# Subsection 1: Learn and use while loops

A loop is another type of execution flow control that we can use to repeat a block of code based on a condition or given a number of times. The *while* and *do/while* loops execute a code block while a condition remains true and *for* loops repeat a code block a given number of times.

Let’s start by looking as *while* loops.

## While loops

*While* loops will repeat a code block while a given condition is true. At some point during the execution of the while loop it is expected that the condition will change to false and the while loop will stop executing and pass control back to the normal execution of the program.

## Infinite While Loops

An important issue you must be aware of is inadvertently creating an infinite loop. As the name suggests, an infinite loop will never exit from the code block because the *while* expression always evaluate to true.

Consider the following example

1. short s = 10;  
2. byte b = 5;  
3. while(s > b){  
4. System.*out*.print("forever ... ");  
5. }

In this example the value of *s* will always be greater that *b* and the *while* expression will always be true.

Infinite loops are not inherently bad, they can be very useful as in the code below, but they become troublesome when they are an unintended consequence. Ensure that you take care to analyse any *while* loop’s logic to ensure you don’t stumble into infinite loop issues.

while(true){  
 // Code that monitors for input  
}

## Unreachable code

In the previous subsection we looked at how unreachable code is allowed for *if-then* statements. The same exception does not apply to *while* loops. Consider the example above in which the *while* expression is always true and will never quit executing the code block. The consequence of this is that the code after the *while* loop will never be executed. The compiler is capable of identifying this issue and will fail to compile the code and report an **Unreachable Statement** error.

A potential solution is to add logic that can break out of the loop based on a condition. Execution of a loop can be stopped by using the *break* keyword. We look at this and other control keywords at the end of this subsection, but for now consider the following code example:

while(true){  
 if(someCondition){  
 break;  
 }  
}

The compiler can determine that there is an opportunity that the *if* condition will be true and the *break* statement is executed and so concludes that the code following the *while* loop is potentially reachable and compilation will succeed.

Let’s look at one further example where the unreachable statements in the while code block itself. Consider the following fragment.

1. final short s = 10;  
2. final byte b = 5;  
3. while(s < b){  
4. // Unreachable Statement  
5. }

In this code snippet the *while* condition is always false and the code on line 4 will never be executed because the variables *s* and *b* can never change because they are declared as final. The compiler can determined that the *while* expression will always be false so compilation will fail. Removing the final keywords makes the code compliable even though we can clearly see that the value of *s* and *b* don’t change and the while expression will always be false. The compiler will err on the side of caution and only declared statements as unreachable if it is absolutely certain.

## Exercises

Which of the following while statements will cause a compile error?

1. while(false){}

2. while(-1){}

3. while(isAlive()){}

4. while( 0 < 0){}

5. boolean x = false; while( x = true ){}

Answers: 1, 2, 4

Which of the following code snippet contains unreachable code?

1.

int s = 10;  
 int b = 5;  
 while(s < b){  
 // Code block  
 }

2.  
while(false){  
 // Code block  
 }  
System.out.println(“Hello World”);

3.

final short s = 10;  
final byte b = 5;  
while(s > b){  
 // Code block  
}

4.

while(true){  
 // Code block  
 }  
System.out.println(“Hello World”);

5.

while(true){  
 // Code block  
 }

Answer: 1, 2, 4

# Subsection 2: Learn and use do/while loops

Exercises:

Which of the following do/while loops will compile?

1.  
do {;} while (true);  
  
2.  
do  
 System.out.println("Hello World");  
while (false);  
  
3.   
do ; while (false);  
  
4.  
while (isAlive()) { System.out.println("Johnny 5 is Alive");} do;  
  
5. None of the above

Answers: 1, 2, 3

# Subsection 3: Learn and use for loops including the enhanced for loop

The last type of loop construct we will look at is the *for* loop. *For* loops repeat a block of code a given a number of times and there are two types: the basic *for* loop and the enhanced *for* loop. The enhanced *for* loop is often referred to as the *for-each* loop.

## For Loop Variation

The simple example that you have seen already is the most common use of for statement however there variations of this that you should be familiar with. Let’s go through each one so you are prepared for them if they appear on the exam.

### Count down

The increment can also be a decrement. Consider the following example that counts down from 10.

for (int i = 10; i > 0; i--) {  
 System.*out*.println(i);  
}

The increment can be either flavour of post and pre increment or decrement and the number of iterations will not be affected.

