Java support for regular expressions

Java library classes for regular expressions

- `java.util.regex.Pattern`
- `java.util.regex.Matcher`
- `java.lang.String`

Example

```
import java.util.regex.*;
...
// simple use for a single match
assert "Java".matches("[A-Z][a-z]+");

// more efficient use for multiple matches
Pattern p = Pattern.compile("[A-Z][a-z]+"); // static factory method
Matcher m = p.matcher("Java"); // instance factory method
assert m.matches();
```

Java support for regular expressions

`Pattern` class in a nutshell

- instances of `Pattern` are immutable objects representing regular expressions
- patterns are created from strings by the static factory method
  `static Pattern compile(String regex)`
- patterns can create matchers with the instance factory method
  `Matcher matcher(CharSequence input)`

Remarks

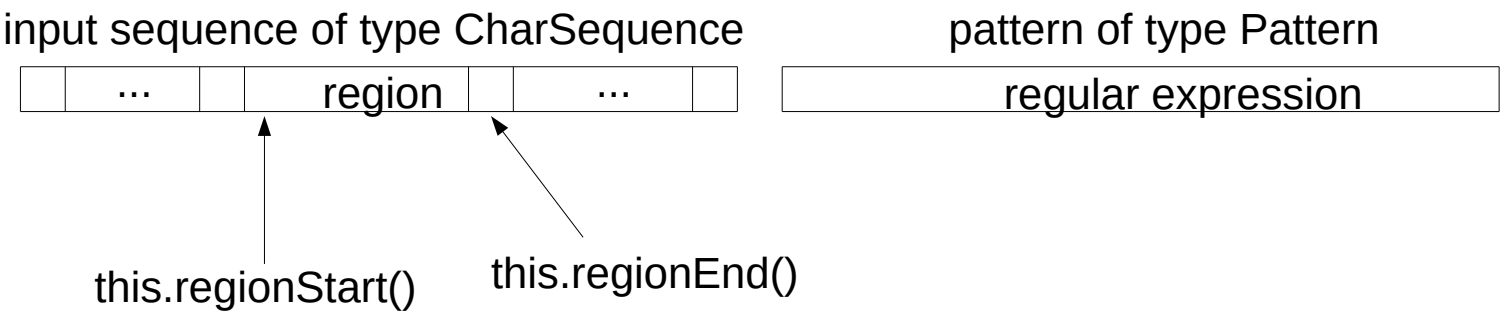
- `compile` may throw `PatternSyntaxException`
- `CharSequence` is an interface defined in `java.lang`
- `String` implements `CharSequence`
- therefore `String`$\leq$`CharSequence`

Java support for regular expressions

`Matcher` class in a nutshell

- a matcher is a mutable object with an input sequence and a pattern
- a matcher works on a subsequence of its input sequence, called region
- the bounds of the region can be modified with
  `Matcher region(int start, int end)`
- the input sequence can be changed with
  `Matcher reset(CharSequence input)`

a matcher object this



Java support for regular expressions

Match operations

- **boolean** `matches()`: tries to match the entire region against the pattern
- **boolean** `lookingAt()`: tries to match a subsequence of the region starting at its beginning
- **boolean** `find()`: tries to find the next subsequence of the input sequence that matches the pattern

Example

```
Pattern pt = Pattern.compile("[A-Z][a-z]+");
Matcher mt = pt.matcher("Java"); // region is the whole input
assert mt.matches();
mt.reset("Java language"); // region is the whole input
assert !mt.matches();
assert mt.lookingAt();
mt.reset("language Java"); // region is the whole input
assert !mt.matches();
assert !mt.lookingAt();
assert mt.find();
```

Java support for regular expressions

Query operations

- `int start()`: returns the start index of the previous match
- `int end()`: returns the index after the last character matched
- `String group()`: returns the string matched by the previous match

Example

```
Pattern pt = Pattern.compile("[A-Z][a-z]+");
Matcher mt = pt.matcher("Java Language");
assert !mt.matches();
assert mt.lookingAt();
assert mt.start() == 0;
assert mt.end() == 4;
assert mt.group().equals("Java");
mt.region(mt.end(), mt.regionEnd()); // moves to " Language" and reset mt
assert !mt.matches();
assert !mt.lookingAt();
assert mt.find();
assert mt.start() == 5;
assert mt.end() == 13;
mt.group().equals("Language");
```

Java support for regular expressions

Remarks on query operations

A query throws `IllegalStateException` if any of the following requirements is not verified:

- a match operation has been called before the query
- the last match operation was successful
- the matcher was not reset after the last match operation

The following instance methods reset the matcher:

- `Matcher reset(CharSequence input)`
- `Matcher region(int start,int end)`

Java support for regular expressions

Interface `java.util.regex.MatchResult`

- the result of the last match operation can be returned with the instance method `MatchResult toMatchResult()`
- the result is unaffected by subsequent operations performed upon this matcher

Example

```
Pattern pt = Pattern.compile("[A-Z][a-z]+");
Matcher mt = pt.matcher("Java Language");
assert mt.lookingAt();
MatchResult res = mt.toMatchResult();
mt.region(res.end(),mt.regionEnd()); // matcher is reset
assert res.start() == 0; // result of the previous query
assert res.end() == 4;
assert res.group().equals("Java");
assert mt.start() == 0; // throws IllegalStateException
```

Java support for regular expressions

Capturing groups

Parentheses force precedence but define also capturing groups

- capturing groups are indexed from left to right, starting from 1
- group 0 is the whole pattern, `mt.group(0)` equivalent to `mt.group()`
- index of a group: the number of open parentheses

Example

```
Pattern pt = Pattern.compile("(0|[1-9]\\d*)([Ll]?)");
Matcher mt = pt.matcher("42L");
mt.lookingAt();
MatchResult res = mt.toMatchResult();
assert res.group(0).equals("42L");
assert res.group(1).equals("42");
assert res.group(2).equals("L");
mt.reset("42");
mt.lookingAt();
res = mt.toMatchResult();
assert res.group(0).equals("42");
assert res.group(1).equals("42");
assert res.group(2).equals("");
```

Java support for regular expressions

A selection of regular-expression constructs in Java

- Logical operators
  - `XY` `X` followed by `Y` (concatenation)
  - `X|Y` Either `X` or `Y` (union)
  - `(X)` `X`, as a capturing group (anyway parentheses force precedence)
- Postfix operators (called greedy quantifiers)
  - `X?` `X`, once or not at all (optionality)
  - `X*` `X`, zero or more times (Kleene star)
  - `X+` `X`, one or more times (Kleene star except the empty string)
- Characters
  - `x` The character `x` (if it is not a special character)
  - `\` Nothing, but quotes the following character
    example: `\\` is the backslash character
  - `\t` The tab character
  - `\n` The newline (line feed) character
  - `\r` The carriage-return character

Java support for regular expressions

A selection of regular-expression constructs in Java

- Character classes
  - `[abc]` `a`, `b`, or `c` (simple class)
  - `[^abc]` Any character except `a`, `b`, or `c` (negation)
  - `[a-zA-Z]` `a` through `z` or `A` through `Z`, inclusive (range)
- Predefined character classes
  - `.` Any character (except line terminators, unless the `DOTALL` flag is specified)
  - `\s` A whitespace character
- Boundary matchers
  - `^` The beginning of a line
  - `$` The end of a line
- Named-capturing and non-capturing
  - `(?<name>X)` `X`, as a named-capturing group
  - `(?:X)` `X`, as a non-capturing group
- See the full documentation in the API documentation

Java support for regular expressions

Remarks on regular expressions and strings

Be careful when using white spaces and special characters in strings!

Example

```
assert " ".matches(" | ");
assert "|".matches("\\|"); // regular expression \|
assert "|".matches("[|]");
```

Java support for regular expressions

`?`, `+` and `*` try to match the longest string, concatenation and `|` do not

Example

```
Pattern pt = Pattern.compile("(\\d\\d\\d)?");
Matcher mt = pt.matcher("234");
assert mt.lookingAt();
assert mt.end() == 3; // longest string matched
pt = Pattern.compile("\\d+");
mt = pt.matcher("234");
assert mt.lookingAt();
assert mt.end() == 3; // longest string matched
pt = Pattern.compile("\\d*");
mt = pt.matcher("234");
assert mt.lookingAt();
assert mt.end() == 3; // longest string matched
pt = Pattern.compile("(\\d\\d)?(\\d\\d\\d)?");
mt = pt.matcher("234");
assert mt.lookingAt();
assert mt.end() == 2; // longest string not matched
pt = Pattern.compile("\\d\\d|\\d\\d\\d");
mt = pt.matcher("234");
assert mt.lookingAt();
assert mt.end() == 2; // longest string not matched
```