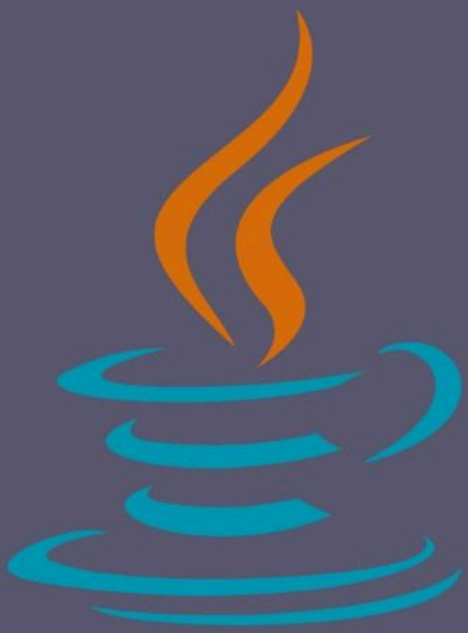
OBJECT ORIENTED PROGRAMMING USING



Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Java - Regular Expressions

Java provides the java.util.regex package for pattern matching with regular expressions.

A regular expression is a special sequence of characters that help you match or find other strings or set of strin using a specialized syntax held in a pattern.

A regular expression is sequence of charachters that forms a search pattern. When you search for data in a text, you can use thi search pattern to descri what you are searching for.

A regular expression can be
- a single charachter
- more complicated pattern

They can be use to
- search
- edit
- manipulate

text and dat

Regular expressions can be used to perfo all types of text search and text replace operations.

Java doesn't have a built-in Regula Expression class, but we can import java.util.regex package to work with regular expressions.

1) Pattern Class
2) Matcher Class
3) PatternSyntaxException Class

Pattern Class:

Defines a pattern (to be used in a search).
The class is compilation of regular expressions that can be used to define various types of patterns. providing no public constructors.

A Pattern object is compiled representation of a regular expression.
The Pattern class provides no public constructors
To create a pattern, you must first invoke one of its public static compile() methods, which will return a Pattern object.
These methods accept a regular expression as first argument.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegexExample{
  public static void main (String [Jargs)
{ String regex = "ab";

  Pattern p = Pattern.compile(regex);

String input = "The word ab is present in text";

Matcher m = p.matcher(input);

while (m.find()) {
System.out.println("Found at index
                    a match: " + m.start()
```

Example: Find out if there are any occurences of word "w3schools" in a sentence.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
public static void main (String [] args)
{ Pattern p = Patter.compile ("w3schools",
                              p.CASE_INSENSITIVE);

Matcher m = p.matcher ("Visit W3schools!");

boolean found = m.find();

if (found){
 System.out.println ("Match found");
}
 else
{ System.out.println ("Match not found");

}}}}
```

Match found

First the pattern is created using the method. Pattern.compile(). The first parameter indicate which pattern is being searched for ("w3schools"), the second parameter has a flag to indicates that search should be case-insensitive. The second parameter is optional.

=> The matcher() method is used to search for the pattern in a string. It returns a Matcher object which contains info. about search that was performed.

=> The find() method returns true if the pattern was found in the string and false if it was not found.

Matcher Class:

Used to search for pattern.

Used for performing match operations on text using patterns.

A matcher object is the engine that interprets the pattern and perform match operations against an input string.

Like Pattern class, Matcher class has no public constructor.

You obtain matcher object by invoking the matcher() method on pattern object.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class MatcherExample {
public static void main (String [] args)
{ String regex = "apple";
  Pattern p = Pattern.compile(regex);
  String input = "I hava an apple";
```

```
Matcher m = p.matcher (input );
while (m.find());
{ System.out.println (" Found 'apple' at
                    index: " + m.start());
}}}
```

Flags:

Flags in the compile() method change how the search is performed.

Pattern.CASE_INSENSITIVE :

The case of letter will be ignored when performing a search.

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

public class Main{
public static void main (String[]args)
{
  String input = "Hello, HeLLo, hELLo, hello!";
  Pattern p = Pattern.compile ("hello", Pattern.
                    CASE_INSENSITIVE
Matcher m = p.matcher(input);
while (m.find())
{ System.out.println ("Matcher found at: "+m.start(
```

Match found at index 0 : Hello
Match found at index 7 : HeLLO
Match found at index 14 : HELLO
Match found at index 21 : hello

- m.start() : This method returns the start index of the last match. It indicates the position in the input string where the matched substring begins.

