# Lab: Regular Expressions

In the current lab we are going to introduce some restrictions in the data for our database. Below you can see the constraints to validate the input entry before adding it to the data structure.

**Input format** – the format for the input should be the following:

**{Course Name}\_{Course Instance}{One or more white spaces}{Username}{One or more white spaces}{Score}**

Our task now is to write a regular expression that matches only valid entries, so we can add them to our data structure safely. Here is some example input data that may be given:

C#\_Feb\_2015 Kiko23\_4144 69

JSApps\_Dec\_2014 Ivo42\_230 17

C#\_Jul\_2016 Kiko23\_4144 94

JSApps\_Dec\_2014 Sten16\_96 41

C#\_Feb\_2015 Desi12\_2001 77

WebFundamentals\_Oct\_2015 Ivo42\_230 238

DataStructures\_Apr\_2016 Ivan23\_923 94

C#\_July\_2016 Rdsauja16\_23 71

JSApps\_Dec\_2014 NiK68\_0192 1

Unity\_Jan\_2016 Sten16\_96 56

unity\_Jan\_2016 Sten16\_96 53

JSApps\_Dec\_2014 Stan21\_23 46

C#\_Feb\_2015 NiK68\_0192 53

DataStructures\_Apr\_2016 Stan21\_23 93

WebFundamentals\_Oct\_2015 Desi12\_2001 81

Java\_May\_2015 Ivo12\_2341 77

C#\_Feb\_15 Sten16\_96 12

C#\_Feb\_2015 Desi12\_2001 93

WebFundamentals\_Oct\_2015 Kiko23\_4144 87

**Course name –** starts with a capital letter and may contain only small and capital English letters, plus ‘+’ or hashtag ‘#’.

**Course instance –** should be in the format ‘Feb\_2015’, e.g. containing the first three letters of the month, starting with a capital letter, followed by an underscore and the year. The year should be between 2014 and the current year.

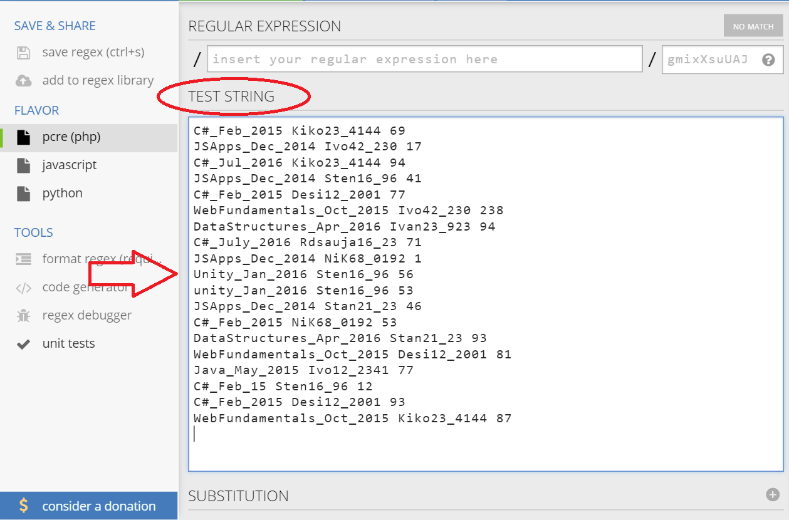
**Username** – starts with a capital letter and should be followed by at most 3 small letters after that. Then it should have 2 digits, followed by an underscore, followed by two to four digits. **Correct:** Ivan23\_234, Nas12\_4215, Re14\_203. **Incorrect:** Ivana33\_123, Stan\_12, Мари31\_421

**Score** – should be in the range from 0 to 100.

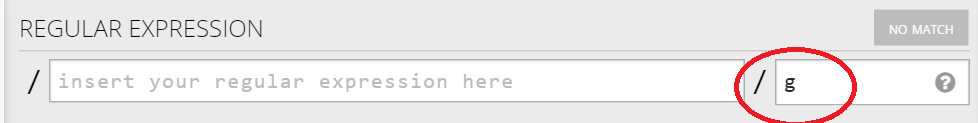
We are going to write a regular expressing for validating the input and implement it in the method for reading data from a file for the database of the university.

