Chapter 5. Strings: Basic Handling

The objective of this chapter is to ensure that you are conversant with string processing in Java, given the fact that string manipulation is a key feature of any programming language. Among the string exercises presented in this chapter are exercises on string variable declaration, concatenation, case modification, equality testing, substring manipulation, as well as the formatting of strings for output. Your understanding of the immutability of `String` objects is also exercised.

The key class on which exercises are presented in this chapter is the class `String`.

Note: Due to the intertwining of the topic of string handling with other topics yet to be discussed, we will deal with strings as far as possible in this chapter and address other aspects of strings again in the chapters on the `StringBuffer` class, the `Char` type and Arrays and also to different degrees in other chapters as appropriate.

1. Write a program which declares a `String` variable named `firstName` and initializes its value to the word *"John"* using the operator `new`.

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

     String firstName = new String("John");
   }
}
```

2. Repeat the same exercise as above, initializing the variable simply by assigning the word *"John"* to the `String` variable using the assignment operator.

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

     String firstName = "John";
   }
}
```

3. *(string concatenation)*
I have the `String` variables `s1` and `s2` with values as indicated:

```
 s1 = "John";
 s2 = "Leavings";
```

Using the addition (+) operator, put into a third `String` variable `s3` the concatenation of these two strings, ensuring that you put a space in-between them. Print `s3` out to the console.

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

     String s1 = "John";
     String s2 = "Leavings";
     String s3 = s1 + " "  + s3;
     System.out.println(s3);
   }
}
```

4. Repeat the preceding exercise, using the `concat` method of the class `String`.

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

    String s1 = "John";
    String s2 = "Leavings";
    String space = " ";

    String s3 = s1;
    s3 = s3.concat(space);
    s3 = s3.concat(s2);
    System.out.println(s3);
  }
}
```

5. Determine and print out the length of the string contained in the `String` variable `firstName` of exercise 2 above.
Hint: Use the `Length` method of the `String` class.

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

    String firstName = "John";
    int stringLength = 0; // We will use this variable for the length.
    stringLength = firstName.length();
    System.out.printf("The length of the first name is %s\n", stringLength);
  }
}
```

Note: We could, without having declared the variable `stringLength` have printed the length of the string directly in the `System.out.printf` statement as shown below:

```
    System.out.printf("The length of the first name is %s\n", firstName.length());
```

6. I have the following string literal, *"Basketball!"*. Print out its length, without declaring any variables.

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

    System.out.printf("The length is %d.\n", "Basketball!".length());
    }
  }
}
```

7. Print out the value of the variable `firstName` from exercise 2 in lowercase.
Hint: Use the `String` method `toLowerCase()`.

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

    String firstName = "John";
    String firstNameLowerCase = firstName.toLowerCase();
    System.out.printf("firstName as all uppercase is %s\n", firstNameLowerCase);
  }
}
```

OR

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

    String firstName = "John";
    System.out.printf("firstName as all lowercase is %s\n", firstName.toLowerCase());
    //We just executed the desired method within the parameter field of the method
    //System.out.printf.
  }
}
```

8. Print out the value of the variable `firstName` from the preceding exercise in uppercase.

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

    String firstName = "John";
    System.out.printf("firstName as all uppercase is %s\n", firstName.toUpperCase());
  }
}
```

9. What does the statement "*a `String` is immutable*" mean?

Before we answer the question, it must first be understood that for variables that pertain to objects, what is contained in the variable is not directly the object itself, but rather a value stating the location in memory where the object that it references is located. Therefore, when you access an object variable, in order to get the object referred to, the Java run-time actually performs two steps; first it looks up what memory address the variable is pointing to and then secondly goes to that memory location to get the contents thereof. Now to the explanation of `String` immutability.

What the statement a *`String` is immutable* means is that once a value is assigned to a `String` object, Java does not actually allow the value contained in the `String` object to be changed. However, this bears explanation since, in a sense, we can "change" the contents of a `String` as shown below:

