LOOPING CONSTRUCTS IN JAVA - FOR AND FOR-EACH

Looping constructs in Java are:

- For statement
- For-each statement
- While statement
- Do while statement

We will see for and for-each here. We will see while, do while, break and continue statements in next note.

The for statement

Syntax of for loop is:

```
for(initialization-code; terminal-condition; increment-decrement)
 //for loop body
```

If there are more than one statement in the body, it needs to be enclosed in braces ({}). If there is only one statement, braces are optional, but is preferred for readability. We can also have an empty statement (;) as the body.

The initialization-code part is executed only once in the beginning, terminal-condition part id checked every time before executing the body and will execute only if it return true, and increment-decrement part is executed every time after executing the body. The increment-decrement part can increment, decrement or otherwise modify the variable as needed.

```
for(int i=0; i< 5;i ++)
{
    System.out.print(i);
}
```

First int i=0 will be executed. Then i<5 is evaluated and since it return true, the body is executed. Once the body is executed, i++ is executed. Again i<5 is evaluated, body is executed and i++ is executed. Once i becomes 5, the execution will come out of the loop. Above statement will print:

01234

You can use comma operator to separate statements within initialization-code part and increment-decrement, but not in the terminal-condition.

```
for(int i=0, j=5; i<5;i++,j--)
 {
   System.out.print(i+j);
 }
```

If you use comma operator in the condition part, you will get compilation error. You need to use a valid logical expression, a simple one like above or a complex one like (i<5) && (j>3).

The index variable used by a for statement can have different scope depending on how it is declared. If it is declared within the initialization-code part of the for loop (as in the above example), its scope and lifetime will be limited to the for loop block and any inner blocks. Instead if we have declared it outside, its scope and life time will be that of the outer block.

```
int i=0, j=5;
for(i=0, j=5; i<5;i++,j--)
{
  System.out.print(i+j);
}
System.out.print(i+j);
```

For a local variable, scope and lifetime is limited to that block and any of its children. For more details on variables and scopes, refer to javajee.com/data-types-and-variables-in-java.

The initial operation, terminal condition, or end loop operation are not required, but optional in a for loop. We can also have an empty statement (;) as the body. Below code is valid, but creates an infinite loop:

```
for(;;)
;
```

The for-each statement

The for-each statement aka the enhanced for loop, provides an easier way to iterate through an array or a class that has implemented the java.util.Iterable interface. This interface consists of a single method, iterator that returns an Iterator. The Iterable interface is the super interface of the java.util.Collection interface, and hence the for-each statement is usually used with arrays and classes that implement the Collection interface like List, Set and Map implementations.

Syntax is:

```
for (<dataType variable>:<collection/array>)

//for each loop body
```

Note that even though this is called a for-each loop, there is no each keyword.

Example – for each with array

```
int[] arr={1,2,3,4,5};

for(int val: arr)

  System.out.print(val); for(MyClass c : myClassList) {}
```

Example – for each with Collection (List)

```
ArrayList<String> list = new ArrayList<>();

list.add("Heartin");

list.add("Sneha");

list.add("June");

for (String str: list)

{

  System.out.println(str);

}
```

Limitations of for-each compared to simple for loop

1. The index variable is not available and hence we cannot modify the value at a particular position for an array or list.

2. If we want to use one loop to access two different arrays or two different collections, the for-each loop cannot be used.
3. If the array/collection is null, you will get a null pointer exception.
4. You can't modify (add or remove elements) a collection you are iterating over. If you try to modify like below, you will get a java.util.ConcurrentModificationException.

   Below code will throw java.util.ConcurrentModificationException:

```
for (String str: list)
{
  if(str.equals("abc"))
    list.remove("abc");
}
```

   Modifying an array element like below will not throw exception.

```
int[] arr={1,2,3,4,5};
for(int val: arr)
{
  System.out.print(val);
  arr[2]=7;
}
```

   However this doesn't make much sense as we need to use the array Index here which is not available with a for-each loop.

   Source : http://javajee.com/looping-constructs-in-java-for-and-for-each