## Start Using a Regex Editor

First we want to open some regex editor that will help us to complete our task. You can use whatever editor you like but you should be already familiar with <https://regex101.com/> so we give you its link here. Next you may want to paste the sample data given above in the TEST STRING box:



Next you need to include the global modifier by simply adding a ‘g’ in the upper right corner:



Ok, that was pretty easy. Let’s proceed with the next task.

## Using Groups

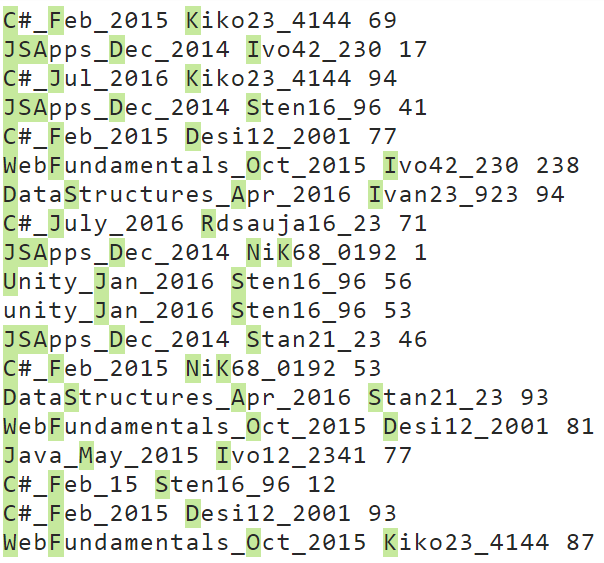
We are going to separate our match into three groups:

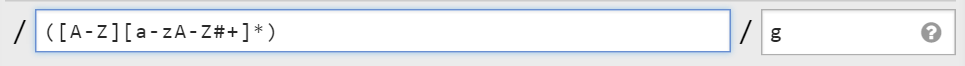
1: Course name and instance

2: Student user name

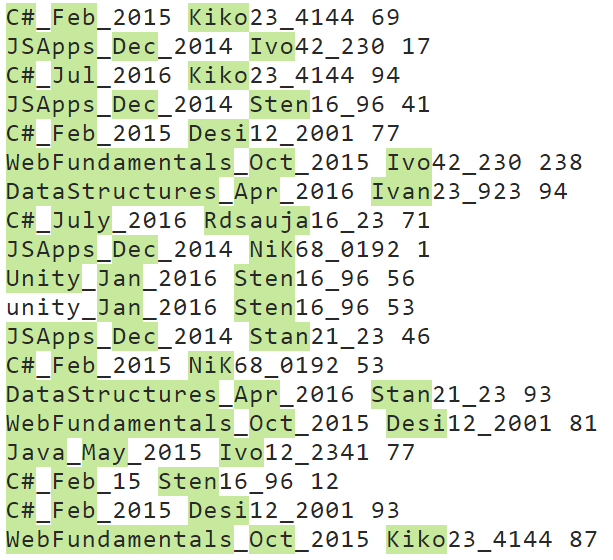
3: Student score on task

We can start with the first group and to be more concrete with the course name. It’s said that it should start with a capital letter so that is the first thing to add and you will be able to see as we go, how some data does not meet the conditions of the regex. So our regex by now should look like this: 

And the matches are still quit unclear:  


