Chapter 8. Iterations/Loops

The facility in a programming language to iterate easily through a set of items (for example the individual elements of an array) and perform common actions on each element is a key enabler of compact code. This chapter contains exercises on different loop mechanisms within Java, such as the `while`, `do/while`, `for` and enhanced `for` loops. Exercises which require the use of "labels" as well as the loop jump keywords `continue` and `break` are also presented, as well as exercises on nested loops and infinite loops.

`while` loop

1. Use a `while` loop to print out the numbers 0 to 9 in order, on sequential lines.

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

    int myCounter = 0; // We are starting at 0
    while (myCounter < 10) {
      System.out.println(myCounter);
      myCounter = myCounter + 1; //we increment the counter here
    }
  }
}
```

2. Use a `while` loop to add the numbers 1 to 15,000. Print out the result.

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

    long myCounter = 1; // We are starting at 1
    long total = 0L;
    while (myCounter <= 15000) {
       total+= myCounter;
       myCounter++; //we increment the counter here
    }
    System.out.println("Total = " + total);
  }
}
```

3. We have an integer array `intArray01` which has 5 elements with the following content in the noted positions in the array:

```
[0] - 18
[1] - 42
[2] - 26
[3] - 44
[4] - 55
```

Using a `while` loop, print out the values contained in this array, in the following format:

`intArray01[<index>]=<value>`

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };
    int loopCounter = 0; // We are starting at element 0
    int arrayLength = intArray01.length; // upper limit

    while (loopCounter < arrayLength) {
      System.out.printf("intArray01[%d] = %d\n",
                        loopCounter,intArray01[loopCounter]);
      loopCounter++;
    }
  }
}
```

4. Print out the contents of the array in the preceding exercise *backwards*, in the same output format, using a `while` loop.

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };
    int loopCounter = intArray01.length - 1; // We are starting at the last element

    while (loopCounter >= 0) {
      System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);
      loopCounter = loopCounter - 1;
```

```
    }
  }
}
```

for loop

5. Explain the components and the functioning of a `for` loop, using the sample code below:

```
int loopCounter;
for(loopCounter=3; loopCounter < 10; loopCounter++)
{
   // do action
}
```

The control section of a `for` loop consists of the following elements:

1. The keyword `for`, which brackets the other control elements of the loop.
2. The other control elements which are semi-colon separated within the brackets.

In the brackets we have, in order, the following:

1. A statement indicating the name of the control variable of the `for` loop, with its initial value specified. In our example above, we state that our loop counting variable named `loopCounter` has an initial value of 3. Note that the initialization of the control variable does not have to be done inside the control section of the `for` loop; in such a case, the initialization section of the `for` loop will be left blank.
2. A conditional statement that states the condition which while true permits the `for` loop to continue running. In our example we are saying "keep performing this loop *while* the value of the variable `loopCounter` is less than 10".
3. Finally, an increment statement, which indicates the incremental value to add to the loop counter after each iteration of the actions that are supposed to be performed in the statements governed by the `for` loop are performed. In our example here, we indicate that the loop counter `loopCounter` should be incremented by 1.

The body of the `for` loop is shown in this example as:

```
{
  // do action
}
```

These braces that immediately follow the `for` loop surround the code that is to be run on each iteration of the `for` loop.

A `for` loop is, in a sense, a repackaged `while` loop in Java. This can be said because its conditional statement is saying *"while this condition is true*, continue looping*"*.

Notes

1. If you only have one line that you want to be performed for each iteration of a `for` loop, there is no need to put braces around the single line to be run. For example we would write:

```
int loopCounter;
for(loopCounter=3; loopCounter < 10; loopCounter++)
  single line of action.
```

2. The loop counter can be declared directly in the control section of the `for` loop. For example we can write:

```
for(int loopCounter = 0; loopCounter < 20; loopCounter++)
{
  //do action
}
```

   The scope of a variable declared in the loop control section is limited to the scope of the `for` loop itself.

3. Another point to note is that it is acceptable for any (or all) of the loop control components to be missing! Only the two semicolons in the control statement are mandatory.

