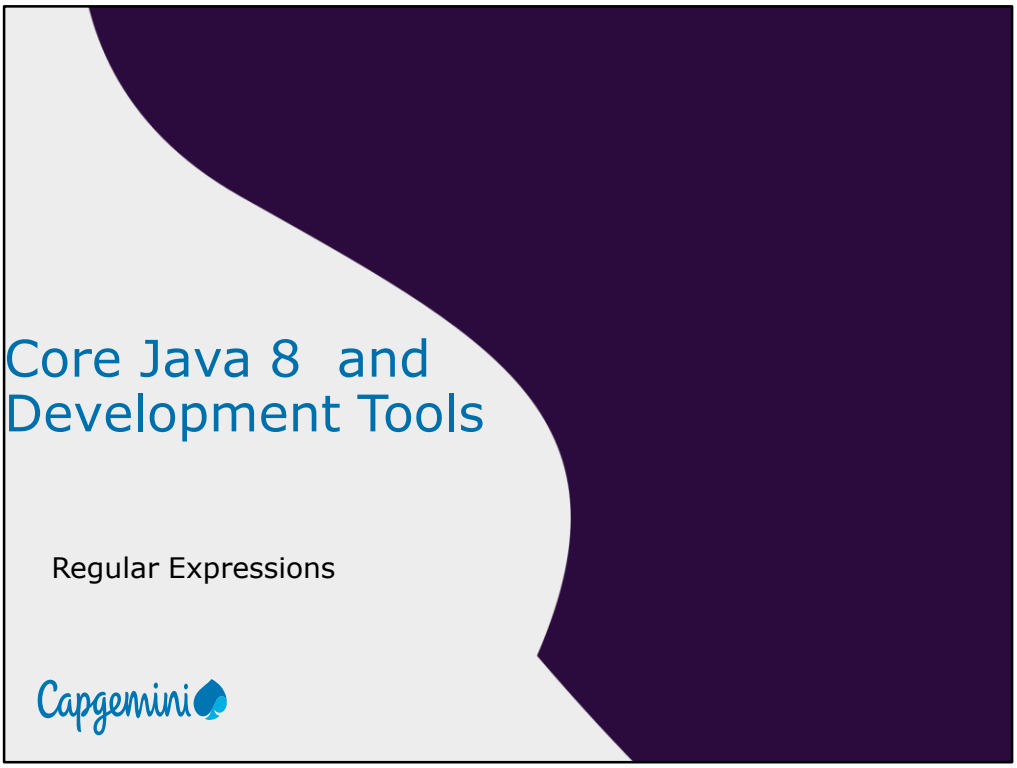
Core Java 8 and
Development Tools
Regular Expressions
Capgemini

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

Lesson Outline:

8.1: Regular Expressions
8.2: Validating data
8.3: Best Practices

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()

Regular Expressions are basically patterns of characters which are used to perform certain useful operations on the given input. The operations include finding particular text, replacing the text with some other text, or validating the given text. For example, we can use Regular Expression to check whether the user input is valid for a field like Email-id or a telephone number.

Supported in most of the common languages like C, Java, Python, C# etc. Support is slightly different for regex in each of these languages though.

The Java String class has several methods that allow you to perform an operation using a regular expression on that string in a minimal amount of code. Some of them are listed above. For example, myString.matches("regex") returns true or false depending whether the string can be matched entirely by the regular expression.

Regular Expressions

java.util.regex package

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

- Pattern
- Matcher
- PatternSyntaxException

The java.util.regex package primarily consists of the following three classes:
Pattern Class: An instance of this class is a compiled representation of a regular expression. This class provides no public constructors. To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object. These methods accept a regular expression as the first argument.
Matcher Class: An instance of this class is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. A Matcher instance is created by invoking the matcher() method on a Pattern object.
PatternSyntaxException: A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern. It has various methods like getDescription(), getIndex(), getMessage() and getPattern() that provide details of the error.

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()

Some methods of the Pattern class:

compile (String regex) : This method returns a new Pattern object which is a compiled form of regular expression pattern. Note that compile() is a static method.

matcher (String input) : This method is used to create new Matcher object for an input for a given pattern, which can be used to perform matching operations.

matches (pattern, inputSequence) : This method returns true only if the entire input text matches the pattern. This method internally depends on the compile() and matcher() methods of the Pattern object. Note that matches() is a static method.

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

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

Please refer to Javadocs for details about the methods of this class.
In the code snippet above, the first output returns false since the entire input string " Shop,Mop,Hopping,Chopping " does not match the regular expression pattern “hop" and hence matches() method returns false.

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

This API supports a number of special characters (metacharacters) that affect the way a pattern is matched.
To treat a metacharacter as an ordinary character, precede it with a backslash (\)

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:

Please browse through the Pattern class specification in Javadocs. There are tables summarizing the supported regular expression constructs.

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");
     }
  }
```

Regular Expressions to validate data

Demo : Regular Expression

Execute the RegularExMatcher .java program

Best practises
Lab

No separate lab for this but included this assignment in layered architecture.

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()