SIENAcollege
Computer Science

Computer Science 225
Advanced Programming
Siena College
Spring 2017

Topic Notes: Regular Expressions

Our next brief topic is on pattern matching using regular expressions.

Pattern Matching

You've done simple *pattern matching* with methods like `String.indexOf`, where you can find the first occurrence of a text pattern in another.

```
String text = "Advanced programming!!";
System.out.println(text.indexOf("gram"));
```

This would perform a search for an exact pattern match. A more general problem is to search for numbers, or words of a certain length. These are the kinds of searches that are facilitated by using *regular expressions*. This very common term is often abbreviated as "regex" or "RE."

Those of you who have studied or are studying theory of computation would know that regular expressions are string sequences based on regular languages. Here, we are not concerned with the theory so much, but how we can use regular expressions in Java to perform pattern matching.

Java's `regex` Package

There are two classes in Java that we will look at a bit today, and that you'll work with some more in lab this week: `java.util.regex.Pattern` and `java.util.regex.Matcher`.

The idea is that a `Pattern` object represents a regular expression, and can generate a `Matcher` object that can be used to find patterns that match.

We will experiment with a series of examples that look like this:

```
String regex = "\\d\\d";
String text = "2-digit numbers 32 8378 hey!";

Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(text);

while (m.find()) {
  System.out.println(m.group());
}
```

The above demonstrates only a subset of the functionality of these classes. See the API documentation for more.

See the code, augmented with extensive comments:

See Example: RegexPractice

Specifying Regular Expressions in Java

As we see in the example, the regular expression is specified with special characters called *metacharacters* that allow the regular expression to match with classes of characters. Above, the regular expression contains the `\d` metacharacter twice. This means we wish to match two consecutive decimal digits.

The metacharacters are written with a double backslash inside a Java `String` literal, since it would otherwise be treated as a special character (remember how you need `\"` to print a `"` character with `System.out.println`?).

`\d` is just one of many metacharacters supported by Java's regex system. Here are a the other predefined character classes:

- `\D` – any non-digit character
- `\w` – any letter, number, or underscore
- `\W` – anything other than a letter, number, or underscore
- `\s` – any whitespace character
- `\S` – any non-whitespace character
- `.` (a period) – any character

We can also specify sets or ranges of characters to match. `[abcd]` or `[a-d]` would each match any character a, b, c, or d.

Like with the original example, we can place these specifiers next to each other and require a sequence of matching characters. `[a-f]\\d\\s` would match a pattern of any one character a through f, followed by a digit, followed by a whitespace character.

Patterns following `^` and `$` will match only the beginning and end of the line, respectively.

Placing other characters after a pattern will allow for repetition, or a greedy matching:

- `*` – 0 or more repetitions of preceding
- `+` – 1 or more repetitions of preceding
- `{x,y}` – between `x` and `y` more repetitions of preceding

- `?` – 0 or 1 repetitions of preceding

Options can be specified with `|`, and groups can be formed by placing part of the pattern in parentheses.

Many more options can be found in the `Pattern` Java API documentation.

Java API Documentation: Pattern at
`http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html`

An example that uses the METAL graph data, reporting on graph vertices that look like interstate highway labels:

See Example: FindInterstates