Chapter 5: Loops in Java

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop.

In Java, there are three types of loops.

- **while loop**
- **do...while loop**
- **for loop**

1. Java while loop

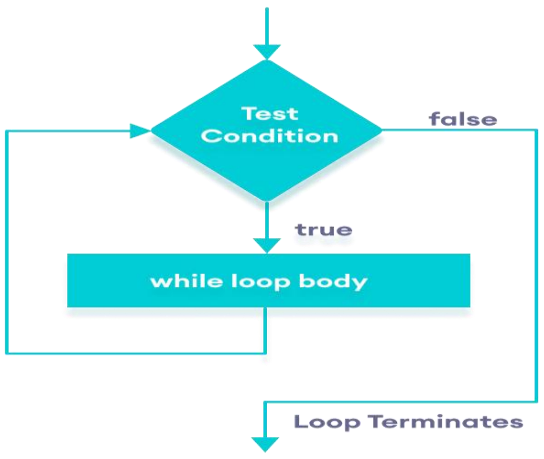
Java `while` loop is used to run a specific code until a certain condition is met. The syntax of the `while` loop is:

```
while (testExpression) {
    // body of loop
}
```

Here,

1. A `while` loop evaluates the **textExpression** inside the parenthesis `()`.
2. If the **textExpression** evaluates to `true`, the code inside the `while` loop is executed.
3. The **textExpression** is evaluated again.
4. This process continues until the **textExpression** is `false`.
5. When the **textExpression** evaluates to `false`, the loop stops.

Flowchart of while loop



Example 1: Display Numbers from 1 to 5

```
// Program to display numbers from 1 to 5
class Main {
  public static void main(String[] args) {
    // declare variables
    int i = 1, n = 5;
    // while loop from 1 to 5
    while(i <= n) {
      System.out.println(i);
      i++;
    }
  }
}
```

Output

```
1
2
3
4
5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

1st	`i = 1`
	`n = 5`
		`true`	`1` is printed.
			`i` is increased to **2**.

2nd	`i = 2`
	`n = 5`
		`true`	`2` is printed.
			`i` is increased to **3**.

3rd	`i = 3`
	`n = 5`
		`true`	`3` is printed.
			`i` is increased to **4**.

4th	`i = 4`
	`n = 5`
		`true`	`4` is printed.
			`i` is increased to **5**.

Iteration	Variable	Condition: i <= n	Action
5th	`i = 5` `n = 5`	`true`	`5` is printed. `i` is increased to **6**.
6th	`i = 6` `n = 5`	`false`	The loop is terminated

---------------------------*************************---------------------------------

Example 2: Sum of Positive Numbers Only

```java
// Java program to find the sum of positive numbers
import java.util.Scanner;
class Main {
  public static void main(String[] args) {
    int sum = 0;

    // create an object of Scanner class
    Scanner input = new Scanner(System.in);

    // take integer input from the user
    System.out.println("Enter a number");
    int number = input.nextInt();

    // while loop continues
    // until entered number is positive
    while (number >= 0) {
      // add only positive numbers
      sum += number;

      System.out.println("Enter a number");
      number = input.nextInt();
    }

    System.out.println("Sum = " + sum);

  }
}
```

Output

```
Enter a number
25
Enter a number
9
Enter a number
5
Enter a number
-3
Sum = 39
```

In the above program, we have used the Scanner class to take input from the user. Here, `nextInt()` takes integer input from the user.

The `while` loop continues until the user enters a negative number. During each iteration, the number entered by the user is added to the `sum` variable.

When the user enters a negative number, the loop terminates. Finally, the total sum is displayed

----------------*******************-----------------

2. Java do...while loop

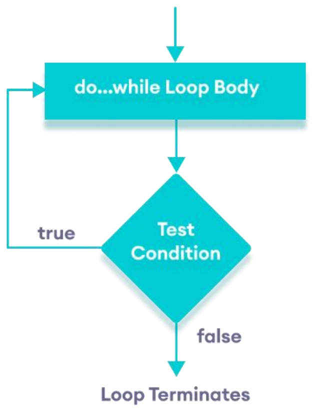
The `do...while` loop is similar to while loop. However, the body of `do...while` loop is executed once before the test expression is checked. For example,

