Util Package

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

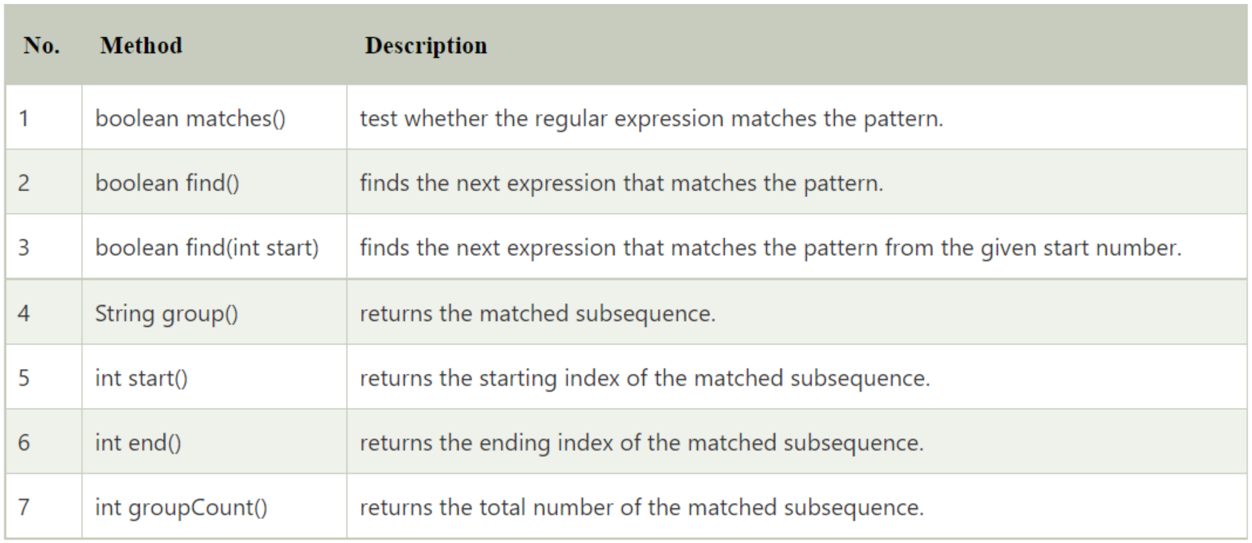
It is widely used to define the constraint on strings such as password and email validation. After learning Java regex tutorial, you will be able to test your regular expressions by the Java Regex Tester Tool.

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

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

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.

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)

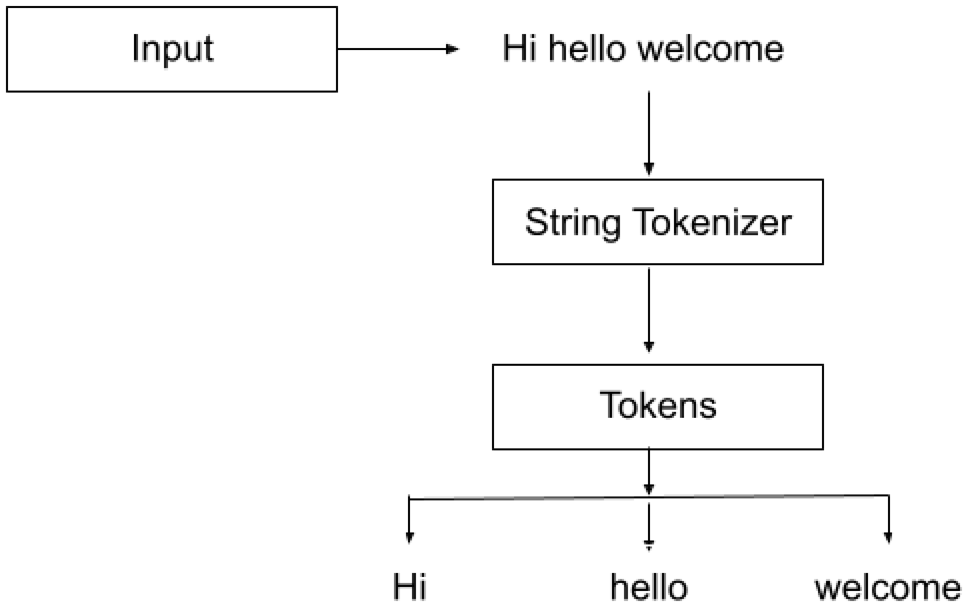
```
import java.util.regex.*;
class RegexExample4{
public static void main(String args[]){
System.out.println("? quantifier ....");
System.out.println(Pattern.matches("[amn]?", "a"));//true (a or m or n comes one time)
System.out.println(Pattern.matches("[amn]?", "aaa"));//false (a comes more than one
time)
System.out.println(Pattern.matches("[amn]?", "aammmnn"));//false (a m and n comes
more than one time)
System.out.println(Pattern.matches("[amn]?", "aazzta"));//false (a comes more than one
time)
System.out.println(Pattern.matches("[amn]?", "am"));//false (a or m or n must come one
time)

System.out.println("+ quantifier ....");
System.out.println(Pattern.matches("[amn]+", "a"));//true (a or m or n once or more
times)
System.out.println(Pattern.matches("[amn]+", "aaa"));//true (a comes more than one
time)
System.out.println(Pattern.matches("[amn]+", "aammmnn"));//true (a or m or n comes
more than once)
System.out.println(Pattern.matches("[amn]+", "aazzta"));//false (z and t are not matching
pattern)

System.out.println("* quantifier ....");
System.out.println(Pattern.matches("[amn]*", "ammmna"));//true (a or m or n may come
zero or more times)

}}
```

