Annotations in Java

Annotation is nothing but a piece of information added to a data or object. While creating an annotation in java , two things are important, one is **@Target and @Retention**.

**Retention Policy**

Retention policy indicates how long annotations with the annotated type are to be retained. In case of no retention policy, the default retention policy will be **RetentionPolicy.CLASS**. It means how long the annotated type will remain.

* **RetentionPolicy.SOURCE**: Retained in source code and discarded during the compile. Example: **@Override, @SuppressWarnings**
* **RetentionPolicy.CLASS**: Retained at the class and discarded during class loading. Useful when doing bytecode-level post-processing. **The default retention policy type is CLASS**.
* **RetentionPolicy.RUNTIME**: Retained at runtime and never discarded. The annotation should be available for reflection at runtime. Example: **@Deprecated, @Autowired**

**@Target** : It specifies where it should be applicable whether the annotated type is applicable to Class, Constructor, Method etc.

* + **ElementType**.**PACKAGE** - Package declaration
  + **ElementType**.**TYPE** - Class, Interface, Annotation, Enum declaration
  + **ElementType**.**FIELD** - Field declaration (includes enum constants)
  + **ElementType**.**CONSTRUCTOR** - Constructor declaration
  + **ElementType**.**METHOD** - Method declaration
  + **ElementType**.**PARAMETER** - Parameter declaration
  + **ElementType**.**LOCAL\_VARIABLE** - Local variable declaration
  + **ElementType**.**ANNOTATION\_TYPE** - Annotation type declaration

An Example of custom annotation at the class level is given below.

@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*TYPE*)  
public @interface Vegetarian {  
 boolean value() default true;  
}

@Vegetarian(value = false)  
public class Lion {  
 private String name;  
  
 get()/set() methods

}

public class Test {  
  
 public static boolean isVegetarian(Object obj) {  
 boolean flag = false;  
 Class<?> clazz = obj.getClass();  
 if (clazz.isAnnotationPresent(Vegetarian.class)) {  
 Vegetarian veg = clazz.getAnnotation(Vegetarian.class);  
 flag = veg.value();  
 }  
 return flag;  
 }  
  
 public static void main(String[] args) {  
 Lion lion = new Lion();  
 lion.setName("African Lion");  
 boolean flag = *isVegetarian*(lion);  
 System.*out*.println("Flag: "+flag);  
  
 }  
}

**An example of Custom Annotation at the Field Level**

public class Employee {  
 @Max(45)  
 private int age;  
 private String name;  
  
 public Employee(String name, int age) {  
 this.age = age;  
 this.name = name;  
 }  
}

**@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*FIELD*)**public **@interface** Max {  
 int value();  
}

public class Processor {  
 public void process(Object obj) throws IllegalAccessException {  
 Field[] fields = **obj.getClass().getDeclaredFields()**;  
 for(Field field : fields) {  
 **field.setAccessible(true);** **if(field.isAnnotationPresent(Max.class))** {

**Max max = field.getDeclaredAnnotation(Max.class);**  
 //**Max max = field.getAnnotation(Max.class); // also you can write**  
 **int maxVal = max.value();**  
 System.*out*.println("Max Value: " + maxVal); *//45* int actualValue = field.getInt(obj);  
 System.*out*.println("Actual Value: "+actualValue); *// 55* }  
 }  
 }  
}

public class Test {  
 public static void main(String[] args) throws IllegalAccessException {  
 Employee emp = new Employee("John", 55);  
 Processor p = new Processor();  
 p.process(emp);  
 }  
}

Eatable and NonEatable products like Bread, Cake, Soap, Washing Powder

Edible or Non-Edible oil etc

**How to create your own Custom Annotation in Spring Boot**

Here we will follow two ways to approach.

* 1. Simply create an annotation and provide the logic to validate.
  2. Create annotation and use it using Spring AOP.

**Simple Custom Annotation in Spring Boot**

A user can’t deposit more than 50K as cash. Base upon this we will create an annotation called ***CashValidation***.

To create an annotation for Spring Boot, 4 things are required to be mentioned in the annotation.

