

Lab: Regular Expressions

Problems for exercises and homework for the ["Programming Fundamentals" course @ SoftUni](#)

You can check your solutions in [Judge](#).

1. Match Full Name

Write a Java Program to **match full names** from a list of names and **print** them on the console.

Writing the Regular Expression

First, write a regular expression to match a valid full name, according to these conditions:

- A valid full name has the following characteristics:
 - It consists of **two words**.
 - Each word **starts** with a **capital letter**.
 - After the first letter, it **only contains lowercase letters afterwards**.
 - **Each** of the **two words** should be **at least two letters long**.
 - The **two words** are **separated** by a **single space**.

To help you out, we've outlined several steps:

1. Use an online regex tester like <https://regex101.com/>
2. Check out how to use **character sets** (denoted with square brackets - "[]")
3. Specify that you want **two words** with a space between them (the **space character** ' ', and **not** any whitespace symbol)
4. For each word, specify that it should begin with an uppercase letter using a **character set**. The desired characters are in a range – **from 'A' to 'Z'**.
5. For each word, specify that what follows the first letter are only **lowercase letters**, one or more – use another character set and the correct **quantifier**.
6. To prevent capturing of letters across new lines, put "\b" at the beginning and at the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

In order to check your RegEx, use these values for reference (paste all of them in the **Test String** field):

Match ALL of these	Match NONE of these
Ivan Ivanov	ivan ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Ivan IvAnov, Ivan Ivanov

By the end, the matches should look something like this:

TEST STRING	SWITCH TO UNIT TESTS ▶
Vankata IvAnov, Ivan ivanov, Ivan Ivanov, ivan ivanov, ivan Ivanov, IVan Ivanov, Ivan Ivanov	

After you've constructed your regular expression, it's time to write the solution in Java.

Implementing the Solution in Java

Create a new Java project and copy your **regular expression** into a **String** variable:

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    String regex = "\\b[A-Z][a-z]+ [A-Z][a-z]+\\b";  
}
```

Now, it's time to **read the input** and create two **classes** to help us work with **regular expressions**:

- **Pattern Class** – A Pattern object is a compiled representation of a regular expression.
- **Matcher Class** – A Matcher object is the engine that interprets the pattern and performs match operations against an input string.

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    String regex = "\\b[A-Z][a-z]+ [A-Z][a-z]+\\b";  
    String input = scanner.nextLine();  
  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(input);  
}
```

Now, it's time to **extract all the matches** from our input and print them. We use matcher method **find()**, which **attempts to find the next subsequence of the input sequence that matches the pattern**. To get our matches, we need to use method **group()**.

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    String regex = "\\b[A-Z][a-z]+ [A-Z][a-z]+\\b";  
    String input = scanner.nextLine();  
  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(input);  
  
    while (matcher.find()) {  
        System.out.print(matcher.group() + " ");  
    }  
}
```

Examples

Input
Ivan Ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Test Testov, Ivan Ivanov
Output
Ivan Ivanov Test Testov

2. Match Phone Number

Write a regular expression to match a **valid phone number** from **Sofia**. After you find all **valid phones**, **print** them on the console, separated by a **comma and a space** ", ".

Compose the Regular Expression

A valid number has the following characteristics:

- It starts with "+359"
- Then, it is followed by the area code (always 2)
- After that, it's followed by the **number** itself:

- The number consists of **7 digits** (separated in **two groups** of **3** and **4 digits** respectively).
- The different **parts** are **separated** by **either a space or a hyphen** ('-').

You can use the following RegEx properties to **help** with the matching:

- Use **quantifiers** to match a **specific number** of **digits**
- Use a **capturing group** to make sure the delimiter is **only one of the allowed characters** (space or hyphen) and **not a combination** of both (e.g. +359 2-111 111 has **mixed delimiters**, it is **invalid**). Use a **group backreference** to achieve this.
- Add a **word boundary** at the **end** of the match to avoid **partial matches** (the last example on the right-hand side).
- Ensure that before the '+' sign there is either a **space** or the **beginning of the string**.

You can use the following table of values to test your RegEx against:

Match ALL of these	Match NONE of these
+359 2 222 2222 +359-2-222-2222	359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222

Implement the Solution in Java

Now it's time to write the solution, so let's start writing!

First, just like in the previous problem, put your RegEx in a variable:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    String regex = "\\+359([- ])2\\1[\\d]{3}\\1[\\d]{4}\\b";
```

Again we need a **Pattern** and **Matcher**.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    String regex = "\\+359([- ])2\\1[\\d]{3}\\1[\\d]{4}\\b";
    String phones = scanner.nextLine();

    Pattern pattern = Pattern.compile(regex);
    Matcher phoneMatcher = pattern.matcher(phones);
```

We can also save our matches in a List if we need.