- m.group() : This method returns the actual substring that matches the pattern during the last call to find(). It represents the text of last matched group.

2) Pattern.LITERAL - Special charachters in the pattern will not have any special meaning and will be treated as ordinary charachters when performing a search.

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

public class Main {
  public static void main(String[] args)
{ String input = "A $100 bonus for 2
                 items (limited offer)";
```

```
String literalPattern = "A $100 bonus for
                        2 times (limited offer)";

Pattern p = Pattern.compile(literalPattern,
                        Pattern.LITERAL);

Matcher m = p.matcher(input);

if (m.find())
{
  System.out.println("Match found: " +
                        m.group());
}
else
{
  System.out.println("No match found");
}}
```

Match found: A $100 bonus for 2 items
(limited offer).

3) Pattern.UNICODE_CASE :

Use it together with the CASE_INSENSITIVE flag to als ignore the case of letters outside of t English alphabet.

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

public class Main {
```

```
public static void main(String [] args)
{ String input = "Café and café are the
                 same";

Pattern p = Pattern.compile ("café", Pattern.
                              CASE_INSENSITIVE);
                              |Pattern.UNICODE_CASE );

Matcher m = p.matcher (input);

while (m.find())
{ System.out.println ("Match found at
                       index:" + m.start()+":"
                       + m.group());
}
}}
```

Regular Expression Patterns

The "first parameter" of the Pattern.compile() method is the pattern.
It describes what is "being searched for".

- [abc] : Find one charachter from the options between the brackets.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
 public static void main(String[] args)
{ String input = "a1b2c3";
//Define the pattern with characters in square bracket
Pattern p = Pattern.compile("[abc]");

Matcher m = p.matcher(input);

while (m.find());
{ System.out.println("Found: " + m.group())
}
```

Found : a
Found : b
Found : c

- [^abc] : Find one character NOT between the brackets.

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

```
public class Main {
 public static void main (String [] args)
{ String input = "a1 b2 c3 XyZ";
 Pattern p = Pattern.compile ("[^abc]");
 Matcher m = p.matcher (input);
 while (m.find())
{ System.out.println ("Match found:"+
                        m.group);
}}}
```

```
Match found: 1
Match found: 2
Match found: 3
Match found: X
Match found: y
Match found: Z
```

- [0-9] : Find one charachter from the range 0 to 9.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
public static void main (String [] args)
{ String input = "The numbers are 1,5 and 9";
 Pattern p = Pattern.compile ("[0-9]");
 Matcher m = p.matcher (input);
```

```
        System.out.println ("Matches:");
        while (m.find())
        { System.out.println (m.group()+ " ");
        }
    }
}
```

- abc : Exact abc occurs then true otherwise false.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
 public static void main (String [] args)
{
  String input = "The sequence is abcdef";
  String regex = "abc";
 Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(input);
if (m.find())
 { System.out.println ("matches:" + m.group);
 }
else
{ System.out.println ("Match not found");
}}}
```

matches : abc

Metacharacters:

Metacharacters are the characters with a special meaning:

| : Find a match for any one of the patterns separated by | as :

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
public static void main (String[] args)
{ String input = "I have a dog and a cat,
                 but I don't have a fish";

Pattern p = Pattern.compile ("cat|dog|fish");

Matcher m = p.matcher (input );

while ( m.find() )
{
System.out.println ("Found match: "+m.group());
}
```

```
Found match : dog
Found match : cat
Found match : fish
```

. : Find just one instance of any character.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class main
{ public static void main (String[] args)
```

```
{ String input = "Hello @ World";
  String regex = ".@";
  Pattern p = Pattern.compile(regex);
  Matcher m = p.matcher(input);
  if (m.find())
  { System.out.println("Found a match:"+m.
                                    group(
  }
  else
{ System.out.println("No match found");
}}}
```

^ : Finds a match as the beginning of a string as in : ^Hello.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
 public static void main(String[]args)
{ String input = "Hello World";
  String regex= "Hello";
 Pattern p = Pattern.compile(regex);
 Matcher m = p.matcher(input);
```

```
if (m.find())
 { System.out.println("Match found at
                    the beginning:" + m.group()
 }
else
{ System.out.println("No match found");
}}}
```

Match found at the beginning : Hello

$: Finds a match at the end of string as in World$.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main {
 public static void main (String [] args)
{ String input = "Hello World";
  String regex = "World$";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(input);
if (m.find())
{ System.out.println("Match found at the
                    end :" + m.group());
}
else
{ System.out.println("No match found");
```

Quantifiers: Quantifiers define quantities.

n+ : Matches any string that contains at least one or more string.

