Chapter 5: Iteration

Learning Outcomes:

- ✓ Identify the importance of using iteration in a program
- ✓ Implement FOR and WHILE loops into a program
- ✓ Implement iteration independently to resolve given problems efficiently

Prerequisite Knowledge:

- ✓ Complete Chapter 4
- ✓ Be able to confidently implement the readLine() method (with appropriate TRY/CATCH) to record responses from a program user
- ✓ Be able to confidently implement selection

Keywords

FOR Loop WHILE Loop DO WHILE Loop Increment

5.1 The Theory: Iteration

If a programmer was asked to write a simple program to print the numbers 1 to 5 to the command-line, what would the code be? Would it be as simple as the code below?

```
public class MyProgram{

	public static void main(String[] args){
		System.out.println("1");
		System.out.println("2");
		System.out.println("3");
		System.out.println("4");
		System.out.println("5");
	}
}
```

There is no doubt about it, the code works and, as a 'rule of thumb', the less code the better. However, what if a programmer was asked to print 1 to 100, or even 1 to 10,000? This method suddenly becomes impractical and a mass of code is needed to achieve something rather trivial. The solution to the problem is iteration; iteration allows code to be repeated numerous times using little code, which makes for a good 'rule of thumb'!

The FOR Loop

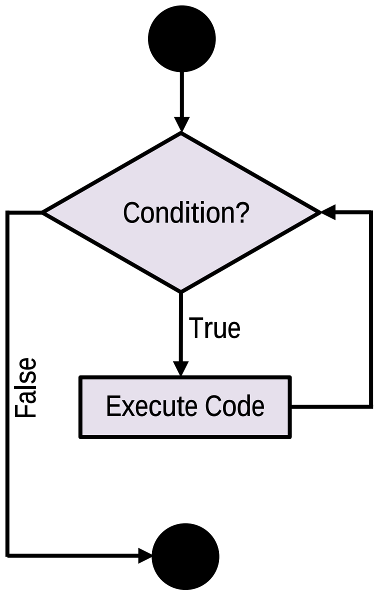
One type of iteration is the 'FOR loop;' these are particularly useful when a programmer needs to execute the same code a fixed number of times. In Java, a FOR loop must have a starting point, a loop condition and an increment value; this will enable the loop to increment from the start value, by the set increment amount, until the loop condition is met.

Start variable and value

Increment value

```
for(start; condition; increment){
	//repeat this code
}
```

Loop condition



5.2 Practical: Using FOR Loops

My FOR Loop Program

Most software languages have some form of FOR loop, although the syntax may differ. A FOR loop has a starting number, which is an integer variable with a set value. The end condition tests the starting number variable to determine whether the condition is still true; once false, the loop ends. The end condition is set using Boolean operators, and may look something like '`x < 5`'.

Programming Tips!

The increment is set to +1 [`i++`] by default, but this can be changed to a greater increment by changing the code, for example, [`i=i+2`] will increment by 2.

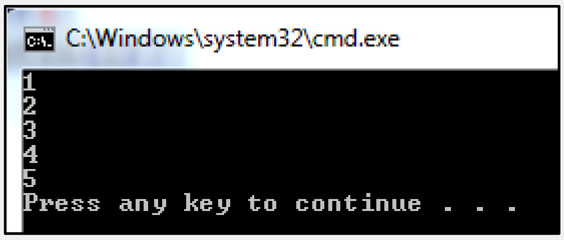
The example below implements a FOR loop to print the numbers 1 to 5 to the command-line; this is a much more efficient method than the previous example. Note that variable `i` has a starting value of 1 and an end condition of [`i < 6`]. The increment for `i` is set to [`i++`] (`i` equals its current value +1), resulting in the loop incrementing a total of 5 times. For each increment of the loop the value of `i` will be printed to the command-line, before it is increased by 1. When `i` reaches the value 6, the condition [`i < 6`] becomes false and the loop ends (the program will continue to execute the remaining code).

Try the code

```
public class MyFirstLoop{

        public static void main(String[] args){
            for(int i=1; i<6; i++){
                System.out.println(i);
            }
        }
}
```

Step by step

1. Type out the above code
2. Save the program as MyFirstLoop.java
3. Compile the program (use the javac compiler)
4. Run the MyFirstLoop.class file
5. Check the results of the program



```
C:\Windows\system32\cmd.exe
1
2
3
4
5
Press any key to continue . . .
```

Activity 5.1

Experiment with the code from the previous example, checking the results of each change within the command-line. Perform the following changes:

- Change the start value to 2
- Change the end condition value to 99
- Change the increment value to 3

Activity 5.2

Odds and Evens Program – write a program that implements two FOR loops. The first loop should print all of the odd numbers between 1 and 100. The second loop should print all of the even numbers between 1 and 100. Pay attention to the starting point value and increment amount; these are the key!

