# Lesson 3.3

# *String and Scanner*

By the end of this lesson you will learn:

- Introduction to String
- How to Create String
- String Methods
- String Literals vs String Object
- String Equality
- Mutable vs Immutable Object
- Why String is Immutable in Java
- StringBuffer & StringBuilder
- What is Scanner Class in Java
- User Input through Scanner

---

## Introduction to String

**String** is probably the most used class in Java's class library. The obvious reason for this is that strings are a very important part of programming.

In Java, a string is a **sequence of characters**. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

The first thing to understand about strings is that every string you create is an **object of type String**. Even string constants are String objects. For example, in the statement

```
System.out.println("This is a String, too");
```

the string "This is a String, too" is a String object.

The second thing to understand about strings is that objects of type **String are immutable**. Once a String object is created, its contents cannot be altered. It will be discussed in detail later in the note.

Strings can be constructed in a variety of ways. The easiest is to use a statement like this:

```
String myString = "this is a test";
```

Once you have created a String object, you can use it anywhere that a string is allowed.
For example, this statement displays myString:

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

Java defines one operator for String objects: **+**. It is used to **concatenate** two strings. For example, this statement

```
String myString = "I" + " like " + "Java.";
```

results in **myString** containing "I like Java."

## How to Create String

A String can be constructed by either:

1. Directly assigning a "**string literal"** to a String reference - just like a primitive
2. Or, via the "**new**" operator and constructor, like any other classes.

For example,

```
String str1 = "Hello World";//Implicit construction via string literal
String str2 = new String("Hello Java");//Explicit construction via new
```

In the first statement, **str1** is declared as a **String reference** and initialized with a string literal "Hello World". In the second statement, str2 is declared as a String reference and initialized via the **new operator** and **constructor** to contain "Hello Java".

**String literals** are stored in a common pool. This facilitates *sharing of storage* for strings with the same contents to conserve storage. **String objects** allocated via new operator are stored in the heap, and there is no sharing of storage for the same contents.

## String Methods

The commonly used method in the String class are summarized below.

Let us create three strings first:

```
String s1 = "Hello";
String s2 = "World";
String s3 = "Hello";
```

| No | Method | Example |
|---|---|---|
| 1 | **length()** - returns the number of characters in the String. | s1.length(); // **5** |
| 2 | **charAt()** - returns the character at the specified index. | s2.charAt(2); // **r** |
| 3 | **equals()** - returns true if two strings have equal contents. | s1.equals(s2); // **false**<br>s1.equals(s3); // **true** |
| 4 | **compareTo()** -returns 0 if equal, Less than zero – if the invoking String is "less than" the other, Greater than | s1.compareTo(s2) // **-15**<br>s1.compareTo(s3) // **0** |

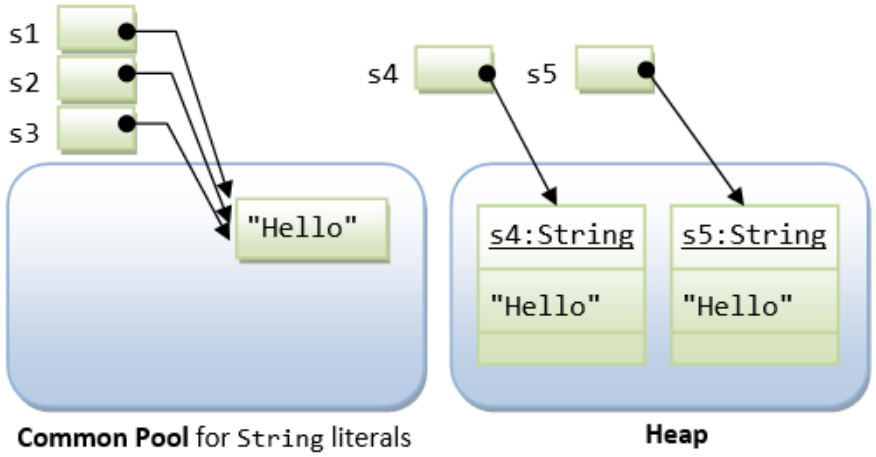
| | | |
|---|---|---|
| | zero - if the invoking String is "greater than" the other. | s2.compareTo(s3) // **15** |
| 5 | **indexOf()** - returns the index within the string of the first occurrence of the specified string. | s1.indexOf('e') // **1**<br>s1.indexOf('l') // **2** |
| 6 | **substring()** - returns a portion of the String's text. | s1.substring(2) // **lo**<br>s1.substring(0, 2) // **Hel** |
| 7 | **toUpperCase(), toLowerCase()** - converts the String to upper- or lower-case characters. | s1.toUpperCase() // **HELLO**<br>s1.toLowerCase() // **hello** |
| 8 | **concat()** - concatenates specified string to the end of this string. | s1.concat(s2) // **HelloWorld** |

## String Literal vs String Object

As mentioned, there are two ways to construct a string: **implicit construction** by assigning a **string literal** or explicitly creating a String object via the **new** operator and constructor. For example,

```
String s1 = "Hello";               // String literal
String s2 = "Hello";               // String literal
String s3 = s1;                    // same reference
String s4 = new String("Hello");   // String object
String s5 = new String("Hello");   // String object
```

Java has provided a special mechanism for keeping the String literals - in a so-called string common pool. If two string literals have the same contents, they will share the same storage inside the common pool. This approach is adopted to conserve storage for frequently used strings. On the other hand, String objects created via the new operator and constructor are kept in the heap. Each String object in the heap has its own storage just like any other object. There is no sharing of storage in heap even if two String objects have the same contents.




