Regular Expression

1. Goals

- The host language does not provide a literal syntax for regular expressions like **`/\w+@\w+\.\w+/`**.
- Avoid to programmatically build regular expressions and avoid runtime overhead of compiling these expressions.
- Add a new literal type, using the syntax known from JavaScript, Ruby, Perl, etc.

2. Parser

- Create a subclass of **`LAAspect`** called **`RegularExpressionAspect`**.
- Define the language extension:

```
RegularExpressionAspect>>advice: aParser
    ^ LAAdvice new
        before: aParser primary;
        parser: ($/ asParser , $/ asParser negate star , $/
asParser) small
```

  - #**`before:`** defines a pointcut into the original grammar, a pointcut into the language model.
  - **`#parser:`** defines the parser to be injected.

3. Test it

- Create a subclass of **`TestCase`** called **`RegularExpressionTest`**.
- On the test class select *Language Aspects* | *Add Aspect* | *RegularExpressionAspect* to use the language extension in the context of this apect.
- Add a test method:

```
RegularExpressionTest>>testRegularExpression
    self assert: (/\w+@\w+\.\w+/ matches: 'renggli@gmail.com')
```

- Test fails (⌘**T**).
- Show decompiled code. **`nil`**, we haven't told the how to compile the parse tree to a host language AST.

4. Compiler

- Add a method to specify compilation:

```
RegularExpressionAspect>>compile: aToken
    ^ (aToken value copyFrom: 2 to: aToken size - 1) asRegex
          lift: aToken
```

  - **`#copyFrom:to:`** removes the slashes
  - **`#asRegex`** parses and compiles the expression
  - **`#lift:`** creates a literal node from the matcher object together with the original token
- Select *Language Aspects* | *Recompile Users* from the context menu on the aspect.

1. Test it

- Test passes (⌘`T`).
- Show decompiled code.

2. Highlight

- Add a method to specify highlighting:

```
RegularExpressionAspect>>highlight: aToken
    ^ aToken -> Color orange
```

- Show test case again.
- Put a `#halt` and show the debugger.

3. Other Examples

- **`LAPathAspect`** and **`LAPathAspectTest`**
  Implement an XPath like query language, that uses Smalltalk blocks as query expressions.
- **`LACrosscuttingTest`**
  Shows the regular expression extension, the XPath extension and Smalltalk code intermixed in a single method.
- Add an ignore case property to the language extension: `/[a-z]+/i`. Change parser and compilation concern.