Regular Expressions in Java

What are Regular Expressions ?

- It is an API for defining String patterns that can be used for searching, manipulating and editing a string in Java.
- Email validation and passwords are few areas of strings where Regex are widely used to define the constraints.
- Regular Expressions are provided under **java.util.regex package**.

java.util.regex Package

This consists of 3 classes and 1 interface :

- MatchResult Interface
- Pattern Class
- Matcher Class
- PatternSyntaxException Class

MatchResult Interface

- Used to determine the result of a match operation for a regular expression.
- It must be noted that although the match boundaries, groups and group boundaries can be seen, the modification is not allowed through a MatchResult.

Methods :

Modifier and Type	Method	Description
int	`end()`	To return the offset after the last character matched.
int	`end(int group)`	To return the offset after the last character of the subsequence captured by the given group during this match.
String	`group()`	To return the input subsequence matched by the previous match.
String	`group(int group)`	To return the input subsequence captured by the given group during the previous match operation.
int	`groupCount()`	To return the number of capturing groups in this match result's pattern.
int	`start()`	To return the start index of the match.
int	`start(int group)`	To return the start index of the subsequence captured by the given group during this match.

1. Pattern Class

- It is a compilation of regular expressions that can be used to define various types of patters, providing no public constructors.
- Created by invoking the `compile()` method which accepts a regular expression as the first argument and returns a pattern after execution.

Basic Syntax:

```
Pattern pattern = Pattern.compile("geeks");
```

Methods :

- `compile(String regex):` - compiles the given regular expression into a pattern.
- `compile(String regex, int flags):` - compiles the given regular expression into a pattern with the given flags.
- `flags():` - return the pattern's match flags.
- `matcher(CharSequence input):` - creates a matcher that will match the given input against the pattern.
- `matches(regex, input):` - compiles the given regular expression and attempts to match the given input against it.
- `pattern():` - returns the regular expression from which this pattern was compiled.
- `quote(String s):` -returns a literal pattern String for the specified String.
- `split(input):` - to split the given input sequence around matches of this pattern.
- `split(input, limit):` - to split the given input sequence around matches of this pattern.
- `toString():` - to return the string representation of the pattern.

Modifier and Type	Method	Description
static Pattern	`compile(String regex)`	To compile the given regular expression into a pattern.
static Pattern	`compile(String regex,int flags)`	To compile the given regular expression into a pattern with given flags.
int	`flags()`	To return this pattern's match flags.
Matcher	`matcher(CharSequence input)`	To create a matcher that will match the given input against this pattern.
static boolean	`matches(String regex, CharSequence input)`	To compile the given regular expression and attempts to match the given input against it.
String	`pattern()`	To return the regular expression from which this pattern was compiled.
static String	`quote(String s)`	To return a literal pattern String for the specified String.
String[]	`split(CharSequence input)`	To split the given input sequence around matches of this pattern.
String[]	`split(CharSequence input,int limit)`	To split the given input sequence around matches of this pattern.
String	`toString()`	To return the string representation of this pattern.

2. Matcher Class

- This object is used to perform match operations for an input string in java, thus interpreting the previously explained patterns.
- This too defines no public constructors and can be implemented by invoking a `matcher()` on any pattern object.

Basic Syntax:

```
// Create a pattern to be searched
Pattern pattern = Pattern.compile("geeks");

// Search above pattern in "geeksforgeeks.org"
Matcher m = pattern.matcher("geeksforgeeks.org");
```

Methods :

Modifier and Type	Method	Description
boolean	`find()`	For searching multiple occurrences of the regular expressions in the text.
boolean	`find(int start)`	For searching occurrences of the regular expressions in the text starting from the given index.
int	`start()`	For getting the start index of a match that is being found using find() method.
int	`end()`	For getting the end index of a match that is being found using find() method. It returns index of character next to last matching character.
int	`groupCount()`	It is used to find the total number of the matched subsequence.
String	`group()`	It is used to find the matched subsequence.
boolean	`matches()`	It is used to test whether the regular expression matches the pattern.

Note: Pattern.matches() checks if the whole text matches with a pattern or not. Other methods are mainly used to find multiple occurrences of pattern in text.

3. PatternSyntaxException class

- This object of Regex is used to indicate a syntax error in a regular expression pattern and is an unchecked exception.

Methods :

Modifier and Type	Method	Description
String	`getDescription()`	To retrieve the description of the error.
int	`getIndex()`	To retrieve the error index.
String	`getMessage()`	To return a multiline string containing description of syntax error and its index, the erroneous regular-expression pattern, and a visual indication of the error index within the pattern.
String	`getPattern()`	To retrieve the erroneous regular-expression pattern.

Important Observations / Facts

1. We create a pattern object by calling Pattern.compile(), there is no constructor. compile() is a static method in Pattern class.
2. Like above, we create a Matcher object using matcher() on objects of Pattern class.
3. Pattern.matches() is also a static method that is used to check if given text as a whole matches pattern or not.
4. find() is used to find multiple occurrences of pattern in text.
5. We can split a text based on a delimiter pattern using split()