**Interpreter:**

Java interpreter is a computer program (system software) that implements the JVM. It is responsible for reading and executing the program. It is designed in such a way that it can read the source program and translate the source code instruction by instruction. **Interpreter in** Java is a computer program that converts high-level program statement into Assembly Level Language.

**Features of an Interpreter in Java**

* The interpreter converts the source code line-by-line during the RUN Time
* You can execute and evaluate a program while execution
* Less amount of time is spent on analyzing and processing the program
* When compared to a compiler, the program execution speed is slower
* An interpreter does not generate an intermediate machine code
* Each error of every line is displayed one by one

## ****Interpreter vs Compiler:****

|  |  |
| --- | --- |
| Interpreter | Compiler |
| Translates Program Line by Line | Translates entire program together |
| Compile-time is Less but Execution is Slower | Compile-time is More but Execution is Faster |
| Will not generate Intermediate Object Code | Generates Intermediate Object Code |
| Program is Compiled until an Error is found | Error is displayed at the end of Compilation |
| Python, PHP, Perl, Ruby use Interpreter. | C,C++,Scala,Java use Compilers |

**String:**

 string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java)

of characters works same as Java string.

**Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

**CharSequence Interface:**

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.

The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

There are two ways to create String object:

By string literal

By new keyword

**String Literal:**

Java String literal is created by using double quotes. Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

**Syntax:**

String s = “Welcome”;

