# Java

## Regular Expressions

FSR Informatik

October 11, 2016

# Overview

# Class String

String is a Java class representing text.

Strings are immutable. If you use methods to *change* a string you actually get another string.

Javadoc:
`https://docs.oracle.com/javase/7/docs/api/java/lang/String.html`

# String equality - 1

```
1       String x = "hello";
2       String y = "hello";
3       if (x.equals(y)) {
4           System.out.println("x equals y");
5       }
6       if (x == y) {
7           System.out.println("x == y");
8       }
9
```

Obviously x equals y because they are containing the same text. (x == y) is true as well because x and y are references to the same object.

# String equality - 2

```
1       String u = new String("hello");
2       String v = new String("hello");
3       if (u.equals(v)) {
4           System.out.println("u equals v");
5       }
6       if (u == v) {
7           System.out.println("u == v");
8       }
9
```

u also equals v.
But u and v are not references to the same String object.
Therefore (u == v) is false.

# Other Classes

Java offers multiple classes to deal with strings according to your needs.

**StringBuilder** offers extensive append methods to build up a string.

`https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html`

**StringBuffer** is a thread-safe implementation. It offers similar methods to StringBuilder.

`https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html`

# BufferedReader

```
1       import java.io.*;
2
3       public class Input {
4
5           public static void main(String[] args) {
6               try (BufferedReader reader = new BufferedReader(
        new InputStreamReader(System.in))) {
7                   System.out.print("input: ");
8                   String input = reader.readLine();
9                   System.out.println("> " + input);
10              } catch (IOException e) {
11                  e.printStackTrace();
12              }
13          }
14      }
15
```

# BufferedReader - In Detail

In a normal try-catch-finally statement you can use the finally block to close your resources. Resources must be closed after the program is finished.
The try-with-resource statement will close the resource in normal case and in exceptional case. For that the resource must implement the interface *AutoCloseable*.

```
1       try (BufferedReader reader = new BufferedReader(new
        InputStreamReader(System.in))) {
2
3       } catch (IOException e) {
4
5       }
6
```

http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

A **regular expression** is a search pattern for text. An often used short version is **regex**.

You can use regex in many other programming languages and in some text editors.

See `https://en.wikipedia.org/wiki/Regular_expression` and `https://xkcd.com/208/` for more information.

# Using RegEx in Java

You can check if a regex matches a string via:

```
1       String example = "Hello World";
2
3       String regex = "Hello World";
4
5       // this regular expression matches
6       if ( example.matches(regex) ) {
7           System.out.println(regex + " matches " + example);
8       }
9
```

The regex (search pattern) is also a string.

An regex matches an identical string.

# Case sensitive

| regex | string | matches? |
|---|---|---|
| hello world | hello world | yes |
| hello | hello world | no |
| world | hello world | no |
| hello world | hello | no |
| Hello World | hello world | no |

# Dot

The dot `.` matches any charachter.

| regex | string | matches? |
|---|---|---|
| hello | hello | yes |
| hell. | hello | yes |
| hel.o | hello | yes |
| he.o | hello | no |
| he..o | hello | yes |
| ..... | hello | yes |

# Brackets

**[**regex**]** matches **one character** according to the regex inside the brackets.

| regex | string | matches? |
|---|---|---|
| hell[o] | hello | yes |
| hell[aeiou] | hello | yes |
| hell[o][o] | hello | no |
| hell[ap] | hello | no |
| hel[lo][lo] | hello | yes |
| hell[] | hello | ? |

# Brackets

Be careful with your regular expressions in general.

| regex | string | matches? |
|---|---|---|
| hell[] | hello | java.util.regex.PatternSyntaxException |

*Be careful with regex as input.*

# Caret

The caret ^ inside brackets matches the inverse expression.

| regex | string | matches? |
|---|---|---|
| hell[^o] | hello | no |
| hell[^e] | hello | yes |
| hell[^eo] | hello | no |

# Range

You can set up ranges for matching. Be careful with the order. [A-z] is possible but [a-Z] throws an exception.
The order is: digits followed by captial letters followed by small letters.

| regex | string | matches? |
|---|---|---|
| hell[a-e] | hello | no |
| hell[a-z] | hello | yes |
| hell[a-cd-f] | hello | no |
| hell[a-mn-z] | hello | yes |
| hell[a-kg-z] | hello | yes |

# Or

You can combine multiple regex with pipe |.

| regex | string | matches? |
|---|---|---|
| hello\|bye | hello | yes |
| hello\|bye | bye | yes |
| hello\|bye\|ciao | hello | yes |
| hello\|bye | BYE | no |
| hello\|bye | cat | no |

# Meta Characters

| regex | matches |
|---|---|
| \d | a digit |
| \D | a non digit |
| \w | a normal character |
| \W | a non character |
| \s | a whitespace |
| \S | a non whitespace |

You have to escape a backslash in *String*. Therefore you need two backslashs for your meta character.

```
1       String regex = "\\wello"
2       // for the regex: \wello to match "hello";
3
```

# Meta Characters - Examples

| regex | string | matches? |
|---|---|---|
| h\wllo | hello | yes |
| hell\D | hello | yes |
| hell\W | hello | no |
| he\slo | hello | no |
| he\Slo | hello | yes |
| \w | ä | no |
| a\wb | a-b | no |
| a\wb | a_b | yes |

# Quantifiers - 1

You can use +, * and ? as quantifiers to match more than one character.

| regex | matches |
|---|---|
| regex+ | one or more appearances of regex |
| a+ | a, aa, aaa, ... |
| regex* | any appearances of regex |
| a* | "", a, aa, ... |
| regex? | zero or one appearances of regex |
| a? | "", a |

"" means the empty string.

# Quantifiers - 2

You can use braces to quantify occurrences.

| regex | matches |
|---|---|
| regex$\{n\}$<br>a$\{4\}$ | n appearances of regex<br>aaaa |
| regex$\{n, m\}$<br>a$\{2, 4\}$ | at least n and at most m appearances of regex<br>aa, aaa, aaaa |
| regex$\{n, \}$<br>a$\{3, \}$ | at least n appearances of regex<br>aaa, aaaa, aaaaa, ... |

"" means the empty string.

# Quantifiers - Examples

| regex | equivalent regex | string | matches? |
|---|---|---|---|
| hel{1,}o | hel+o | hello | yes |
| hello{0,1} | hello? | hello | yes |
| hellom{0,1} | hellom? | hello | yes |
| hel{0,}o | hel*o | hello | yes |
| hel{2}o | *no equivalent* | hello | yes |

# Start and End

The caret ^ matches the begin and the dollar sign **$** matches the end of a string.

| regex | string | matches? |
|---|---|---|
| hello^ | hello | no |
| hello$ | hello | yes |
| ^hello | hello | yes |
| ^hello$ | hello | yes |

# Groups

Parentheses **()** can be used to group regex.

| regex | string | matches? |
|---|---|---|
| (hello)* | hellohello | yes |
| (hello){2,} | hellohello | yes |
| (hello){2,} | hello | no |
| (he[l]*o)+ | hellohello | yes |

There are more things to know about regular expressions.

`http://docs.oracle.com/javase/tutorial/essential/regex/index.html`