## String Equality

You can use the method **equals()** of the String class to **compare the contents** of two Strings. You can use the relational equality operator '==' to compare the references of two objects. Let

us examine the following codes:

```
s1 == s1;         // true, same pointer
s1 == s2;       // true, s1 and s1 share storage in common pool
s1 == s3;         // true, s3 is assigned same pointer as s1
s1.equals(s3);    // true, same contents
s1 == s4;         // false, different pointers
s1.equals(s4);    // true, same contents
s4 == s5;         // false, different pointers in heap
s4.equals(s5);    // true, same contents
```

## Mutable vs Immutable Object

| Mutable | Immutable |
|---|---|
| Fields can be changed after the object creation | Fields cannot be changed after object creation |
| Generally, provides a method to modify the field value | Does not have any method to modify the field value |
| Has Getter and Setter methods | Has only Getter method |
| Example: StringBuilder, java.util.Date | Example: String, Boxed primitive objects like Integer, Long and etc |

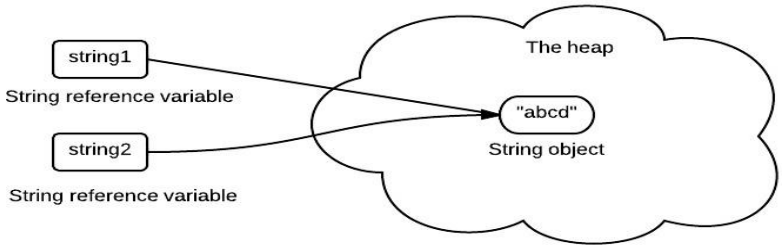
## Why String is Immutable in Java

String pool is a special storage area in Java heap. When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object and returning its reference.

The following code will create only one string object in the heap.

```
String string1 = "abcd";
String string2 = "abcd";
```

Here is how it looks:



If string is not immutable, changing the string with one reference will lead to the wrong value for the other references.

## StringBuffer & StringBuilder

As explained earlier, Strings are immutable because String literals with same content share the same storage in the string common pool. Modifying the content of one String directly may cause adverse side-effects to other Strings sharing the same storage.

JDK provides two classes to support mutable strings: StringBuffer and StringBuilder (in core package java.lang). A StringBuffer or StringBuilder object is just like any ordinary object, which are stored in the heap and not shared, and therefore, can be modified without causing adverse side-effect to other objects.

StringBuilder class was introduced in JDK 5. It is the same as StringBuffer class, except that StringBuilder is not synchronized for multi-thread operations. However, for single-thread program, StringBuilder, without the synchronization overhead, is more efficient.

Methods of string mutable classes are given below:

| **Method** | **Description** |
|---|---|
| length() and capacity() | The length of a mutable string can be calculated using length() method and corresponding capacity can be calculated using capacity(). |
| append(String str)<br>append(int number) | This method is used for adding new text at the end of an existing mutable string. |
| insert(int index, String str)<br>insert(int index, char ch) | Used for inserting text at a specified position in a given string. In the given syntax index specifies the starting index of at which the string will be inserted. |
| reverse() | Used for reversing the order of character in a given string. |
| delete(int start, int end) and deleteCharAt(int index) | Used for deleting characters from a mutable string. Start indicates the starting index of the first character to be removed and the end index indicates an index of one past the last character to be removed. |
| replace(int startindex, int endindex, String str) | Used for replacing character sequence between startindex and endindex-1 with the specified string. |

## What is Scanner Class in Java

JDK 1.5 introduces **java.util.Scanner class**, which greatly simplifies formatted text input

from input source (e.g., files, keyboard, network). Scanner, as the name implied, is a simple text scanner which can parse the input text into **primitive types** and **strings** using regular expressions. It first breaks the text input into **tokens** using a **delimiter pattern**, which is by **default the white spaces** (blank, tab and newline). The tokens may then be converted into primitive values of different types using the various nextXxx() methods (nextInt(), nextByte(), nextShort(), nextLong(), nextFloat(), nextDouble(), nextBoolean(), next() for String, and nextLine() for an input line). You can also use the hasNextXxx() methods to check for the availability of a desired input.

## User Input through Scanner

To create an object of Scanner class, we usually pass the predefined object **System.in**, which represents the standard input stream.

N.B: We may pass an object of class File if we want to read input from a file.

Let us examine the following example:

```
import java.util.Scanner;

public class ScannerDemo1{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);

        String name = sc.nextLine(); // String input
        char gender = sc.next().charAt(0); // Character input
        int age = sc.nextInt(); // Integer input
        long mobileNo = sc.nextLong(); // Long input
        double cgpa = sc.nextDouble(); // Double input

        System.out.println("Name: "+ name);
        System.out.println("Gender: "+ gender);
        System.out.println("Age: "+ age);
        System.out.println("Mobile Number: "+ mobileNo);
        System.out.println("CGPA: "+ cgpa);
    }
}
```

## Exercises

1. Write a Java program to find the length of a given string. The string is “Hello Java”.
2. Write a Java program to concatenate a given string to the end of another string. First String is “This is a simple text”, and the second string is “concatenated string”.
3. Write a Java program to check whether two strings are equal or not.
4. Write a Java program to reverse a string. (take user input using scanner).