

Chapter 6: Nesting, Algorithms and More String

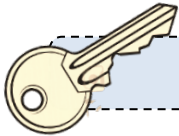
Learning Outcomes:

- ✓ Identify the importance of nesting in code
- ✓ Identify the importance of writing good algorithms, as opposed to hard coding
- ✓ Implement nesting techniques
- ✓ Implement methods from the `String` class



Prerequisite Knowledge:

- ✓ Complete Chapter 5
- ✓ Be able to confidently implement selection and iteration techniques in a program
- ✓ Be able to use appropriate indentation in a program



Nesting

Algorithm

Argument

Control Structures

Keywords

6.1 The Theory: Nesting Techniques

When writing algorithms, sometimes the standard iteration and/or selection control structures do not provide enough flexibility to meet the project requirements. Thankfully, however, additional flexibility can be added to selection/iteration structures by simply 'nesting' them inside of other selection/iteration structures. For example, an IF statement can be nested inside of another IF statement, or loop. In fact, control structures can be 'nested' inside of each other as many times as required. In the previous chapter, nesting was used to complete 'the triangle' and 'the square' programs; these programs implemented a loop inside of a second loop, providing the additional functionality that was needed to meet the project requirements. If necessary, however, these programs could have nested a third and fourth loop further still.

Nesting Examples

The example below is an extension of the driving licence activity that was completed in Chapter 3. In this example, nesting is used to add additional conditions (extended functionality) to the program; the 'extended functionality' uses the number of points accumulated on the driving licence to determine the length of the ban too. Note that the inner IF statements will not be tested/executed unless the condition(s) of the outer IF statement return true.



```
int points = 15;
int years = 3;

if(years > 2 && points >= 12){
    if(points < 15){
        System.out.println("You are banned for 3 years");
    }else{
        System.out.println("You are banned for 4 years");
    }
}else if(years <= 2 && points >= 6){
    if(points < 12){
        System.out.println("You are disqualified");
    }else{
        System.out.println("You are banned for 1 year");
    }
}
```

Below is an alternative example of nesting; this example illustrates how an IF statement can be implemented inside of a WHILE loop. In short, the program asks the user to guess a 'lucky number' in order to end the game. The code uses an IF statement to determine if the user has correctly guessed the lucky number, to be specific the number 7. If the user's guess is true, the IF statement changes the value of the `counter` variable to 1, resulting in the WHILE loop's condition becoming false and thus exiting the loop on the next iteration.

```
int counter = 0;
BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
String luckyNum = null;

while(counter == 0){
    System.out.println("Enter a lucky number?");
    try{
        luckyNum = x.readLine();
    }catch(IOException e){
        System.out.println("Error: " + e);
    }
    if(luckyNum.equals("7")){
        System.out.println("Great Guess - well done!");
        counter = 1;
    }
}
```

The next example is a possible solution to 'the square' activity as featured in Chapter 5. The code demonstrates how a loop can be nested inside of another loop. Be aware that the inner loop will execute all of its iterations entirely, on a single iteration of the outer loop. In total, the inner loop will iterate 25 times, whereas the outer loop will iterate just 5 times.

```
for(int i=1; i<6; i++){
    for(int y=1; y<6; y++){
        System.out.print("* ");
    }
    System.out.println();
}
```



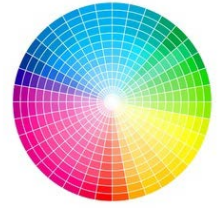
Programming Tips!

As mentioned in an earlier chapter, indentation is important. However, when nesting control structures inside of other control structures, indentation becomes even more important! Indentation enables the start and the end of a structure to be clearly identified; it also makes nested techniques visually clearer for programmers, resulting in code being easier to work with!

6.2 Practical: Nesting and Algorithms

The Colour Mixer Program

Use the knowledge learnt from the previous text to complete the colour mixer program. Remember to use appropriate indentation so that each control structure (i.e. loop or IF statement) can be easily distinguished. To complete this exercise entirely, skills learnt from former chapters, such as the `[readLine()]` method and `[try/catch]` technique, will need to be implemented too. In programming, it is important for beginners to regularly review previous programs built; being able to write programs from 'scratch' is something that develops over time.



