CHAPTER 8

Loops

There are four looping structures in C#. These are used to execute a code block multiple times. Just as with the conditional `if` statement, the curly brackets for the loops can be left out if there is only one statement in the code block.

While Loop

The `while` loop runs through the code block only if its condition is true and will continue looping for as long as the condition remains true. Note that the condition is only checked at the beginning of each iteration (loop).

```
int i = 0;
while (i < 10) {
  System.Console.Write(i++); // 0-9
}
```

Do-While Loop

The `do-while` loop works in the same way as the `while` loop, except that it checks the condition after the code block and will therefore always run through the code block at least once. Bear in mind that this loop ends with a semicolon.

M. Olsson, *C# 8 Quick Syntax Reference*, https://doi.org/10.1007/978-1-4842-5577-3_8

```
int j = 0;
do {
  System.Console.Write(j++); // 0-9
} while (j < 10);
```

For Loop

The `for` loop is used to go through a code block a specified number of times. It uses three parameters. The first parameter initializes a counter and is always executed once before the loop. The second parameter holds the condition for the loop and is checked before each iteration. The third parameter contains the increment of the counter and is executed at the end of each iteration.

```
for (int k = 0; k < 10; k++) {
  System.Console.Write(k); // 0-9
}
```

Several variations of the `for` loop are possible. For instance, the first and third parameters can be split into several statements using the comma operator.

```
for (int k = 0, m = 5; k < 10; k++, m--) {
  System.Console.Write(k+m); // 5 (10x)
}
```

There is also the option of leaving out one or more of the parameters. For example, the third parameter may be moved into the body of the loop.

```
for (int k = 0; k < 10;) {
  System.Console.Write(k++); // 0-9
}
```

Foreach Loop

The foreach loop provides an easy way to iterate through arrays. At each iteration, the next element in the array is assigned to the specified variable (the iterator) and the loop continues to execute until it has gone through the entire array.

```
int[] a = { 1, 2, 3 };
foreach (int n in a) {
  System.Console.Write(n); // "123"
}
```

Note that the iterator variable is read-only and can therefore not be used to change elements in the array.

Break and Continue

There are two special keywords that can be used inside loops – break and continue. The break keyword ends the loop structure, and continue skips the rest of the current iteration and continues at the start of the next iteration.

```
for (int i = 0; i < 10; i++) {
  if (i == 5) break; // end loop
  if (i == 3) continue; // start next iteration
  System.Console.Write(i); // "0124"
}
```