Practical Class nº 3

Regular Expressions, Wrappers

1. Bibliography

http://www.regular-expressions.info/java.html
http://www.regular-expressions.info/reference.html

2. Introduction to Regular Expressions using Java

In this worksheet it is intended that students explore the use of ER in Java.
In addition to the REs covered in the theoretical classes, the following table summarizes some expressions that can be used in a Java program:

Regular Expression	Descripriom
`\d`	Any digit, it's equivalent to [0-9]
`\D`	Every char except a digit it's equivalent to [^0-9]
`\s`	Any space char, it's equivalent to [\s\t\n\x0b\r\f]
`\S`	Every char except a space char, it's equivalent to [^\s]
`\w`	A alphanumeric char, its equivalent to [a-zA-Z_0-9]
`\W`	A char except a alphanumeric char, its equivalent to [^\w]

NOTE 1: Putting **(?i)** at the beginning of an ER makes it 'case insensitive'

NOTE 2:
The \ character is an escape character used in strings in Java. If you want to use this character in an ER, it must appear duplicated. For example, using "\\" defines a single character "\". To use \ w in an RE, you must type \\ w, to use \ d, type \\ d, etc.

2.1. Classes Pattern and Matcher

For advanced use of ER, the classes **java.util.regex.Pattern** and **java.util.regex.Matcher** should be used
The **Pattern** class must first be used to define the regular expression. The object created with the Pattern class allows you to create a Matcher object for a given string. This **Matcher** object allows you to execute regex methods on the string.
The description of the methods of these classes can be found at:
http://docs.oracle.com/javase/tutorial/essential/regex/matcher.html

The steps for creating the RE and looking for the respective pattern in a data source are as follows:

1) Create the regular expression in a String:
   `String er = “[a-zA-z]+”;`

2) Initialize a String with the data source where the RE will be applied, for instance:
   `String text = “Mary is 12 years old”;`

3) Create pattern class variable to compile the RE:
   `Pattern p = Pattern.compile(er);`

4) Create Matcher type variable to link the RE to the text:
   `Matcher m = p.matcher(text);`

5) Looking for matches in the text:
   ```
   while (m.find()){
         System.out.println(m.group());
   }
   ```

A complete example:

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

public class TestingRE {


    public static void main(String[] args) {

        String line = " Line 1 comes before line 2 in the middle of 100 lines ";

       //RE to find numbers in a data source
       String re="[0-9]+"; // equivalent to \\d+
       Pattern p = Pattern.compile(re);
       Matcher m = p.matcher(line);
       while(m.find()){
             System.out.println("number : " + m.group() + “\n”)
       }
   }
}
```

3. Tasks to perform in the class

Create a new Java Application named **DIClass3**.

a) Write a function **`static void class3a()`** that:

- initialize a variable of the String class with several Portuguese phone numbers (mobile and fixed)

```
String phones="919191919 929992221 91111111111 239494582 9199999999 967779999";
```

  - initialize a String with a RE that validates only mobile phone numbers using the rule:
    - starting with 91, 92, 93, 96
    - total number of digits is 9
  - In the string ***phones***, the blue ones are the correct Portuguese mobile numbers.

- Use the Classes **`Pattern`** e **`Matcher`** to compile and find the correct matches. Print the found matches in the output console.
- Call the implemented function in *main* and check the result.

b) Write a function **`static void ficha3b(String fileIn, String fileOut)`** that:

- Initialize a variable of the String class with an RG that looks for dates using the following formats:
  - dd-mm-yyyy ou dd/mm/yyyy
- Use the Classes Pattern and Matcher as before
- Read the text file ***fileIn*** line by line
- In each line use the Matcher to look for dates. When you find a valid date write it in the ***fileOut***
  - 31-03-2002 – Valid Date
  - 04/12/2007 - Valid Date
  - 01-01-2005 - Valid Date
  - 01/03/2007 - Valid Date a
- Call the function in ***main*** using these files as arguments **"dates.txt"** e **"out.txt"**
- After running the function check if the **out.txt** file was correctly created.

c) Write the following Wrappers to look in the source ***people.html*** for the following information:

a) **`static ArrayList search_names(String search)`**

Looks for names in **people.html** that contains the search string.
The function returns a ArrayList with all the found names.

Example: search for **"Ana"** the function returns the list with *Ana Martins*, *Anabela Lopes*
Test the function in ***main()***

b) **`static ArrayList search_names_using_city(String search)`**

using a search city, returns the names of the people that live there. The source data is the file **people.html**.
The function returns the list of the names found. Example: search for **"Coimbra"** the function returns the list with *Anabela Lopes, Anibal Costa, Daniel Costa, Roberto Salvador, Mariana Matos, Suzete Franco.*

Test the function in ***main()***

c) **`static ArrayList search_names_for_job (String search)`**

using a search job, returns all the names of people that have that job. The source data is the file **people.html**.
The function returns the list of the names found. Example: search for **"Professor"** the function returns the list with *Anabela Lopes, Daniel Costa, Roberto Salvador.*

d) **`static String search_name(String id)`**

using a search ID returns the name of the person with that id or **null** if not found

e) **`static int search_age(String id)`**

using a search ID returns the age of the person with that id or **-1** if not found

f) **`static String search_city(String id)`**

using a search ID returns the city of the person with that id or **null** if not found

g) **`static String search_job(String id)`**

using a search ID returns the job of the person with that id or **null** if not found