Activity 5.3

Odds and Evens Program Extended – Remove the even FOR loop from the program and amend the odd FOR loop so that the user can choose the odd numbers' start point and end point (as opposed to them being set to 1 to 100). To achieve this, use the [`readLine()`] method, from the previous chapter, to ask the user for the start and end values. Integer variables will need to replace the values in the FOR loop, i.e. `i = myStartPoint`, as opposed to `i = 1`.

The Maths Program

The example below implements a FOR loop to print a series of addition sums to the command-line. The two variables, `myStartPoint` and `myEndPoint`, are used to indicate the starting point and end condition of the FOR loop; their values can be set independently, as opposed to using hard-coded values. The line of code inside of the loop [`println(i + " + " + 5 + " = " + (i+myZ));`] uses concatenation to join several parts together, enabling both the sum question and answer to be printed to the command-line. Note that variable `i` will be substituted for its current value on each cycle of the loop; as the loop increments, the value of `i` will increase by 1. Finally, note that the addition symbol and equals symbol are placed between quotations; this is so they are not used in the operation, but rather to make the output more human-friendly.

Activity 5.4

The Times Table – write a program that displays a specified times table to the command-line. The user should be able to choose a start point, an end point (both between 1 and 50) and a desired multiplier for the times table. The program should then print the specified times table, in full, to the command-line. For example, if the user entered the start point as 3, the end point as 6 and the multiplier as 2, then the output will be:

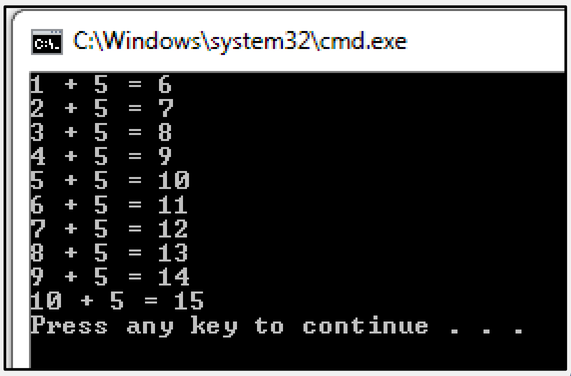
```
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
```

Try the code

```
public class TheAddition{

        public static void main(String[] args){

            int myStartPoint = 1;
            int myEndPoint = 10;
            for(int i= myStartPoint; i< (myEndPoint+1); i++){
                System.out.println(i + " + " + 5 + " = " + (i+5));
            }
        }
}
```

Step by step

1. Type out the above code
2. Save the program as TheAddition.java
3. Compile the program (use the javac compiler)
4. Run the TheAddition.class file
5. Check the results of the program



```
1 + 5 = 6
2 + 5 = 7
3 + 5 = 8
4 + 5 = 9
5 + 5 = 10
6 + 5 = 11
7 + 5 = 12
8 + 5 = 13
9 + 5 = 14
10 + 5 = 15
Press any key to continue . . .
```

Programming Tips!

Note when changing the increment of a loop that [`i=i+2`] is the same as the shorthand version of [`i+=2`]. Crazy stuff!

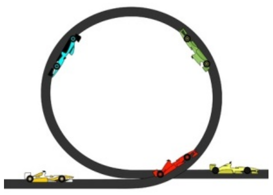
Programming Tips!

Loops can be difficult to understand; the best way to learn how they work is to imagine what the code is doing on each cycle of the loop.

5.3 The Theory: WHILE and DO WHILE Loops

More Loops

Another useful loop is the 'WHILE loop'; again, like the FOR loop, this type of loop is also found in most software languages. This loop works by repeating the contained code so long as the condition is true; the moment the condition becomes false, the loop ends. The logic of a WHILE loop is shown below:



Loop condition

```
while(condition){
      //repeat this code
}
```

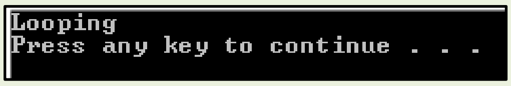
DO WHILE loops are helpful if a programmer is unsure about the number of times code will need to be repeated. This is because a condition can be used to control the number of iterations in the loop, as opposed to a fixed numeric value. The example below demonstrates the WHILE loop syntax:

```
public class TheWhileLoop{

  public static void main(String[] args){

      int x = 1;
      while(x < 5){
          System.out.println(x);
          x = x + 1;
      }
  }
}
```

Programming Tips!