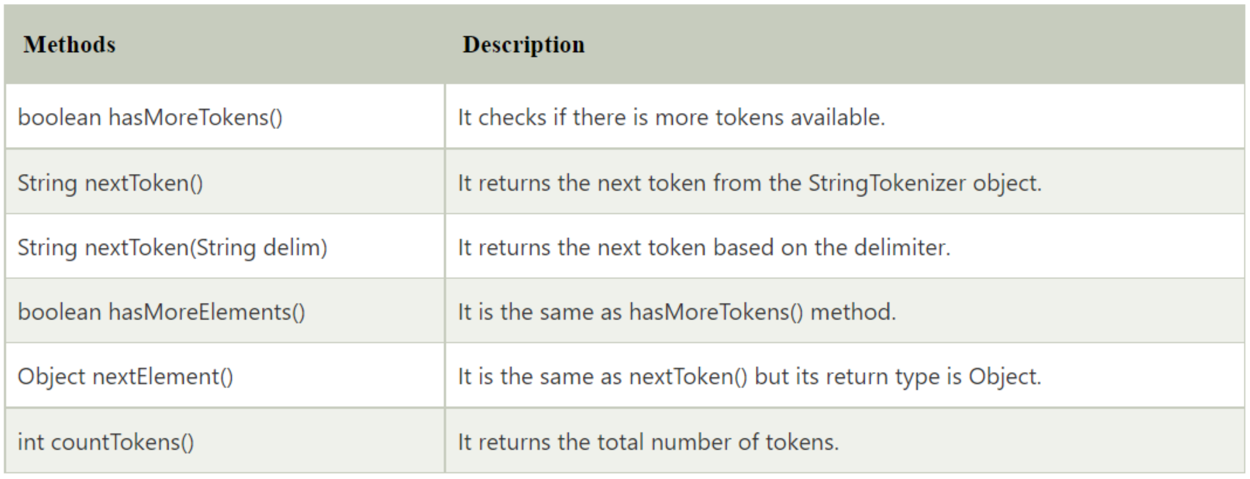
StringTokenizer

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

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.





Methods	Description
boolean hasMoreTokens()	It checks if there is more tokens available.
String nextToken()	It returns the next token from the StringTokenizer object.
String nextToken(String delim)	It returns the next token based on the delimiter.
boolean hasMoreElements()	It is the same as hasMoreTokens() method.
Object nextElement()	It is the same as nextToken() but its return type is Object.
int countTokens()	It returns the total number of tokens.

