Core Java 8 and Development Tools

Lesson 08 : Regular Expressions

Lesson Objectives

After completing this lesson, participants will be able to:

- Understand concept of Regular Expressions
- Use the java.util.regex package
- Validate input data

8.1: Regular Expressions
Text Processing using Regular Expression

Regular expressions or RegEx is a mechanism of allowing text processing. It is a special text string for performing search, edit, or manipulate text and data.

Regex API is available in the java.util.regex package

The String class in java also allows a regular expression operation with minimal code

- String.replaceAll()
- String.matches()
- String.split()

8.1: Regular Expressions
java.util.regex package

The java.util.regex package primarily consists of the following three classes:

- Pattern
- Matcher
- PatternSyntaxException

8.1: Regular Expressions
Pattern class

java.util.regex.Pattern precompiles regular expressions so they can be executed more efficiently. Example:

- String consisting of ‘a’ in the beginning and ‘b’ in the end with any number of characters in between
  - Pattern pattern = Pattern.compile("a*b");
- Number consisting of one or more digits
  - Pattern pattern = Pattern.compile("(\\d+)");

Some methods of the Pattern class are compile(), matches(), matcher()

8.1: Regular Expressions
Pattern class : Example

```
public class RegExpTest {
    public static void main(String[] args) {
        String inputStr = “Test String”;
        String pattern = “Test String”;
        boolean patternMatched =
                                Pattern.matches(pattern,
inputStr);
        System.out.println(patternMatched);
    }
}
```

```
Output: true
```

8.1: Regular Expressions
Matcher class

java.util.regex.Matcher interprets the pattern and performs match operations against an input string.

It provides a full set of methods to do the scanning.

```
String input = "Shop,Mop,Hopping,Chopping";
Pattern pattern = Pattern.compile("hop");
Matcher matcher = pattern.matcher(input);
System.out.println(matcher.matches());
while (matcher.find()){
    System.out.println(matcher.group() + ": " +matcher.start() + ": " +
matcher.end());
}
```

Displays : false

```
Displays:
hop: 1: 4
hop: 18: 21
```

8.1: Regular Expressions
Regular Expression guide

Construct	Matches
\d	A digit
\D	A non digit
\s	A white space character
\S	A non-whitespace character
^	Beginning of a line
$	The end of a line
.	Any character
*	Any no of characters
\	Escape character

8.1: Regular Expressions
Regular Expression guide

construct	**Matches**
[abc]	a, b, or c
[^abc]	Any character except a, b, or c
[a-zA-Z]	a through z or A through Z, inclusive
[a-d[m-p]]	a through d, or m through p:
[a-z&&[def]]	d, e, or f
[a-z&&[^bc]]	a through z, except for b and c
[a-z&&[^m-p]]	a through z, and not m through p:

8.2: Regular Expressions to validate data
Example

```
public static void validateCode(String args) throws Exception{
    String input = “Exo1";
  //Checks for string that start with upper case alphabet and end with
digit.
     Pattern p = Pattern.compile(“^[A-Z][0-9]&");
     Matcher m = p.matcher(input);
     if (!m.find()) {
        System.err.println(“Enter  code which  start with upper case
alphabet and end with a digit");
     }
   }
```

8.2: Regular Expressions to validate data
Demo : Regular Expression

Execute the RegularExMatcher .java program

Summary

In this lesson, you have learnt the following:

- What are Regular Expressions
- Use the java.util.regex package
- Use regular expressions for manipulating strings

Review Question

Question 1 : To suppress the special meaning of metacharacters, use ____________

Question 2 : This method returns a new Pattern object :

- **Option 1 :** compile()
- **Option 2 :** matches()
- **Option 3** : matcher()