6. Predict the output of the following `for` loop.

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

    for (int loopCounter = 0; loopCounter < 5; loopCounter = loopCounter + 1) {
      System.out.println("Hello!");
    }
  }
}
```

Output:

```
Hello!
Hello!
Hello!
Hello!
Hello!
```

7. Write a program which using a `for` loop will output the values 2 to 10.

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

    for (int i = 2; i < 11; i++) {
      System.out.println(i);
    }
  }
}
```

8. What is the difference between the following two `for` loop code samples?

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

vs.

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

The difference between the loops is that in the second code snippet, the loop control variable is defined within the control structure of the `for` loop. Otherwise the loops are equivalent in function. Also note that in the second code sample, the lifetime of the loop control variable expires at the end of the `for` loop since it was defined within the control structure of the loop.

9. What is the difference between the following two `for` loop code samples?

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

vs.

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

The difference between the loops is that in the second code snippet, the loop control variable is initialized outside/before the control structure of the `for` loop. Otherwise the loops are completely equivalent in function.

10. What is the difference between the following two `for` loop code samples?

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

vs.

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

The difference between the loops is that in the second code snippet the loop increment variable is incremented within the code block, rather than in the control section. Otherwise the loops are completely equivalent in function.

11. Predict and explain the output of the following code snippet:

```
for (int i=0; i < 3; i++)
  System.out.println(i);
  System.out.println("hello");
```

The output is:

```
0
1
2
hello
```

Explanation

<table>
<tr><td></td><td></td><td>If there are no braces around lines of code following a <code>for</code> loop, only the first line of code immediately following the <code>for</code> loop is considered to be the body of the loop.</td></tr>
<tr><td>12.</td><td>Predict and explain the output of the following code snippet:
<code>for(int i=0;i<10;i++);</code></td><td>Nothing is output, however the code is valid. Note the semi-colon at the end of the statement.</td></tr>
<tr><td>13.</td><td colspan="2">Predict the output of this program.
<pre>public class PracticeYourJava {
  public static void main(String[] args) {

    for(int j=0; j < 10; j++, System.out.println("Hello"));
  }
}</pre></td></tr>
<tr><td></td><td colspan="2">It prints out the word “Hello” 10 times.
Normally we may have written the <code>for</code> loop as follows:
<pre> for(int j=0; j < 10; j++)
   System.out.println("Hello");</pre>However, the compact form shown in the exercise, showing that we can perform other actions in the incrementing part of the control section of the loop is also valid (however it might be deemed to be less readable though).</td></tr>
<tr><td>14.</td><td colspan="2">We have the following array definition: <code>int[] intArray01 = new int[]{18,42,26,44,55};</code>
Using a <code>for</code> loop, print out the values contained in this array, in the following format:
<pre> intArray01[<index>] = <value></pre></td></tr>
<tr><td></td><td colspan="2"><pre>public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };
    int loopCounter;
    for (loopCounter = 0; loopCounter < intArray01.length; loopCounter = loopCounter + 1) {
      System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);
    }
  }
}</pre>Notes/Explanation of the <code>for</code> loop in Java
Really, in Java, the <code>for</code> loop can be thought of as a <code>while</code> loop of the following structure (using the answer above as an example):
<pre>loopCounter=0; // initial condition
while(loopCounter < intArray01.length) // the termination condition
{
  System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);
  loopCounter=loopCounter+1; // the increment of the counter as specified
}</pre>If you are having trouble with formulating a <code>for</code> loop, you can first formulate its <code>while</code> loop counterpart.</td></tr>
<tr><td>15.</td><td colspan="2">Print the same array of the preceding exercise out backwards, in the same output format, using a <code>for</code> loop.
Hint: decrement the loop control variable</td></tr>
<tr><td></td><td colspan="2"><pre>public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };
    int loopCounter;
    for (loopCounter = intArray01.length - 1; loopCounter >= 0; loopCounter-=1)
      System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);
  }
}</pre></td></tr>
<tr><td>16.</td><td colspan="2">Using a <code>for</code> loop, fill an <code>int</code> array of 15 elements with random integers between the values of 0 and 50,000. After filling all the elements, print them out using a <code>while</code> loop.</td></tr>
</table>