There is also a 'DO WHILE loop'; this is the same as the WHILE loop, except the condition is checked at the end, as opposed to the start. This means that the code will execute at least once, before checking the condition (regardless of whether the condition is true or not). This differs from the WHILE loop which will not execute any code if the condition is found to be false. The syntax for a DO WHILE loop is shown below:

```
int x = 2;
do{
   System.out.println("Looping");
}while(x==3);
```



```
Looping
Press any key to continue . . .
```

The example above has a condition that states 'variable x must have a value less than 5'. As x has the value 1 to start with the given condition is true, and thus the code will execute. The second line of code [x = x + 1;] inside the loop (after the [println()] method) will increment the value of x by 1, before the loop returns to the start and checks the condition for a second time. The loop will continue until x has the value of 5, because at that point the condition becomes false (if x equals 5 then this is no longer less than 5) and the code ceases to execute. When the condition is false, the program flow returns to the rest of the program code. The example above will, in total, loop 4 times and print the numbers 1 to 4 to the command-line; if a programmer wanted the loop to iterate 5 times then the condition needs to be amended to 'x <= 5'

5.4 Practical: Using WHILE Loops

Exiting a Loop

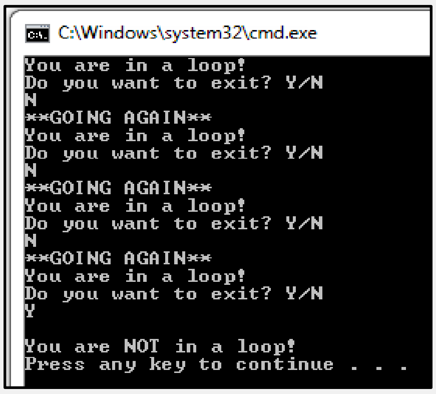
The example below demonstrates how an IF statement and WHILE loop can be implemented together to create a repeating program. In brief, this program will continue to loop until the user decides to exit the loop by typing 'Y' (or 'y') into the command-line:1 at which point the variable counter is set to false, resulting in the WHILE condition no longer being true and thus the end of the loop. Note that boolean variables make excellent counters when working with WHILE loops.

Try the code

```
import java.io.*;
public class ExitingALoop{

       public static void main(String[] args){
           boolean counter = true;
           BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
           String myText = null;
           while(counter==true){
                 System.out.println("You are in a loop!");
                 System.out.println("Do you want to exit? Y/N");
                 try{
                         myText = x.readLine();
                 }catch(IOException e){
                         System.out.println("Error: " + e);
                 }
                 if(myText.equals("Y") || myText.equals("y")){
                         counter = false;
                 }else{
                         System.out.println("**GOING AGAIN**");
                 }
           }
           System.out.println("");
           System.out.println("# # You are NOT in a loop! # #");
       }
}
```

Step by step

1. Type out the above code
2. Save the program as ExitingALoop.java
3. Compile the program (use the javac compiler)
4. Run the ExitingALoop.class file
5. Check the results of the program



```
You are in a loop!
Do you want to exit? Y/N
N
**GOING AGAIN**
You are in a loop!
Do you want to exit? Y/N
N
**GOING AGAIN**
You are in a loop!
Do you want to exit? Y/N
N
**GOING AGAIN**
You are in a loop!
Do you want to exit? Y/N
Y

You are NOT in a loop!
Press any key to continue . . .
```

Activity 5.4

The Area App – write a program that asks the user to choose a shape (from the list below) and once, chosen, prompts the user to enter the necessary information to calculate the area of the selected shape. Once the area of the shape has been calculated, the program should print the answer to the command-line. Finally, the program should prompt the user if they wish to choose another shape; if they do, the program should repeat, else it should end. The shapes are as follows:

- Circle – require radius from user
- Square – require length from user
- Rectangle – require length and width from user
- Triangle – require base and height from user
- Trapezium – require base, top and height from user

Programming Tips!

