Quantifiers in Java

Difficulty Level : Hard Last Updated : 06 Dec, 2021

Prerequisite: Regular Expressions in Java

Quantifiers in Java allow users to specify the number of occurrences to match against. Below are some commonly used quantifiers in Java.

```
X*        Zero or more occurrences of X
X?        Zero or One occurrences of X
X+        One or More occurrences of X
X{n}      Exactly n occurrences of X
X{n, }    At-least n occurrences of X
X{n, m}   Count of occurrences of X is from n to m
```

The above quantifiers can be made Greedy, Reluctant, and Possessive.

Greedy Quantifier (Default)

By default, quantifiers are Greedy. Greedy quantifiers try to match the longest text that matches a given pattern. Greedy quantifiers work by first reading the entire string before trying any match. If the whole text doesn't match, remove the last character and try again, repeating the process until a match is found.

```
// Java program to demonstrate Greedy Quantifiers

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

class Test
{
    public static void main(String[] args)
    {
        // Making an instance of Pattern class
        // By default quantifier "+" is Greedy
        Pattern p = Pattern.compile("g+");

        // Making an instance of Matcher class
        Matcher m = p.matcher("ggg");
```

```
        while (m.find())
            System.out.println("Pattern found from " + m.start() +
                               " to " + (m.end()-1));

    }
}
```

Output

```
Pattern found from 0 to 2
```

Explanation: The pattern **g+** means one or more occurrences of **g**. Text is **ggg**. The greedy matcher would match the longest text even if parts of the matching text also match. In this example, **g** and **gg** also match, but the greedy matcher produces **ggg**.

Reluctant Quantifier (Appending a ? after quantifier)

This quantifier uses the approach that is the opposite of greedy quantifiers. It starts with the first character and processes one character at a time.

```
// Java program to demonstrate Reluctant Quantifiers

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

class Test
{
    public static void main(String[] args)
    {
        // Making an instance of Pattern class
        // Here "+" is a Reluctant quantifier because
        // a "?' is appended after it.
        Pattern p = Pattern.compile("g+?");

        // Making an instance of Matcher class
        Matcher m = p.matcher("ggg");

        while (m.find())
            System.out.println("Pattern found from " + m.start() +
                               " to " + (m.end()-1));
```

```
    }
}
```

Output

```
Pattern found from 0 to 0
Pattern found from 1 to 1
Pattern found from 2 to 2
```

Explanation: Since the quantifier is reluctant, it matches the shortest part of the test with the pattern. It processes one character at a time.

Possessive Quantifier (Appending a + after quantifier)

This quantifier matches as many characters as possible, like a greedy quantifier. But if the entire string doesn't match, then it doesn't try removing characters from the end.

```
// Java program to demonstrate Possessive Quantifiers

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

class Test
{
    public static void main(String[] args)
    {
        // Making an instance of Pattern class
        // Here "+" is a Possessive quantifier because
        // a "+' is appended after it.
        Pattern p = Pattern.compile("g++");

        // Making an instance of Matcher class
        Matcher m = p.matcher("ggg");

        while (m.find())
            System.out.println("Pattern found from " + m.start() +
                               " to " + (m.end()-1));
    }
}
```

Output

```
Pattern found from 0 to 2
```

Explanation: We get the same output as Greedy because the whole text matches the pattern.

Difference Between Greedy and Possessive Quantifiers

```
// Java program to demonstrate difference
// between Possessive and Greedy Quantifiers

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

class Test
{
    public static void main(String[] args)
    {
        // Create a pattern with Greedy quantifier
        Pattern pg = Pattern.compile("g+g");

        // Create same pattern with possessive quantifier
        Pattern pp = Pattern.compile("g++g");

        System.out.println("Using Greedy Quantifier");
        Matcher mg = pg.matcher("ggg");
        while (mg.find())
            System.out.println("Pattern found from " + mg.start() +
                               " to " + (mg.end()-1));

        System.out.println("\nUsing Possessive Quantifier");
        Matcher mp = pp.matcher("ggg");
        while (mp.find())
            System.out.println("Pattern found from " + mp.start() +
                               " to " + (mp.end()-1));

    }
}
```

Output

```
Using Greedy Quantifier
Pattern found from 0 to 2

Using Possessive Quantifier
```

In the above example, since the first quantifier is greedy, **g+** matches the whole string. If we match **g+** with whole string, **g+g** doesn't match, the Greedy quantifier removes the last character, matches **gg** with **g+**, and finds a match. In the Possessive quantifier, we start like Greedy. **g+** matches the whole string, but matching **g+** with the whole string doesn't match **g+g** with **ggg**. Unlike Greedy, since quantifier is possessive, we stop at this point.

This article is contributed by **Rahul Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above