```
import java.util.*;
class QuantifierExample{
 public static void main (String[]args)
{ String input = "Quantifier";
  Pattern p = Pattern.compile ("n+");
  Matcher m = p.matcher (input);
  if (m.find())
{ System.out.println ("Match found:"+input)
}
 else
{ System.out.println ("Match not found");
 }
  input = "Hello";
  m = p.matcher (input);
  if (m.find())
{ System.out.println ("Match found:"+input
}
 else
{ System.out.println ("Match not found");
}
}
```

Match found : Quantifier
Match not found.

n* : Matches any string that contains zero or more occurencies of n.

```
import java.util.*;
class QuantifierExample{
public static void main (String[]args)
{  String input = "Quantifier";

   Pattern p = Pattern.compile ("n*");

   Matcher m = p.matcher (input);

   if (m.find())
  { System.out.println ("Match found:"+input);
  }
   else
  { System.out.println ("Match not found");
  }

   input = "Hello";
     m = p.matcher (input);

   if (m.find())
  { System.out.println ("Match found:"+input);
  }
   else
  { System.out.println ("Match not found");
  }
}
```

Match found : Quantifier
~~Match not found:~~
Match found : Hello

n{x}: Matches any String that contains a sequence of x n's.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class QuantifierExample {
 public static void main (String [] args)
{ String input1 = "nnn"; // should match as there're exactly 3 'n
 String input2 = "nnnn"; // shouldn't match a there are more than 3 n
String input3 = "Hello"; // shouldn't match as there are no n's.
 int x = 3;
Pattern p = Pattern.compile ("n[" + x + "]");

Matcher m1 = p.matcher (input1);
Matcher m2 = p.matcher (input2);
Matcher m3 = p.matcher (input3);

 if (m1.find())
{ System.out.println ("Input1 : Match found"
}
 else
{ System.out.println ("Match not found");
}

 if (m2.find())
{ System.out.println ("Input 2 : Match found
}
 else
{ System.out.println ("Match not found");
}
```

```
if (m3.find())
{ System.out.println("Input 3: Matchfound");
}
else
{ System.out.println("Input 3: Match not
                      found");
}}}
```

Input 1 : Match found
Input 2 : Match not found
Input 3 : Match not found

n{x,y} : Matches any string that contains a sequence of XtoY n's.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Main{
 public static void main (String[]args)
{
 String input1 ="nnnnn";
 String input2 = "nnnnnnn";
 String input3 = "nn";
 String input4 = "nnnnnnnnn";

int min =4;
int max = 8;

String patternString = "n{"+ min + "," +max+"}";

Pattern p= Pattern.compile(patternString);
```

```
Matcher m1 = p.matcher(input1);
Matcher m2 = p.matcher(input2);
Matcher m3 = p.matcher(iput3);
Matcher m4 = p.matcher(input4);

 if (m1.find())
{ System.out.println ("Input 1: Match found");
 }
 else
{ System.out.println ("Input 2: Match not
                        found");
 }
 if (m2.find())
{ System.out.println ("Input 2: Match found");
 }
 else
{ System.out.println("Input 2: Match not
                        found");
 }
 if (m3.find())
{ System.out.println ("Input 3: Match found");
 }
 else
{ System.out.println("Input 3: Match not
                        found");
}}}
```

Input1 : Match found
Input2 : Match found
Input3 : Match not found
Input4 : Match not found

n{x,} : Matches any string that contains a sequence of at least x n's.

```
import java.util.regex.*;
class Main {
public static void main (String[]args)
{ String regex = "n{3,}";

 Pattern p = Pattern.compile(regex);
String input1 = "nnn";
String input2 = "nn";
String input3 = "nnnnn";
String input4 = "n";

testMatch (pattern, ~~test~~input1);
testMatch (pattern, input2);
testMatch (pattern, input3);
testMatch (pattern, input4);

public Static void testMatch (Pattern p,
                              String input)
{ Matcher m = p.matcher (input);
{ System.out.println ("Testing String:"+input);
  if (m.find())
  { System.out.println ("Match found:"
                        + m.group());
  }
  else
  { System.out.println ("No match found");
  }
}
```

Match found
No match found
Match found
No match found