ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Regular Expressions
Κανονικές εκφράσεις

Αθ. Ανδρούτσος

Κανονικές Εκφράσεις (1)

- Με τις κανονικές εκφράσεις (regular expressions) μπορούμε να ελέγξουμε **εάν μια συμβολοσειρά είναι ίδια με κάποια άλλη ή περιέχει ένα συγκεκριμένο μοτίβο (pattern)**
- Τα regular expressions συχνά χρησιμοποιούνται για αναζήτηση σε log files, για validation των δεδομένων εισόδου ή για την υλοποίηση scanners σε μεταγλωττιστές
- Οι κλάσεις στην Java που υλοποιούν κανονικές εκφράσεις βρίσκονται στο package java.util.regex

String.matches

- Μία απλή εκδοχή για τη χρήση Regular Expressions είναι η μέθοδος ***String.matches(String regex)***
- Είναι παρόμοια με την ***equals*** μόνο που εδώ συγκρίνουμε με regular expression και επομένως συγκρίνουμε με πολλαπλές τιμές
- Έστω για παράδειγμα ότι θέλουμε να ελέγξουμε αν ένα String περιέχει τη λέξη Red (βλ. επόμενη διαφάνεια)

Απλή εφαρμογή της .matches


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει
 * αν ένα String είναι το "Red".
 *
 * @author A. Androutsos
 * @see #isRed(String)
 */
public class RegExEquals {

    public static boolean isRed(String s) {
        return (s != null) && (s.matches("Red"));
    }
}
```

- Εδώ η .matches λειτουργεί όπως η equals
- Ελέγχει απλώς αν το String s είναι “Red”

- Αν θα θέλαμε να ελέγξουμε αν το String s είναι “Red” ή “Green” ;
- Με την .equals θα είχαμε: s.equals(“Red”) || s.equals(“Green”)
- Με Regular Expression δείτε την επόμενη διαφάνεια

Alternation -- Red or Green


```
1   package gr.aueb.elearn.ch3;
2
3   /**
4    * Περιέχει μία μέθοδο που ελέγχει αν
5    * ένα String είναι το "Red" ή το "Green".
6    *
7    * @author A. Androutsos
8    * @see #isRedOrGreen(String)
9    */
10  public class RegExRedOrGreen {
11
12      public static boolean isRedOrGreen(String s) {
13          return (s != null) && (s.matches("Red|Green"));
14      }
15  }
```

- Παρατηρήστε ότι ***δεν αφήνουμε κενά μεταξύ των λέξεων και του |***

- Αν θέλουμε να μπορεί να ταιριάζει το String s με τα Red ή red ή Green ή green ;
- Δείτε την επόμενη διαφάνεια.

Character Sets


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει αν
 * ένα String είναι το "Red" ή "red" ή
 * "Green" ή "green".
 *
 * @author A. Androutsos
 * @see #isRredOrGgreen(String)
 */
public class RegExRedOrGreenFirstLetter {

        public static boolean isRredOrGgreen(String s) {
            return (s != null) && (s.matches("[Rr]ed|[Gg]reen"));
        }
}
```

- Έλεγχος για Red/red/Green/ green
- Μέσα σε [ ] δίνουμε τους χαρακτήρες που μπορεί να βρίσκονται στην 1η θέση του Red ή του Green

- Αν θα θέλαμε μία λέξη να ξεκινάει με οποιοδήποτε κεφαλαίο γράμμα ;
- Δείτε την επόμενη διαφάνεια

Έλεγχος περιοχής χαρακτήρων (1)


