The Java™ Tutorials

Trail: Essential Classes
Lesson: Regular Expressions

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Predefined Character Classes

The `Pattern` API contains a number of useful *predefined character classes*, which offer convenient shorthands for commonly used regular expressions:

Construct	Description
`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

In the table above, each construct in the left-hand column is shorthand for the character class in the right-hand column. For example, `\d` means a range of digits (0-9), and `\w` means a word character (any lowercase letter, any uppercase letter, the underscore character, or any digit). Use the predefined classes whenever possible. They make your code easier to read and eliminate errors introduced by malformed character classes.

Constructs beginning with a backslash are called *escaped constructs*. We previewed escaped constructs in the String Literals section where we mentioned the use of backslash and `\Q` and `\E` for quotation. If you are using an escaped construct within a string literal, you must precede the backslash with another backslash for the string to compile. For example:

```
private final String REGEX = "\\d"; // a single digit
```

In this example `\d` is the regular expression; the extra backslash is required for the code to compile. The test harness reads the expressions directly from the `Console`, however, so the extra backslash is unnecessary.

The following examples demonstrate the use of predefined character classes.

```
Enter your regex: .
Enter input string to search: @
I found the text "@" starting at index 0 and ending at index 1.

Enter your regex: .
Enter input string to search: 1
I found the text "1" starting at index 0 and ending at index 1.

Enter your regex: .
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \d
Enter input string to search: 1
I found the text "1" starting at index 0 and ending at index 1.

Enter your regex: \d
Enter input string to search: a
No match found.

Enter your regex: \D
Enter input string to search: 1
```

```
No match found.

Enter your regex: \D
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \s
Enter input string to search:
I found the text " " starting at index 0 and ending at index 1.

Enter your regex: \s
Enter input string to search: a
No match found.

Enter your regex: \S
Enter input string to search:
No match found.

Enter your regex: \S
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \w
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \w
Enter input string to search: !
No match found.

Enter your regex: \W
Enter input string to search: a
No match found.

Enter your regex: \W
Enter input string to search: !
I found the text "!" starting at index 0 and ending at index 1.
```

In the first three examples, the regular expression is simply `.` (the "dot" metacharacter) that indicates "any character." Therefore, the match is successful in all three cases (a randomly selected `@` character, a digit, and a letter). The remaining examples each use a single regular expression construct from the Predefined Character Classes table. You can refer to this table to figure out the logic behind each match:

- `\d` matches all digits
- `\s` matches spaces
- `\w` matches word characters

Alternatively, a capital letter means the opposite:

- `\D` matches non-digits
- `\S` matches non-spaces
- `\W` matches non-word characters

Previous page: Character Classes
Next page: Quantifiers