runs 3 times, and the inner loop runs 3 times for each iteration of the outer loop.

The inner loop completes its iterations for each value of the outer loop.

The output will be:

```
// The output of the above example
i: 1, j: 1
i: 1, j: 2
i: 1, j: 3
```

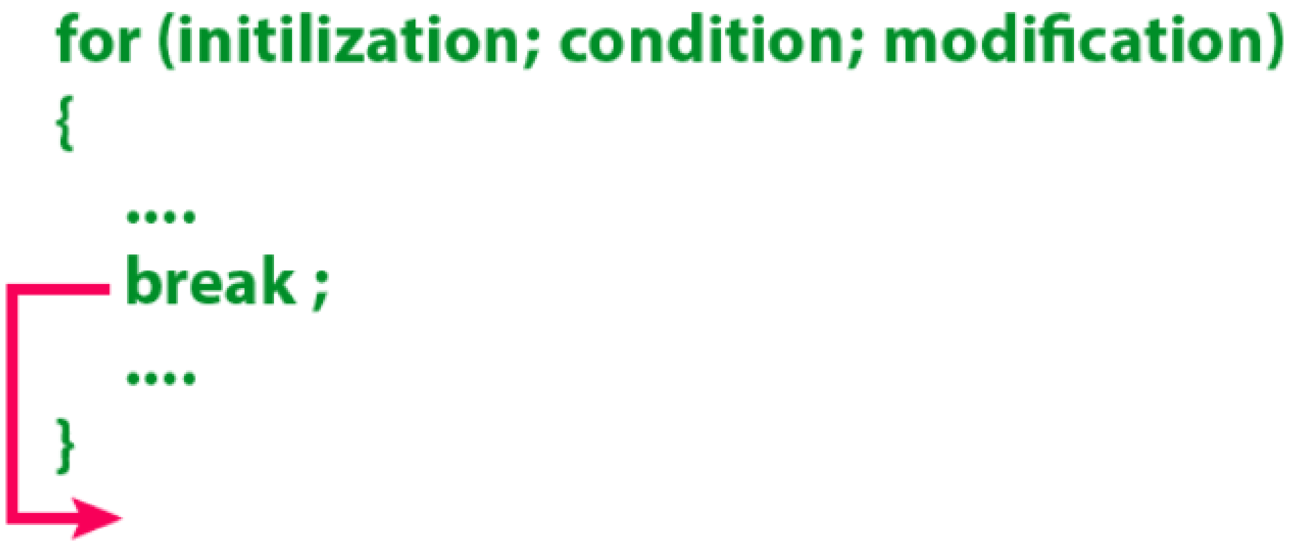
```
i: 2, j: 1
i: 2, j: 2
i: 2, j: 3
i: 3, j: 1
i: 3, j: 2
i: 3, j: 3
```

Break & Continue

The `break` and `continue` keywords are used to **alter the flow of control in conditionals or loops**.

- `break` is used to break out of a code block
- `continue` is used to skip the remaining code in a code block and continue the loop

Break



```
for (initilization; condition; modification)
{
    ....
    break ;
    ....
}
```

The `break` statement is used in a `for` loop to prematurely terminate the loop and exit its execution.

When `break` is encountered, the control flow immediately exits the loop, and the program continues with the next statement after the loop.

Here's an example to demonstrate the usage of `break` in a `for` loop:

Example 1: Simple Break

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break; // Exit the loop when i equals 5
    }
    System.out.println(i);
}
```

```
// output
1
2
3
4
```

In this example, the `for` loop iterates from 1 to 10. However, when `i` reaches the value of 5, the `break` statement is encountered, and the loop is immediately terminated. As a result, the loop will only print the numbers 1 to 4.

The `break` statement is often used in conjunction with an `if` condition to exit a loop based on a specific condition. It allows you to control the flow of execution within the loop and break out of it when a certain condition is met.

It's worth noting that if there are nested loops, the `break` statement will only exit the innermost loop that contains it. If you want to break out of multiple nested loops simultaneously, you can use labeled `break` statements.

Example 2: Labeled Break