1. A default error message.
2. Define groups like **public** Class<?>[] groups() **default** {};
3. Define Payload like **public** Class<? **extends** Payload>[] payload() **default** {};
4. Annotation must have a validation class like @Constraint(validatedBy = CashValidator.**class**)

The above are mandatory while creating annotations for Spring Boot. Besides, you can provide application specific value.

**Creating an annotation**

@Documented

@Retention(***RUNTIME***)

@Target({ ***FIELD***, ***PARAMETER*** })

**@Constraint(validatedBy = CashValidator.class)**

**public** **@interface** CashValidation {

**public** **int** value(); // Specific to application

// Default error message

**public String message() default "Cash deposit amount can't be greater";**

// represents group of constraints

**public Class<?>[] groups() default {};** // Mandatory

// represents additional information about annotation

**public Class<? extends Payload>[] payload() default {};** // Mandatory

// public Class<?>[] payload() default {}; // Mandatory, it can be written

}

**Validation Class for define annotation**

**public** **class** CashValidator **implements** ConstraintValidator<CashValidation, Integer> {

@Override

**public** **boolean** isValid(Integer value, ConstraintValidatorContext context) {

**boolean** flag = **true**;

**if** (value > 50000) {

flag = **false**;

**context.disableDefaultConstraintViolation();**

**context.buildConstraintViolationWithTemplate(value+" greater than 50K")**

**.addConstraintViolation();**

}

**return** flag;

}

}

If you want t build your own application specific error message, you have to define like this.

**context.disableDefaultConstraintViolation();**

**context.buildConstraintViolationWithTemplate(value+" greater than 50K") .addConstraintViolation();**

**How to use Annotation**

@Data

**public** **class** Customer {

**private** String actNo;

**private** String name;

**@CashValidation(value = 50000)**

**private** **int** amount;

}

**How to define in Controller**

@PutMapping(path="/cash")

**public** ResponseEntity<String> depositCash**(@Valid** @RequestBody Customer customer) {

System.***out***.println("Customer depositing ..."+customer.getAmount());

**return** **new** ResponseEntity<String>("Cash Deposited successfully.", HttpStatus.***OK***);

}

Since we are using custom annotations, we must provide the **@Valid** annotation before the object.

**Define ExceptionHandler**

@ControllerAdvice

**public** **class** AppExceptionHandler {

@ExceptionHandler(MethodArgumentNotValidException.**class**)

@ResponseBody

**public** ResponseEntity<List> processUnmergeException(**final** **MethodArgumentNotValidException** ex) {

List list = **ex.getBindingResult().getAllErrors().stream()**

**.map(fieldError -> fieldError.getDefaultMessage())**

**.collect(Collectors.*toList*());**

**return** **new** ResponseEntity<>(list, HttpStatus.***BAD\_REQUEST***);

}

}

**Create annotation and use it using Spring AOP - Define an annotation**

@Retention(***RUNTIME***)

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

**public** **@interface** AppLogger {

}

Define the aspect

@Aspect

@Component

@Slf4j

**public** **class** AppLoggerAspect {

@Autowired

**private** HttpServletRequest request;

@Around("@annotation(AppLogger)")

**public** Object logRequest(**ProceedingJoinPoint joinPoint**) **throws** Throwable {

***log***.debug("Incoming Request: {}", request.getRequestURI());

**CodeSignature codeSignature = (CodeSignature) joinPoint.getSignature();**

***log***.debug("Invoked Method Name : {}", codeSignature.getName());

// Extracting method arguments name

String[] parameterNames = codeSignature.getParameterNames();

// Extracting method arguments value

Object[] args = joinPoint.getArgs();

**for** (**int** i = 0; i < parameterNames.length; i++) {

***log***.debug("Argument Name: {}, Argument value: {}", parameterNames[i], args[i]);

}

Object obj = joinPoint.proceed();

**return** obj;

}

}

**How to use**

@RestController

**public** **class** CashDepositController {

@AppLogger

@PutMapping(path="/cash")

**public** ResponseEntity<String> depositCash(@Valid @RequestBody Customer customer) {

System.***out***.println("Customer depositing ..."+customer.getAmount());

**return** **new** ResponseEntity<String>("Cash Deposited successfully.", HttpStatus.***OK***);

}

}