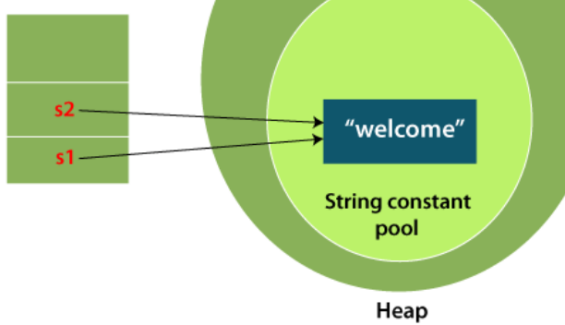
String in java is immuatable

A String in Java is actually an object

Java String literal is created by using double quotes

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance  due to memory effcient

it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.



objects are stored in a special memory area known as the "string constant pool".

Methods

* **length()**
* toUpperCase() **&** toLowerCase()
* **indexOf()**
* concat()

The String class trim() method eliminates white spaces before and after the String.

**CharAt Method**

String myStr = "Hello";

char result = myStr.charAt(0);

System.out.println(result);

**Find out if a string contains a sequence of characters:**

String myStr = "Hello";

System.out.println(myStr.contains("Hel")); // true

System.out.println(myStr.contains("e")); // true

System.out.println(myStr.contains("Hi")); // false

isEmpty() Method

Find out if a string is empty or not:

String myStr1 = "Hello";

String myStr2 = "";

System.out.println(myStr1.isEmpty());

System.out.println(myStr2.isEmpty());

replace & replaceAll Method

Returns a new string where all "l" characters are replaced with "p" characters:

String myStr = "Hello";

System.out.println(myStr.replace('l', 'p'));

**split()**

Splits a string into an array of substrings

And returns String[].

public class Main {

public static void main(String[] args) {

String myStr = "Split a string by spaces, and also punctuation.";

String[] myArray = myStr.split(“ “);

for (String s : myArray) {

System.out.println(s); } }

}

Split by space

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

The + operator can be used between strings to combine them. This is called **concatenation**:

String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

String firstName = "John ";

String lastName = "Doe";

System.out.println(firstName.concat(lastName));

|  |  |  |
| --- | --- | --- |
| **Escape character** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

# Java StringBuffer Class & StringBuilder Class

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.

#### Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

class StringBufferExample{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello ");

**sb.append("Java");//now original string is changed**

**sb.insert(1,"Java");//now original string is changed**

**insert(int offset, String s)**

**replace(int startIndex, int endIndex, String str)**

**delete(int startIndex, int endIndex)**

**reverse()**

**capacity()**

**Other Normal string methods**

System.out.println(sb);//prints Hello Java

}

}

The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized

|  |  |
| --- | --- |
| String class uses String constant pool. | StringBuffer uses Heap memory |

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is less efficient than StringBuilder. | StringBuilder is more efficient than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

# How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members

The ****java.util.StringTokenizer**** class allows you to break a String into tokens. It is simple way to break a String.

Use Split() as StringTokenizer is deprecated.

**String to int**

  String str1 = "5";

**int** result = Integer.parseInt(str1); // Using Integer.parsrInt()

        System.out.println(result);

**Int to String**

**int** x = 5;

        //1st way

        String str = Integer.toString(x); // using Integer.toString()

        System.out.println(str);

        //2nd way

        String str2 = String.valueOf(x); // using String.valueOf()

        System.out.println(str2);

Java Regex

The **Java Regex** or Regular Expression is an API to *define a pattern for searching or manipulating strings*.

It is widely used to define the constraint on strings such as password and email validation.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

java.util.regex package

The Matcher and Pattern classes provide the facility of Java regular expression. The java.util.regex package provides following classes and interfaces for regular expressions.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

## **Matcher class**

It implements the **MatchResult** interface. It is a regex engine which is used to perform match operations on a character sequence.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | boolean matches() | test whether the regular expression matches the pattern. |
| 2 | boolean find() | finds the next expression that matches the pattern. |
| 3 | boolean find(int start) | finds the next expression that matches the pattern from the given start number. |
| 4 | String group() | returns the matched subsequence. |
| 5 | int start() | returns the starting index of the matched subsequence. |
| 6 | int end() | returns the ending index of the matched subsequence. |
| 7 | int groupCount() | returns the total number of the matched subsequence. |

## **Pattern class**

It is the compiled version of a regular expression. It is used to define a pattern for the regex engine.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | static Pattern compile(String regex) | compiles the given regex and returns the instance of the Pattern. |
| 2 | Matcher matcher(CharSequence input) | creates a matcher that matches the given input with the pattern. |
| 3 | static boolean matches(String regex, CharSequence input) | It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern. |
| 4 | String[] split(CharSequence input) | splits the given input string around matches of given pattern. |
| 5 | String pattern() | returns the regex pattern. |

### **Example of Java Regular Expressions**

There are three ways to write the regex example in Java.

1. **import** java.util.regex.\*;

**public** **class** RegexExample1{

**public** **static** **void** main(String args[]){

//1st way

Pattern p = Pattern.compile(".s");//. represents single character

Matcher m = p.matcher("as");

**boolean** b = m.matches();

//2nd way

**boolean** b2=Pattern.compile(".s").matcher("as").matches();

//3rd way

**boolean** b3 = Pattern.matches(".s", "as");

System.out.println(b+" "+b2+" "+b3);

}}

## **Regex Character classes**

|  |  |  |
| --- | --- | --- |
| **No.** | **Character Class** | **Description** |
| 1 | [abc] | a, b, or c (simple class) |
| 2 | [^abc] | Any character except a, b, or c (negation) |
| 3 | [a-zA-Z] | a through z or A through Z, inclusive (range) |
| 4 | [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| 5 | [a-z&&[def]] | d, e, or f (intersection) |
| 6 | [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| 7 | [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |

## **Regular Expression Question 1**

1. /\*Create a regular expression that accepts alphanumeric characters only.

Its length must be six characters long only.\*/

**import** java.util.regex.\*;

**class** RegexExample6{

**public** **static** **void** main(String args[]){

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun32"));//true

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32"));//false (more than 6 char)

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2"));//true

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun$2"));//false ($ is not matched)

}}

1. All string methods and code it.
2. Escape charater
3. String builder immutable and non-syncronized -> non thread safe.