java.util.regex

Interface MatchResult

All Known Implementing Classes:

Matcher

```
public interface MatchResult
```

The result of a match operation.

This interface contains query methods used to determine the results of a match against a regular expression. The match boundaries, groups and group boundaries can be seen but not modified through a `MatchResult`.

Since:

1.5

See Also:

`Matcher`

Method Summary

Methods

Modifier and Type	Method and Description
`int`	`end()` Returns the offset after the last character matched.
`int`	`end(int group)` Returns the offset after the last character of the subsequence captured by the given group during this match.
`String`	`group()` Returns the input subsequence matched by the previous match.
`String`	`group(int group)` Returns the input subsequence captured by the given group during the previous match operation.

`int`	`groupCount()` Returns the number of capturing groups in this match result's pattern.
`int`	`start()` Returns the start index of the match.
`int`	`start(int group)` Returns the start index of the subsequence captured by the given group during this match.

Method Detail

start

`int start()`

Returns the start index of the match.

Returns:

The index of the first character matched

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

start

`int start(int group)`

Returns the start index of the subsequence captured by the given group during this match.

Capturing groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression *m.*`start(0)` is equivalent to *m.*`start()`.

Parameters:

`group` - The index of a capturing group in this matcher's pattern

Returns:

The index of the first character captured by the group, or `-1` if the match was successful but the group itself did not match anything

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

`IndexOutOfBoundsException` - If there is no capturing group in the pattern with the given index

end

```
int end()
```

Returns the offset after the last character matched.

Returns:

@return The offset after the last character matched

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

end

```
int end(int group)
```

Returns the offset after the last character of the subsequence captured by the given group during this match.

Capturing groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression *m.*`end(0)` is equivalent to *m.*`end()`.

Parameters:

`group` - The index of a capturing group in this matcher's pattern

Returns:

The offset after the last character captured by the group, or `-1` if the match was successful but the group itself did not match anything

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

`IndexOutOfBoundsException` - If there is no capturing group in the pattern with the given index

group

```
String group()
```

Returns the input subsequence matched by the previous match.

For a matcher *m* with input sequence *s*, the expressions *m*.`group()` and *s*.`substring(`*m*.`start(),` *m*.`end())` are equivalent.

Note that some patterns, for example `a*`, match the empty string. This method will return the empty string when the pattern successfully matches the empty string in the input.

Returns:

The (possibly empty) subsequence matched by the previous match, in string form

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

group

`String group(int group)`

Returns the input subsequence captured by the given group during the previous match operation.

For a matcher *m*, input sequence *s*, and group index *g*, the expressions *m*.`group(`*g*`)` and *s*.`substring(`*m*.`start(`*g*`),` *m*.`end(`*g*`))` are equivalent.

Capturing groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression `m.group(0)` is equivalent to `m.group()`.

If the match was successful but the group specified failed to match any part of the input sequence, then `null` is returned. Note that some groups, for example `(a*)`, match the empty string. This method will return the empty string when such a group successfully matches the empty string in the input.

Parameters:

`group` - The index of a capturing group in this matcher's pattern

Returns:

The (possibly empty) subsequence captured by the group during the previous match, or `null` if the group failed to match part of the input

Throws:

`IllegalStateException` - If no match has yet been attempted, or if the previous match operation failed

`IndexOutOfBoundsException` - If there is no capturing group in the pattern with the given index

groupCount

```
int groupCount()
```

Returns the number of capturing groups in this match result's pattern.

Group zero denotes the entire pattern by convention. It is not included in this count.

Any non-negative integer smaller than or equal to the value returned by this method is guaranteed to be a valid group index for this matcher.

Returns:

The number of capturing groups in this matcher's pattern

Overview Package Class Use Tree Deprecated Index Help *Java™ Platform Standard Ed. 7*

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Submit a bug or feature
For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.