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει αν
 * ένα String ξεκινάει με ένα γράμμα κεφαλαίο
 * και τελειώνει με -ing.
 *
 * @author A. Androutsos
 * @see #hasOneCapitalLetterFirstAndEndsWith_ing(String)
 */
public class RegExRange {

    public static boolean hasOneCapitalLetterFirstAndEndsWith_ing(String s) {
        return (s != null) && (s.matches("[A-Z]ing"));
    }
}
```

Με παύλα ορίζουμε την περιοχή

Για παράδειγμα με **[A-Z]** ορίζουμε ένα κεφαλαίο χαρακτήρα

- Αν θα θέλαμε να ξεκινάει είτε με κεφαλαίο ή με μικρό γράμμα ;
- Δείτε την επόμενη διαφάνεια

Έλεγχος περιοχής χαρακτήρων (1)


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει αν
 * ένα String ξεκινάει με ένα γράμμα πεζό
 * ή κεφαλαίο και τελειώνει με -ing.
 *
 * @author A. Androutsos
 * @see #hasOneCapitalLetterFirstAndEndsWith_ing(String)
 */
public class RegExRange2 {

    public static boolean hasOneCapitalLetterFirstAndEndsWith_ing(String s) {
        return (s != null) && (s.matches("[a-zA-Z]ing"));
    }
}
```

- Με παύλα ορίζουμε την περιοχή, π.χ. με **[a-zA-Z]** ορίζουμε ένα είτε πεζό ή κεφαλαίο χαρακτήρα

Μεταχαρακτήρες (1)

- **.** Οποιοσδήποτε χαρακτήρας
- **^** Αρχή μιας γραμμής
- **$** Τέλος μιας γραμμής
- **\w** Χαρακτήρας λέξης από a-z, A-Z, 0-9, καθώς και _ (underscore)
- **\W** Οποιοσδήποτε χαρακτήρας εκτός από χαρακτήρα λέξης

Μεταχαρακτήρες (2)

- **\s** Χαρακτήρας διαστήματος
- **\S** Οποιοσδήποτε χαρακτήρας εκτός από χαρακτήρα διαστήματος
- **\d** Οποιοδήποτε ψηφίο
- **\D** Οποιοσδήποτε χαρακτήρας εκτός από ψηφίο

Οποιοσδήποτε χαρακτήρας - Κενό - Ψηφίο


```
1   package gr.aueb.elearn.ch3;
2
3   /**
4    * Περιέχει μία μέθοδο που ελέγχει αν
5    * ένα String έχει τη μορφή ένας οποιοσδήποτε
6    * χαρακτήρας ακολουθούμενος από ένα κενό διάστημα
7    * ακολουθούμενος από ένα ψηφίο.
8    *
9    * @author A. Androutsos
10   * @see #isAnySpaceDigit(String)
11   */
12  public class RegExAnyCharSpaceDigit {
13
14      public static boolean isAnySpaceDigit(String s) {
15          return (s != null) && (s.matches(".\\s\\d"));
16      }
17  }
```

- Θέλουμε ως 1ο χαρακτήρα οποιοδήποτε χαρακτήρα. Για αυτό χρησιμοποιούμε τον μεταχαρακτήρα τελεία.
- Στη συνέχεια θέλουμε ένα κενό (space \s) και μετά ένα ψηφίο (digit \d)
- Ως ψηφίο θα μπορούσαμε να χρησιμοποιήσουμε και το [0-9]

- Παρατηρήστε πως επειδή το \ είναι ειδικός χαρακτήρας τον κάνουμε escape με \ μπροστά, οπότε έχουμε \\s για το κενό διάστημα που είναι \s και \\d για το ψηφίο που είναι \d

Γράμμα - Ψηφίο


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει αν
 * ένα String έχει τη μορφή ένα γράμμα
 * ή ψηφίο ή κάτω παύλα ακολουθούμενο
 * από ένα ψηφίο.
 *
 * @author A. Androutsos
 * @see #isLetterDigit(String)
 */
public class RegExLetterDigit {

    public static boolean isLetterDigit(String s) {
        return (s != null) && (s.matches("\\w\\d"));
    }
}
```

- Ένα οποιοδήποτε γράμμα δηλαδή [a-zA-Z0-9_] αναπαρίσταται ως \w και ένα ψηφίο [0-9] ως \d
- Ενναλακτικά θα μπορούσαμε να το γράψουμε στην πλήρη του μορφή

Γράμμα και ψηφίο οπουδήποτε μέσα στο String


```java
package gr.aueb.elearn.ch3;

/**
 * @author A. Androutsos
 * @see #isLetterDigitAnywhere(String)
 */
public class RegExLetterDigitAnywhere {