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

    int numElements =  15;
    int[] array01 = new int[numElements];

    for(int i=0;i<array01.length;i++)
      array01[i] = (int)Math.ceil(Math.random() * 50000);

    int counter=0;
    while(counter < numElements) {
      System.out.printf("intArray01[%d] = %d\n", counter, array01[counter]);
      counter++;
    }
  }
}
```

Enhanced for loop

17. We have an integer array `intArray01` which has 5 elements with the following content in the noted positions:

    [0] - 18
    [1] - 42
    [2] - 26
    [3] - 44
    [4] - 55

    Using an enhanced `for` loop, print out the contents of the array. *(don't print out the index, just print the values).*

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

    for(int item : intArray01) {
       System.out.println(item);
    }
  }
}
```

Note: The enhanced `for` loop is more useful when you have to perform an action on each and every element in the array under consideration.

Unlike the `for` and `while` loops, the enhanced `for` loop does not present an accessible index.

18. Using the same array as in the exercise above, use an enhanced `for` loop to print out the values contained in this array, in the following manner:

    `intArray01[<index>] = <value>`

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

    int index = 0;
    for(int item : intArray01) {
      System.out.printf("intArray01[%d] = %d\n", index, item);
      index = index + 1;
    }
  }
}
```

Notes/Explanation: The enhanced `for` loop does not present an index to the user; rather, in each iteration it gets the next item in the sequence/array under assessment into the variable that is present in its declaration statement (in this exercise for example, it sequentially puts the content that is in the array `intArray01` element by element into the variable `item`). Therefore, we have to create our own index if we want to print an index out.

19. With respect to the array in the exercise above, use an enhanced `for` loop to print out every *2nd value* (i.e. elements 0, 2 and 4 only) contained in this array, in the following format:

    `intArray01[<index>] = <value>`

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

    int index = 0;
```

```
    for(int item : intArray01) {
      if (index % 2 == 0) {
        System.out.printf("intArray01[%d] = %d\n", index, item);
      }
      index = index + 1;
    }
  }
}
```

Explanation: Again, due to the nature of the enhanced `for` loop statement, it does not lend itself as smoothly as using any of the other loop constructs for this same scenario.

20. Repeat the preceding exercise, this time using a `for` loop.

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

    for (int j = 0; j < intArray01.length; j+=2)
      System.out.printf("intArray01[%d] = %d\n", j, intArray01[j]);
  }
}
```

Note: Note how the loop index was incremented by 2 to achieve the specified output effect of printing every 2nd entry.

21. Repeat the preceding exercise using a `while` loop.

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

    int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

    int counter = 0; // We will be starting at the 0th index.
    int arrayLength = intArray01.length;

    while (counter < intArray01.length) {
      System.out.printf("intArray01[%d] = %d\n", counter, intArray01[j]);
      counter = counter + 2;
    }
  }
}
```

do/while loop

22. Under what circumstances is a `do/while` loop a good candidate to use?

When you have something that must run at least once before loop conditions are checked. Think of this loop as: *you have to do first and then check conditions for potential subsequent actions.*

23. Write a program which uses a `do/while` loop to print out the numbers 1 to 10.

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

    int index = 1;
    do{
      System.out.println(index);
      index++;
    }while (index <= 10);
  }
}
```

24. Using a `do/while` loop, calculate and print out the factorial of 10.

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

    int iNumToCalculateFactorial = 10;
    long result = 1;
    int index = 1;

    do {
      result = result * index;
```

```
      index++;
    }while (index <= iNumToCalculateFactorial);
    System.out.printf("%d! = %d\n", iNumToCalculateFactorial, result);
  }
}
```

25. Using an *infinite* `while` loop, print the following string out indefinitely:

"Hello!"

```
public class PracticeYourJava {
  public static void main(String[] args) {
    while (true) {
      System.out.println("Hello!");
    }
  }
}
```

Note: Recall that a `while` loop runs while whatever it evaluates in its control statement equals the boolean value `true`. Therefore, we simply need to put the boolean value `true` directly into the loop.