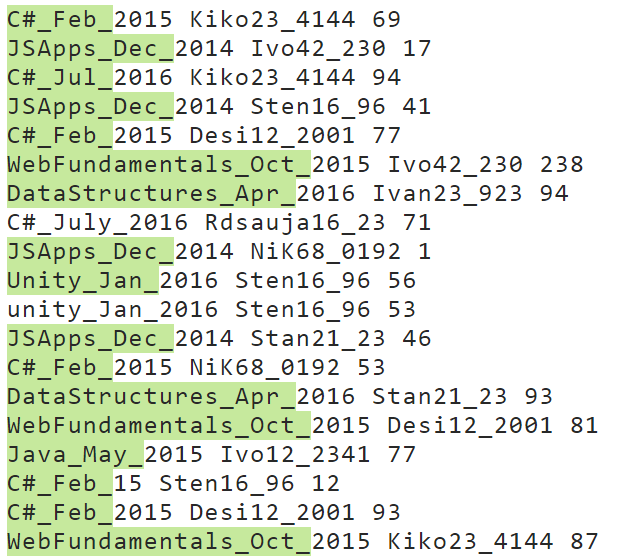
As you can see even from the first condition we don’t catch the **unity** course written with small letters. The next thing needed is that it **can** contain small and capital letters and also the symbols ‘+’ and ‘#’:   


We have put an asterisk ‘\*’ after the range, because the name of the course may be only of one capital letter. The result of the current modification looks like this:



Now it’s time to add an underscore and the condition for the month name that follows, followed by an underscore:   

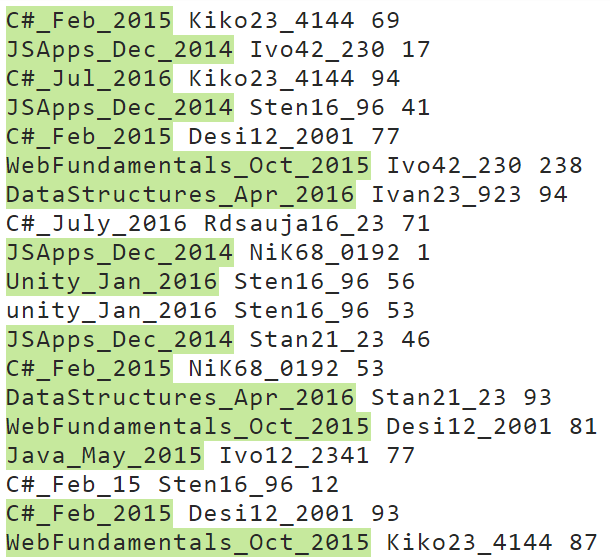

Here the condition after the range should be exactly two letters, because in the conditions the total number of letter for the month name is 3 and the first one should be capital. The result after this filter gives is now clearly showing where we are headed:



As you can see, now the C# course in July is no longer valid because the month is written with four letters.

Finally, it is time to add the year to the matching regular expression:  

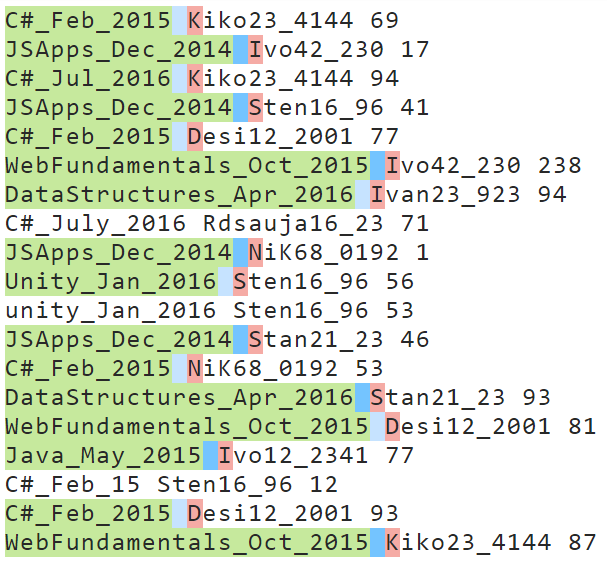

Now we should be ready with the group for the course and the result that we match by now should be as the one below:



