Interpreter Part 1: Regex

Submit Assignment

Due Wednesday by 11:59pm **Points** 100 **Submitting** a file upload
File Types java **Available** after Aug 31 at 8:30am

In this assignment, you will practice working with regular expressions and JUnit tests to build up an understanding of grammars and JUnit testing.

Additionally, see the **Interpreter Project Overview** and **Interpreter Project Setup** pages for information on the project itself and how to setup your workspace for this assignment.

Submission

You will submit both `Regex.java` and `RegexTests.java`, which should contain your work for all three parts below. In total, there should be 7 regex's each of which least 5 successful (does match) and 5 unsuccessful (doesn't match) test cases. Do not modify the names of fields in the `Regex` class since they are used in the JUnit tests. You should retest your regex's with the provided test class prior to submitting.

Your test cases should make a reasonable attempt to test the different areas of the regular expression. For example, with the email regex example testing only `gmail.com` addresses will not be sufficient - you should try other domains as well as erroneous input like `@gmailcom` (missing dot).

Regex Overview

A Regex (REGular EXpression) defines a grammar that can be used to pattern match strings. There are many flavors of regex, and some include features such as backreferrencing which is not actually a regular grammar. In this assignment we're going to focus on the main features of regex which are standard across nearly all implementations.

When testing a regex (really anything for that matter), a good strategy is to break it up into pieces and test one piece at a time. For example, the regex `#?[a-z]+[1-9]*` can be broken up into three concatenated pieces: `#?`, `[a-z]+`, and `[1-9]*`. You can then write test cases for each of these individually, taking into account what characters are allowed and how many times they can be repeated. Using the railroad diagram viewer (linked below) is a good way to see what paths in your regex exist.

Some useful links for working with regex:

- **regexr** **(https://regexr.com/)** : Realtime regex matching + reference sheet

- **regexper** (https://regexper.com/#(%3F%3ARailroad%7CRegex) : Railroad diagram visualization of regex's

JUnit Overview

JUnit is a testing framework for Java. In this class, we'll be using JUnit 5 (Jupiter) which includes a couple new features that will be helpful to us - in particular, parameterized tests.

To run tests, either click the run icon on the left margin when viewing the test file or execute the gradle test task, which can be done by clicking the Gradle tab in the right sidebar of IntelliJ and navigating to Tasks > verification > test (double click to run).

Additionally, here are two links with info on JUnit - while you can probably get by for now with just copying the examples, you should make sure you have an understanding of what's happening as we'll be using JUnit throughout the class.

- **JUnit5 User Guide** (https://junit.org/junit5/docs/current/user-guide/)
- **JUnit5 Parameterized Tests Tutorial** (https://www.baeldung.com/parameterized-tests-junit-5)

Regex Part 1

First, write test cases for the provided regex which matches an email. Pay close attention to the implementation of the email regex - it may not cover all possible valid emails, and you should take this into account in your tests! While the minimum is 5 successful and 5 unsuccessful cases, part of your grade is based on how effective those cases are at testing the given regex - I highly recommend including more!

- Email Regex: `[A-Za-z0-9._-]+@[A-Za-z0-9-]*\.[a-z]{2,3}`
  - Note: Some characters, like `.`, have special meaning in regex and need to be escaped like `\.`. This requires passing a literal `\` character to the `Pattern`, so in Java this would be written in a string as `"\\."`
  - Examples: `thelegend27@gmail.com`, `otherdomain@ufl.edu`
  - Non-Examples: `missingdot@gmailcom`, `symbols#$%@gmail.com`

Regex Part 2

Next, write regex's and test cases (the methods are provided) for the three problems below. As above, you should have at least 5 successful and 5 unsuccessful test cases.

- File names with the extension `java` or `class`. Your regex should capture the base name of the file into a group `name`. If a file has multiple extensions, like `base.tar.gz`, the base name is just `base` and the extension is `gz` (thus, `tar` is not part of either).
  - Examples: `Regex.tar.java`, `RegexTests.class`
  - Non-Examples: `directory`, `scrippy.py`
- Strings between 10 and 20 characters (inclusive) which have **even** lengths.

  - Examples: `thishas14chars`, `i<3pancakes!`
  - Non-Examples: `6chars`, `i<3pancakes!!`
- A list of comma-separated positive integers surrounded by square brackets `[]`. A single space character is allowed after the comma. Trailing commas are not allowed.
  - Examples: `[]`, `[1]`, `[1,2,3]`
  - Non-Examples: `1,2,3`, `[1 2 3]`, `[1,2,3,]`

Regex Part 3

Finally, write regex's and tests (both the methods and cases) for the following language constructs. These will be used in the next assignment as the basis for lexer tokens. As above, you should have at least 5 successful and 5 unsuccessful test cases.

- Identifiers
  - Contains alphanumeric characters, underscores, and any of `+-*/.:!?<>=`.
  - Cannot begin with a digit.
  - A period on its own (`.`) is not allowed.
  - Examples: `getName`, `is-empty?`, `<=>`
  - Non-Examples: `42=life`, `why,are,there,commas,`
- Numbers
  - Begins with one or more digits, or one of `+-` followed by one or more digits
  - Optionally is followed by a decimal point and one or more digits
  - Examples: `1`, `-1.0`, `007.000`
  - Non-Examples: `1.`, `.5`
- Strings
  - Starts and ends with a double quote (`"`)
  - Supports escape sequences starting with a backslash (`\`) followed by one of `bnrt'"\`.
  - Examples: `""`, `"abc"`, `"Hello,\nWorld!"`
  - Non-Examples: `"unterminated`, `"invalid\escape"`

Provided Code

The following files are provided for this assignment: `Regex.java`, containing constants for regex, and `RegexTests.java`, for tests. As mentioned above, you must keep the structure of the `Regex` class the same since it's used by the JUnit tests.

- Source Files (`src/main/java/plc/interpreter`)
  - **Regex.java**
- Test Files (`src/test/java/plc/interpreter`)
  - **RegexTests.java**