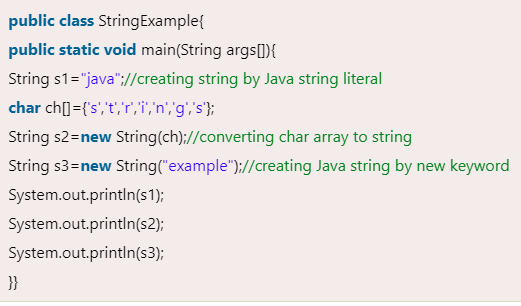
**New Keyword:**

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

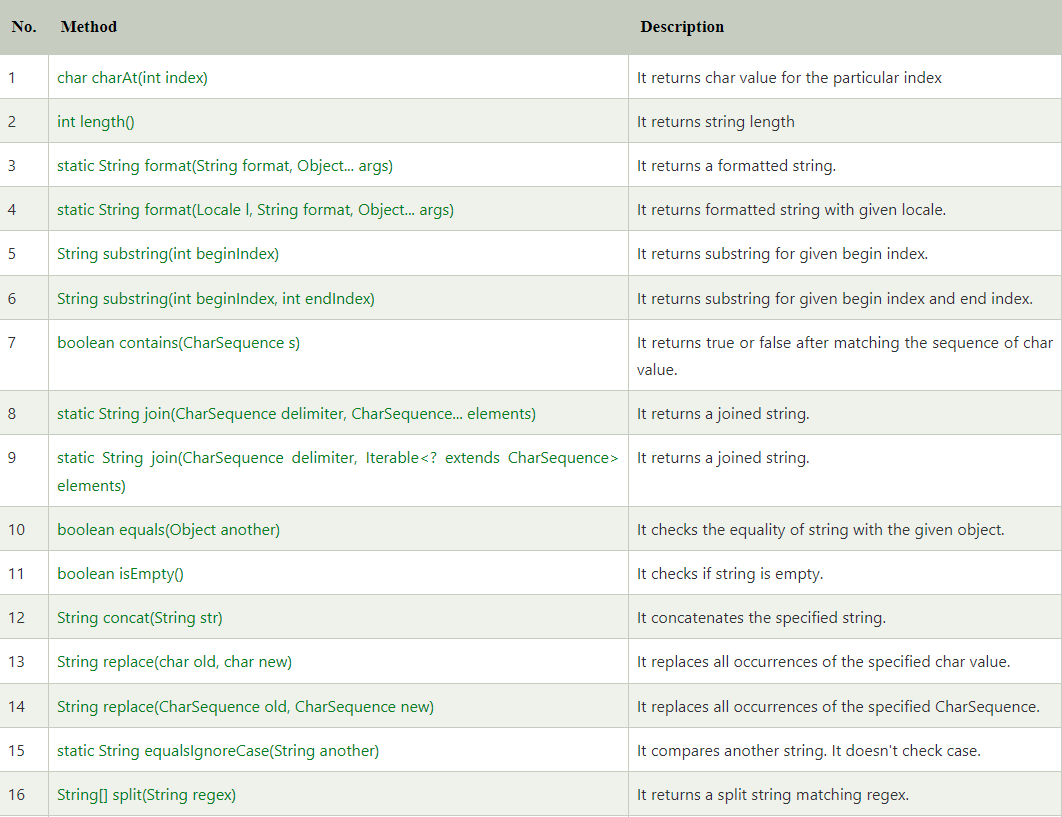
**Syntax:**

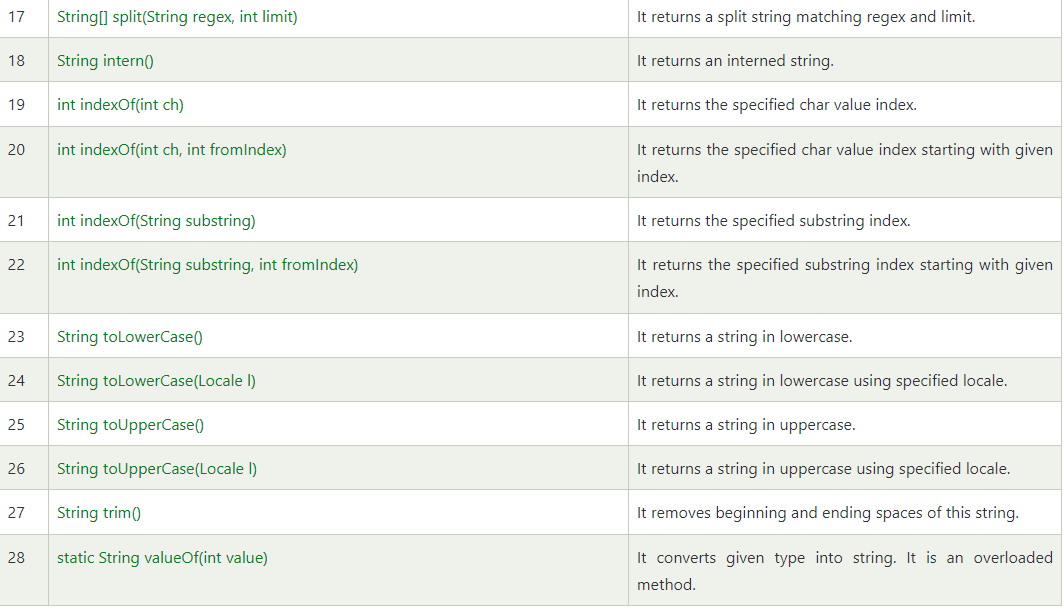
String s = new String(“Welcome”);

**String example program:**

****

**String class methods**:





**Constructors:**

**String(byte[] byte\_arr)** – Construct a new String by decoding the byte array. It uses the platform’s default character set for decoding.

**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

String s\_byte =new String(b\_arr); //Geeks

**String(byte[] byte\_arr, Charset char\_set)** – Construct a new String by decoding the byte array. It uses the char\_set for decoding.

**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

Charset cs = Charset.defaultCharset();

String s\_byte\_char = new String(b\_arr, cs); //Geeks

**String(byte[] byte\_arr, String char\_set\_name)** – Construct a new String by decoding the byte array. It uses the char\_set\_name for decoding.  
It looks similar to the above constructs and they appear before similar functions but it takes the String(which contains char\_set\_name) as parameter while the above constructor takes CharSet.  
**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, "US-ASCII"); //Geeks

**String(byte[] byte\_arr, int start\_index, int length)** – Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number ofcharacters from starting location).  
**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, 1, 3); // eek

**String(byte[] byte\_arr, int start\_index, int length, Charset char\_set)** – Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number of characters from starting location).Uses char\_set for decoding.  
**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

Charset cs = Charset.defaultCharset();

String s = new String(b\_arr, 1, 3, cs); // eek

**String(byte[] byte\_arr, int start\_index, int length, String char\_set\_name)**– Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number of characters from starting location).Uses char\_set\_name for decoding.

