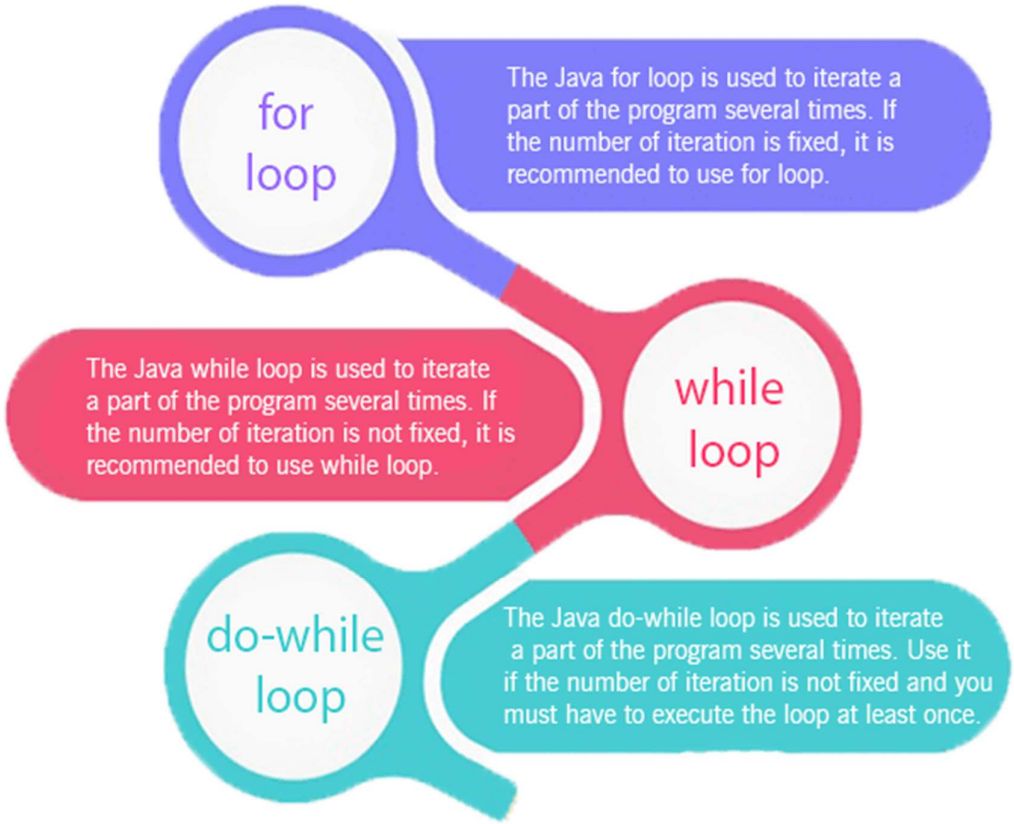
16. Loops in Java

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. for-each (or) enhanced for loop
3. while loop
4. do-while loop



16.1 for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

Syntax:

```
for(initialization, condition, increment/decrement)
{
//block of statements
}
```

for loop Example Program:

```java
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample
{
	public static void main(String[] args)
	{
		//Code of Java for loop
		for(int i=1;i<=10;i++)
		{
			System.out.println(i);
		}
	}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

16.1.1 Nested for loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Nested for loop Example:

```
public class NestedForExample
{
      public static void main(String[] args)
      {
            //loop of i
            for(int i=1;i<=3;i++)
            {
                  //loop of j
                  for(int j=1;j<=3;j++)
                  {
                     System.out.println(i+" "+j);
                  }//end of i
            }//end of j
      }
}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

16.1.2 Infinite for loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Syntax:

for(;;)

{

//code to be executed

}

Infinite for loop Example:

```
//Java program to demonstrate the use of infinite for loop
//which prints an statement
```

```
public class InfiniteForExample
{
      public static void main(String[] args)
      {
    //Using no condition in for loop
    for(;;)
    {
        System.out.println("infinitive loop");
    }
    }
}
```

Output:

```
infinitive loop
infinitive loop
infinitive loop
infinitive loop
infinitive loop
infinitive loop
.
.
.
.
```

16.2 for-each (or) enhanced for loop

The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation. It works on the basis of elements and not the index. It returns element one by one in the defined variable.

Syntax:

for(data_type variable : array_name)

{

//code to be executed

}

for-each Example Program:

```
//Java For-each loop example which prints the
//elements of the array
public class ForEachExample
{
      public static void main(String[] args)
      {
```

```
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr)
    {
        System.out.println(i);
    }
    }
}
```

Output:

```
12
23
44
56
78
```

16.3 while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop. It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed. The Java while loop is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops. The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

Syntax:

while (condition)

{

//code to be executed

Increment / decrement statement;

}

while loop Example:

```
public class WhileExample
{
      public static void main(String[] args)
      {
            int i=1;
            while(i<=10)
            {
                  System.out.println(i);
```

```
                i++;
            }
        }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

16.3.1 Nested while loop

When a loop is inside a loop then it is called as nested loop.

Syntax:

while(condition1)
{
//statements
while(condition2)
{
//statements
Increment/Decrement;
}
Increment/Decrement;
}

Nested while loop Example:

```
public class NestedWhileExample
{
      public static void main(String[] args)
      {
            int i=1;
            while(i<=3)
            {
                  int j=1;
                  while(j<=3)
                  {
                        System.out.print(i+" "+j);
                      j++;
```

```
                    System.out.println();
                }
                i++;

            }
        }
}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

16.3.2 Infinite while loop

If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

while(true)

{

//code to be executed

}

Infinite while loop Example:

```
public class InfiniteWhileExample
{
public static void main(String[] args)
     {
     // setting the infinite while loop by
     // passing true to the condition
    while(true)
    {
        System.out.println("infinitive while loop");
    }
    }
}
```

Output:

```
infinitive while loop
infinitive while loop
```

```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
.
.
.
.
.
```

16.4 do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop. It is also known as the exit-controlled loop since the condition is not checked in advance. The Java do-while loop is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop. Java do-while loop is called an exit control loop. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java do-while loop is executed at least once because condition is checked after loop body.

Syntax:

do

{

//code to be executed / loop body

//update statement

}while (condition);

do-while Example Program1:

```
public class dowhileExample1
{
public static void main(String[] args)
      {
      do
      {
            System.out.println("Iam printed once");
      }
    while(1>2);

      }
}
```

Output:

```
Iam printed once
```

do-while Example Program2:

```
public class dowhileExample2
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			System.out.println(i);
			i++;
		}while(i<=10);
	}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

16.4.1 Nested do-while loop

When a loop inside a loop then it is called as nested loop.

Syntax:

```
do
{
    //statements
        do
        {
            //statements
            //update
        }while(condition2);
    //update
}while(condition1);
```

Nested do-while Example:

```
public class NesteddowhileExample
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			int j=1;
			do
			{
			System.out.print(i+" "+j);
			j++;
			System.out.println();
			}
			while(j<=3);
			i++;
		}while(i<=3);
	}
}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

16.4.2 Infinite do-while loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

Syntax:

```
do
{
    //code to be executed
}while(true);
```

Infinite do-while Example:

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

```
	do
	{
		System.out.println("infinitive do while loop");
	}while(true);
	}
}
```

Output:

```
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
.
.
.
.
.
.
.
```