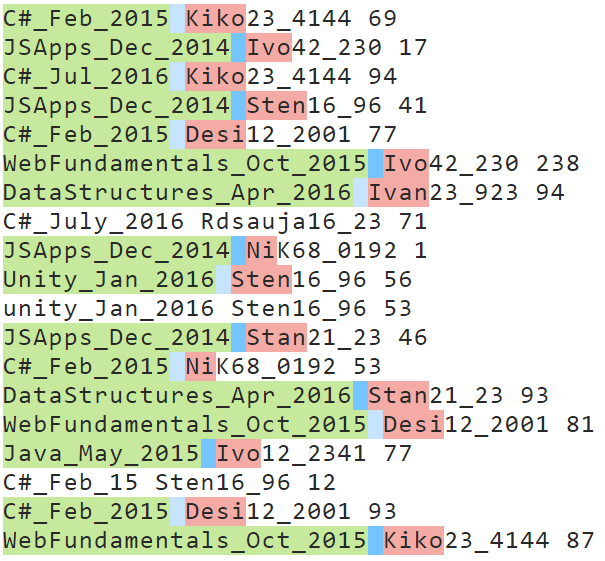
As you can see, now we don’t catch the C#\_Feb\_15, because the year is not in the valid format.

It is time to write the group for the user name which is similar to the one we created. So now we must add a separator for one or more spaces, followed by the group, starting with a first capital letter:



The result after this filter is pretty much the same for the input we’ve chosen, but there could have been a person whose name starts with a smaller letter or an entry where is no space between the course and the user name:  


Now we should finish the regex for the rest of the name before the two numbers that follow. We should have in mind that the we can have **at most** 3 letters after the first capital letter, but there may be 3 letters as there may be no letters after the first one. So this may be expressed as follows in our regular expression

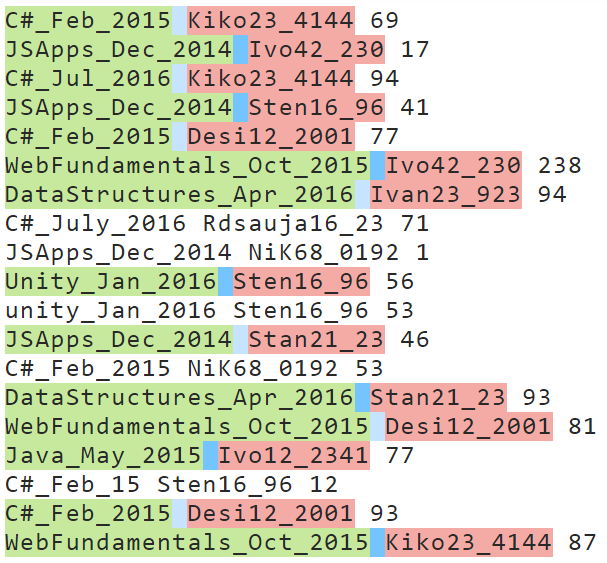


After this addition we can note that even if C#\_July\_2016 was written following the conditions, the user name would still be incorrect and of course the whole entry will be invalid.

So the only thing left for the user name is the **two digits** that follow after the letters, followed by an **underscore**, followed by **two to four digits:**

****

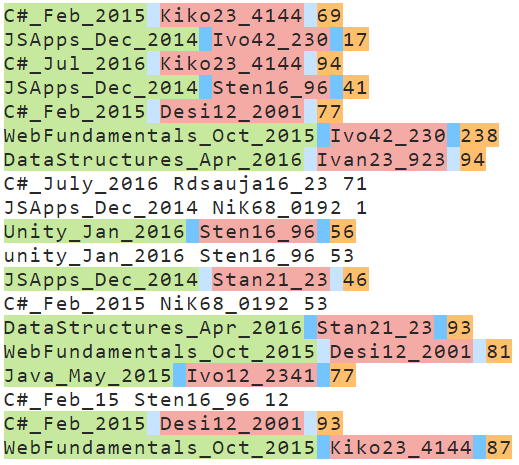
You may see below that 2 more matches are now invalid, because they don’t match the required format for the user name and more specifically for the numbers that are in the user name:



The final group we need to catch is the group for the score on a task for the given person and the given course:



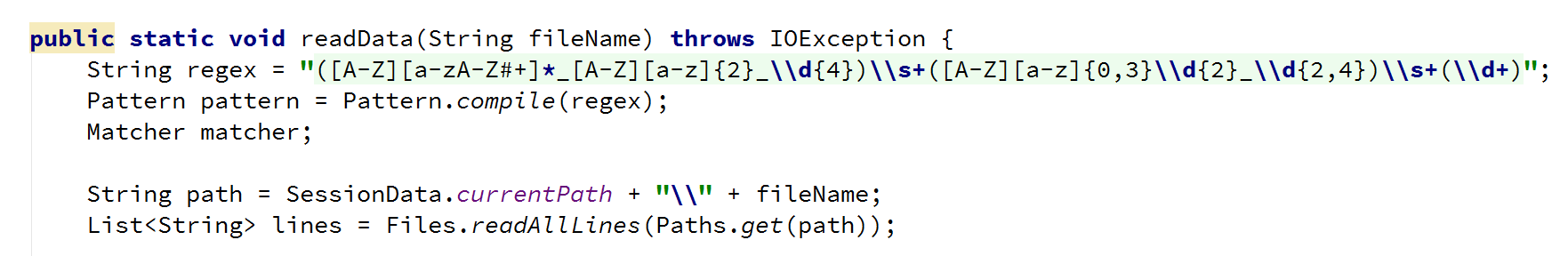
The result with the given matches from the example should now look like this:



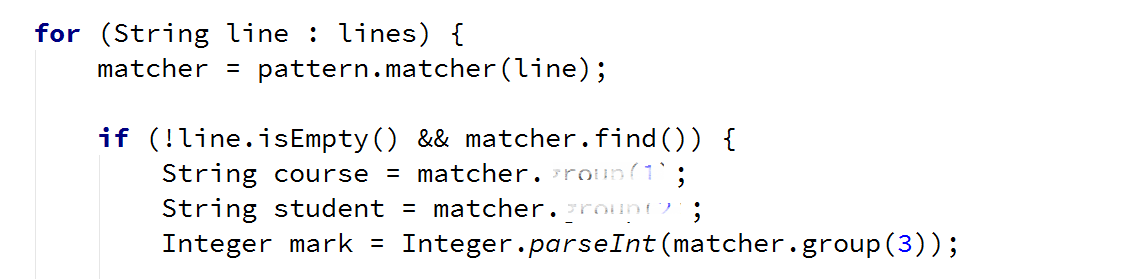
So now that we’ve finally written the whole regular expression, it is time to implement it in Java in the method that reads all the students from the “database” file, but now you will be given a new file that will contain entries that do not match the given format.

We are going to refactor some code in the readData() method in the StudentRepository class because we will now have to get the data from the groups of the current match.

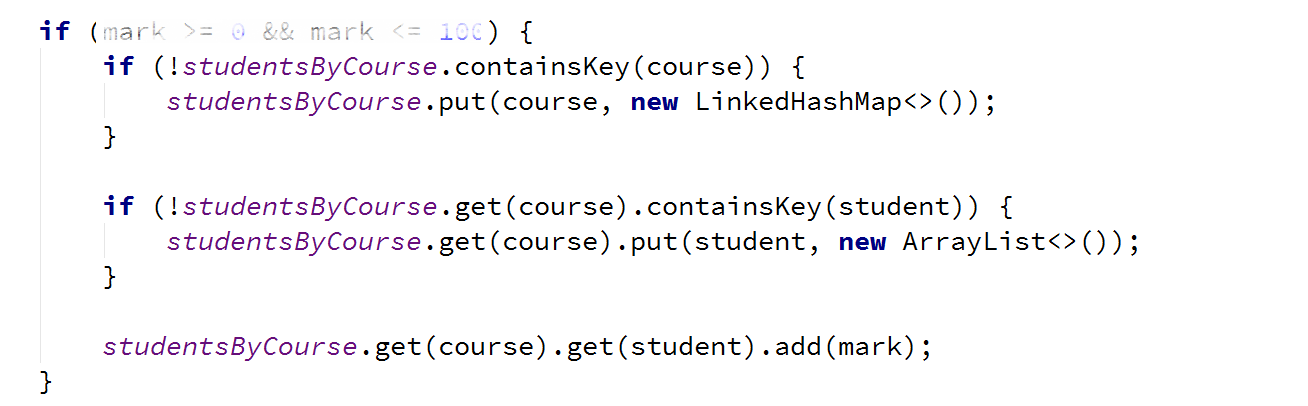
First thing is to copy the pattern for the match of the entries and also create a new String with the given pattern and then compile it into a Pattern class:



Now that we have this the Matcher and the Pattern classes, we can use them to check if there is a match with the current input line and if there is such, to get the data that we need from it in order to:



So the only thing left is to check if the score is in the range between 0 and 100 and if all the three conditions are true, we can insert the data that we have extracted.



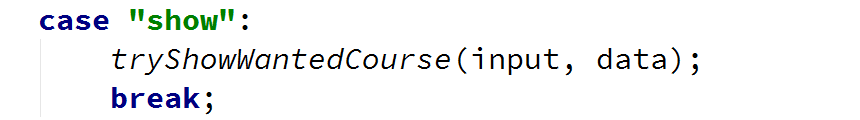
## Adding Features to the Command Interpreter

**Before** **testing** there is just one little thing we can **add** **to** **Command** **Interpreter**, because obviously we forgot adding it in the previous piece. The functionality is going to be a new command in the following format:

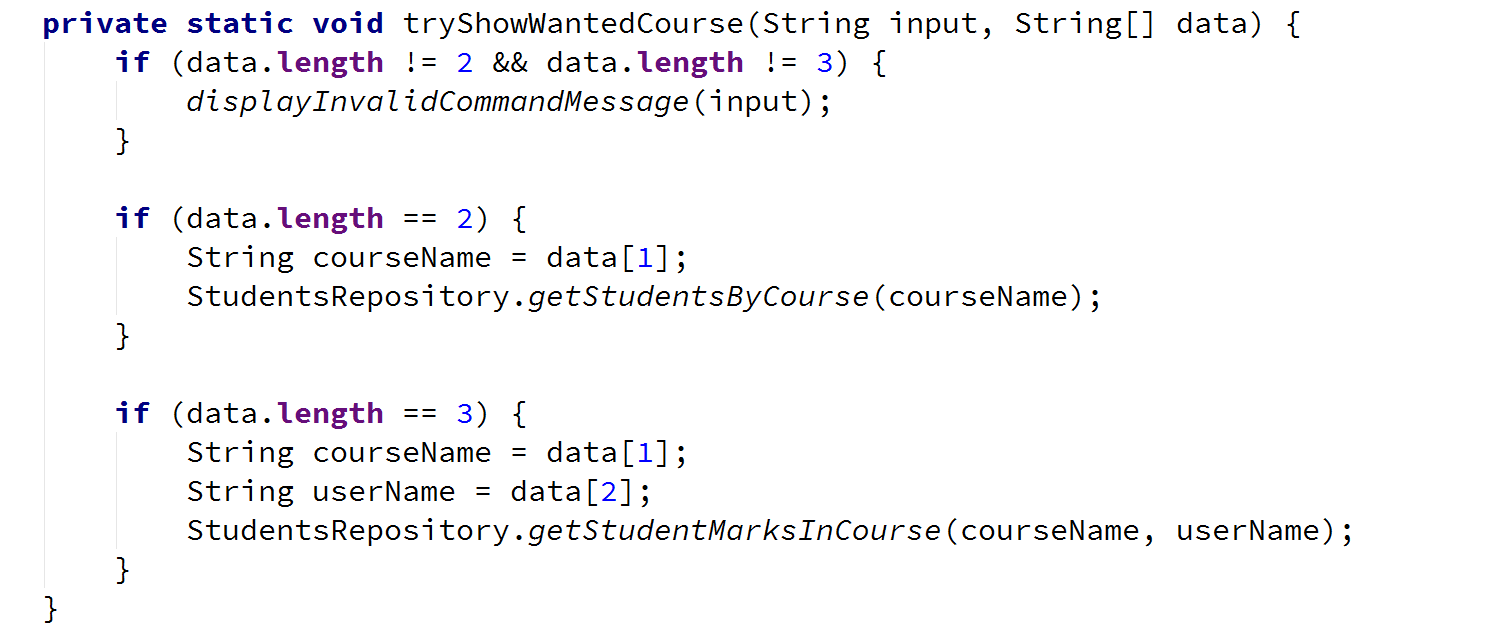
**show courseName (username) – user name may be omitted**

It shows information for a given course or a given user in a course from our data.

In the switch of the command interpreter, the case looks like the following:

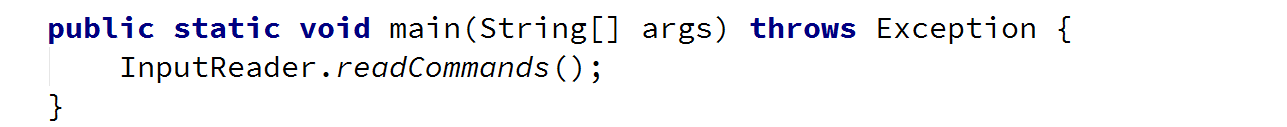


Time to implement its functionality. We should only check for the number of parameters and depending on this, call the corresponding method.

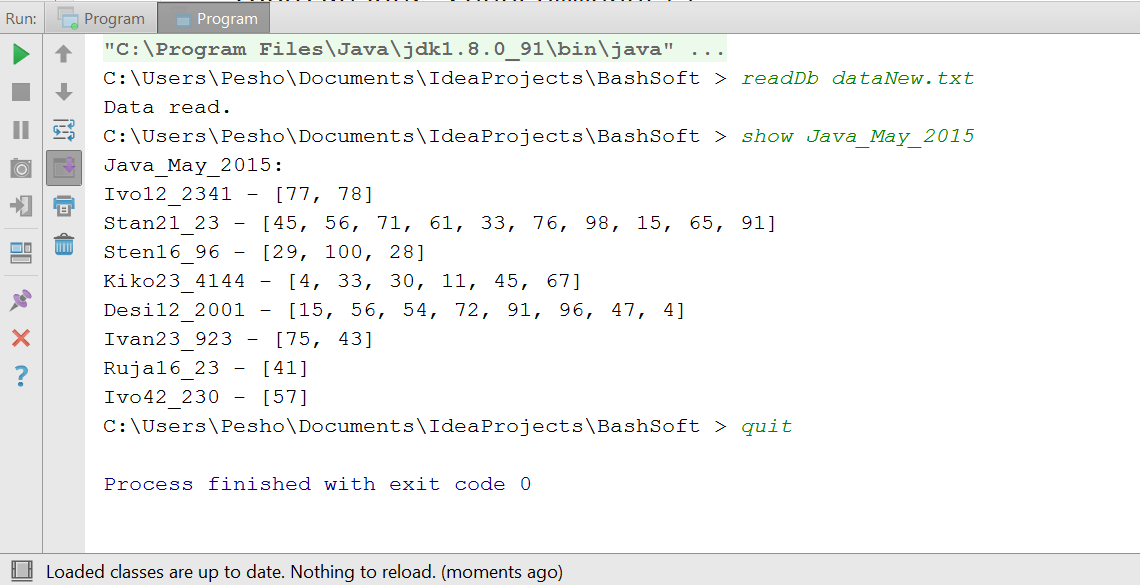


## Testing Your Code

Finally, we should be ready with the **implementation** of the **regular** **expressions**. Now you can test your code with the provided dataNew.txt file. If it is not placed in your project folder you can navigate to it with the appropriate commands after running your project:



And you should get an output like that:



Congratulations! You’ve completed the lab exercise for Regular Expressions.