Java Loops

Loops are a fundamental concept in programming that allow us to execute a block of code multiple times based on a condition. In Java, we have several types of loops that serve different purposes. These include the **for loop**, **while loop**, **do-while loop**, and **for-each loop**. Additionally, the `break` and `continue` statements can control the flow of these loops.

1. For Loop

The **For Loop** is used when you know how many times you want to execute a statement or a block of statements. It consists of three parts: initialization, condition, and iteration.

Syntax of `for` Loop:

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

- **Initialization**: Used to initialize a counter variable.
- **Condition**: Defines the condition that must be true for the loop to continue.
- **Iteration**: Updates the counter variable after each loop iteration.

Example:

```
for (int i = 0; i < 5; i++) {
    System.out.println("The value of i is: " + i);
}
```

- **Output**:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

Explanation:

- Starts with `i = 0`, checks if `i < 5`, then prints the value of `i`.
- After each iteration, `i` is incremented by 1 (due to `i++`).

---

2. While Loop

The **While Loop** is used when you want to execute a block of code as long as a specific condition is true. The condition is checked before entering the loop.

Syntax of `while` Loop:

```
while (condition) {
    // Code to be executed
}
```

Example:

```
int i = 0;
while (i < 5) {
    System.out.println("The value of i is: " + i);
    i++;  // Increment the counter
}
```

- **Output**:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

Explanation:

- The loop continues as long as the condition `i < 5` is true.
- The counter variable `i` is incremented inside the loop to ensure the loop terminates.

---

3. Do-While Loop

The **Do-While Loop** is similar to the `while` loop, but with one key difference: the condition is evaluated after the code block is executed. This ensures that the loop is executed at least once.

Syntax of `do-while` Loop:

```
do {
    // Code to be executed
```

```
} while (condition);
```

Example:

```
int i = 0;
do {
    System.out.println("The value of i is: " + i);
    i++;  // Increment the counter
} while (i < 5);
```

- **Output**:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

Explanation:

- The code block is executed once before checking the condition (`i < 5`), ensuring the loop runs at least once.

---

4. For-Each Loop

The **For-Each Loop** is a simplified version of the `for` loop and is used to iterate over elements in arrays or collections (such as Lists, Sets, or Maps). It eliminates the need for manual indexing.

Syntax of `for-each` Loop:

```
for (type variable : collection) {
    // Code to be executed
}
```

Example:

```
String[] fruits = {"Apple", "Banana", "Cherry", "Date"};
for (String fruit : fruits) {
    System.out.println(fruit);
}
```

- **Output**:

```
Apple
Banana
Cherry
Date
```

Explanation:

- The `for-each` loop iterates through each element in the `fruits` array, and the value of each element is assigned to the `fruit` variable.

---

5. Break and Continue

Both `break` and `continue` are used to control the flow of loops, but in different ways:

Break Statement

The `break` statement is used to exit the loop prematurely when a certain condition is met.

Syntax of `break`:

```
if (condition) {
    break;  // Exits the loop
}
```

Example:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;  // Exit the loop when i equals 5
    }
    System.out.println("The value of i is: " + i);
}
```

- **Output**:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

Explanation:

- The loop terminates when `i == 5` because of the `break` statement.

Continue Statement

The `continue` statement is used to skip the current iteration of the loop and move on to the next iteration when a condition is met.

Syntax of `continue`:

```
if (condition) {
    continue;  // Skip this iteration
}
```

Example:

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue;  // Skip the iteration when i equals 2
    }
    System.out.println("The value of i is: " + i);
}
```

- **Output**:

```
The value of i is: 0
The value of i is: 1
The value of i is: 3
The value of i is: 4
```

Explanation:

- When `i == 2`, the `continue` statement skips the print statement for that iteration, and the loop continues with the next value of `i`.

Key Points

- **For Loop**:
  - Useful when you know the exact number of iterations.
  - Contains initialization, condition, and iteration steps.
- **While Loop**:

    - Useful when you want to loop while a condition is true.
    - Condition is checked before the loop body is executed.

- **Do-While Loop**:
    - Similar to `while` loop, but guarantees the loop is executed at least once.
    - Condition is checked after the loop body is executed.

- **For-Each Loop**:
    - Simplifies iteration over arrays or collections.
    - Ideal for iterating through elements without using an index.

- **Break**:
    - Used to exit a loop prematurely when a certain condition is met.

- **Continue**:
    - Skips the current iteration and proceeds to the next iteration of the loop.

Conclusion

- Loops are fundamental for automating repetitive tasks in programming.
- **For**, **while**, and **do-while** loops serve different purposes depending on whether you know the number of iterations or need to run the loop as long as a condition is true.
- **For-each** is a more readable and concise way to iterate over collections or arrays.
- **Break** and **continue** provide control over the loop flow for more precise behavior.