**Example:**

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, 1, 4, "US-ASCII"); // eeks

**String(char[] char\_arr)** – Allocates a new String from the given Character array

**Example:**

char char\_arr[] = {'G', 'e', 'e', 'k', 's'};

String s = new String(char\_arr); //Geeks

**String(char[] char\_array, int start\_index, int count)** – Allocates a String from a given character array but choose count characters from the start\_index.  
**Example:**

char char\_arr[] = {'G', 'e', 'e', 'k', 's'};

String s = new String(char\_arr , 1, 3); //eek

**String(int[] uni\_code\_points, int offset, int count)** – Allocates a String from a uni\_code\_array but choose count characters from the start\_index.  
**Example:**

int[] uni\_code = {71, 101, 101, 107, 115};

String s = new String(uni\_code, 1, 3); //eek

**String(StringBuffer s\_buffer)** – Allocates a new string from the string in s\_buffer  
**Example:**

StringBuffer s\_buffer = new StringBuffer("Geeks");

String s = new String(s\_buffer); //Geeks

**String(StringBuilder s\_builder)** – Allocates a new string from the string in *s\_builder*  
**Example:**

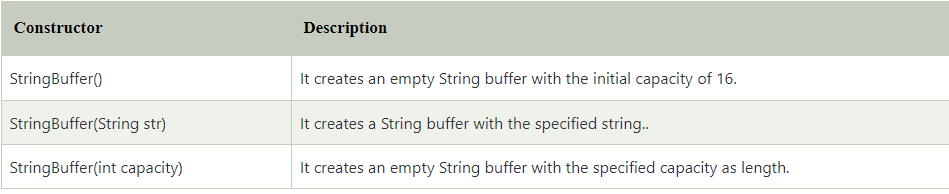
StringBuilder s\_builder = new StringBuilder("Geeks");