    /**
     * Ελέγχει αν ένα String ξεκινάει με ένα οποιοδήποτε
     * χαρακτήρα 0 ή περισσότερες φορές βρίσκει ένα χαρσκτήρα
     * λέξης και στη συνέχεια ένα ψηφίο και τελειώνει με ένα
     * οποιοδήποτε χαρακτήρα 0 ή περισσότερες φορές.
     *
     * @param s το προς έλεγχο String
     * @return true, αν γίνει match το regex, αλλιώς false.
     */
    public static boolean isLetterDigitAnywhere(String s) {
        return (s != null) && (s.matches(".*\\w\\d.*"));
        /*
         * Είναι παρόμοιο με:
         * return (s != null) && (s.matches(".*[a-zA-Z0-9_][0-9].*"));
         */
    }
}
```

- Η διαφορά με την προηγούμενη διαφάνεια είναι ότι έχουμε προσθέσει στην αρχή και το τέλος το (.*) που σημαίνει ένας οποιοσδήποτε χαρακτήρας μηδέν η περισσότερες φορές

Το * είναι ποσοδείκτης που σημαίνει μηδέν η περισσότερες φορές αυτό που προηγείται, εν προκειμένω η τελεία που σημαίνει οποιοσδήποτε χαρακτήρας
Strings που κάνουν match είναι για παράδειγμα τα: ***F4*** , ABC***D8***89! , ***55***AA

Γράμμα και ψηφίο ως λέξη


```java
package gr.aueb.elearn.ch3;

/**
 * @author A. Androutsos
 * @see #isLetterDigitAsWholeWord(String)
 */
public class RegExLetterDigitAsWord {

    /**
     * Ελέγχει αν ένα String ξεκινάει με ένα οποιοδήποτε
     * χαρακτήρα 0 ή περισσότερες φορές βρίσκει ένα χαρακτήρα
     * word boundary, μετά ένα χαρακτήρα λέξης, μετά ένα ψηφίο,
     * μετά ένα χαρακτήρα word boundary (άρα βρίσκει λέξεις)
     * και τελειώνει με ένα οποιοδήποτε χαρακτήρα 0 ή περισσότερες
     * φορές.
     *
     * @param s το προς έλεγχο String
     * @return true, αν γίνει match το regex, αλλιώς false.
     */
@   public static boolean isLetterDigitAsWholeWord(String s) {
        return (s != null) && (s.matches(".*\\b\\w\\d\\b.*"));
    }
```

- Η διαφορά με την προηγούμενη διαφάνεια είναι ότι έχουμε προσθέσει το \\b στην αρχή και το τέλος που σημαίνει word boundary άρα ***αναζητούμε λέξεις*** που αποτελούνται από ένα γράμμα και ένα ψηφίο

Strings που κάνουν match είναι για παράδειγμα τα: P ***F4*** , ABC ***D8*** 89! , 55 ***A0***

Ποσοδείκτες

- Όπως το * που είδαμε στο προηγούμενο παράδειγμα, οι ποσοδείκτες εφαρμόζονται αμέσως μετά από τον χαρακτήρα, ή το regex (ή το group)
  - **?** Μηδέν ή μία εμφάνιση
  - **+** Μία ή περισσότερες εμφανίσεις (occurrence indicator)
  - * 0 ή περισσότερες εμφανίσεις
  - {n} n εμφανίσεις
  - {n, m} Μεταξύ n και m εμφανίσεων
  - {n, } Τουλάχιστον n εμφανίσεις
  - { ,m} Όχι περισσότερες από m εμφανίσεις

Έγκυρη διεύθυνση e-mail


```java
package gr.aueb.elearn.ch3;

/**
 * Περιέχει μία μέθοδο που ελέγχει αν
 * ένα String αναπαριστά μία έγκυρη email
 * διεύθυνση που τελειώνει σε com ή gr.
 *
 * @author A. Androutsos
 * @see #isValidEmail(String)
 */
public class RegExEmail {