```
outerLoop: for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i * j == 6) {
            break outerLoop; // Exit both loops when i*j equals 6
        }
        System.out.println("i: " + i + ", j: " + j);
    }
}
```

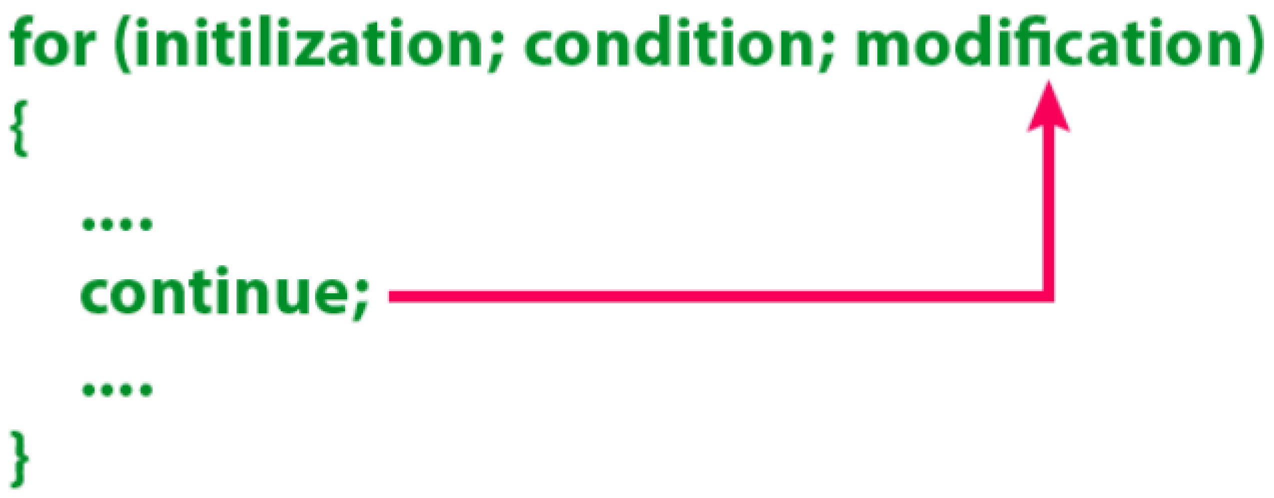
In this example, the labeled `break` statement `break outerLoop;` is used to **exit both the outer and inner loops when the condition `i * j == 6` is satisfied**. This allows you to break out of multiple levels of nested loops in a single step.

```
// output
```

```
i: 1, j: 1
i: 1, j: 2
i: 1, j: 3
i: 2, j: 1
i: 2, j: 2
```

The `break` statement provides flexibility in controlling the flow of a `for` loop, allowing you to terminate the loop prematurely based on specific conditions.

Continue



The `continue` statement is used in a `for` loop to skip the remaining code within the loop for the current iteration and proceed to the next iteration.

When `continue` is encountered, the loop execution jumps to the next iteration's evaluation step.

Here's an example to demonstrate the usage of `continue` in a `for` loop:

Example 1: Simple Continue

```java
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue; // Skip even numbers
    }
    // .. code
    System.out.println(i);
}
```

```
// output
1
3
5
7
9
```

In this example, the `for` loop iterates from 1 to 10. However, when `i` is an even number (i.e., the condition `i % 2 == 0` is true), the `continue` statement is encountered. As a result, the remaining code within the loop is skipped for that iteration, and the loop proceeds to the next iteration. Therefore, only odd numbers will be printed.

The `continue` statement allows you to control the flow of a `for` loop and selectively skip certain iterations based on specific conditions. It is often used in conjunction with an `if` condition to skip certain processing steps for particular loop iterations.

Example 2: Labeled Continue

Similar to the `break` statement, if there are nested loops, the `continue` statement only affects the innermost loop that contains it. If you want to skip iterations in outer loops as well, you can use labeled `continue` statements.

Here's an example of using a labeled `continue` statement in nested loops:

```
outerLoop: for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            continue outerLoop; // Skip to the next iteration of the outer loop
        }
        System.out.println("i: " + i + ", j: " + j);
    }
}
```

```
// output
i: 1, j: 1
i: 1, j: 2
i: 1, j: 3
i: 2, j: 1
i: 3, j: 1
i: 3, j: 2
i: 3, j: 3
```

Challenge

- Counting the number of char p

```java
String searchMe = "peter piper picked a peck of pickled peppers";
int num = 0;

for (int i = 0; i < searchMe.length(); i++) {
    if (searchMe.charAt(i) != 'p') {
        continue;
    }
    num++;
}

System.out.println("Found " + num + " p's in the string."); // num = ?
```