CSC2620 Object Oriented Design — Spring 2024
Unit 8 - Regular Expressions

8.1 Regular Expressions

- Regular expressions are used to perform *pattern matching*, where you match strings and substrings to some pattern you have supplied.
- There are two categories of characters in patterns:
  1. Normal characters (match themselves)
  2. Metacharacters (can have special meanings in patterns–do not match themselves)
     `\ | ( ) [ ] { } ^ $ * + ? .`
     - A metacharacter is treated as a normal character if it is backslashed
     - A period is a special metacharacter - it matches any character except newline

8.1.1 Regular Expression Syntax

- Character classes
  - Put a sequence of characters in brackets, and it defines a set of characters, any one of which matches
    `[abcd]`
  - Dashes can be used to specify spans of characters in a class
    `[a-z]`
  - A caret at the left end of a class definition means the opposite (does not include the following characters)
    `[^0-9]`
  - Character class abbreviations

Abbr.	**Equiv. Pattern**	**Matches**
`\d`	`[0-9]`	a digit
`\D`	`[^0-9]`	not a digit
`\w`	`[A-Za-z_0-9]`	a word character
`\W`	`[^A-Za-z_0-9]`	not a word character
`\s`	`[ \r\t\n\f]`	a whitespace character
`\S`	`[^ \r\t\n\f]`	not a whitespace character

  - How would you create a regular expression for the following?
    1. Any digit, followed by a period, followed by two digits.
    2. Any character that is not a "word character" followed by white space and then a digit.

- Quantifiers
  - Quantifiers enable you to repeat a part of a pattern
  - Quantifiers in braces

Quantifier	**Meaning**
`{n}`	exactly `n` repetitions
`{m,}`	at least `m` repetitions
`{m, n}`	at least `m` but not more than `n` repetitions

  - The braces come after the pattern to be repeated:
    `xyyyz` matches `"xy{3}z"` and `"xy{1,4}z"`, for example
  - Other quantifiers (just abbreviations for the most commonly used quantifiers)
    - `*` means zero or more repetitions
      e.g., `\d*` means zero or more digits
    - `+` means one or more repetitions
      e.g., `\d+` means one or more digits
    - `?` Means zero or one
      e.g., `\d?` means zero or one digit

- Anchors
  - The pattern can be forced to match only at the left end with `^`; at the end with `$`
    e.g., `"^Lee"` matches "Lee Ann" but not "Mary Lee Ann"
    `"Lee Ann$"` matches "Mary Lee Ann", but not "Mary Lee Ann is nice"
- Pattern modifiers
  - The `?i:` modifier tells the matcher to ignore the case of letters
    `"?i:oak"` matches "OAK" and "Oak" and ...
  - The `x` modifier tells the matcher to ignore whitespace in the pattern (allows comments in patterns)

8.1.2 Pattern Matching Methods on Strings

- The `String` method `matches` takes in a regular expression and returns `true` if the string it's called on matches the given regular expression:

```
String str = "Gluckenheimer";
System.out.print(str.matches("^Gluck(.)*"); // prints true
System.out.print(str.matches("Gluck"));      // prints false
```

- `split(String regex)` returns an array of tokens, pieces of the original string, broken up and separated by `regex`. We'll do more with this method later.
- `replaceFirst` and `replaceAll` also work when the first argument is a regular expression.

8.1.3 The Matcher Approach

- You can use a `Pattern` and a `Matcher` object to do some more powerful work with regular expressions.

```
  /**
  * A method demonstrating the use of matcher, described in
section 4.6.3
  */
 public static void matcherTest( ) {
   // String to be scanned to find the pattern.
   String line = "This is a weighty search string";
   // The regex to use (what substrings of line does it match?)
   String pattern = "[aeiou]{2}";

   // Create a Pattern object from the regex - must be compiled
   Pattern r = Pattern.compile(pattern);

   // Now create matcher object from the pattern
   //   (call .matcher on your pattern)
   Matcher m = r.matcher(line);

   // Find all instances of pattern in line and report them
   while (m.find( )) {
       // The 0 group uses the entire regular expression
       System.out.println("Found value: " + m.group(0) );
   }
}
```

For more information, check out:
https://www.tutorialspoint.com/java/java_regular_expressions.htm

8.2 Capturing Groups

- See here for tutorial on **capturing groups**
- **Capturing groups** are a way to treat multiple characters as a single unit, created by placing characters to be grouped inside a set of parentheses.
- For example, the regular expression `(dog)` creates a single group consisting of 'd', 'o', and 'g'
- Capturing gorups are *numbered* from left to right by counting the left parenthesis
  - Let's take a look at the example `((A)(B(C)))`. There are four groups:
    1. ((A)(B(C)))
    2. (A)
    3. (B(C))
    4. (C)
  - Start a new numbering at each left parenthesis
  - The `groupCount` method of the `Matcher` object retunrs an `int` with how many total groups exist in a given `Pattern`
- Groups are numbered starting at 0
  - The `Matcher` object's `group` method takes in a number for the group and returns the string matching that group.
  - `group(0)` is reserved for the entire string

```java
public class RegexMatches {

   public static void main( String args[] ) {
      // String to be scanned to find the pattern.
      String line = "This order was placed for QT3000! OK?";
      String pattern = "(.*)(\\d+)(.*)";

      // Create a Pattern object
      Pattern r = Pattern.compile(pattern);

      // Now create matcher object.
      Matcher m = r.matcher(line);
      if (m.find( )) {
         System.out.println("Found value: " + m.group(0) );
         System.out.println("Found value: " + m.group(1) );
         System.out.println("Found value: " + m.group(2) );
      }else {
         System.out.println("NO MATCH");
      }
   }
}
```

This code produces:

```
Found value: This order was placed for QT3000! OK?
Found value: This order was placed for QT300
Found value: 0
```

8.3 Practice

1. Create a regular expression to match each of the following scenarios:
   - You want to determine if a string is a phone number.
   - You want to determine if a string is a first name, middle initial, and last name.
   - You want to determine if a string is a complete sentence, starting with a capital and ending with a punctuation mark.
2. Open a new Java project and create and test a method called `indexOf(String searchString, String regex)`
   - The first argument is a search string
   - The second argument is a regular expression
   - The method's job is to return the index of the first instance of a pattern matching the regular expression in the search string. For instance, the call `indexOf("This is a search string", "[aeiou]{2}");` should return 11 (why?).

8.4 More Practice

1. Create a new Java project.
2. Download `words.txt` and add it to your project. Use regular expressions to determine the number of words in the file that:
   - end in 'ing'
   - contain three vowels in a row
   - contain exactly one vowel
   - start and end with the same letter
3. Create a method that takes in a full sentence as an input argument and returns an array list of all the abbreviations found in the sentence. Note that, if the last word of the sentence is an abbreviation, the sentence will end with two periods.