SCALA STRINGS

Consider the following simple example where we assign a string in a variable of type val:

```
object Test {
   val greeting: String = "Hello, world!"

   def main(args: Array[String]) {
      println( greeting )
   }
}
```

Here, the type of the value above is **java.lang.String** borrowed from Java, because Scala strings are also Java strings. It is very good point to note that every Java class is available in Scala. As such, Scala does not have a String class and makes use of Java Strings. So this chapter has been written keeping Java String as a base.

In Scala, as in Java, a string is an immutable object, that is, an object that cannot be modified. On the other hand, objects that can be modified, like arrays, are called mutable objects. Since strings are very useful objects, in the rest of this section, we present the most important methods class java.lang.String defines.

Creating Strings:

The most direct way to create a string is to write:

```
var greeting = "Hello world!";

or

var greeting:String = "Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value, in this case, "Hello world!', but if you like, you can give String keyword as I have shown you in alternate declaration.

```
object Test {
   val greeting: String = "Hello, world!"

   def main(args: Array[String]) {
      println( greeting )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Hello, world!

C:/>
```

As I mentioned earlier, String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters then you should use String Builder Class available in Scala itself.

String Length:

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the length method, which returns the number of characters contained in the string object.

After the following two lines of code have been executed, len equals 17:

```
object Test {
   def main(args: Array[String]) {
      var palindrome = "Dot saw I was Tod";
      var len = palindrome.length();
      println( "String Length is : " + len );
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
String Length is : 17

C:/>
```

Concatenating Strings:

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end. You can also use the concat method with string literals, as in:

```
"My name is ".concat("Zara");
```

Strings are more commonly concatenated with the + operator, as in:

```
"Hello," + " world" + "!"
```

Which results in:

```
"Hello, world!"
```

Let us look at the following example:

```
object Test {
   def main(args: Array[String]) {
      var str1 = "Dot saw I was ";
      var str2 =  "Tod";
      println("Dot " + str1 + str2);
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Dot Dot saw I was Tod

C:/>
```

Creating Format Strings:

You have printf and format methods to print output with formatted numbers. The String class has an equivalent class method, format, that returns a String object rather than a PrintStream object. Let us look at the following example, which makes use of printf method:

```
object Test {
```

```
   def main(args: Array[String]) {
      var floatVar = 12.456
      var intVar = 2000
      var stringVar = "Hello, Scala!"
      var fs = printf("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar)
      println(fs)
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
The value of the float variable is 12.456000, while the
value of the integer var iable is 2000, and the
string is Hello, Scala!()

C:/>
```

String Methods:

Following is the list of methods defined by **java.lang.String** class and can be used directly in your Scala programs:

SN	Methods with Description
1	**char charAt***intindex* Returns the character at the specified index.
2	**int compareTo***Objecto* Compares this String to another Object.
3	**int compareTo***StringanotherString* Compares two strings lexicographically.
4	**int compareToIgnoreCase***Stringstr* Compares two strings lexicographically, ignoring case differences.
5	**String concat***Stringstr* Concatenates the specified string to the end of this string.
6	**boolean contentEquals***StringBuffersb* Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	

static String copyValueOf*char*[]*data*

Returns a String that represents the character sequence in the array specified.

8

static String copyValueOf*char*[]*data, intoffset, intcount*

Returns a String that represents the character sequence in the array specified.

9

boolean endsWith*Stringsuffix*

Tests if this string ends with the specified suffix.

10

boolean equals*ObjectanObject*

Compares this string to the specified object.

11

boolean equalsIgnoreCase*StringanotherString*

Compares this String to another String, ignoring case considerations.

12

byte getBytes

Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

13

byte[] getBytes(String charsetName

Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

14

void getChars*intsrcBegin, intsrcEnd, char*[]*dst, intdstBegin*

Copies characters from this string into the destination character array.

15

int hashCode

Returns a hash code for this string.

16

int indexOf*intch*

Returns the index within this string of the first occurrence of the specified character.

17

int indexOf*intch, intfromIndex*

Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

18

int indexOf*Stringstr*

Returns the index within this string of the first occurrence of the specified substring.

19

int indexOf*Stringstr, intfromIndex*

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

20

String intern

Returns a canonical representation for the string object.

21

int lastIndexOf*intch*

Returns the index within this string of the last occurrence of the specified character.

22

int lastIndexOf*intch, intfromIndex*

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

23

int lastIndexOf*Stringstr*

Returns the index within this string of the rightmost occurrence of the specified substring.

24

int lastIndexOf*Stringstr, intfromIndex*

Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

25

int length

Returns the length of this string.

26

boolean matches*Stringregex*

Tells whether or not this string matches the given regular expression.

27

boolean regionMatches*booleanignoreCase, inttoffset, Stringother, intooffset, intlen*

Tests if two string regions are equal.

28

boolean regionMatches*inttoffset, Stringother, intooffset, intlen*

Tests if two string regions are equal.

29

String replace*charoldChar, charnewChar*

Returns a new string resulting from replacing all occurrences of oldChar in this string with

newChar.

30

String replaceAll(String regex, String replacement

Replaces each substring of this string that matches the given regular expression with the given replacement.

31

String replaceFirst*Stringregex, Stringreplacement*

Replaces the first substring of this string that matches the given regular expression with the given replacement.

32

String[] split*Stringregex*

Splits this string around matches of the given regular expression.

33

String[] split*Stringregex, intlimit*

Splits this string around matches of the given regular expression.

34

boolean startsWith*Stringprefix*

Tests if this string starts with the specified prefix.

35

boolean startsWith*Stringprefix, inttoffset*

Tests if this string starts with the specified prefix beginning a specified index.

36

CharSequence subSequence*intbeginIndex, intendIndex*

Returns a new character sequence that is a subsequence of this sequence.

37

String substring*intbeginIndex*

Returns a new string that is a substring of this string.

38

String substring*intbeginIndex, intendIndex*

Returns a new string that is a substring of this string.

39

char[] toCharArray

Converts this string to a new character array.

40

String toLowerCase

Converts all of the characters in this String to lower case using the rules of the default locale.

41

String toLowerCase*Localelocale*

Converts all of the characters in this String to lower case using the rules of the given Locale.

42

String toString

This object *whichisalreadyastring*! is itself returned.

43

String toUpperCase

Converts all of the characters in this String to upper case using the rules of the default locale.

44

String toUpperCase*Localelocale*

Converts all of the characters in this String to upper case using the rules of the given Locale.

45

String trim

Returns a copy of the string, with leading and trailing whitespace omitted.

46

static String valueOf*primitivedatatypex*

Returns the string representation of the passed data type argument.