* Le folosim pentru a face verificari mai avansate.
* Pentru a adauga, folosim @Pattern(regexp=””, message=”mesaj eroare”); deasupra la membrul din Clasa
* Ceea ce este intre „” de la regexp este anume si Regular Exception

@Pattern(regexp = "^[a-zA-Z0-9]{5}",message = "Invalid Postal Code")  
private String postalCode;

**Integer Field Required**

* Utilizarea datelor primitive nu e o idee buna, deoarece ele nu sunt Null la crearea obiectelor si nici nu pot fi efectuate multe operatii asupra lor. E mai bine sa utilizam Wrapper Class.
* De exemplu, daca avem un field pentru int, adica data primitiva, o valaorea goala din form va fi ehivalenta cu Null si nu va putea sa-i trimita lui Null
* Mai mult ca atat, formele lucreaza cu String, si String cu wrapper class poate interactiona. Orice introducem intr-un field e luat ca String, si apoi convertit in ce e necesar, dar nu poate fi convertit in int fara Integer.
* Pentru Wrapper Class, nu mai folosim @NotBlank, dar @NotNull, caci nu are de unde avea spatii libere ori lipsa, caci “” nu e Integer, ci String

@NotNull(message = "Can't be empty")  
@Min(value = 12, message="You're too young")  
@Max(value = 120, message = "Invalid age")  
private Integer age;

**Handle String input for Integer field**

* Daca in field bazat pe membrul de sus vom introduce litere, de ex, vom avea o eroare legata de conversii ce va aparea pe pagina, si e cam urata.

**Create Custom Error Message**

* Cream un fisier .properties
* Cream pentru fiecare eroare atribute, respectand regula generala:

typeMismatch.customer.age="Invalid Age"



TypeError.SpringModelAttributeName.FieldName = “message”, unde “ “ sunt optionale

Dupa, adaugam in context:

<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  
 <property name="basename" value="properties.messages"/>  
</bean>

Lasam to**tul e**xact asa! Doar value modificam. **Atentie la locatia fisierului .properties!Atentie si ca nu punem .properties la numele fisierului, caci automat se va pune!**



* **BindingResult,** parametrul din requestMapping, ne poate oferi multe informatii despre validarile esuate.
* System.*out*.println(bindingResult);

De exemplu, daca introducem in fieldul legat de

@NotNull()  
@Min(value = 12, message="You're too young")  
@Max(value = 120, message = "Invalid age")  
private Integer age;

Cifre, ne va aparea o eroare legata de conversie, una mare.E greu sa stim ce fel de eroare e asta, ca sa o putem utiliza in fisierul .proeprties pentru mesage. Insa obiectul de tip BindingResult ne ajuta. De ex, daca introducem litere la age, el va arata:

org.springframework.validation.BeanPropertyBindingResult: 1 errors

Field error in object 'client' on field 'age': rejected value [dfdffd]; codes [typeMismatch.client.age,typeMismatch.age,typeMismatch.java.lang.Integer,typeMismatch]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [client.age,age]; arguments []; default message [age]]; default message [Failed to convert property value of type 'java.lang.String' to required type 'java.lang.Integer' for property 'age'; nested exception is java.lang.NumberFormatException: For input string: "dfdffd"]

Acelea in rosu sunt tipuri de erori ce s-au depistat. Ele pot fi de la generale la particulare. Daca una din ele e suprascrisa, adica o scriem noi in fisierul .properties, mesajul nostru va fi afisat in loc de cel default afisat de MVC validation messages.

Putem avea errori supra scrise:

* particulare, adica pentru membrul din obiectul nostru typeMisatch.client.age
* generale, pentru orice membru de oriunde typeMismatch.age
* pentru erori misMatch legate de Integer - typeMismatch.java.lang.Integer
* foarte generale – pentru orice eroare de tip typeMismatch typeMismatch