**Create Custom Java Annotation**

* Putem crea propriile anotatii, ca @CourseCode

Pasi:

1. Cream @CourseCode annotation **intr-un fisier .java**.

Pentru a o crea, vom folosi @interface, care arata ca cream o annotatie:

(1)@Constraint(validateBy = CourseCodeConstraintValidator.class)

(2)@Target( { ElementType.METHOD, ElementType.FIELD} )

(3)@Retention(RetentionPolicy.RUNTIME)

Public @interface CourseCode { }

1. Aici aratam care clasa are regulile de validare
2. Aici aratam unde putem aplica anotatia. ElementType.METHOD inseamna ca pe o metoda, si .FIELD pe un field.
3. @Retention face ca anotatia data sa fie retinuta in fisierul Java al clasei. Ea arata cat de mult timp ea va fi retinuta sau folosita. In cazul la RetentionPolicy.RUNTIME, ea va fi procesata la runtime, adica pastrata in compiled Java byte code. Deci, poate fi folosita liber cand codul ruleaza. Si totodata, poate fi folosita in orice fisier din proiect.

@Constraint(validatedBy = CourseCodeConstraintValidator.class)  
@Target({ElementType.*METHOD*, ElementType.*FIELD*})  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface CourseCode{  
  
}

1. Configuram anotatia:

{

Public String value() default “LUV”;

Public String message() default “Must start with LUV”;

}

Aici spunem ca anotatia are un parametru numit value, si daca userul nu ofera o value, se va folosi default value “LUV” si un alt parametru message, care daca nu e scris de user, are valoarea default setata. Acesti parametri sunt cei utilizati cand folosim anotatia, asa cum gen Bean(message=””) are message ca parametru, asa si noi putem crea ale noastre.

public @interface CourseCode{  
  
 public String value() default "LUV";  
  
 public String message() default "must start with LUV";  
  
 public Class<?>[] groups() default {};  
  
 public Class<? extends Payload>[] payload() default {};  
}

* public Class<?>[] groups() default {}; - este pentru grupurile la care va apartine anotatia, daca vom hotara sa adaugam group validator.

Acum anotatia va putea fi folosita cu parametrii value si message:

@CourseCode(value=””, message=””);

Facem publici membrii din anotatie, ca sa-i putem accesa din clasa ce re regulile de finite.

1. Cream clasa de validare. Anume ea e helper class care contine bussiness rules pentru validare.

public class CourseCodeConstraintValidator implements ConstraintValidator<CourseCode,String> {  
  
 private String coursePrefix;  
 @Override  
 public void initialize(CourseCode theCourseCode) {  
 coursePrefix = theCourseCode.value();  
 }  
  
 @Override  
 public boolean isValid(String theCode, ConstraintValidatorContext theConstraintValidatorContext) {  
boolean result ;  
  
if(theCode != null) {  
 result = theCode.startsWith(coursePrefix);  
}  
else {  
 result = true;  
}  
return result;

}

* <CourseCode, String> - anotatia si tipul de date de validat. Nu neaparat String, poate fi orice alt tip de obiect.
* NU FOLOSIM MODELUL DAT DE METODE DE AUTO OVERRIDE
* String coursePrefix – e valoarea data de anotatie, adica @CourseCode(value=”de aici”)
* Void initialize – e folosita pentru a seta valoarea din value a anotatiei in clasa. Aici e anume obiectul anotatiei creat de noi.
* isValid method:
* String theCode – textul introdus de user in HTML form.Atentie, daca intre <Anotatie, NuString> nu punem un String, nici in metoda asta primul parametru nu va fi string, ea e bazata pe Generic
* ConstraintValidatorContext theConstraintValidatorContext – e folosit pentru a da mesaje de erori aditionale.
* Metodata data e apelata la RUNTIME de Spring MVC

1. Adaugam anotatia, care e numele dupa @interface, unde avem nevoie de ea asa, cu parametrii setati de noi,

@CourseCode(value=”regula”, message=”text error”)

Private String courseCode;

**Annotation for other objects**

* Putem crea anotatii nu numai pentru obiecte de tip String, dar oricare.
* Obiectul pe care vrem anotatia sa il prea is verifice il punem aici:

public class ValidateNameValidator implements ConstraintValidator<ValidateName,OurObject> {



* **Deci, daca vom avea:**



**@ValidateName**

**OurObject myObject;**

acest obiect va fi preluat de anotatie, si deci evident, metoda isValid va avea un alt parametru ca primul:

@Override  
public boolean isValid(OurObject value, ConstraintValidatorContext context) {

Ce obiect e pus ca al 2 intre <>, acela si se pune in isValid method ca prim parametru.

**Putem verifica si Array de String**

1. Change the value entry to an array of Strings:

// define default course code

public String[] value() default {"LUV"};

**2. Update CourseCodeConstraintValidator.java to validate against array of strings**

Change the field for coursePrefixes to an array

private String[] coursePrefixes;

Update the isValid(...) method to loop through the course prefixes. In the loop, check to see if the code matches any of the course prefixes.

@Override

public boolean isValid(String theCode,

ConstraintValidatorContext theConstraintValidatorContext) {

boolean result = false;

if (theCode != null) {

//

// loop thru course prefixes

//

// check to see if code matches any of the course prefixes

//

for (String tempPrefix : coursePrefixes) {

result = theCode.startsWith(tempPrefix);

// if we found a match then break out of the loop

if (result) {

break;

}

}

}

else {

result = true;

}

return result;

}

**3. Update Customer.java to validate using array of strings**

@CourseCode(value={"TOPS", "LUV"}, message="must start with TOPS or LUV")

private String courseCode;