    /**
     * Ελέγχει αν ένα String αναπαριστά μία έγκυρη email
     * διεύθυνση.
     *
     * @param s το προς έλεγχο String
     * @return true, αν γίνει match το regex, αλλιώς false.
     */
    public static boolean isValidEmail(String s) {
        return (s != null) && (s.matches("\\w*\\.?\\w+@\\w+\\.(com|gr)"));
    }
}
```

- Ελέγχει αν το s περιέχει έγκυρη διεύθυνση e-mail που τελειώνει σε com ή gr και έχει τη μορφή 0 ή περισσότεροι χαρακτήρες, 0 ή μία τελεία –επειδή η τελεία είναι ειδικός χαρακτήρας για να πάρουμε την τελεία την κάνουμε escape \\.
- Στη συνέχεια πάλι χαρακτήρες ένα ή περισσότερους, μετά το @, μετά πάλι ένα ή περισσότερους χαρακτήρες , μετά τελεία και μετά com ή gr

- Το com|gr το βάζουμε σε παρενθέσεις επειδή πρόκειται για ξεχωριστό regex group

Επαλήθευση format


```java
package gr.aueb.elearn.ch3;

/**
 * @author A. Androutsos
 * @see #isTwoLettersDashSevenDigits(String)
 */
public class RegExTwoLettersDashSevenDigits {

    /**
     * Ελέγχει αν ένα String ξεκινάει με δύο γράμματα
     * κεφαλαία στη συνέχεια έχει παύλα και μετά 7 ψηφία.
     *
     * @param s το προς έλεγχο String
     * @return true, αν γίνει match το regex, αλλιώς false.
     */
    public static boolean isTwoLettersDashSevenDigits(String s) {
        return (s != null) && (s.matches("[A-Z]{2}-\\d{7}"));
    }
}
```

- Κάνουμε validate ένα string ότι περιέχει τη μορφή δύο γράμματα κεφαλαία ακολουθούμενα από παύλα – ακολουθούμενα από 7 αριθμούς

Split με RegEx


```java
package gr.aueb.elearn.ch3;

/**
 * Επιδεικνύει την String.split με
 * Regular Expression.
 *
 * @author A. Androutsos
 */
public class RegExSplit {

    /**
     * Κάνει parse το String και το επιστρέφει
     * τα tokens που διαχωρίζονται με ένα ή
     * περισσότερα κενά διαστήματα.
     */
    public static void main(String[] args) {
        String s = "Java    Advanced Level";

        String[] tokens = s.split("\\s+");

        for (String token : tokens) {
            System.out.println(token);
        }
    }
}
```

```
"C:\Progr
Java
Advanced
Level

Process f
```

- Στην split δίνουμε το regex \\s+ που σημαίνει το να διαχωρίσει το string σε tokens που με βάση ένα ή περισσότερα κενά διαστήματα

replaceAll με RegEx


```java
package gr.aueb.elearn.ch3;

/**
 * Επιδεικνύει την String.split με
 * Regular Expression.
 *
 * @author A. Androutsos
 */
public class RegExReplaceAll {

    /**
     * Κάνει parse το String και το επιστρέφει
     * τα tokens που διαχωρίζονται με ένα ή
     * περισσότερα κενά διαστήματα.
     */
    public static void main(String[] args) {
        String s = "Java    Advanced Level";

        s = s.replaceAll("\\s+", " ");
        System.out.println(s);
    }
}
```

```
"C:\Program Files\Ja
Java Advanced Level

Process finished wit
```

- Αντικαθιστούμε τα κενά με ένα κενό

Αμοιβαία ανταλλαγή με backreference


```
1   package gr.aueb.elearn.ch3;
2
3   public class RegExBackReference {
4
5       public static void main(String[] args) {
6
7           String s = "Thanassis Androutsos";
8
9           s = s.replaceAll("(.*)\\s(.*)", "$2 $1");
10
11          System.out.println(s);
12      }
13  }
```