```
do {
    // body of loop
} while(textExpression)
```

Here,

1. The body of the loop is executed at first. Then the **textExpression** is evaluated.
2. If the **textExpression** evaluates to `true`, the body of the loop inside the `do` statement is executed again.
3. The **textExpression** is evaluated once again.
4. If the **textExpression** evaluates to `true`, the body of the loop inside the `do` statement is executed again.
5. This process continues until the **textExpression** evaluates to `false`. Then the loop stops.

Flowchart of do...while loop



Example 3: Display Numbers from 1 to 5

```
// Java Program to display numbers from 1 to 5

import java.util.Scanner;

// Program to find the sum of natural numbers from 1 to 100.

class Main {
  public static void main(String[] args) {

    int i = 1, n = 5;

    // do...while loop from 1 to 5
    do {
      System.out.println(i);
      i++;
    } while(i <= n);
  }
}
```

Output

```
1
2
3
4
5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

	`i = 1`
	`n = 5`
		not checked	`1` is printed.
			`i` is increased to **2**.

1st	`i = 2`
	`n = 5`
		`true`	`2` is printed.
			`i` is increased to **3**.

2nd	`i = 3`
	`n = 5`
		`true`	`3` is printed.
			`i` is increased to **4**.

3rd	`i = 4`
	`n = 5`
		`true`	`4` is printed.
			`i` is increased to **5**.

4th	`i = 5`
	`n = 5`
		`true`	`6` is printed.
			`i` is increased to **6**.

5th	`i = 6`
	`n = 5`
		`false`	The loop is terminated

Example 4: Sum of Positive Numbers

Example 3: Display Numbers from 1 to 5

```
// Java Program to display numbers from 1 to 5

import java.util.Scanner;

// Program to find the sum of natural numbers from 1 to 100.

class Main {
  public static void main(String[] args) {
    int i = 1, n = 5;

    // do...while loop from 1 to 5
    do {
      System.out.println(i);
      i++;
    } while(i <= n);
  }
}
```

Output

```
1
2
3
4
5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

	`i = 1`
	`n = 5`
		not checked	`1` is printed.
			`i` is increased to **2**.

1st	`i = 2`
	`n = 5`
		`true`	`2` is printed.
			`i` is increased to **3**.

2nd	`i = 3`
	`n = 5`
		`true`	`3` is printed.
			`i` is increased to **4**.

3rd	`i = 4`
	`n = 5`
		`true`	`4` is printed.
			`i` is increased to **5**.

4th	`i = 5`
	`n = 5`
		`true`	`6` is printed.
			`i` is increased to **6**.

5th	`i = 6`
	`n = 5`
		`false`	The loop is terminated

Example 4: Sum of Positive Numbers

```java
// Java program to find the sum of positive numbers
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    int sum = 0;
    int number = 0;

    // create an object of Scanner class
    Scanner input = new Scanner(System.in);

    // do...while loop continues
    // until entered number is positive
    do {
      // add only positive numbers
      sum += number;
      System.out.println("Enter a number");
      number = input.nextInt();
    } while(number >= 0);

    System.out.println("Sum = " + sum);

  }
}
```

Output 1

```
Enter a number
25
Enter a number
9
Enter a number
5
Enter a number
-3
Sum = 39
```

Here, the user enters a positive number, that number is added to the `sum` variable. And this process continues until the number is negative. When the number is negative, the loop terminates and displays the sum without adding the negative number.

Output 2

```
Enter a number
-8
Sum is 0
```

Here, the user enters a negative number. The test condition will be `false` but the code inside of the loop executes once.

-------------------------------********************-------------------------------

Infinite while loop

If **the condition** of a loop is always `true`, the loop runs for infinite times (until the memory is full). For example,

```
// infinite while loop
while(true){
    // body of loop
}
```

Here is an example of an infinite `do...while` loop.

