DEVELHOPE

#codeforimpact

Regular Expressions

DEVELHOPE

#codeforimpact

What is a Regular Expression

A *regular expression* (*regex*) is a pattern of characters used to do a search in a string.

Regex are very useful when working with strings and their methods.

Let's see a basic example:

```
import java.util.Arrays;

public class TestingRegex {
     public static void main(String args[]) {
          String s1 = "How are you today?";
          String[] newArray = s1.split("are");     // this is the most basic regex
          System.out.println(Arrays.toString(newArray)); // prints [How ,  you today?]
     }
}
```

Meta Characters

Meta character	Description
.	Period matches any single character except a line break.
[ ]	Character class. Matches any character contained between the square brackets.
[^]	Negated character class. Matches any character that is not contained between the square brackets
*	Matches 0 or more repetitions of the preceding symbol.
+	Matches 1 or more repetitions of the preceding symbol.
?	Makes the preceding symbol optional.

Meta Characters

Meta character	Description
`{n,m}`	Braces. Matches at least "n" but not more than "m" repetitions of the preceding symbol.
`(xyz)`	Character group. Matches the characters xyz in that exact order.
`|`	Alternation. Matches either the characters before or the characters after the symbol.
`\`	Escapes the next character. This allows you to match reserved characters `[ ] ( ) { } . * + ? ^ $ \ |`
`^`	Matches the beginning of the input.
`$`	Matches the end of the input.

Full Stop

The full stop matches any single character except line break.

```
String s1 = "How are you today?";
String[] newArray = s1.split(".day");
System.out.println(Arrays.toString(newArray)); // prints [How are you t, ?]
```

```
String s1 = "How are you today?";
String[] newArray = s1.split("to.");
System.out.println(Arrays.toString(newArray)); // prints [How are you , ay?]
```

Character Sets (Classes)

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class TestingRegex {
    public static void main(String args[]) {

    String s1 = "How are you today?";
    String s2 = "how are you today";

    Pattern p = Pattern.compile("[Hh]ow");

    Matcher m1 = p.matcher(s1);
    System.out.println(m1.find()); // true

    Matcher m2 = p.matcher(s2);
    System.out.println(m2.find()); // true
    }
}
```

The character sets is specified using the square brackets.

The order of the characters inside the sets is not relevant.

To test the regex here we are using the `java.util.regex.Pattern` and `java.util.regex.Matcher`.

This means an uppercase `H` or a lowercase `h` followed by character `o` and `w`.

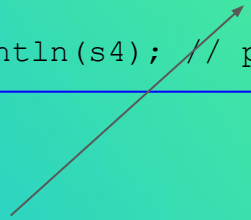
Negated Character Sets

When the caret symbol is after the opening square bracket, it negates the character set.

```
String s3 = "Cow low is the sea level?";

String s4 = s3.replaceAll("[^l]ow", "How");

System.out.println(s4); // prints How low is the sea level?
```



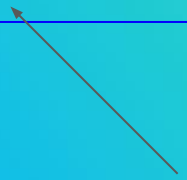
We exclude the `l` character

Repetitions: star *

The * symbol matches **zero or more** repetitions of the preceding matcher.

For example:

`"[a - z]*"`	`Hello how are you???`
`"ba*"`	`bam baaaam baas`
`"\s*java\s*"`	`Well, java is different from javascript`



`\s` indicates a space character

Repetitions: plus +

The + symbol matches **one or more** repetitions of the preceding character.

For example:

`"\s+java\s+"`	`Well, java is different from javascript`
`"j.+script"`	`Well, java is different from javascript`

Repetitions: the question mark ?

Meta character `?` makes the preceding character optional.
It matches zero or one instance of the preceding character.

For example:

`"[M]?ore"`	`More lorem ipsum`
`"[h|s]?w"`	`hw is different from sw`

Braces: `{n,m}`

Braces are quantifiers used to match at least `n` but not more than `m` repetitions of the preceding symbol.

For example:

`"l{3}"`	`Hollywood is correct. Hollllywood is not correct.`
`"l{2}"`	`Hollywood is correct. Hollllywood is not correct.`
`"[ll]{2,3}"`	`Hollywood is correct. Hollllywood is not correct.`

Capturing group `(...)` and `Alternation`

A capturing group is a group of subpatterns in the exact order that is written inside parenthesis.

As we have already seen, alternation `|` is used as a logic `OR`.

For example:

```
"(lll)"          Hollywood is correct. Hollllywood is not correct.

"(wh|th)ere"     where you there? ←  matches and captures the group

"(?:wh|th)ere"   where you there? ←  matches and doesn't capture the
group
```

Escaping special characters

The backslash \ symbol is used for escaping the next reserved character.

These are the reserved characters: `{ } [ ] / \ + * . $ ^ | ?`

For example:

```
"\."          Hollywood is correct.

"\?"          where you there?
```

Anchors: caret ^ and dollar sign $

The caret symbol `^` is used to check if a matching character is the first character of the input string.

The dollar sign `$` is used to check if a matching character is the last character in the string.

For example:

`"^(the)"`	`the yellow bus on the road`
`"(the)$"`	`the yellow bus on the road`
`"(the)$"`	`the yellow bus on the`

Shorthand characters

`.`	Any character except new line
`\w`	Matches alphanumeric characters: `[a-zA-Z0-9_]`
`\W`	Matches non-alphanumeric characters: `[^\w]`
`\d`	Matches digits: `[0-9]`
`\D`	Matches non digits: `[^\d]`
`\s`	Matches whitespace characters: `[\t\n\f\r\p{Z}]`
`\S`	Matches non-whitespace characters: `[^\s]`

Flags

`i`	Case insensitive
`g`	Global search, match all instances, not just the first
`m`	Multiline, the anchor meta characters will work on each line

```
"/the/gi"          The mirror in the room
```

DEVELHOPE

#codeforimpact

Regular Expressions