JAVA – Integrazione RegExp

Metodi Avanzati di Programmazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Docente: Pierpaolo Basile

Il metodo `split` di `String`

- Il metodo `split` della classe `String` suddivide una stringa in tante sottostringhe in base all'espressione regolare passata come parametro
- Per esempio se si vuole suddividere una stringa in base agli spazi:

```
String test1 = "sdfkljsd lkjsd fkl    lksdjf  fslkdjf f  sldkj";
String[] split = test1.split("\\s+");
for (String s : split) {
   System.out.println(s);
}
```

Il metodo `replaceAll` di `String`

- Il metodo `replaceAll` della classe `String` sostituisce tutte le sottostringhe che corrispondono ad una espressione regolare con una stringa passata come parametro
- Per esempio se si vogliono sostituire sequenze di numeri con uno spazio:

```
String test2 = "sdfkljsd824lkjsd76fkl2765lksdjf549fslkdjf";
String tt2 = test1.replaceAll("[0-9]+", " ");
```

Classe `Pattern`

- La classe `Pattern` è una rappresentazione compilata di un'espressione regolare
- Un'espressione regolare, specificata come stringa, deve prima essere compilata in un'istanza di questa classe
- Il modello risultante può quindi essere utilizzato per creare un oggetto `Matcher` capace di fare il matching tra qualsiasi sequenza di caratteri e l'espressione regolare compilata
- Un esempio tipico di invocazione è:

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

Classe `Matcher`

- Esegue le operazioni di matching su una sequenza di caratteri in funzione dell'espressione regolare compilata
- Un matcher viene creato da un pattern invocando il metodo matcher del pattern. Una volta creato, un matcher può essere utilizzato per eseguire tre diversi tipi di operazioni di match:
  - Il metodo **`match`** tenta di abbinare l'intera sequenza di input al pattern
  - Il metodo **`lookingAt`** cerca la sequenza di input corrispondente all'espressione regolare, a partire dall'inizio della stringa
  - Il metodo **`find`** analizza la sequenza di input cercando la sottosequenza successiva che corrisponde all'espressione regolare
- Ognuno di questi metodi restituisce un valore booleano che indica l'esito positivo o negativo. Maggiori informazioni possono essere ottenute interrogando lo stato del matcher

Classe `Matcher` (stato)

- Un matcher trova corrispondenze in un sottoinsieme del suo input chiamato regione. Per impostazione predefinita, la regione contiene tutto l'input del matcher
  - La regione può essere modificata tramite il metodo `region` e interrogata tramite i metodi `regionStart` e `regionEnd`
- Lo stato esplicito di un matcher include gli indici di inizio e fine dell'ultimo match (`start` ed `end`)
- Include anche gli indici di inizio e fine della sottosequenza dell'ultimo match, nonché un conteggio totale (`groupCount`) di tali sottosequenze
  - Per comodità, vengono forniti anche metodi per restituire queste sottosequenze in forma di stringa (`group`)

Utilizzo di Matcher

```
Matcher matcher1 = pattern.matcher("dlkflsASDaslsdSD");
//deve corrispondere l'intera stringa
System.out.println(matcher1.matches());


Matcher matcher2 = pattern.matcher("dlkflsASDaslsdSD 8798767");
//la corrispondenza deve partire dall'inizio ma non è
necessario che corrisponda l'intera stringa
System.out.println(matcher2.lookingAt());
```

Utilizzo di Matcher

```
Matcher matcher3 = pattern.matcher("dlkflsASDaslsdSD");
//cicla su tutti i matching
while (matcher3.find()) {
   System.out.println(matcher3.group() + ": " +
    matcher3.start() + "-" + matcher3.end());
}
```

Utilizzo di Matcher

```
//una sequenza di numeri seguita da 1 o massimo 3 lettere min.
String regexp = "([0-9]+)([a-z]{1,3})";
Pattern pattern2 = Pattern.compile(regexp);
String str = "9843989jf 39203920jie 32122i";
Matcher matcher4 = pattern2.matcher(str);
while (matcher4.find()) {
   //restituisce il numero di gruppi (individuati dalle
   parentesi nella regexp)
   int gc = matcher4.groupCount();
   //il gruppo 0 corrisponde all'intero matching
   for (int i = 0; i <= gc; i++) {
      System.out.println(matcher4.group(i) + ": " +
      matcher4.start(i) + "-" + matcher4.end(i));
   }
   System.out.println();
}
```