```
// infinite do...while loop
int count = 1;
do {
   // body of loop
} while(count == 1)
```

In the above programs, the **textExpression** is always `true`. Hence, the loop body will run for infinite times.

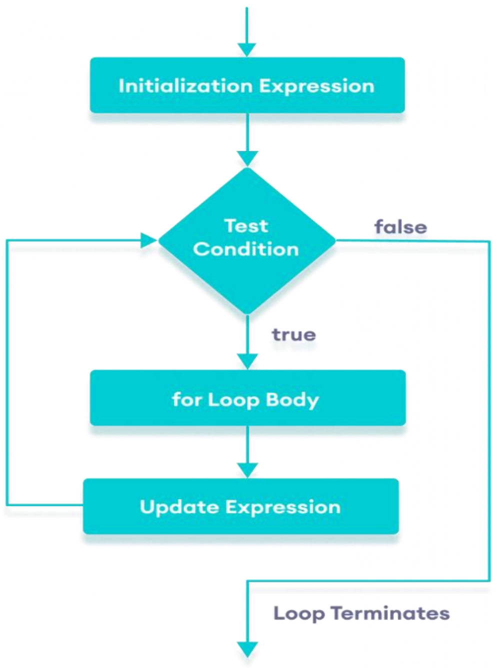
3. Java for loop

Java for loop is used to run a block of code for a certain number of times. The syntax of for loop is:

```
for (initialExpression; testExpression; updateExpression) {
    // body of the loop
}
```

Here,

1. The **initialExpression** initializes and/or declares variables and executes only once.
2. The **condition** is evaluated. If the **condition** is true, the body of the for loop is executed.
3. The **updateExpression** updates the value of **initialExpression**.
4. The **condition** is evaluated again. The process continues until the **condition** is false.



Example 5: Display a Text Five Times

```
// Program to print a text 5 times

class Main {
  public static void main(String[] args) {

    int n = 5;
    // for loop
    for (int i = 1; i <= n; ++i) {
      System.out.println("Java is fun");
    }
  }
}
```

Output

```
Java is fun
Java is fun
Java is fun
Java is fun
Java is fun
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

1st	`i = 1`
	`n = 5`
		`true`	`Java is fun` is printed.
			`i` is increased to **2**.

2nd	`i = 2`
	`n = 5`
		`true`	`Java is fun` is printed.
			`i` is increased to **3**.

3rd	`i = 3`
	`n = 5`
		`true`	`Java is fun` is printed.
			`i` is increased to **4**.

4th	`i = 4` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **5**.
5th	`i = 5` `n = 5`	`True`	`Java is fun` is printed. `i` is increased to **6**.
6th	`i = 6` `n = 5`	`False`	The loop is terminated.

Example 6: Display numbers from 1 to 5

```
// Program to print numbers from 1 to 5

class Main {
  public static void main(String[] args) {

    int n = 5;
    // for loop
    for (int i = 1; i <= n; ++i) {
      System.out.println(i);
    }
  }
}
```

Output

```
1
2
3
4
5
```

Here is how the program works.

Iteration	Variable	Condition: i <= n	Action
1st	`i = 1` `n = 5`	`True`	`1` is printed. `i` is increased to **2**.
2nd	`i = 2` `n = 5`	`True`	`2` is printed. `i` is increased to **3**.
3rd	`i = 3` `n = 5`	`True`	`3` is printed. `i` is increased to **4**.
4th	`i = 4` `n = 5`	`True`	`4` is printed. `i` is increased to **5**.
5th	`i = 5` `n = 5`	`True`	`5` is printed. `i` is increased to **6**.
6th	`i = 6` `n = 5`	`False`	The loop is terminated.

Example 7: Display Sum of n Natural Numbers

```java
// Program to find the sum of natural numbers from 1 to 1000.

class Main {
  public static void main(String[] args) {
    int sum = 0;
    int n = 1000;

    // for loop
    for (int i = 1; i <= n; ++i) {
      // body inside for loop
      sum += i;     // sum = sum + i
    }
    System.out.println("Sum = " + sum);
  }
}
```

Output:

```
Sum = 500500
```

Here, the value of `sum` is **0** initially. Then, the for loop is iterated from `i = 1 to 1000`. In each iteration, `i`is added to `sum` and its value is increased by **1**.

When `i` becomes **1001**, the test condition is `false` and `sum` will be equal to `0 + 1 + 2 + .... + 1000`.

The above program to add the sum of natural numbers can also be written as

```
// Program to find the sum of natural numbers from 1 to 1000.

class Main {
  public static void main(String[] args) {

    int sum = 0;
    int n = 1000;

    // for loop
    for (int i = n; i >= 1; --i) {
      // body inside for loop
      sum += i;     // sum = sum + i
    }

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

The output of this program is the same as the **Example 7**.

➢ Java break and continue Statements

While working with loops, it is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

The break statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.

Here is the syntax of the break statement in Java:

```
break;
```

After the continue statement, the program moves to the end of the loop. And, test expression is evaluated (update statement is evaluated in case of the for loop).

Here's the syntax of the continue statement.