```
"C:\Program Files\Jav
Androutsos Thanassis

Process finished with
```

- Groups ορίζουμε μέσα σε παρενθέσεις, ώστε να μπορούμε να αναφερθούμε σε αυτά αργότερα (backreference)
- Τα $1 , $2 αναφέρονται στα δύο groups

Pattern - Matches

- Με την matches δεν μπορούμε να χειριστούμε πολλαπλές εμφανίσεις ενός pattern μέσα σε ένα string
- Χρησιμοποιούμε τις κλάσεις Pattern / Matcher
- Η κλάση Pattern μας δίνει τη δυνατότητα να κάνουμε compile ένα regex μία φορά και να το χρησιμοποιήσουμε όσες φορές θέλουμε και επομένως είναι πολύ αποδοτικότερη

Μετατροπή Ημερομηνίας


```java
package gr.aueb.elearn.ch3;

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

/**
 * Μετατρέπει ημερομηνίες από mm/dd/yyyy
 * σε dd/mm/yyyy.
 *
 * @author A. Androutsos
 */
public class PatternDate {

    public static void main(String[] args) {
        String s = "12/05/1996";

        Pattern pattern = Pattern.compile("(\\d{2})/(\\d{2})/(\\d{4})");
        Matcher matcher = pattern.matcher(s);

        //Attempts to match the entire region against the pattern.
        if (matcher.matches()) {
            String month = matcher.group(1);
            String day = matcher.group(2);
            String year = matcher.group(3);
            String out = day + "/" + month + "/" + year;
            System.out.println(out);
        } else {
            System.out.println("Pattern not matches");
        }
    }
}
```

- Μέσα σε παρενθέσεις είναι τα groups ώστε στη συνέχεια να τα επεξεργαστούμε

Greedy Ποσοδείκτης


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class PatternGreedy {
7
8      public static void main(String[] args) {
9          String s = "login:thanos;pass:123456;";
10
11         Pattern pattern = Pattern.compile(".*;");
12
13         Matcher matcher = pattern.matcher(s);
14
15         while (matcher.find()) {
16             System.out.println(matcher.group(0));
17         }
18     }
19 }
```

```
"C:\Program Files\Java\jdk
login:thanos;pass:123456;

Process finished with exi
```

- Το match γίνεται μέχρι το τελευταίο ; Δηλ. As much as possible (greedy – άπληστο match)
- Η find() δίνει το next match

- Στο pattern δεν έχουμε ορίσει groups, οπότε όλο το match θεωρείται group(0) ή σκέτο group()

Reluctant (non-greedy) ποσοδείκτης


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class PatternReluctant {
7
8      public static void main(String[] args) {
9          String s = "login:thanos;pass:123456;";
10
11         Pattern pattern = Pattern.compile(".*?;");
12
13         Matcher matcher = pattern.matcher(s);
14
15         while (matcher.find()) {
16             System.out.println(matcher.group(0));
17         }
18     }
19 }
```

```
"C:\Program Fi
login:thanos;
pass:123456;

Process finish
```

- Το match γίνεται μέχρι το πρώτο ; Δηλαδή
  As little as possible (non-greedy – όχι άπληστο match)

Start of String


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class PatternStartOfString {
7
8      public static void main(String[] args) {
9          String s = "B31:F456:D55";
10
11         Pattern pattern = Pattern.compile("^[A-Z].*?:");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             System.out.println(matcher.group());
16         }
17     }
18 }
```

```
"C:\
B31:

Proce
```

- Με το ^ έξω από τα [] εννοούμε την αρχή του string
- Επομένως ψάχνουμε για pattern που ξεκινάει από την αρχή με A-Z και φτάνει μέχρι τα πρώτα :

Αφαίρεση συνόλου


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class PatternNegation {
7
8      public static void main(String[] args) {
9          String s = "A31:D456:X55:H78:";
10
11         Pattern pattern = Pattern.compile("[A-Z&&[^ADX]].*?:");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             System.out.println(matcher.group());
16         }
17     }
18 }
```

```
"C:\Pr
H78:
```

Δίνει H78: μιας και τα υπόλοιπα substrings εμπίπτουν στο ^ADX

Με το ^ μέσα στα [ ] εννοούμε άρνηση (negation)

- Το [A-Z&&[^ADF]] δίνει την αφαίρεση από το A-Z του ^ADX

Τομή συνόλων


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class RegExIntersection {
7
8      public static void main(String[] args) {
9          String s = "A31:D456:X55:H78:";
10
11         Pattern pattern = Pattern.compile("[A-Z&&[ADX]].*?:");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             System.out.println(matcher.group());
16         }
17     }
18 }
```

```
"C:\Pr
A31:
D456:
X55:

Proces
```