String s = new String(s\_builder); //Geeks

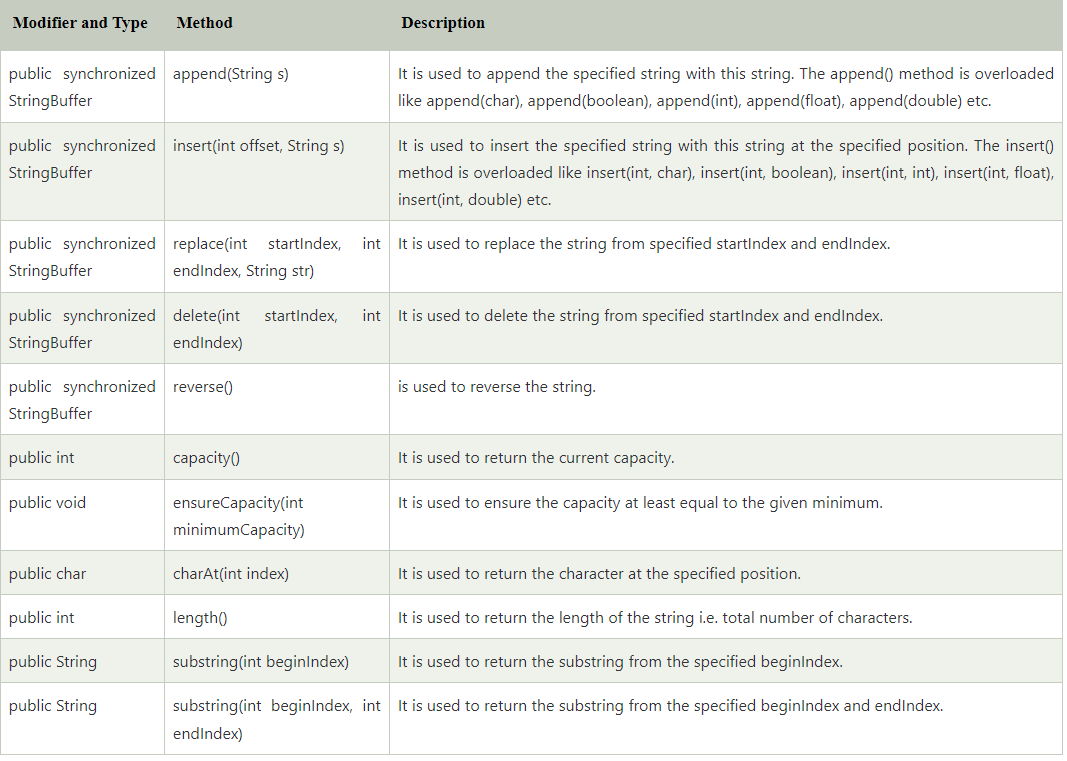
**StringBuffer:**

Java StringBuffer class is used to create mutable (modifiable) String objects.  The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

**Constructor of StringBuffer class:**



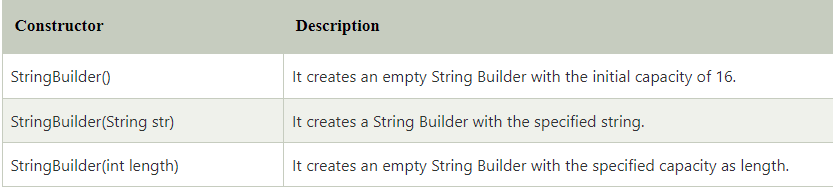
**Method of StringBuffer class:**



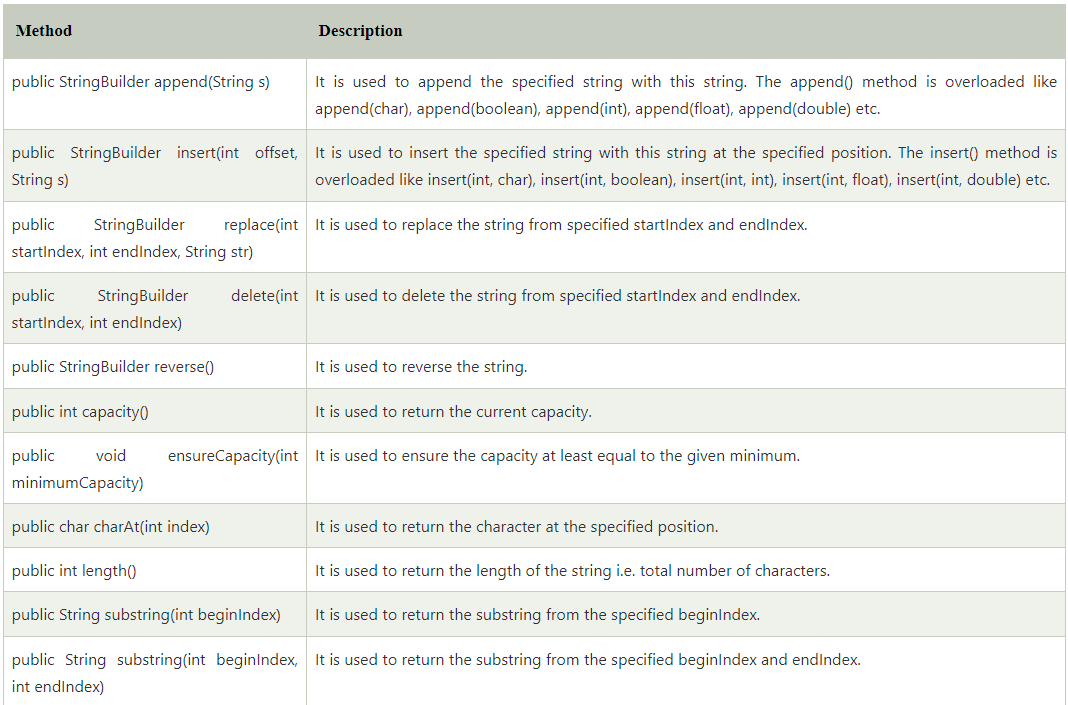
**StringBuilder class:**

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

**Constructors of StringBuilder class**:

****

**Methods of StringBuilder class:**

****





**Difference between StringBuffer and StringBuilder**:

