Lab 7: Regular Expressions
12:00 PM, Mar 4, 2020

Contents

1 Introduction 1

2 Fun with RegEx (The Mystery Continues...) 2

3 Regexes in Java 2
   3.1 A Regex REPL (REPLex?) 3

4 Debugging (Part 2) 4

Objectives

By the end of this lab, you will know:

- How regular expressions are useful
- How to make and use regular expressions

1 Introduction

A regular expression (regex for short) is a string of text that describes a pattern to search for. The pattern defined by the regex, depending on the regex and the string to perform the search on, may match once, several times, or not at all. They are powerful tools for finding particular words or series of characters in strings, and are widely used in search engines, text editors, and much more. They will come in very handy in Search, your next project!

In the course website, we have a detailed regex guide on how regex syntax works. You can find it here. The guide and the lab slides will help you throughout this lab.

Task: Read through our guide on regular expressions (link above), up to but not including the Java or Scala sections.

Task: Now that you’re a bit more familiar with the syntax of regular expressions, go ahead and write down (on paper) what you think the following regexes would match:

- “A+H+”
- “(h[ae])*”
- “.*\..*”
Another useful tool for creating and testing regular expressions is the Internet! We recommend using regex.com, debuggex.com, or regex101.com. There you can input a regex and see what your regex matches to in a sample text.

Task: Try out our regexes from the last task using one of these websites, and correct your answers for the last task if needed.

Now that you have learned about what regexes are and how they are useful, let’s put them to use!

2 Fun with RegEx (The Mystery Continues. . . )

You’re working in the MSLab when your CS 18 TA Ian walks in, looking harried. He doesn’t seem to notice you working in the corner and he sits down at mslab3d, plugging in a flash drive and logging in. After a lot of typing, he seems frustrated and looks around, finally seeing you, the resident CIT detective. He asks you for your help, but refuses to let you see the details of what he’s working on, insisting that it’s secret.

Ian has a giant text file to sift through, but he doesn’t want to do it all by hand (he has too much important research to do). Luckily, he knows what he is searching for, but he needs your help coming up with the regexes to search with!

Task: Open up a blank text file. Create a regex for each of the following searches and write it in the file. Then, using either regex101.com, debuggex.com, or regex101.com, test the regex you have created and make sure it is matching with the correct strings.

- All space separated words.
- All words that start with a capital letter.
- Any capitalization of the word train (ie. TRAIN, train, tRAIn, TrAiN, etc.)
- All words of length 4 (Note: 4 word characters)
- All words inside quotes (Hint: think escaped characters!)

As his (now improved, thanks to you) search is running, the computer starts to smell of smoke, and the flash drive suddenly bursts into flames. You gasp and jump back, but Ian quickly kicks the flash drive to the ground and stomps it out before the smoke detector can even go off. You’re shocked, but Ian seems completely unfazed. He tells you that it’s not surprising, considering the MSLab fire of 2018. You’ve heard legends of it, but always thought it was just a story that was just for fun. You ask Ian for more information about it (was it real? who did it? what happened?), but he just picks up his smoldering flash drive and walks out, offering you one question as he leaves: have you ever wondered what happened to mslab3e?

You’ve reached a checkpoint! Please call over a lab TA to review your work.

3 Regexes in Java

Java already has two implemented classes that make it helpful to deal with regexes, Patterns and Matchers.
We can use Patterns to see if a regex matches to a particular string using `matches(String regex, CharSequence input)`. However, this only checks if the entire input matches the regex. If we’re looking for all matches that a regex has within a string, we’ll need to use Matchers. Matchers are a way to iterate through an input string, find if matches exist, and find what those matches are.

For this lab, you might find the following functions useful. Our regular expressions guide has more information on how to use them!

**Pattern:**
- `static Pattern compile(String regex)`
- `Matcher matcher(CharSequence input)`
- `static boolean matches(String regex, CharSequence input)`

**Matcher:**
- `boolean find()`
- `String group()`

**Task:** Go back to our regex guide and read the Java section to learn about how to put regexes to use in a Java program.

For the following tasks, create a class called `RegexMatcher`. In it you will implement the following methods.

**Task:** Write a static method called `regexifier` that takes in two Strings. The method should return the result (a space separated String) of the second String applied as a regex to the first String. For example, if the regex returned matches on some strings A, B, C, and D, the method would return the concatenation of A, B, C, and D with spaces separating each. The method signature should look like the following:

```java
public static String regexifier(String input, String regex)
```

**Hint:** Make sure there’s no trailing space, else our provided tests (info below) won’t pass! Try using the Java String `trim()` method to deal with this.

**Note:** Before attempting to make a `Pattern` or `Matcher` out of the input Strings, check to ensure that both inputs are not `null`. If either of them are `null`, return an empty String!

**Task:** We have provided a test file for you; go ahead and copy it over at this time:

```
cp /course/cs0180/sol/lab07/sol/* ~/course/cs0180/workspace/javaproject/sol/lab07/sol
```

**Task:** Run the test file to ensure your `regexifier` method works.

**Task:** Now, create a static method called `brownEmails` that takes in a String and finds brown.edu email addresses that are separated by word boundaries. This method should return a String of all the matching addresses, separated by one space between each. For this problem, a Brown email address contains any number (at least 1) of names separated by underscores, followed by a group of zero or more numbers and then “@brown.edu”, where a name is a group of lowercase letters and dashes.

**Hint:** Remember, a word boundary is represented by \b in regexes!

**Hint:** Use `regexifier` as a helper!

**Task:** Uncomment the tests for `brownEmails`, and run the tests to ensure your implementation works!
You’ve reached a checkpoint! Please call over a lab TA to review your work.

3.1 A Regex REPL (REPLex?)

Now that you’re familiar with using regular expressions in Java programs, let’s use what we have learned in previous labs to make a REPL!

As a Brown student, you’re probably familiar with our course codes. These are, to name a few, “CSCI0180”, “COLT1411B”, “MCM0750A”, and “PHP0030”. In this final task involving regular expressions, you will write a REPL where a user will write any number of things, and you will print out all the course codes that were included in their input, even if the course codes are not space separated!

**Task:** Create a new class called Courses. In it, write a REPL that reads from System.in and then prints out (with spaces separating them) all the course codes contained in the line. This should terminate on EOF.

You should write the REPL in a method called `run`, which takes no arguments. You should write a separate helper method, `findCourseCodes`, which takes user input and returns the course codes found in the input. In the main method, you should simply call `run()`.

**Hint:** Courses consist of 3 to 4 capital letters, followed by 4 numbers, with an optional capital letter at the end!

**Hint:** If you’d like to use your regexifier method as a helper, you can do so by statically calling it in your Courses class, like this: `RegexMatcher.regexifier(arg1, arg2)`.

**Hint:** Remember, use a BufferedReader! To make one that reads in from System.in, you would do `BufferedReader r = new BufferedReader(new InputStreamReader(System.in));`

**Hint:** Don’t forget try with resources, and printing informative errors if an exception is encountered!

**Task:** Uncomment the `findCourseCodes` tests in our provided test class, and check that it works! Also, test your REPL by running it to ensure its expected user interaction behavior.

4 Debugging (Part 2)

Remember the debugging lab from a few weeks ago? As lines for TA hours get longer and longer, knowing how to use the debugger in IntelliJ is going to become really important and will save everyone a lot of head-banging. However, it seems that many students are still not totally sure how to use the debugger to actually find issues in their code, which is why we are bringing back the debugging lab - Part 2!!!!

**Task:** In this section, we’d like you to figure out (using the debugger) what might be wrong about some implementations of quicksort (yes, this is what you worked on last week). To start, please move the faulty implementations over to your directory from

/course/cs0180/src/lab07/src/*

**Hint:** Refer back to lab04 or the slides from this lab if you don’t remember the different debugging tools

**Hint:** If you don’t remember what quicksort is, refer back to the materials from last week’s lab

Once a lab TA signs off on your work, you’ve finished the lab! Congratulations! Before you leave, make sure both partners have access to the code you’ve just written.
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: https://cs.brown.edu/courses/cs018/feedback.