Activity 6.1

The Colour Mixer – write a program that will inform the user of new colours that can be created by mixing other colours together. The program must demonstrate nesting by implementing IF statements inside of each other. The program should inform the user of the following colour mixes:

- Red + Blue = Magenta
- Red + Red = Red
- Red + Black = Dark Red
- Blue + Red = Magenta
- Blue + Blue = Blue
- Blue + Black = Dark Blue
- Black + Red = Dark Red
- Black + Blue = Dark Blue
- Black + Black = Black

```
C:\Windows\system32\cmd.exe
##### The Colour Mixer #####
Choose a colour:
<A> Red
<B> Blue
<C> Black
B
Choose a second colour to mix:
<A> Red
<B> Blue
<C> Black
C
RED + BLACK = DARK RED
Press any key to continue . . .
```

Activity 6.2

The Colour Mixer extended – amend the colour mixing program by adding an outer loop to repeat the colour mixing process, if the user chooses to do so; otherwise the program should end.

The Shapes Program – Writing Algorithms

```
int z = 2;
int j = 5;
for(int i=1; i<6; i++){
    for(int k=1; k<j; k++){
        System.out.print(" ");
    }
    for(int y=1; y<z; y++){
        System.out.print("*");
    }
    System.out.println();
    z=z+1;
    j=j-1;
}
System.out.println();
```



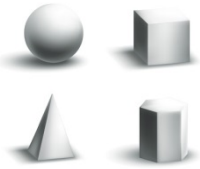
Try the code



Step by step

1. Type out the code into the main method
2. Save the program as `MyTriangle.java`
3. Compile the program (use the `javac` compiler)
4. Run the `MyTriangle.class` file
5. Check the results of the program

```
C:\Windows\system32\cmd.exe
*
**
***
****
*****
Press any key to continue . . .
```



The previous code generates a right-angled triangle using the asterisk character and the space character. To achieve the triangle shape, a clever algorithm is used, as opposed to using the `[println()]` method numerous times to print each line of the triangle. Using an algorithm comes with its own advantages, especially when coping with future change. The triangle algorithm above uses variables as counters in the FOR loop condition, instead of using fixed

numbers. The variable's values could even be set using a `[readLine()]` method, resulting in the algorithm being flexible enough to generate a right-angled triangle of any specified size by the user. The lesson here is that code should be written as independently as possible, meaning that code should be robust and able to cope with change, with little or no 'tinkering'.

The algorithm above implements two FOR loops inside of an outer FOR loop; what a headache! This time the two inner loops will execute fully for each cycle of the outer loop. The 'clever part' about this algorithm is that the first inner loop (which generates the space characters) will shrink by one space character per cycle of the outer loop, whereas the second inner loop (which generates the asterisk characters) will increment by one asterisk character per cycle of the outer loop. Finally, note the use of the `[print()]` and `[println()]` methods in the algorithm; the `[print()]` method continues characters on the same line, whereas the `[println()]` method creates an entirely new line.

Activity 6.3

The Shapes Program – write an algorithm, in a program, that generates a right-angled triangle to a specified size by the user. The base of the triangle should be printed before the tip.

```

C:\Windows\system32\cmd.exe
Please enter a shape size <Between 2 and 20>;
10

*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

Activity 6.4

The Shapes Program 2 – write an algorithm, in a program, that generates a right-angled triangle (with spaces on the left margin) to a specified size by the user. The base of the triangle should be printed before the tip.

