COMP 6481: Programming and Problem Solving

Tutorial 05:

Regular Expression

What is Regular Expression in JAVA?

- A Regular Expression (regex) is a special sequence of characters that specifies a search pattern in text
- In Java is an API to define a pattern for search, edit, or manipulating strings
- Java Regex API provides 1 interface and 3 classes in java.util.regex package

java.util.regex package

- Regex in Java provides 3 classes and 1 interface
  - Pattern Class
  - Matcher Class
  - PatternSyntaxException Class
  - MatchResult Interface

java.util.regex package

- **Pattern** Class - Used as a compiled representation of a regular expression.It provides no public constructors.
- **Matcher** Class - Used as an engine which interprets the pattern and also performs match operations against an input string.
- **PatternSyntaxException** - Used in indicating a syntax error in a regular expression pattern.

Pattern and Matcher class

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

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

The Matcher Class

- Matcher Class performs match operations on a character sequence.
  - **boolean matches()**: tests the regular expression which matches the pattern
  - **boolean find()**: finds the next expression which matches the pattern
  - **boolean find(int start)**: finds the next expression which matches the pattern from given the start number
  - **String group()**: returns the matched subsequence
  - **int start()**: returns the starting index of the matched subsequence.
  - **int end():** returns the ending index of the matched subsequence
  - **int groupCount():** returns the total number of the matched subsequences

Complete Example

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
  public static void main(String[] args) {
    Pattern pattern = Pattern.compile("COMP", Pattern.CASE_INSENSITIVE);
    Matcher matcher = pattern.matcher("This is COMP 6481");
    boolean matchFound = matcher.find();
    if(matchFound) {
      System.out.println("Match found");
    } else {
      System.out.println("Match not found");
    }
  }
}
```

Regular Expression Syntax

The first parameter of the Pattern.compile() method is the pattern. It describes what is being searched for.

- “” → Exact match
- [] → Matches any single character e.g. [a-z] → a to z
- && → AND e.g. [a-c && x-z] → a to c and x to z
- ^ → NOT OR String begins with e.g. [^ab] → any single character except a or b OR “^Hello” → String begins with
- $ → end of line “Hello$” → String ends with Hello
- * → 0 or more occurrences e.g. a* → a or aa or aaa etc.
- \ → an escape character e.g. → \n, \t etc.

Predefined Character Classes

- .   ->   Any character
- \d ->   A digit: [0-9]
- \D ->   A non-digit: [^0-9]
- \s ->   A whitespace character: [\t\n\x0B\f\r]
- \S ->   A non-whitespace character: [^\s]
- \w ->   A word character: [a-zA-Z_0-9]
- \W ->   A non-word character: [^\w]

- Pattern.matches(“\\d”, “7”)
- Pattern.matches(“\\w”, “k”)
- Pattern.matches(“.p” , “ap”)

Regex Quantifiers

Regex	Description
X?	X occurs once or not at all
X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

Back References

Back References are regular expression commands which refer to previous matched regular expressions.

- Example - “(\d)\1{2}”.
- Java Example -
  - Pattern.*matches*("(\\d)\\1{2}", "111") -> true
  - Pattern.*matches*("(\\d)\\1{2}", "122") -> false
  - Pattern.*matches*("(\\d)\\1{2}", "222") -> true
- Here \d represents a subexpression and \1 is a back reference which means exact subexpression is expected after the actual subexpression, and {2} represents it is needed exactly twice.

Question 1: Get the Output for the following matches

```
► System.out.println("? quantifier ....");
  System.out.println(Pattern.matches("[ajn]?", "a"));
  System.out.println(Pattern.matches("[ajn]?", "aaa"));
  System.out.println(Pattern.matches("[ajn]?", "aammmnn"));
  System.out.println(Pattern.matches("[ajn]?", "aazzta"));
  System.out.println(Pattern.matches("[ajn]?", "aj"));
```

Q1 Continue

- System.out.println("+ quantifier ....");

```
System.out.println(Pattern.matches("[ajn]+", "a"))
System.out.println(Pattern.matches("[ajn]+", "aaa"));
System.out.println(Pattern.matches("[ajn]+","aammmnn"));
System.out.println(Pattern.matches("[ajn]+","aazzta"));
```

- System.out.println("* quantifier ....");

```
System.out.println(Pattern.matches("[ajn]*", "ajjjna"));
```

Regex Exercises

- ➢ Create a regex that accepts 10 digit numeric characters that start with 5 ,6 or 7.
- ➢ Create a regex that accepts alphanumeric characters only with length of 8.
- ➢ Create a regex expression that check if a string is exactly ‘true’, ‘True’, ‘Yes’ or ‘yes’.
- ➢ Create a regex that check if a text contains ‘sam’ or ‘sung’ or both.
- ➢ What would the regex [a-z&&[d-f]] capture?
- ➢ What would the regex [a-z&&[^abc]] capture?

Regex Exercises

- Create a Regular expression that find quoted expression - single quoted(‘...’) or double quoted(“...”).
- Write a regular expression which checks if text is an email or not.