```
continue;
```

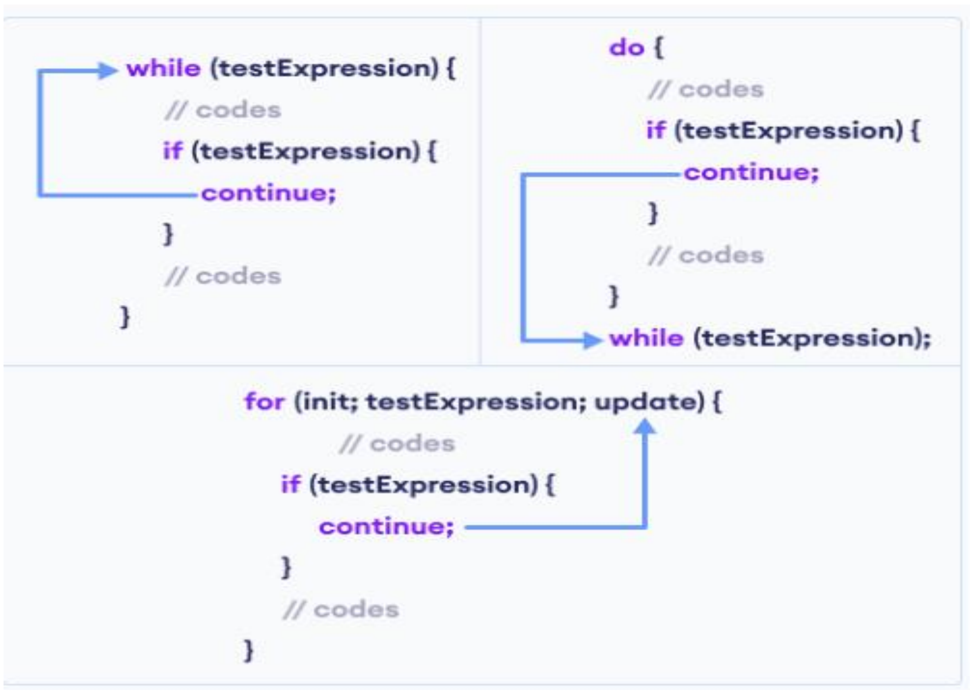
Example 8: Java break statement

```
class Test {
    public static void main(String[] args) {
        // for loop
        for (int i = 1; i <= 10; ++i) {
            // if the value of i is 5 the loop terminates
            if (i == 5) {
                break;
            }
            System.out.println(i);
        }
    }
}
```

Output:

```
1
2
3
4
```

Working of Java continue statement



Example 9: Java continue statement

```java
class Main {
  public static void main(String[] args) {
    // for loop
    for (int i = 1; i <= 10; ++i) {
      // if value of i is between 4 and 9
      // continue is executed
      if (i > 4 && i < 9) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

Output:

```
1
2
3
4
9
10
```

In the above program, we are using the `for` loop to print the value of `i` in each iteration. Here, the `continue` statement is executed when the value of `i` becomes more than **4** and less than **9**. It then skips the print statement inside the loop. Hence, we get the output with values **5**, **6**, **7**, and **8** skipped.

Example 10: Compute the sum of 5 positive numbers

```java
import java.util.Scanner;
class Main {
  public static void main(String[] args) {

    Double number, sum = 0.0;
    // create an object of Scanner
    Scanner input = new Scanner(System.in);
    for (int i = 1; i < 6; ++i) {
      System.out.print("Enter number " + i + " : ");
      // takes input from the user
      number = input.nextDouble();
      // if number is negative
      // continue statement is executed
      if (number <= 0.0) {
        continue;
      }

      sum += number;
    }
    System.out.println("Sum = " + sum);
  }
}
```

Output:

```
Enter number 1: 2.2
Enter number 2: 5.6
Enter number 3: 0
Enter number 4: -2.4
Enter number 5: -3
Sum = 7.8
```

In the above example, we have used the for loop to print the sum of 5 positive numbers. Here, when the user enters a negative number, the `continue` statement is executed. This skips the current iteration of the loop and takes the program control to the update expression of the loop.