### Infinite Loop

You must ensure that if you create an infinite loop it was done so intentionally. Here are a few examples of what infinite loops look like.

for (int x = 0; ; x++) {  
 System.*out*.println(x);  
}

for (int y = 0; true; y++) {  
 System.*out*.println(y);  
}

for (; ; ) {  
 System.*out*.println("Hello World");  
}

Remember that the code after an infinite loop will be unreachable and the class will not compile.

### Multiple Terms

Multiple variables and incrementors can be specified at the same time and it is not a requirement to use all initialised variables. The following code fragment demonstrated the use of multiple terms.

for (int i = 10, j = 0, k = 0; i > 0 && j < 10; --i, ++j) {  
 System.*out*.println(i + " " + j);  
}

You can see how the variables *i*, *j* and *k* are declared and set to zero, but only variables *i* and *j* are used. It is perfectly legal to do this although of little value. The boolean expression makes use of both *i* and *j* variables and both of these variable are updated in the increment part.

It is not possible to declare additional variables in the initialisation part as shown in the following example:

for (int i = 0, int j = 0; i < 10; i++) {  
 System.*out*.println(i);  
}

This code will not compile.

### Spaces and indentations

As you have probably already guessed the *for* loop can be constructed without braces. Just as with the *if* and *while* statements you must be careful with the placement of the code block line. Here are a few examples of *for* statements without braces.

for (int x = 0; x < 5; x++)  
 System.*out*.println(x);  
  
for (int x = 0; x < 5; x++)  
  
 System.*out*.println(x);  
  
for (int x = 0; x < 5; x++)  
 // comments here  
 System.*out*.println(x);  
  
for (int x = 0; x < 5; x++)  
 for (int y = 0; y < 5; y++)  
 for (int z = 0; z < 5; z++)  
 System.*out*.println(x);

Review these variation and take care when asked questions in the exam that include code without braces.

### Unreachable Statement

There are two ways that cause the **Unreachable Statement** error: the *for* loop never executes because the *booleanExpression* is certain to be false and an infinite *for* loop. Consider the following example:

1. for (int y = 0; false; y++) {  
2. System.*out*.println(y);  
3. }

The code on line 2 is never executes and compilation will fail.

## Nested Loops

Nested loops allow you to build more complex looping structures. A typical example is to loop over a two-dimensional array and print its contents to the console. The following code snippet does just that.

1. String[][] simpleFlag = new String[][]{  
2. {"\*", "\*", "\*", "\*"},   
3. {"\*", "0", "0", "\*"},   
4. {"\*", "0", "0", "\*"},   
5. {"\*", "\*", "\*", "\*"}};  
6. for (int x = 0; x < simpleFlag.length; x++) {  
7. for (int y = 0; y < simpleFlag.length; y++) {  
8. System.*out*.print(simpleFlag[x][y]);  
9. }  
10. System.*out*.println("");  
11. }

In this code snippet the outer loop on line 6 loop four times, once for each element in the one-dimension of the 2-D array, and the inner loop iterates over the array four times. The code on line 8 uses the values of x and y to reference an element in the array and prints it to the console.

Exercises:

1. Which of the following code snippets are infinite loops?

1

for (int x = 0; ; x++) {  
 System.*out*.println(x);  
}

2

for (int x = 0; x > 10; x++) {  
 System.*out*.println(x);  
}

3

for (int y = 0; true; y++) {  
 System.*out*.println(y);  
}

4

for (int x = 0; false ; x++) {  
 System.*out*.println(x);  
}

5

for (; ; ) {  
 System.out.println("Hello World");  
}

Answer: 1, 3, 5

2. Which of the following code snippets are invalid?

1

for (int i = 10, j = 0, k = 0; i > 0 && j < 10; --i, ++j) {  
 System.*out*.println(i + " " + j);  
}

2

for (int y = 0; false; y++) {  
 System.out.println(y);  
}

3

for (int i = 0, int j = 0; i < 10; i++) {  
 System.out.println(i);  
}

4

for (; ; ) {  
 System.out.println("Hello World");  
}

5

for (int x = 0; x < 5; x++)  
 for (int y = 0; y < 5; y++)  
 for (int z = 0; z < 5; z++)  
 System.out.println(x);

Answers: 2, 3