REGULAR EXPRESSIONS IN JAVA

e-mail: mrdevilbynature@gmail.com

What is a Regular Expression

- a **Regular Expression** (abbreviated as RegEx) is a sequence of characters that forms a search pattern, mainly for use in string matching.

- RegEx is kind of a language within a language.

Basic Knowledge

Metacharacters:

A metacharacter is a character that has a special meaning to a computer program, such as a regular expression engine.

A few metacharacters supported by Java RegEx Engine are:

- \d : Represents a digit.
- \D : Represents a non-digit
- \s : Represents a whitespace.
- \S : Represents a non-whitespace
- \w : Represents a word Character (letters, numbers or an underscore).
- \W : Represents a non-word Character.
- “.” : Represents any character

Structure of a RegEx

The basic structure of a RegEx is bounded by the below mentioned classes.

- Literal escape \x
- Grouping [...]
- Range a-z
- Union [a-e][i-u]
- Intersection [a-z&&[aeiou]]

The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that is in both of its operand classes.

Quantifiers

Used to specify the number of occurrences.

X?	*X*, once or not at all
*X**	*X*, zero or more times
X+	*X*, one or more times
X{*n*}	*X*, exactly *n* times
X{*n*,}	*X*, at least *n* times
X{*n*,*m*}	*X*, at least *n* but not more than *m* times

A Few Examples…

- [abc] Searches for a's or b's or c's.
- \d+ Searches for a Number.
- 0[xX]([0-9a-fA-F])+ Searches for a Hexadecimal Number.
- [a-fn-s] Searches for a character between a to f OR n to s.

Writing Regular Expressions in JAVA

- The Classes that help in implementing RegEx in Java are:

  1. Pattern class
  2. Matcher class

  (Both the classes are present in the java.util.regex package)

The Pattern Class

A regular expression, specified as a string, must first be compiled into an instance of this class.

The resulting pattern can then be used to create a Matcher object that can match arbitrary character sequences against the regular expression.

All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

The Matcher Class

An engine that performs match operations on a character sequence by interpreting a Pattern.

A Matcher object is created from a pattern by invoking the pattern's matcher method.

Once created, a matcher can be used to perform three different kinds of match operations:

- The matches method attempts to match the entire input sequence against the pattern.
- The lookingAt method attempts to match the input sequence, starting at the beginning, against the pattern.
- The find method scans the input sequence looking for the next subsequence that matches the pattern.

  This method starts at the beginning of this matcher's region, or, if a previous invocation of the method was successful and the matcher has not since been reset, at the first character not matched by the previous match.

Each of these methods returns a boolean indicating success or failure.

Implementing the Classes

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaaabababaab");
```

The compile method in the Pattern class is a static method which takes a Regular Expression in the form of a String as the argument

A Matcher object is created by invoking the matcher method of the Pattern class

[The matcher method is a Non-static method, so it requires a Pattern object for its invocation]

Output Options

Once a match has been found by using one of the matches, lookingAt and find method, the following Matcher class methods can be used to display the results:

- public int start() — Returns the start index of the previous match.
- public int end() last — Returns the offset after the character matched.
- public String group() — Returns the input subsequence matched by the previous match.

Implementing the matches method

```
import java.util.regex.*;
public class Regex
{
public static void main(String[] arg)
{
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
if(m.matches())
System.out.println(m.start()+" "+m.end()+" "+m.group());
}
}
```

Output:

```
0  6   aaaaab
```

Implementing the lookingAt method

```
import java.util.regex.*;
public class Regex
{
public static void main(String[] arg)
{
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("b aab");
if(m.lookingAt())
System.out.println(m.start()+" "+m.end()+" "+m.group());
}
}
```

Output:

```
0 1   b
```

Implementing the find method

```
import java.util.regex.*;
public class Regex
{
public static void main(String[] arg)
{
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("ab aabcba");
while(m.find())
System.out.println(m.start()+" "+m.end()+" "+m.group());
}
}
```

Output:

```
0  2    ab
3  6    aab
7  8    b
```

Regular Expressions as Strings

Writing the statement:

String regex="\d";

would produce a COMPILER ERROR!

Whenever the compiler sees a '\' inside a String, it expects the next character to be an escape sequence.
The way to satisfy the compiler is to use double backslashes.

The statement:

String regex="\\d";

is a valid RegEx metacharacter in the form of a String.

Regular Expressions as Strings Contd…

Suppose, if we want the dot (.) to represent its usual meaning while writing a Regular Expression as a String.

String s = "."; would be considered as a legal RegEx metacharacter

Also,

String s="\."; would be considered as an illegal escape sequence by the compiler

The workaround is to use double backslashes :

String s="\\.";

Represents a dot, and not some RegEx metacharacter.

References

- Java api documentation SE6
- SCJP – Kathy Sierra and Bert Bates
- Wikipedia

THANK YOU!