```
  String s1 = "Hello. ";
  s1 = s1 + " How are you?";
```

When we add the string "`How are you?`" to the string in the object reference `s1`, what really happens is that under the covers Java creates a brand new string *"Hello. How are you?"* somewhere else in memory and in the

	background assigns a reference to the location of this new string to the variable `s1`! Why is this knowledge of the immutability of objects of the `String` class important to know? It is important to know for a number of reasons, including (1) the fact that modifying a `String` (which we now know causes a new string to be created, and the variable holding a reference to it to be updated to the location of the new string) is relatively expensive timewise. Therefore, for situations where we have to manipulate a string quite often in a program, it is best that the class **`StringBuilder`** which is mutable be used rather than the class `String` and (2) if the particular memory reference to which a given `String` variable pertains is important to you then you should know that it will be changed when you modify the contents of the given `String` variable. Really, strings can be referred to as *immutable reference* types. They have unique behavior, in particular the following: 1. (Previously stated) They cannot be changed once created. If you modify a `String` (appending, trimming, etc), a new string is created and presented to the program in which it was changed, *with the same variable name*, giving you the appearance that the contents of the `String` variable in hand were changed (As stated earlier, in a sense it is changed by Java creating a new string and making the variable `s1` contain a reference to the location in memory of the new string). For many programs, this fact has no negative impact on the construction of the program as this throwing away and re-creation is hidden from the programmer who requested the modification. 2. Even though they are reference types, due to their immutability, if you pass a `String` to another method which then tries to modify its contents, unlike other reference types, the contents of the original `String` in the calling method will not be modified. **Note:** Despite the extensive explanation above, for general intents and purposes, you can program with the mentality that you *can* change the contents of a `String` in a method, however you must be aware of what is going on in the background.	
10.	I have a string object reference `s1`, initialized as follows: `String s1 = null;` What does this initialization to the value `null` mean?	
	The initialization to the value `null` is saying in effect that "this `String` object reference `s1` does not contain a reference to any object". The value `null` does not only apply to `String` variables; it can be used for any object reference variable. When any object reference has the value of `null`, it is saying that the object reference variable in question does not contain a reference to any object.	

11.	What is the difference between the following two `String` objects?
	`String s1 = null;`
	`String s2 = "";`	

	The `String` variable `s1` does not point to any object at all, whereas `s2` points to an empty string. An empty string is still regarded as a `String` object, whereas `null` really means nothing; not even empty. If we run the following code: `System.out.println(s1.length());` It will result in an exception, `java.lang.NullPointerException`, for the action is on an unassigned variable. However, `System.out.println(s2.length());` will result in the value `0` being output.	
String Equality		

12.	Given two strings `s1` & `s2`, using the `String.equals` *instance method*, write a line of code to test whether `s1` & `s2` are equal and put the result into a `boolean` variable `x`.	`boolean x = s1.equals(s2));`
		OR
		`boolean x = s2.equals(s1));`

Substrings		
13.	What is the index of the first character in a `String` in Java?	
	The index of the first character in a Java `String` object is 0. Therefore if for example we have 10 characters in a string, the indexes of the individual characters therein from the 1st to the 10th will be from 0 to 9 in order.	
14.	Given the string *"Mary had a little*	The character is at position 5.

lamb", manually determine the position of the character 'h' in the string.

This is because character position enumeration in Java for strings starts from 0. Therefore the positions in this string would be;

```
M  0
a  1
r  2
y  3
   4
h  5
a  6
d  7
```

15. Given the string *"Mary had a little lamb"*, write a program to determine the position of the letter '`i`' in this string. Print this position out to the console stating: *The position of letter* '*i*' *is : <position>*.
Hint: `String` *method* `<String>.indexOf`

Solution #1

```java
public class PracticeYourJava {
  public static void main(String[] args) {
  String rhymeLine = "Mary had a little lamb";
  int letterPosition = rhymeLine.indexOf('i'); // note single quotes for a single letter
  System.out.printf("The position of the letter \'i\' is: %d\n.", letterPosition);
  }
}
```

Note: a single character literal is referred to using single quotes, not double quotes, as seen in this example when we look for the index of `'i'` using the `indexOf` method.

Solution #2

```java
public class PracticeYourJava {
  public static void main(String[] args) {
    String rhymeLine = "Mary had a little lamb";
    System.out.printf("The position of the letter \"i\" is %d.\n", rhymeLine.indexOf('i'));
  }
}
```

Note: In this variant of the answer, we simply call the `indexOf` method directly within the parameter section of the `System.out.printf` method.

Solution #3

```java
import java.util.*;
public class PracticeYourJava {
  public static void main(String[] args) {
    int letterPosition = "Mary had a little lamb".indexOf('i');
    System.out.printf("The position of the letter \"i\" is %d.\n", letterPosition);
  }
}
```

Note: Observe carefully how in this solution we show how a `String` method can be applied *directly to the literal string* without declaring a variable to hold the string itself.

Solution #4

```java
public class PracticeYourJava {
  public static void main(String[] args) {
    System.out.printf("The position of the letter \"i\" is %d.\n",
                      "Mary had a little lamb".indexOf('i'));
  }
}
```

Note(s): In this variant of the answer, note how no variables are declared at all; we just apply the `indexOf` method directly to the string in the parameter section of the `System.out.printf` method.

16. Given the string "*Mary had a little lamb*", what is the position of the 2nd letter '**a**' in this string?
Hint: the overloaded `String.indexOf` *method.*