A double value can be formatted to two decimal places, by importing the `DecimalFormat` class. To use the class, [import `java.text.DecimalFormat;`] must be added to the top of a program. The following code will format a double value to two decimal places, in this case 2.34:

```
double myNum = 2.34651223;
DecimalFormat f = new
DecimalFormat("##.00");
System.out.println(f.format(myNum));
```

The Guessing Game

Learning techniques in programming is the ‘easy part’, learning how and when to apply them is the ‘tricky part’. Use the skills and theory learnt in this chapter (in particular WHILE loops) and previous chapters to complete Dave’s Guessing Game:

Activity 5.6a

Dave’s Guessing Game – write a program that generates a random number between 1 and 50 before prompting the user to guess what the random number is.

The rules of the game are simple:

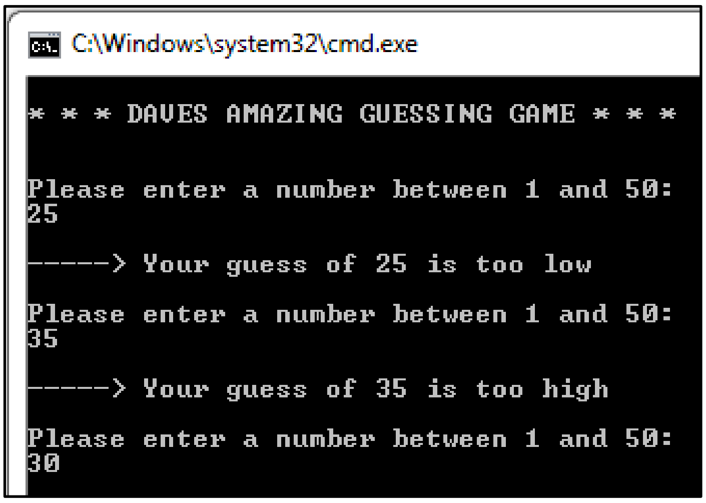
- The user must keep guessing until they guess the random number, at which point the program should end
- Every time the user guesses incorrectly, the program should hint to the user whether their guess was too low or too high

Activity 5.6b

Amend Dave’s Guessing Game – modify the program so that a second WHILE loop is used to ensure that the user’s guess is between 1 and 50. Any other numbers should result in the user being prompted for another response, as well as a message to the user that their input was invalid.

Activity 5.6c

Amend Dave’s Guessing Game – modify the program so that the user has 7 chances to guess the random number. If the 7 chances expire before the user guesses correctly, the program should end. After each incorrect guess the user should be made aware of how many chances they have remaining.


C:\Windows\system32\cmd.exe

```
* * * DAVES AMAZING GUESSING GAME * * *

Please enter a number between 1 and 50:
25

-----> Your guess of 25 is too low

Please enter a number between 1 and 50:
35

-----> Your guess of 35 is too high

Please enter a number between 1 and 50:
30
```

Programming Tips!

When creating the guessing game, the programmer will need to be able to generate a random number. The `Random` class already has the methods programmed to achieve this. To use the Random class it must first be imported, by typing [`import java.util.Random;`] at the top of the program code. The following code can be used to generate a random integer between 1 and 10. Note that +1 is needed, else the code will return a random number between 0 and 9 instead.

```
Random rand = new Random();
int num = rand.nextInt(10) + 1;
```

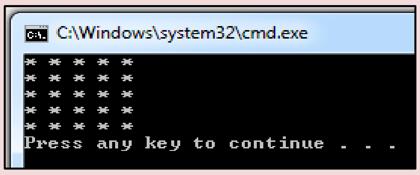
Simple Shapes Program

Based on previous learning from this chapter, refine skills and knowledge learnt by completing the final two activities below:

Activity 5.7

The Square App – write a program that uses two loops (one inside of another) to output a square shape, which is 5 by 5, to the command-line. The programmer can use no more than one asterisk character and one space character.

The Square App Extended – amend the program so that the user can specify the size of the square.



```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
Press any key to continue . . .
```

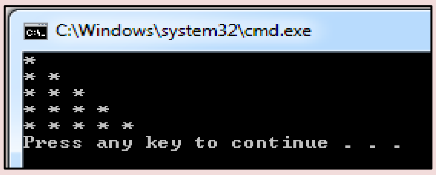
Programming Tips!

To accomplish this task use the [`print("* ")`] method to output the width of a shape and [`println()`] method to output the length of a shape!

Activity 5.8

The Triangle App – write a program that uses two loops (one inside of another) to output a triangle shape, which is 5 high, to the command-line. The programmer can use no more than one asterisk character and one space character.

The Triangle App Extended – amend the program so that the user can specify the height of the triangle.



```
*
* *
* * *
* * * *
* * * * *
Press any key to continue . . .
```