```
List<String> matchedPhones = new LinkedList<>();

while (phoneMatcher.find()) {
    matchedPhones.add(phoneMatcher.group());
}
```

After that, just print the valid phone number list, using **string.Join()**:

```
System.out.println(String.join(" ", matchedPhones));
```

Examples

Input
+359 2 222 2222, 359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222 +359-2-222-2222
Output
+359 2 222 2222, +359-2-222-2222

3. Match Dates

Write a program, which matches a date in the format “**dd{separator}MMM{separator}yyyy**”. Use **named capturing groups** in your regular expression.

Compose the Regular Expression

Every valid date has the following characteristics:

- Always starts with **two digits**, followed by a **separator**
- After that, it has **one uppercase** and **two lowercase** letters (e.g. **Jan, Mar**).
- After that, it has a **separator** and **exactly 4 digits** (for the year).
- The separator could be either of three things: a period (“.”), a hyphen (“-”) or a forward slash (“/”)
- The separator needs to be **the same** for the whole date (e.g. 13.03.2016 is valid, 13.03/2016 is **NOT**). Use a **group backreference** to check for this.

You can follow the table below to help with composing your RegEx:

Match ALL of these	Match NONE of these
13/Jul/1928, 10-Nov-1934, 25.Dec.1937	01/Jan-1951, 23/sept/1973, 1/Feb/2016

Use **named capturing groups** for the **day**, **month** and **year**.

Since this problem requires more complex RegEx, which includes **named capturing groups**, we’ll take a look at how to construct it:

- First off, we don’t want anything at the **start** of our date, so we’re going to use a **word boundary** “\b”:

REGULAR EXPRESSION

:/ \b

- Next, we’re going to match the **day**, by telling our RegEx to match **exactly two digits**, and since we want to **extract** the day from the match later, we’re going to put it in a **capturing group**:

REGULAR EXPRESSION

:/ \b(\d{2})

We’re also going to give our group a **name**, since it’s easier to navigate by **group name** than by **group index**:

REGULAR EXPRESSION

:/ \b(?<day>\d{2})

- Next comes the separator – either a **hyphen**, **period** or **forward slash**. We can use a **character class** for this:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})[-.\//]
```

Since we want to use the separator we matched here to match the **same separator** further into the date, we're going to put it in a **capturing group**:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})([-.\//])
```

- Next comes the **month**, which consists of a **capital Latin letter** and **exactly two lowercase Latin letters**:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})([-.\//])(?:<month>[A-Z][a-z]{2})
```

- Next, we're going to match the **same separator we matched earlier**. We can use a **backreference** for that:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})([-.\//])(?:<month>[A-Z][a-z]{2})\2
```

- Next up, we're going to match the year, which consists of **exactly 4 digits**:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})([-.\//])(?:<month>[A-Z][a-z]{2})\2(?:<year>\d{4})
```

- Finally, since we don't want to match the date if there's anything else **glued to it**, we're going to use another **word boundary** for the end:

REGULAR EXPRESSION

```
// \b(?:<day>\d{2})([-.\//])(?:<month>[A-Z][a-z]{2})\2(?:<year>\d{4})\b
```

Now it's time to find all the **valid dates** in the input and **print each date** in the following format: "Day: {day}, Month: {month}, Year: {year}", each on a **new line**.

Implement the Solution in Java

First off, we're going to put our RegEx in a variable.

```
String regex = "\\b(?:<day>\\d{2})([-.\//])(?:<month>[A-Z][a-z]{2})\\2(?:<year>[\\d]{4})\\b";
String datesStrings = scanner.nextLine();
Pattern pattern = Pattern.compile(regex);
Matcher dates = pattern.matcher(datesStrings);
```

Next, we're going to **iterate** over every single **Match** and **extract** the **day**, **month** and **year** from the **groups**.

```
while (dates.find()){  
    String day = dates.group( name: "day");  
    String month = dates.group( name: "month");  
    String year = dates.group( name: "year");  
  
    System.out.printf("Day: %s, Month: %s, Year: %s\n", day,month,year);  
}
```

Examples

Input
13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,f 25.Dec.1937 23/09/1973, 1/Feb/2016
Output
Day: 13, Month: Jul, Year: 1928 Day: 10, Month: Nov, Year: 1934 Day: 25, Month: Dec, Year: 1937