```java
public class PracticeYourJava {
  public static void main(String[] args) {
     String rhymeLine = "Mary had a little lamb";
    int positionOfFirstOccurrence = rhymeLine.indexOf('a');
    // Now, that we have the position of the 1st letter 'a' we start looking for
    // the very next letter 'a' after that.
    int positionOfSecondOccurrence = rhymeLine.indexOf('a',(positionOfFirstOccurrence + 1));
    System.out.printf("The position of the 2nd letter \"a\" is %d.\n",
positionOfSecondOccurrence);
  }
}
```

17. Given the string *"Mary had a little lamb, little lamb, little lamb, Mary had a little lamb that was as white as snow"*, write a program which will determine and output to the console the position of the 2nd occurrence of the word *"little"* in the string.
Hint: determine the position of the end of the 1st occurrence and then start searching after that.

```java
public class PracticeYourJava {
  public static void main(String[] args) {
    String rhymeLine = "Mary had a little lamb, little lamb, little lamb, Mary had a little lamb
                        that was as white as snow";
    // So what we have to do is first look for the 1st occurrence of the word "little",
    // and then start searching after that 1st occurrence.
    int iFirstAppearancePosition = rhymeLine.indexOf("little");
    int iSearchStartPosition = iFirstAppearancePosition + "little".length();
    // See that we start searching at the end of the 1st instance of the word "little"
    int iSecondAppearancePosition = rhymeLine.indexOf("little", iSearchStartPosition);
    System.out.printf("The 2nd occurrence of \"little\" is at position: %d.\n",
                        iSecondAppearancePosition);
  }
}
```

18. Given the string *"Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow"*, write a program to determine and print out the position of the *last* occurrence of the word *"little"* in the string.
Hint: `String.lastIndexOf`

```java
public class PracticeYourJava {
  public static void main(String[] args) {
    String rhymeLine = "Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a
                        little lamb that was as white as snow";
    int wordPosition = rhymeLine.lastIndexOf("little");
    System.out.printf("The position of the word \"little\" is %d.\n", wordPosition);
  }
}
```

19. Given the string *"Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow"*, write a program to determine and print out the position of the *last* occurrence of the letter 'w'.
Hint: `String` *method* `lastIndexOf(char)`

```java
public class PracticeYourJava {
  public static void main(String[] args) {
   String rhymeLine = "Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a
                       little lamb that was as white as snow";
   int letterPosition = rhymeLine.lastIndexOf('w');
   System.out.printf("The last position of the letter 'w' is %d.\n", letterPosition);
  }
}
```

20. Given a Java `String` that contains the rhyme *"Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little lamb that was as white as snow"*, write code to replace the word *"little"* with the phrase *"big big"* *(to keep the rhyme ☺)* everywhere the word *"little"* appears. Print out the modified string.
Hint: `String` *method* `replace`

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

    String rhymeLine = "Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little
                        lamb that was as white as snow";
    String newRhymeLine = rhymeLine.replace("little", "big big");
    System.out.printf("The new rhyme is:\n%s\n", newRhymeLine);
  }
}
```

21. Given the rhyme *"Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow"*, write a program which *removes* the word *"little"* wherever it appears in the rhyme. Print out the resulting string.
Hint: replace the word with an empty string

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

    String rhymeLine = "Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little
                        lamb that was as white as snow";
    String newRhymeLine = rhymeLine.replace("little", "");
    // "little" is replaced with a blank String. The double-quotes with nothing
    // in-between is a blank String.
    System.out.printf("The new rhyme is:\n%s\n", newRhymeLine);
  }
}
```

22. We have a phone number given as the string `111-222-3333`, of which the first 3 digits are the area code. Extract and print out the area code.
Hint: ***`String.substring`***

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

    String phoneNumber = "111-222-3333";
    int startPosition = 0; // We are starting at the beginning of the String
    int numberOfCharacters = 3;
    String areaCode = phoneNumber.substring(startPosition, (startPosition + numberOfCharacters));
    System.out.printf("The area code is :%s.\n", areaCode);
  }
}
```

23. We have a phone number given as the string `111-222-3333`, of which the last 7 digits are the subscriber number. Extract and print out the subscriber number, that is, the substring `222-3333`.