```

C:\Windows\system32\cmd.exe
Please enter a shape size <Between 2 and 20>;
10

*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

Activity 6.5

The Shapes Program 3 – write an algorithm, in a program, that generates a diamond shape to a specified size by the user; the size is measured by the number of rows to the middle. Note that for the diamond shape to work, the value from the user must be an odd number!

```

C:\Windows\system32\cmd.exe
Please enter an odd number <Between 3 and 19>;
11

  *
 ***
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

Programming Tips!

To create the diamond shape, break the program down into two triangles: the top triangle and the bottom triangle. Also, note that the character spaces must start large and decrease by 1 per cycle, whereas the asterisks must grow by 2 characters per cycle.

6.3 Practical: Working with Strings

String Methods – The Anagram and The Enigma Game

A programmer must have a full 'toolbox', and part of that toolbox is being able to work with strings. In Chapter 4, the Java API was discussed in detail, including the use of pre-defined classes that can be utilised to improve the development time of a project. The `String` class is no different, and contains useful methods that can be used to manipulate string values. The table below outlines a range of string methods and their given purposes:

Ww
Xx



Programming Tips!

When a programmer passes a value to a method, it is referred to as passing an 'argument'. For example, when using the `[println("hi")]` method, the given string between the brackets, `("hi")` in this case, is referred to as an argument!

Activity 6.6a



The Anagram – write a program that prints an anagram to the command-line. The user should try to guess the given anagram; the game should continue until the user guesses correctly or until the user runs out of chances.

After each guess the program should offer a hint to the user. The answer should be stored in a variable and either the `[substring()]` or `[charAt()]` method should be used to reveal part of the anagram.

charAt()		
<pre>String x = "Joseph"; System.out.println(x.charAt(2));</pre>	Code Output: s	Returns the character found at a specified position in a string. Positions start at 0 and continue to the number of characters in the string minus 1. For example, the letter 'h' is at position 5.
equals() and equalsIgnoreCase()		
<pre>String x = "Joe"; System.out.println(x.equals("Joe"));</pre>	Code Output: true	Returns a true or false value dependent on whether two strings are equal in value (if they match). Use the <code>[equalsIgnoreCase()]</code> method if the caps sensitivity of the value being matched is not important.
substring(x) – 1 Argument		
<pre>String x = "Joseph"; System.out.println(x.substring(2));</pre>	Code Output: seph	Returns a substring (a section of a string) starting at a specified position. Like other methods, the positions start at 0 and continue to the number of characters in the string minus 1.
substring(x,y) – 2 Arguments		
<pre>String x = "Joseph"; System.out.println(x.substring(2,5));</pre>	Code Output: sep	Returns a substring (a section of string) between a start position and end position. The positions start at 0 and continue to the number of characters in the string minus 1.

Activity 6.6b

The Enigma – write a program that prints an encrypted word to the command-line. The user must try to guess the letters of the encrypted word to reveal the answer; the user should have 10 chances to reveal the encrypted word. Each time the user guesses a correct letter, an updated version of the encryption must be printed to the command-line.

```
C:\Windows\system32\cmd.exe
<> <> <> THE ENIGMA <> <> <>
Crack the enigma code <guess a letter>: $2x824*!1!8
M
Crack the enigma code <guess a letter>: $2x824MM1!8
Y
-----> Guesses Remaining: 9
-----
Crack the enigma code <guess a letter>: $2x824MM1!8
R
Crack the enigma code <guess a letter>: $R28R4MM1!8
P
```



Programming Tips!

To complete the enigma activity, use two variables called `myWord` and `myEncryption`. The variables should hold the word and the encryption alike; if a letter is repeated in the encrypted word, use the same encryption symbol for that letter. Use the `[contains()]` method to determine whether the given letter is found in the `myWord` variable. If the given letter is present, use an IF statement and the `[replace()]` method to update the `myEncryption` variable. Use either the `[toUpperCase()]` or `[toLowerCase()]` method to remove any case sensitivity issues, and use the `[trim()]` method to remove any white space from the user's guess. Finally, place the entire code into a WHILE loop to keep the game going until the appropriate condition becomes false!

replace()

```
String x = "joseph";
System.out.println(x.replace("j", "J"));
```

Code Output:

Joseph

Finds a specified character or characters in a string and replaces it with a specified character or characters.

endsWith() and startsWith()

```
String x = "Joseph";
System.out.println(x.endsWith("h"));
```

Code Output:

true

Returns a true or false value dependent on whether the string ends/starts with a specified character or characters.

toUpperCase() and toLowerCase()

```
String x = "joseph";
System.out.println(x.toUpperCase());
```

Code Output:

JOSEPH

Turns all characters in a string to either UPPER-CASE or lower-case. This can be useful when matching strings given from a user.

trim()

```
String x = "    joseph    ";
System.out.println(x.trim());
```

Code Output:

joseph

Removes all character spaces at the beginning and end of a string. Note that this will not remove any character spaces that appear in the middle of a string.

contains()

