Java Looping Constructs

Java offers several looping constructs that allow a set of instructions to be executed repeatedly based on a condition.

1. Why Use Loops?

Loops help us:

- Repeat tasks without writing the same code again.
- Handle tasks like printing numbers, adding scores, or processing arrays efficiently.

2. Types of Loops in Java

a) while Loop

Syntax:

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

The statements inside run as long as the condition is true.

Example:

```
int i = 1;
while (i <= 5) {
   System.out.println(i);
   i++;
}
// Output: 1 2 3 4 5
```

b) do-while Loop

Syntax:

```
do {
// Code to be repeated
} while (condition);
```

The statements inside run at least once, even if the condition is false at first, because the test happens after running the block once. (This is why, while is called *"Entry-Controlled Loop"* while the do-while is *called "Exit-Controlled Loop"*)

Example:

```
int i = 1;
do {
   System.out.println(i);
   i++;
} while (i <= 5);
// Output: 1 2 3 4 5
```

c) for Loop

Syntax:

```
for (initialization; condition; increment/decrement) {
 // Code to be repeated
}
```

Most commonly used when the number of repetitions is known beforehand.

Example:

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

}
```

3. Key Points to Remember

- Use while when you do not know how many times the loop will run.
- Use for when you know the number of repetitions while writing the code.
- The do-while loop always executes at least once.
- The keyword "break" stops the loop immediately.

```
// Program to illustrate the use of "break"
int i = 1;
while (i<=10) {
    if (i==5) {
        break;
    }
    System.out.println(i);
    i++;
}
// Output: 1 2 3 4    (Loop breaks when i is 5)
```

- The keyword "continue" skips to the next loop iteration.

```
public class ContinueExample
{
    public static void main(String[] args)
    {
        System.out.println("Printing odd numbers between 1 and 10 using continue:");
        for (int i = 1; i <= 10; i++)
        {
            // If the number is even, skip the rest of the loop iteration
            if (i % 2 == 0)
            {
                continue;  // Skip even numbers
            }
            System.out.println(i);
        } // end of for
    }     // end of main()
}         // end of class
```

- Always check the loop's condition to avoid infinite loops.
- Increment or decrement variables correctly.
- Loops can be nested (a loop inside another loop)
- Modern Java also include enhanced for-each loops for collections/arrays.

```
public class ForEachExample
{
    public static void main(String[] args)
    {
        // Create an array of numbers
        int[] numbers = {10, 20, 30, 40, 50};
        System.out.println("Printing numbers using for-each loop:");
        // For-each loop to print each number
        for (int num : numbers)
        {
            System.out.println(num);
        }
    }
}
```

Learning loops is a building block for most programming concepts and almost always asked in all the examinations involving computer programming.

Loops are essential for tasks like processing arrays (A type of data structure), reading data from a file, implementing different algorithms, and many other situations where repetitive actions are needed.