Date

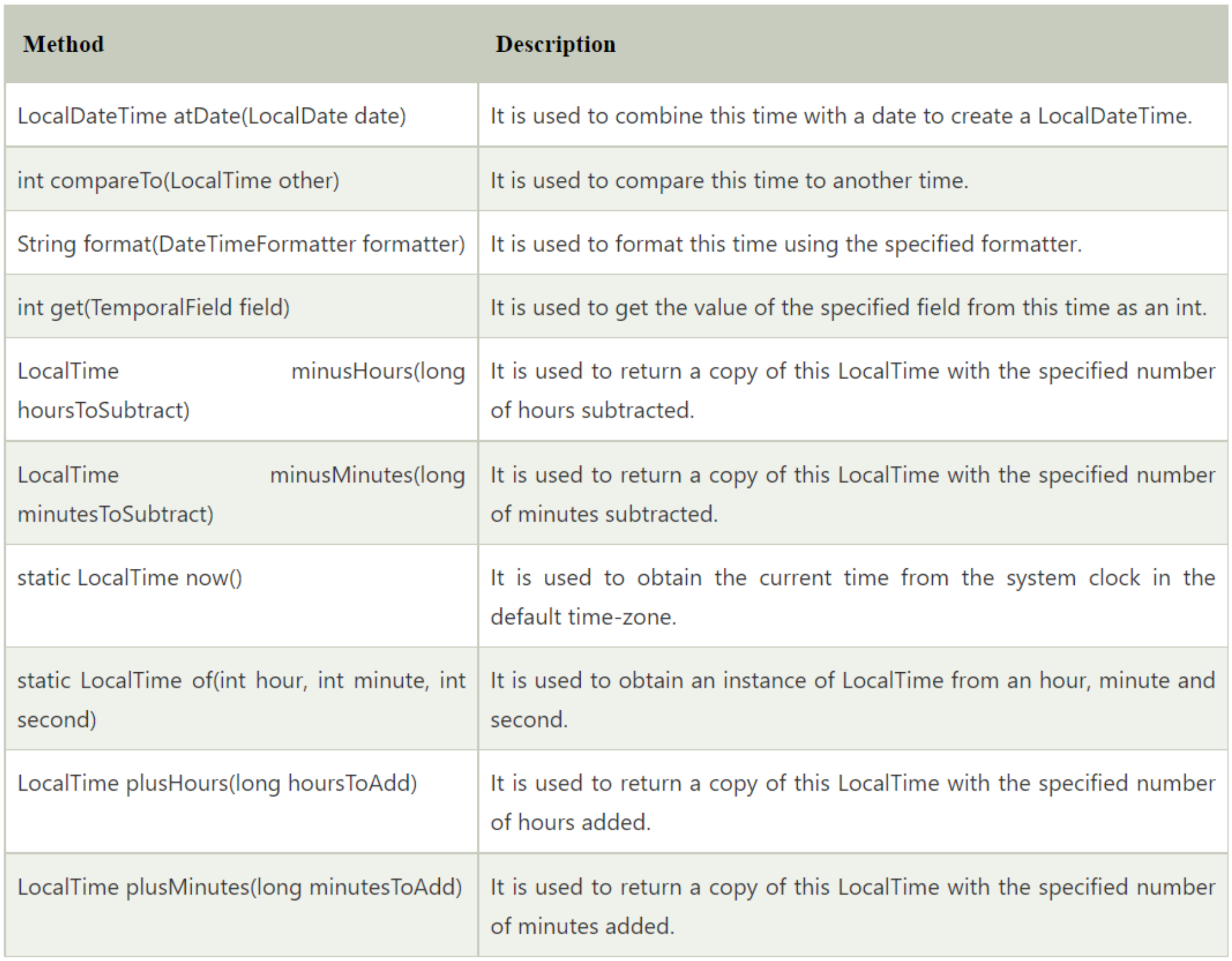
Java LocalDate class is an immutable class that represents Date with a default format of yyyy-mm-dd. It inherits Object class and implements the ChronoLocalDate interface.

LocalDateTime atTime(int hour, int minute)	It is used to combine this date with a time to create a LocalDateTime.
int compareTo(ChronoLocalDate other)	It is used to compares this date to another date.
boolean equals(Object obj)	It is used to check if this date is equal to another date.
String format(DateTimeFormatter formatter)	It is used to format this date using the specified formatter.
int get(TemporalField field)	It is used to get the value of the specified field from this date as an int.
boolean isLeapYear()	It is used to check if the year is a leap year, according to the ISO proleptic calendar system rules.
LocalDate minusDays(long daysToSubtract)	It is used to return a copy of this LocalDate with the specified number of days subtracted.
LocalDate minusMonths(long monthsToSubtract)	It is used to return a copy of this LocalDate with the specified number of months subtracted.
static LocalDate now()	It is used to obtain the current date from the system clock in the default time-zone.
LocalDate plusDays(long daysToAdd)	It is used to return a copy of this LocalDate with the specified number of days added.

```
import java.time.LocalDate;

public class LocalDateExample {
        public static void main(String[] args) {
                LocalDate date = LocalDate.now();
                LocalDate yesterday = date.minusDays(1);
                LocalDate tomorrow = yesterday.plusDays(2);
                System.out.println("Today date: " + date);
                System.out.println("Yesterday date: " + yesterday);
                System.out.println("Tomorrow date: " + tomorrow);

                LocalDate date2 = LocalDate.of(2022, 11, 02);
           System.out.println(date.isLeapYear());
        }
}
```

LocalTime

Java LocalTime class is an immutable class that represents time with a default format of hour-minute-second. It inherits Object class and implements the Comparable interface.



Method	Description
LocalDateTime atDate(LocalDate date)	It is used to combine this time with a date to create a LocalDateTime.
int compareTo(LocalTime other)	It is used to compare this time to another time.
String format(DateTimeFormatter formatter)	It is used to format this time using the specified formatter.
int get(TemporalField field)	It is used to get the value of the specified field from this time as an int.
LocalTime minusHours(long hoursToSubtract)	It is used to return a copy of this LocalTime with the specified number of hours subtracted.
LocalTime minusMinutes(long minutesToSubtract)	It is used to return a copy of this LocalTime with the specified number of minutes subtracted.
static LocalTime now()	It is used to obtain the current time from the system clock in the default time-zone.
static LocalTime of(int hour, int minute, int second)	It is used to obtain an instance of LocalTime from an hour, minute and second.
LocalTime plusHours(long hoursToAdd)	It is used to return a copy of this LocalTime with the specified number of hours added.
LocalTime plusMinutes(long minutesToAdd)	It is used to return a copy of this LocalTime with the specified number of minutes added.

```
import java.time.LocalTime;

public class LocalTimeExample {
        public static void main(String[] args) {
                LocalTime time1 = LocalTime.of(10, 43, 12);
                System.out.println(time1);
                LocalTime time2 = time1.minusHours(2);
                LocalTime time3 = time2.minusMinutes(34);
                System.out.println(time3);
        }
}
```