Δίνει τα παραπάνω μιας και το H78: δεν εμπίπτει στο ADX

- Το [A-Z&&[ADF]] δίνει την τομή του A-Z και του ADX, άρα δίνει τα ADX

Ένωση Συνόλων


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class RegExUnion {
7
8      public static void main(String[] args) {
9          String s = "A31:D456:X55:H78:";
10
11         Pattern pattern = Pattern.compile("[A-C[M-Z]].*?:");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             System.out.println(matcher.group());
16         }
17     }
18 }
```

```
"C:\
A31:
X55:

Proc
```

Δίνει τα παραπάνω μιας και τα D465, H78: δεν εμπίπτουν στο [A-C[M-Z]]

- Το [A-Z[M-Z]] δίνει την ένωση του A-C και του M-Z

Groups & Capturing

- Όπως είδαμε στα προηγούμενα παραδείγματα, βάζοντας μία έκφραση μέσα σε παρενθέσεις δημιουργούμε capturing groups δηλ. named groups στα οποία μπορούμε στη συνέχεια να αναφερθούμε (backreference), όπως στο παράδειγμα με τις ημερομηνίες
- Ο τρόπος αναφοράς είναι με αριθμούς ξεκινώντας από το 1 - group(1)

Named Groups

- Σε ένα group μπορούμε να δώσουμε και ένα όνομα. Όταν δώσουμε όνομα σε ένα group μπορούμε στη συνέχεια να αναφερθούμε στο group με το όνομά του αλλά εξακολουθητικά και με την αρίθμηση που αποδίδεται αυτόματα

```
1   package gr.aueb.elearn.ch3;
2
3   import java.util.regex.Matcher;
4   import java.util.regex.Pattern;
5
6   /**
7    * Μετατρέπει ημερομηνίες από mm/dd/yyyy
8    * σε dd/mm/yyyy με τη χρήση named groups.
9    *
10   * @author A. Androutsos
11   */
```

Δείτε τη συνέχεια στην επόμενη διαφάνεια

Named Groups


```
12  public class PatternDateNamed {
13
14      public static void main(String[] args) {
15          String s = "12/05/1996";
16
17          Pattern pattern = Pattern.compile("(?<month>\\d{2})/(?<day>\\d{2})/(?<year>\\d{4})");
18          Matcher matcher = pattern.matcher(s);
19
20          //Attempts to match the entire region against the pattern.
21          if (matcher.matches()) {
22              String month = matcher.group("month");
23              String day = matcher.group("day");
24              String year = matcher.group("year");
25              String out = day + "/" + month + "/" + year;
26              System.out.println(out);
27          } else {
28              System.out.println("Pattern not matches");
29          }
30      }
31  }
```

Named groups

Αναφορές στα named groups Backreferences

- Παρατηρήστε τα named groups και στη συνέχεια τα backreferences με ***matcher.group("groupName")***

Capturing vs non-capturing groups

- Τα groups που είδαμε με τον τρόπο που τα ορίζουμε μέσα σε παρενθέσεις είναι capturing groups δηλ. καταναλώνουν τους χαρακτήρες και μπορούμε να αναφερθούμε σε αυτά, άρα αποτελούν μέρους του αποτελέσματος του regex
- Υπάρχουν όμως περιπτώσεις που θα θέλαμε μόνο να ελέγξουμε την ύπαρξη ενός regex group χωρίς το group να καταναλώσει χαρακτήρες, αλλά απλώς να μας πει αν υπάρχει ή όχι αυτό το pattern στο string. Αυτά τα groups ονομάζονται ***non-capturing και δεν μπορούμε να αναφερθούμε σε αυτά με backrefernce***

Non-capturing groups

- Μπορούμε να δηλώσουμε ένα non-capturing group ως εξής:
- (Hello)(?:World)+
- Αν έχουμε τα String HelloWorld, τότε γίνεται match αλλά αποδίδεται στο match μόνο το Hello

Capturing groups - HelloWorld


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class RegExNonCapturingHelloWorld {
7
8      public static void main(String[] args) {
9          String s= "HelloWorld HelloWorldWorld";
10
11         Pattern pattern = Pattern.compile("(Hello)(World)+");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             for (int i=1; i <= matcher.groupCount(); i++) {
16                 System.out.print(matcher.group(i) + " ");
17             }
18             System.out.println();
19         }
20     }
21 }
```

```
"C:\Program F
Hello World
Hello World

Process finis
```

