**Package**

* Java ofera un package pentru regex: java.util.regex
* Aceasta are 3 clase de baza:

- Pattern - este un compiled regex. Nu ofera public constructori, ci e creat prin metoda statica compile(), care returneaza un Patterm object. Ea preia ca argument un regular expression.

- Matcher - interpreteaz[ pattern si efectueaza match operations pe un String. Tot nu are constructori publici. Obtinem un Matcher obj prin apelarea la matcher() a unui obiect de tip Pattern

- PatternSyntaxException - obiect returnat, fiind unchecked exception, care indica ca exista o eroare de sintax in regular expression pattern

**Utilizare**

* Intai cream obiectul Pattern, ce va contine regex

Pattern pattern = Pattern.compile("regexp");

* Apoi cream Matcher, obiect ce va fi creat de Pattern si va contine mesajul de verificat

Matcher matcher = pattern.matcher("mesaj de verificat")

* Folosim metda matcher.find(), care returneaza true de fiecare data cand se gaseste un string ce corespunde la regexp

Pattern pattern = Pattern.*compile*("..Text");  
Matcher matcher = pattern.matcher("MyText");  
  
System.*out*.println(matcher.find());

* Totusi, find face ca sa se cauta un singur match. Dar, daca o vom apela, ia va cauta din nou mai departe:

Pattern pattern = Pattern.*compile*("..Text");  
Matcher matcher = pattern.matcher("MyText and a Text");  
  
while(matcher.find())  
 System.*out*.println("found");

S-a afisat:

found

found

**Meta Characters**

* Meta Characters afecteaza modul in care un pattern este matched. Ele adauga o logica la pattern de cautat. Java ofera difertie meta characters:

. - puct inseamna un caracter oricare, obligatoriu

Pattern pattern = Pattern.*compile*("...Text");  
Matcher matcher = pattern.matcher("MyText");

Nu se potriveste! Avem 3 puncte, dar My sunt 2 caractere

[ - creaza o lista de elemente ce pot fi continute

Pattern pattern = Pattern.*compile*("[yet]");  
Matcher matcher = pattern.matcher("MyText");  
  
while(matcher.find())  
 System.*out*.println("Gasit");

Deci avem:

Gasit

Gasit

Gasit

* { - numar de caractere

a[0,3] - 0 de a sau cel mult 3

a[5] - exact 5 de a

* \d - vezi mai jos
* ^ - arata ca anume elementele din interval nu trebuie continute
* ^ - mai arata ca caracterele trebuie sa fie la inceput de cuvant

^User

[^abc] - deci anume a b c nu trebuie sa se gaseasca

* - arata un interval [a-e] de la a la e toate literele

[a-zA-Z] - avem deci 2 intervale, toate literele mici de la a la z si cele mari de la A la Z

Putem folosi si numere [1-5] sau [1-3[7-9]], deci de la 1 la 3 si de la 7 la 9, deci fara 4,5,6

* =
* $ - caracterele trebuie sa fie anume la final

dog$

* \* - zero, unul sau mai multe
* ? - un caracter urmeaza sau nici unul
* + - exact unul
* && intersectia

[1-6&&[3-9]] - deci se vor folosi doar elementele lor comune, adica 3,4,5,6

**Predefined characters (\)**

* \d - echivalent cu [0-9]

Pattern pattern = Pattern.*compile*("\\d");  
Matcher matcher = pattern.matcher("User123");  
  
while(matcher.find())  
 System.*out*.println("Gasit");

GASIT GASIT GASIT

* \D - echivalent cu [^0-9]
* \s - gaseste white spice
* \S - gaseste orice numai nu whitespace
* \w - echivalent cu [a-zA-Z\_0-9], adica word characters
* \W - echivalent cu [^a-zA-Z\_0-9]

**Atentie cu BandingResult!!!**

* **Daca** apare vreo problema la validare, adica cand dam @Valid, se va arunca o exceptie ce contine toate datele legate de fielduri si mesaje
* Daca avem ca parametru un BindingResult, el va lua in el exceptia aparuta, si va colecta datele pe baza lui, ca fieldurile, si mesajele de eroare.
* Metoda se va executa tot oke, daca avem un BindingResult, chiar si cu tot cu erori de validare.
* Gen:

@PostMapping("/add")  
public String add(@RequestBody @Valid User user, BindingResult bindingResult) {  
 System.*out*.println(user);  
 return "Success!";  
}

**Chiar de apar erori de validare, exceptia aruncata este preluata de BindingResult, si oricum metoda se executa si se returneaza "Success"**

* Dar, daca nu avem un BindingResult ca parametru, totul se schimba. Daca validarea e cu probleme, exceptia nu se mai duce in BindingResult, ci se arunca direct,si va continee in ea deja un obiect de tip BindingResult si asa metoda nu mai e executata si nici return nu se mai face. Gen:

@PostMapping("/add")  
public String add(@RequestBody @Valid User user) {  
 System.*out*.println(user);  
 return "Success!";  
}

**Aici nu avem BindingResult ca parametru, si daca ceva nu e bine in validare, se arunca o exceptie ce are un BindingResult in e si ea nu este luata de nimeni, deci metoda nu se va executa si nici return nu se va face. Nu cel dorit de noi.**

* Tipul exceptiei aruncate la validare gresita este:

MethodArgumentNotValidException

Asa ar arata un exception handler in @ControllerAdvice, cand BindingResult nu este ca parametru pus:

@ExceptionHandler  
public BindingResult bindingResult(MethodArgumentNotValidException exception){  
 return exception.getBindingResult();  
}

Observam ca are metoda **getBindingResult()**

* **In @Controller si in @RestController, BindingResult e tratat putin diferit. In Controller, el se va trimite la pagina chiar si de il punem ca parametru,in RestController, el nu se trimite daca e pus ca parametru. Ideea e ca pe pagina trebuie sa afisam exceptiile cumva si de aceea el tot se trimite.**