Set 10: *for* Loops

Skill 10.1: Explain how *for* loops work
Skill 10.2: Explore special features of *for* loops
Skill 10.3: Write nested *for* loops

Skill 10.1: Explain how *for* loops work

Skill 10.1 Concepts

One of the most important control structures in Java is the *for-loop*. A *for-loop* is a block of code that is repeated with certain rules about how to start, increment, and end the process.

Suppose for example we want to sum all the integers from 3 to 79. The rules for starting, incrementing, and stopping are as follows,

- start = 3
- incrementing = +1
- stop = 79

This information can be translated to the *for-loop* shown below

```
for(int n = 3; n <= 79; n++) {
}
```

Now, to sum all the numbers from 3 to 79, we can let the for-loop do the work,

Code

```
int sum = 0;
for (int n = 3; n <= 79; n++) {
     sum += n;
}
System.out.println(sum);
```

Output

```
3157
```

The above example illustrates the following components of a *for-loop*

1. **Initializing expression.** The for loop above was initialized by setting j = 3. If we had wanted to start summing at 19, this part of the for loop would have read j = 19.
2. **Control expression.** The control expression, j <=79 indicates how long to continue looping. This is a boolean expression. As long as the expression is true, the loop will continue.

© Pluska

Warning: There is something really bad that can happen here. You must write your code so as to ensure that this control statement will eventually become false, thus causing the loop to terminate. Otherwise you will have an endless loop which will crash your program.

3. **Step expression.** The step expression, j++, tells us how our variable should change each time through the loop. In this case we are incrementing j each time. However, other possibilities could include,
    - j--
    - j = j + 4
    - j = j * 3
    - etc.

    Note: the step expression occurs at the bottom of the loop. Consider the pseudocode below,

Code

```
int sum = 0;
for (int n = 3; n <= 79; …) {
     sum += n;
     n++;//Just think of the n++ as the last line of code inside the braces
}
System.out.println(sum);
```

Output

```
3157
```

Skill 10.1 Exercises 1 and 2

Skill 10.2: Explore special features of *for* loops

Skill 10.2 Concepts

The break command: If the keyword *break* is executed inside a *for-loop*, the loop is immediately exited (regardless of the control statement). Execution continues with the statement immediately following the closing brace of the for-loop.

Declaring the start loop variable: The start loop variable can be declared in the parenthesis of the *for-loop* or outside the loop. Both situations are illustrated below,

Code – start variable declared in loop	Code – start variable declared outside of loop

`int sum = 0;`
`for (int n = 3; n <= 79; n++) {`
`      sum += n;`
`}`
`System.out.println(sum);`
	`int sum = 0;`
	`// The start variable is declared outside`
	`the loop`
	`int n = 0;`
	`for (n = 3; n <= 79; n++) {`
	`      sum += n;`
	`}`
	`System.out.println(sum);`

Output	**Output**
3157	3157

If *n* is declared in the parenthesis of the loop, its scope is limited to the interior of the loop and is not recognized outside the loop as illustrated below.

Code

`int sum = 0;`
`// The start variable is declared in the parenthesis`
`for (int n = 3; n <= 79; n++) {`
`      sum += n;`
`}`
`System.out.println(sum);`
`System.out.println(n);`

Output
Will not compile because n is declared inside the for loop and cannot be accessed

No curly brackets. If there is only one line of code or just one basic structure (an if-structure or another loop) inside a loop, then the curly brackets are unnecessary. The below example is correct, however it is still highly recommended to include braces to avoid confusion.

Code

```
int sum = 0;
for (int n = 3; n <= 79; n++)
      sum += n;
System.out.println(sum);
```

Output

```
3157
```

Skill 10.3: Write nested *for* loops

Skill 10.3 Concepts

Nested loops is the term used when one loop is placed inside another as in the following example,

Code

```
for (int outer = 0; outer < 5; outer++) {
     System.out.println(outer);// outer is printed 5 times
     for (int inner = 0; inner < 8; inner++) {
          System.out.print(inner);// inner is printed 40 times
      } // end inner loop
System.out.println();
} // end outerloop
```

Output

```
0
01234567
1
01234567
2
01234567
3
01234567
4
01234567
```

In the example above, the inner loop executes 8 times for each of the five iterations of the outer loop. Therefore, the code inside the inner loop will execute 40 times.

Skill 10.3 Exercise 1

© Pluska