- Σε κάθε match (εδώ έχουμε 2 match) έχουμε groupCount groups (εδώ έχουμε δύο groups)

- Όπως βλέπετε γίνεται capture τόσο το HelloWorld, όσο και το HelloWorldWorld, και αποδίδεται το Hello World και στις δύο περιπτώσεις

Non-capturing groups - HelloWorld


```
1  package gr.aueb.elearn.ch3;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class RegExNonCapturingHelloWorld {
7
8      public static void main(String[] args) {
9          String s= "HelloWorld HelloWorldWorld";
10
11         Pattern pattern = Pattern.compile("(Hello)(?:World)+");
12         Matcher matcher = pattern.matcher(s);
13
14         while (matcher.find()) {
15             for (int i=1; i <= matcher.groupCount(); i++)
16             System.out.println(matcher.group(i));
17         }
18     }
19 }
```

```
"C:\Pr
Hello
Hello

Proces
```

- Το ***?:*** κάνει match το Hello World αλλά δεν κάνει capture το World (κάνει το group non-capturing)

- Όπως βλέπετε ενώ γίνεται capture τόσο το HelloWorld, όσο και το HelloWorldWorld, ***αποδίδεται μόνο το Hello*** και στις δύο περιπτώσεις

Non-capturing groups

- Τα non-capturing groups λειτουργούν ως **zero length assertions** δηλαδή μπορεί να κάνουν match αλλά δεν καταναλώνουν του χαρακτήρες και δεν αποδίδουν το match (zero-length match)
- Αν θέλουμε να δηλώσουμε ότι ένα regex **ακολουθείται από ένα non-capturing group το οποίο δεν κάνει match** χρησιμοποιούμε το **?=**
- **Το ?: κάνει match αλλά δεν κάνει capture**
- **Το ?= δεν κάνει ούτε match ούτε capture**
- Για παράδειγμα το **^(?=.*?[a-z]).{6,}$** σημαίνει ένα string που περιέχει τουλάχιστον 6 οποιουσδήποτε χαρακτήρες. Το group είναι non-capturing και κάνει assert ότι το string περιέχει τουλάχιστον ένα μικρό γράμμα
- Το ? πριν το [a-z] κάνει το regex reluctant και βελτιστοποιεί την αναζήτηση

Παράδειγμα


```java
package gr.aueb.cf.ch20.regexapp;

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

/**
 * Gets a string containing 8 or more chars and asserts that the string includes
 * at least one lowercase letter, one uppercase letter and one digit.
 */
public class LowercaseValidationApp {

    public static void main(String[] args) {

        String s = "0Sqwer6ty";

        Pattern pattern = Pattern.compile("(?=.*?[a-z])(?=.*?[A-Z])(?=.*?[0-9])^.{8,}$");
        Matcher matcher = pattern.matcher(s);

        while (matcher.find()) {
            System.out.println(matcher.group());
        }
    }
}
```

- Ελέγχει αν το string περιέχει ένα τουλάχιστον πεζό γράμμα, ένα τουλάχιστον κεφαλαίο και ένα τουλάχιστον ψηφίο

Μικρή εργασία

- Έστω ότι ο χρήστης δίνει ένα password.
- Επαληθεύστε ότι έχει τη μορφή:
  - Τουλάχιστον 8 χαρακτήρες
  - Τουλάχιστον 1 μικρό γράμμα
  - Τουλάχιστον 1 κεφαλαίο γράμμα
  - Τουλάχιστον 1 αριθμό
  - Τουλάχιστον 1 ειδικό χαρακτήρα #?!@$%^&*-