Java Regex

Lab Setup

1) Add the provided code to a Java project called, JavaRegex

   The provided program asks the user to enter a regular expression as a String. It then compiles the regular expression into a Pattern and allows the user to enter any number of Strings. If a provided String matches the Pattern, it will print a message saying so. Otherwise, it will print a message stating that it does not match the provided regular expression.

   The user can also enter the "-exit-" to end the program or "-new-" to compile a different regular expression.

2) The provided code does not currently compile. We will add some code throughout the lab.

Regular expressions

A regular expression is a sequence of characters that specifies a pattern that a set of strings follow.

The most basic regular expression is a string literal. The string "apple" is a regular expression for the set of strings which exactly match "apple" a.k.a the set containing only the word "apple".

A character class (surrounded by square brackets) denotes legal characters at a position.

"[bcp]at" matches the set of strings "bat", "cat", and "pat" because the first character in the string can be either 'b', 'c', or 'p'.

"[^bcp]at" matches the set of strings ending in "at" which do not begin with 'b', 'c', or 'p'.

"`[a-zA-Z]`" matches the set of single upper or lower-case letter strings.

"`"[a-zA-Z]*"`" will match any word surrounded by quotation marks.

Symbols used for other purposes in regex, such as { and \, need to be escaped using the backslash ('\'). For example, "\{" will match the string { and "\\" will match the string \.

A period matches to any character so to match it literally we need to escape it as well.

Pattern class

A `Pattern` is a compiled representation of a regular expression. The regular expression is passed as a String to a method `compile`. You can find its specification and supported constructs in the Java documentation (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/regex/Pattern.html)

1. A typical invocation sequence is

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

   A Pattern for the regular expression "a*b" is compiled. This regex matches strings which begin with any number of 'a's and end with 'b'.

2. In the provided code, the user is asked to enter a String representing a regular expression.

   We can compile that String into a Pattern using the line above.
   ```
   Pattern p = Pattern.compile(regex);
   ```

3. Later the user can enter another regular expression, so we can compile that expression in the same way.
   ```
   Pattern p = Pattern.compile(regex);
   ```

4. Then when matching the String to the Pattern we can use the Matcher class and its method `matches()`
   ```
   Matcher m = p.matcher(regex);
   ```

Graded Submission Task 1:

Write a regular expression which matches ODU student emails. You can write your regex for just the beginning MIDAS ID portion or include the "@odu.edu". Test your regular expression by compiling a Pattern and matching some student emails you may know. When you are confident with your regular expression, add a menu to the program with two options. Option 1 will run the program as normal (as described above). Option 2 will set the variable `regex` to be your regular expression for ODU student emails and allow the user to enter emails to test against your regular expression. It should ouput whether a given email matches the regular expression or not

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Lab wrap-up

In this lab we discussed regular expressions in Java. Regex is implemented in Java with the Pattern and Matcher classes. Pattern represents a compiled regular expression and Matcher returns whether a String matches the regular expression. We included an introduction to writing a regular expression to match ODU student emails.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder “Lab19_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics
Task 1 (points allotted) - Compilation (2) - Menu implemented (2) - Given regex reasonably matches ODU student emails (6)

Appendix

This section will include some useful code snippets.

Parsing a String using a regex

Assume we want to parse a String for all occurrences of a particular pattern. In this example, we are searching a nucleic acid sequence for the α-amino acid Serine. We use the Pattern and Matcher class to build a map which maps String to Integer, i.e., the subsequence mapped to its first index in the sequence.

```
// Sequence we are searching
String sequence = "GGTGAGGGTATCATCCCATCTGACTACACCTCATCGGAGACGGAGCAGT";
// Regex pattern for Serine
/*
  Serine is a 3 character String beginning with TC and
  ending with one of T, C, A, or G
*/
String pattern = "TC[TCAG]";
Pattern serine = Pattern.compile(pattern);
Matcher serMatcher = serine.matcher(sequence);
HashMap<String, Integer> seqMap = new HashMap<>();

// Find all matching subsequences and build the map
while (serMatcher.find()) {
    String seq = sequence.substring(serMatcher.start(), serMatcher.end());
    seqMap.put(seq, serMatcher.start());
}
```

The `find()` method returns true if it finds a substring matching the given Pattern. If `find()` returns true, we can use the methods `start()` and `end()` to find the first index and last index of the matching substring.