```
String x = "Joseph";
System.out.println(x.contains("se"));
```

Code Output:

true

Returns a true or false value dependent on whether a string contains the values from another string. The matching string must be exact; for example, 'SE' or 's e' would have returned false.

More String Methods

There are many more methods available in the `String` class too; it is worthwhile having a look at the Java API documentation to see the other string methods that are available. Once familiar with the `String` class, don't stop there; have a look at the `Integer` class to see what methods are available when working with whole numbers. Once finished with the `Integer` class, look at the `Double` class and see what methods are available when working with decimal values. The Java API documentation is a programmer's best friend; it holds all the key information about every class and method in the Java API!



Activity 6.6a

Top 3 – write a program that asks the user to enter their top three favourite films; the program should store all three films in a single variable and separate them with a dash (-). The program should then use the `[split()]` method to print the three films, each on a new line, to the command-line. Use the `[length()]` method to print the original length of the string variable to the command-line before it is split.

The table below outlines two more methods available in the `String` class. Note that the `[split()]` method uses something called an 'array' to store the returned value from the method; arrays are examined later in Chapter 8.

length()		
<pre>String x = "Joseph"; System.out.println(x.length());</pre>	Code Output: 6	Returns an integer value that represents the number of characters found in the specified string.
split()		
<pre>String x = "Joseph - James - John"; String[] y = x.split("-"); System.out.println(y[0]); System.out.println(y[1]); System.out.println(y[2]);</pre>	Code Output: Joseph James John	Splits a string into smaller parts; the split occurs when a given character is encountered. The method returns a string array that contains all of the new strings that were created by the split.



Programming Tips!

There are 'escaped constructs' available in Java that are quick tools when working with strings. For example, `[\n]` can be typed inside of a string ("Joseph \n James \n John") to create a new line, as opposed to using numerous `[println()]` methods in the code. The same applies for `[\t]` which can be used to add a tab space inside of a string.

When using doubles quotes ("") or the backslash character (\\) inside of a string, they must be escaped by preceding each character with a backslash. For example, the string "The directory is: c:\\temp\\myfolder" would print to the command-line: **The directory is: c:\temp\myfolder**

6.4 Practical: Date Formatting

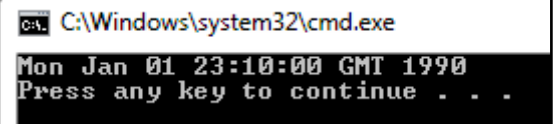
Working with Dates in a Program

Another data type that a programmer must be familiar with is 'Date', which is used to store a date value. The `Date()` class also holds many methods that can be used to manipulate a date value; see the Java API documentation for full details. When using the date data type, it must first be imported by using the following code [`import java.util.Date;`] at the top of a program. The example below demonstrates how a date data type might be used; as date is not a primitive type, an object has to be created with the [`new`] keyword (this was also discussed earlier in Chapter 4). The five arguments [`Date(90,0,1,23,10)`] that are passed to the date object when it is instantiated are used to set the date and time of the object. In order, the arguments are: year (90 as in 1990), month (months start at 0 for January), date, hour and minute.

```
import java.util.Date;

public class WorkingWithDates{

    public static void main(String[] args){
        Date date = new Date(90,0,1,23,10);
        System.out.println(date.toString());
    }
}
```



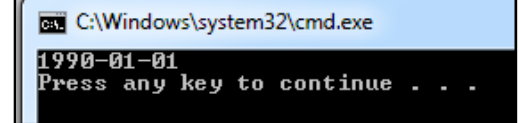
```
C:\Windows\system32\cmd.exe
Mon Jan 01 23:10:00 GMT 1990
Press any key to continue . . .
```

The previous example prints the date and time out in full, including the time zone. However, there is a `SimpleDateFormat` class which can be used to manipulate the appearance of a date value. To use the class it must first be imported using the following code [`import java.text.*;`] at the top of a program. An example of how the `SimpleDateFormat` class might be used is shown below:

```
import java.text.*;
import java.util.Date;

public class WorkingWithDates{

    public static void main(String[] args){
        Date date = new Date(90,0,1,23,10);
        SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");
        System.out.println(ft.format(date));
    }
}
```



```
C:\Windows\system32\cmd.exe
1990-01-01
Press any key to continue . . .
```

Activity 6.8

Use the `SimpleDateFormat` class in a program to determine what character formats, in the code, would produce the following date formats:

- 1990-01-01 11:10 GMT
- 1990-01-01 Mon 11:10:00
- Mon, 01-01-1990 GMT
- Mon, 01-01-1990 – 11:10 PM

The above format is [`yyyy-MM-dd`], which is short for year, month and date. However, any desired format can be created by changing the key letters. For example, `HH` could be added to display the hours and `mm` could be added to display the minutes. Other useful characters are `z` to display the time zone, `s` to display seconds, `a` to display the am/pm marker and `E` to display the day of the week.