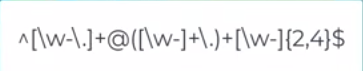
Espressioni regolari

Le espressioni regolari è semplicemente del testo, sono un modo per descrivere o formattare una stringa o un modello e può contenere caratteri o combinazioni di caratteri che hanno un significato speciale.

**Un espressione regolare non è altro che semplice test**o, può contenere caratteri o combinazioni di caratteri che hanno un significato speciale, e vengono chiamati **meta-caratteri**.  
Queste combinazioni **vengono interpretate da un elaboratore** di modelli di espressioni regolari.

Altro termine → le espressioni regolari chiamate RegEx o RE, **non sono altro che strighe**

**che possono essere utilizzare per contrallare se determinati campi vengono formattati correttamente, definendo uno schema, un pattern, una stringa che descriva come dev’essere un altra stringa**, come possiamo immaginare un campo in cui si desidera inserire un indirizzo email, per questo campo sarà applicato un confronto con un espressione regolare di questo tipo,



in modo da capire se è stato formattato correttamente.

Con le espressioni regolari possiamo manipolare un testo di stringhe che contiene numeri e simboli speciali.  
  
Come lavora dietro le quinte il motore del linguaggio di programmazione, queste RE possono essere applicate a qualsiasi linguaggio di programmazione, possiamo dire qualche cenno storico, queste sono state introdotte per la prima volta da clean un matematico, che aveva introdotto le espressioni rapportate agli insiemi, quindi consideriamo che abbiamo il nostro testo, ed immaginiamo che questo sia il nostro sistema applicata una RE, si crea un sotto-insieme a partire dalla formattazione dell RE quindi ad esempio come contrare l’occorrenza delle parole ‘acqua’ in un testo.

**È possibile** verificare che qualcosa sia formattato correttamente.

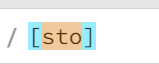
**È possibile** trovare le occorrenze di modelli nel testo.

**È possibile** sostituire le occorrenze corrispondenti dei modelli nel testo.

**È possibile** estrarre dal testo le occorrenze corrispondenti.

Ed inoltre **è possibile** suddivere del testo in base al modello, memorizzano solo alcune sequenze di metacaratteri.

Java supporta le espressioni regolari in diversi modi:   
Esistono classi con metodi che accettano espressioni regolari, stringhe o modelli come parametri, come la classe Scanner, esistono classi nel package java.regex che consentono di implementare le proprie funzionalità, **si tratta di pattern e matcher**.  
  
  
Lo scopo delle regex è quello di definire dei pattern con cui andiamo a strutturare lo schema delle stringhe che vogliamo ricercare all’interno di un testo  
  
  
  
  
  
  
  
Dal sito regex101 possiamo considerare la scelta di selezionare diverse opzioni per le regex:  
  
**global →** ci permette di tenere traccia di tutte le occorrenze per quella stringa, e quindi non si fermerà al primo confronto ma lo effettuerà globalmente su tutto il testo  
**multi-line →** il testo si distribuisce su più righe  
**insensitive →** le regex sono keySentistive impostandolo quando ci sarà un confronto di quella parola in minuscolo, ma la regex è formattata in maiuscolo **allora verrà considerata** rendendo infulente se il testo è minuscolo o maiuscolo

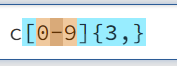
Partendo dalla corince **[]** →  qui scriveremo i caratteri che vogliamo che vengano individuati nel testo, nel senso che la cornice va intesta in questo senso, all’interno elenchiamo tutti i caratteri di cui dev’essere fatto il match cioè di cui deve essere trovata una corrispondenza.  
Quindi verranno individuati non la coppia, ma i singoli caratteri, se abbiamo bisogno di trovare la **coppia** allora togliamo la corinice[]  
  
Abbiamo la possibilità di aggiungere degli intervalli per trovare i match come [a-z] || [a-t] ||

**[a-zA-Z]** → tutte le lettere maiuscole e minuscole  
**[0-9]** → va a controllare se nel testo ci sono numeri  
**[a-zA-Z0-9] →** vengono considerati tutti i caratteri compresi minuscoli e maiuscoli o numeri

Possiamo considerare un carattere speciale che nelle RE viene considerato come una negazione ( **^** )  
  
Un altro carattere speciale ( **?** ), il quale ha un significato **opzionale** ossia che quella parola che stiamo cercando nel testo, può essere 0, **quindi non esserci o esserci al massimo 1 volta**.

( **\*** ) può comparira **da 0 a n** volte

( **+** ) può comparire **1 o più** volte

Abbiamo altri quantificatori per andare a selezionare una parola o carattere che dev’esserci almeno n volte questo lo facciamo tramite le { } in cui adiamo a specificare quante volte deve esserci  
 con la virgola stiamo dicendo che vogliamo un carattere in questo caso ma anche una parola che inizi con la c, che abbiamo dei numeri, e **considerare solo caratteri o parole che abbiamo dalle 3 cifre a salire**, per aggiungere degli intervalli **c[0-9]{1,5}**

Un design pattern è una soluzione generale ad un problema ricorrente, mentre un algoritmo è una sequenza specifica di istruzioni per risolvere un problema computazionale. Un design pattern può quindi essere visto come una guida per la progettazione di algoritmi e software, mentre un algoritmo è l’implementazione specifica di una soluzione ad un problema.  
design pattern è un modo per risolvere i problemi  
  
La potenza delle regex non è quella di combinare gli alias con i quantificatori ma ci permette di definire un pattern, infatti loro operano a livello di caratteri ossia di simboli non esiste nelle regex la differenza tra un carattere dell’alfabeto e un altro cosi come tra un carattere e un numero è questo è il motivo di risultati particolari  
  
**La classe Pattern** permette di definire una regex, ossia rappresenta un modello di espressione regolare  
Inizializzi un oggetto pattern tramite il metodo statico ‘compile’ e gli passi come parametro del metodo la stringa regex che vuoi utilizzare per le corrispondenze, e qui può essere specificata l’opzione Pattern.MULTILINE

**La classe Matcher** ci permette di cercare il modello di una stringa definita dalla regex, questa classe è associata a un oggetto pattern e viene utilizzata per eseguire corrispondenze specifiche di una data stringa. **Crei un oggetto matcher invocando il metodo matcher() sull’oggetto pattern** e passando la stringa di cui desideri eseguire le corrispondenze