26. Repeat the preceding exercise using an infinite `for` loop.

```
public class PracticeYourJava {
  public static void main(String[] args) {
    for ( ; ; ) {
      System.out.println("Hello!");
    }
  }
}
```

Nested loops

27. Predict the output of the following code:

```
public class PracticeYourJava {
  public static void main(String[] args) {
    int j, k;
    for (j = 1; j < 5; j++) {
      for (k = 1; k < 5; k++) {
        System.out.printf("(%d,%d) ", j, k);
      }
      System.out.println();
    }
  }
}
```

Output:

```
(1,1) (1,2) (1,3) (1,4)
(2,1) (2,2) (2,3) (2,4)
(3,1) (3,2) (3,3) (3,4)
(4,1) (4,2) (4,3) (4,4)
```

Note: The objective of this exercise was to show the application of **nested loops** in outputting matrices.

28. Write a program which will output the following pattern:

```
(A,1) (A,2) (A,3) (A,4)
(B,1) (B,2) (B,3) (B,4)
(C,1) (C,2) (C,3) (C,4)
(D,1) (D,2) (D,3) (D,4)
(E,1) (E,2) (E,3) (E,4)
```

```
public class PracticeYourJava {
  public static void main(String[] args){
    int j, k;
    String[] letters = new String[] {"A","B","C","D", "E"};
    // OR we can use a char array instead: char[] letters   = new char[] {'A','B','C','D','E'};

    for (j = 0; j < letters.length; j++){
       for (k = 1; k < 5; k++){
         System.out.printf("(%s,%d) ", letters[j], k);
```

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

-OR-

```
public class PracticeYourJava {
   public static void main(String[] args){
     int j, k;
     int start = (int)'A';
     for (j = start; j < (start + 5); j++){
       for (k = 1; k < 5; k++){
         System.out.printf("(%s,%d) ", (char)j, k);
       }
       System.out.println();
     }
   }
 }
```

The keywords `continue` and `break` and the use of labels

29. Write a `for` loop that can print out the values 1 to 1000 in sequence; however, using a `break` statement halt the running of the loop when the value of the loop counter is 10.

```
public class PracticeYourJava {
  public static void main(String[] args) {
    for (int i = 0; i <= 50; i++) {
      if (i == 10) break;
      System.out.println(i);
    }
  }
}
```

30. What does the `continue` operator do?

The `continue` operator is an operator which when present in a loop causes the code to immediately go back to the start of the loop.

31. Write a `for` loop which loops from 1 to 50, printing out each value; however, using a `continue` statement, skip every value that is divisible by 5.

```
public class PracticeYourJava {
  public static void main(String[] args) {
    for (int i = 0; i <= 50; i++) {
      if (i % 5 == 0) continue;
      System.out.println(i);
    }
  }
}
```

32. What is a label in Java and how is it used?

A label is a name applied to a block of code.
The label is placed at the beginning of the code block and is always terminated with a colon.
The naming convention for a label is simply any valid identifier (i.e. any name that is valid for a variable).
Note that you can put labels at any point in your code.

33. Describe the functioning of the labeled `break` statement.

The labeled `break` statement is a statement that allows you to transfer control from a given point in a block of code to the *end* of the *labeled block* that the labeled `break` appears within. The usage of the labeled break statement is as follows:

```
break <label>;
```

The labeled break statement has the benefit of allowing you to move around in code in a non-sequential manner if so desired. The labeled break statement even gives you the ability to jump out of loops.

Also note that the labeled break statement is not exclusively used in loops, but can be used in any other code.

The short example below can be run in order to observe the functioning of the labeled break statement.

```
import java.util.*;

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

```
  POINT_A:{
    System.out.println("1");
    if (1==1) break POINT_A; //Of course 1 == 1 so the break will be triggered
    System.out.println("2");
    } // This is where the labeled break jumps to; the end of the labeled block
   System.out.println("3");
  }
}
```

As can be seen when this code is run, the output is as follows:

```
1
3
```

The labeled break statement jumped to the end of the code block labeled `POINT_A`, thus skipping the line which was supposed to print out the string "2".
Note that loop blocks (for example `for` blocks) are also deemed to be blocks that can be labeled.

34. I want to output the following pattern:

```
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4
```

Write a program to print the pattern out using a nested `for` loop and a `break` that breaks out of the loop to the end of the labeled block that it is in. Outside/after the end of the labeled block, print out *"\nJust saw a C5"*.

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

    String[] sArray01 = { "A", "B", "C" };
POINT_A: for (int i = 0; i < sArray01.length; i++) {
           for (int j = 1; j <= 10; j++) {
             String sf01 = String.format("%s%d", sArray01[i], j);
             if (sf01.equals("C5") == true) break POINT_A;
             System.out.print(sf01); System.out.print(" ");
           }
           System.out.println();
         }
        System.out.println("\nJust saw a C5");
    }
 }
```

35. Describe the purpose of the `break` statement (not the labeled break) within the context of a loop.

A `break` statement within the context of a loop causes the loop to end processing immediately and for control to go to the statement immediately after the loop.

36. I want to output the following pattern:

```
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10
```

Write a nested `for` loop to implement this.
Hint: Use a `break` *statement within the inner loop when* `C5` *is seen. No label is necessary for this solution.*

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

    String[] sArray01 = { "A", "B", "C", "D", "E" };
    for (int i = 0; i < sArray01.length; i++) {
      for (int j = 1; j <= 10; j++) {
        String sf01 = String.format("%s%d", sArray01[i], j);
        if (sf01.equals("C5") == true) break;
        System.out.print(sf01); System.out.print(" ");
      }
```

```
      System.out.println(); // that break statement jumps to this line
    }
  }
}
```

37. Write a program which uses a nested for loop to add the following two 2×3 matrixes and prints out the resulting matrix in the same 2×3 format.

```
1   2   3
4   5   6
    +
5   5   8
3   1   3
```

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

    int[][] matrix1 = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };
    int[][] matrix2 = new int[][] { { 5, 5, 8 }, { 3, 1, 3 } };
    int[][] resultMatrix = new int[2][3];

    for (int a = 0; a < 2; a++) {
      for (int b = 0; b < 3; b++) {
        resultMatrix[a][b] = matrix1[a][b] + matrix2[a][b];
        System.out.printf(" %d", resultMatrix[a][b]);
      }
       System.out.println();
    }
  }
}
```

38. Using a `String` array with the following elements:

```
o
oo
ooo
oooo
ooooo
oooooo
ooooooo
oooooooo
```

Draw the following pattern:

```
o                o
oo              oo
ooo            ooo
oooo          oooo
ooooo        ooooo
oooooo      oooooo
ooooooo    ooooooo
oooooooooooooooo
ooooooo    ooooooo
oooooo      oooooo
ooooo        ooooo
oooo          oooo
ooo            ooo
oo              oo
o                o
```

Hint: Use `String.format` to build the formatting string as well as the output.

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

    String[] patternArray = new String[] { "o", "oo", "ooo", "oooo", "ooooo", "oooooo",
                                           "ooooooo", "oooooooo" };

    int maxLength = 0;
    for (String s : patternArray) {
       if (s.length() > maxLength) maxLength = s.length();
    }

    int fieldWidth = maxLength;

    for (int i = 0; i < patternArray.length; i++) {
       String formatStr ="%-" + fieldWidth + "s" + "" + "%"  + fieldWidth + "s";
       String outputStr = String.format(formatStr, patternArray[i], patternArray[i]);
       System.out.println(outputStr);
    }
    // Now the lower part of the pattern
    for (int i = (patternArray.length - 2); i >= 0; i--) {
      String formatStr ="%-" + fieldWidth + "s" + "" + "%"  + fieldWidth + "s";
      String outputStr = String.format(formatStr, patternArray[i], patternArray[i]);
      System.out.println(outputStr);
    }
  }
}
```

E1	Using the same source pattern as the preceding exercise, write code to output the following pattern: o o oo oo ooo ooo oooo oooo ooooo ooooo oooooo oooooo ooooooo ooooooo oooooooo oooooooo
E2	Using the same source pattern as above, write code to output the following pattern: o oooooooo oo ooooooo ooo oooooo oooo ooooo ooooo oooo oooooo ooo ooooooo oo oooooooo o
E3	I have an array of strings, each string therein of undeterrmined length. Write out the array of strings, with the same width for each, bounding the strings left & right with the '|' character. *Hint: determine the length of the longest string. That will be your field width. Then use* `String.format` *to output each line as appropriate.*