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

    String phoneNumber = "111-222-3333";
    int startPosition = 4;
    // We are starting after the first dash. Remember start counting at 0.
    String SubscriberNum = phoneNumber.substring(startPosition);
    // Extracts to the end of the String
    System.out.printf("The subscriber number is : %s\n ", SubscriberNum);
  }
}
```

24. We have a phone number, `111-222-3333`, of which the first 3 characters are the area code. The next 3 numbers are the "Central Office number". Extract this number and print it out to the console.

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

    String phoneNumber = "111-222-3333";
    int startPosition = 4; // We are starting after the first dash
    int NumberOfCharsToExtract = 3;
    String CONumberStr = phoneNumber.substring(4, startPosition+NumberOfCharsToExtract);
    // Extracts to the end of the String
    System.out.printf("The Central Office number is : %s\n", CONumberStr);
  }
}
```

Note: If desired you can write this in a more generic way; determine the position of the first and second dashes and extract what is between them.

25. We have a phone number given as the string `111-222-3333`, of which the first 3 characters are the area code. Find and print out the *last* position of the number **`1`** within the area code.

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

    String phoneNumber = "111-222-3333";
    int lastSearchPosition = 2;
    int foundPosition = phoneNumber.lastIndexOf('1', lastSearchPosition);
    System.out.printf("Last position of the number 1 in the area code is : %d\n", foundPosition);
  }
}
```

26. I have the following string:
"   *how are you?*   "
Write a program which gets rid of both the leading and trailing spaces and print the result out. Confirm that you have erased the trailing spaces by printing out the letter 'X' right behind the trimmed string.

Hint: `String.trim`

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

    String s1 = "        how are you?     ";
    String ending = "X";
    String s2 = s1.trim();
    System.out.printf("%s%s\n", s2, ending);
  }
}
```

27. We have the following string:

`">>>>>>How are you<<<<<<"`

Using the `replace` method of class `String`, erase the extraneous characters bounding the phrase "*`How are you?`*". Print the trimmed string out.

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

    String s1 = ">>>>>>>how are you?<<<<<<<<";
    String s2 = s1.replace(">","");
    s2 = s2.replace("<","");

    System.out.println(s2);
  }
}
```

28. The string *"Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little lamb that was as white as snow"* prints out as:

Mary had a little lamb,
little lamb,
little lamb,
Mary had a little lamb that was as white as snow

Write a program which determines the position relative to the start of the 2nd line of the rhyme of the word *"lamb"*.

Code logic
(This is one possible solution) Extract the 2nd line (the \n is the delimiting point) and then look for the word "lamb" in the extracted line.

```java
public class PracticeYourJava {
  public static void main(String[] args) {
    String rhymeLine = "Mary had a little lamb,\nlittle lamb,\n little lamb,\n Mary had a
                     little lamb that was as white as snow";
    int startPos = rhymeLine.indexOf('\n') + 1;
    String rhymeFrom2ndLine = rhymeLine.substring(startPos);

    int endPos = rhymeFrom2ndLine.indexOf('\n');
    String rhyme2ndLine = rhymeFrom2ndLine.substring(0, (endPos+1));

    // We have now extracted the 2nd line into the variable rhymeFrom2ndLine
    int wordPos = rhyme2ndLine.indexOf("lamb");
    System.out.printf("The 1st position of the word \"lamb\" in line 2 = %d\n", wordPos);
  }
}
```

Formatting String Output

29. I have the following `String` variables; `s1="Mary", s2="had", s3="a", s4="little"` and `s5="lamb"`. Print each of them out, right-justified, in a field that is of width 10 characters.

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

     String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";
     System.out.printf("%10s\n", s1);
     System.out.printf("%10s\n", s2);
     System.out.printf("%10s\n", s3);
     System.out.printf("%10s\n", s4);
     System.out.printf("%10s\n", s5);
   }
 }
```

30. Repeat the preceding exercise, however with the output *left-justified*. Also, bound each individual justified string with the character `'|'` on both the left & right.

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

    String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";
    System.out.printf("|%-10s|\n", s1);
    System.out.printf("|%-10s|\n", s2);
    System.out.printf("|%-10s|\n", s3);
    System.out.printf("|%-10s|\n", s4);
    System.out.printf("|%-10s|\n", s5);
  }
}
```

31. What is the use of the **`String.format`** method?

The **`String.format`** method is a method whose behavior is essentially like the `System.out.println/print` methods, however, its output is put into a `String`, rather than written to the console.

This method can be useful in scenarios where at given points in your code you produce intermediate output and prefer to append the intermediate output to a string with the intent of putting the final string out to the console or to a file later in your processing.

32. Rewrite exercise 30, this time, putting all the intermediate output into a `String` variable `s`. Print `s` out at the end of all the processing.
Hint: *`String.format`*

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

    String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";
    String s;
    s = String.format("|%-10s|\n", s1);
    s = s + String.format("|%-10s|\n", s2);
    s = s + String.format("|%-10s|\n", s3);
    s = s + String.format("|%-10s|\n", s4);
    s = s + String.format("|%-10s|\n", s5);
    System.out.println(s);
  }
}
```