

Subscribe to DeepL Pro to edit this document.  
Visit www.DeepL.com/Pro for more information.



Memento Annotations Spring, JPA and Hibernate

Author: François-Xavier COTE[) (fxcote@clelia.frRévision:](mailto:fxcote@clelia.fr) 1.17.1

JPA, Hibernate

Validation

Spring

|  |  |  |
| --- | --- | --- |
| Background | Annotation | Description |
| JPA,  Hibernate ORM  *( to be used in business classes by importing the annotations of the javax.persistence package)* | @Entity | Defines a persistent POJO class  @Entity  **public class** Enquete {}  Hibernate will create an Investigation table. Each object of type Enquete will have a correspondence with a record in the Enquete table.  Forgetting this annotation on a business class gives an exception IllegalArgumentException :Not a managed type: class fr.telecom\_st\_etienne.fx.enquetes.business.enquiry |
|  | @Table | Defines the name of the table associated with the persistent POJO class. |
|  |  | @Entity |

|  |  |  |
| --- | --- | --- |
|  |  | @Table(name="users" )  **public class** User {} |
| @NamedQuery | Declares a HQL query that will be accessible by all classes of the project.  @Entity  @NamedQuery(name="Player.findAll" , query="SELECT j FROM Player j" )  **public class** Player {} |
| @Id | Specifies that the annotated attribute will give rise to a column that will be the primary key of the associated table.  @Id  **private Long** id; |
| @GeneratedValue | Indicates how to generate the identifier. It is possible to specify a generation strategy thanks to a strategy attribute.  @Id @GeneratedValue(strategy=GenerationType. ***IDENTITY***) **private Long** id;  strategy=GenerationType.IDENTITY uses an identity specific to the DBMS (auto\_increment of MySQL, Oracle sequence)  strategy=GenerationType.AUTO: Hibernate takes care of the generation by creating a sequence unique to the schema.  strategy=GenerationType.SEQUENCE: The generation of the primary key is based on a sequence strategy=GenerationType.TABLE: The generation of the primary key uses values stored in a table named hibernate\_sequences by default. |
| @Inheritance | Allows to specify the strategy in the way to create tables related to the notion of inheritance strategy=InheritanceType.TABLE\_PER\_CLASS : duplicates data to avoid join operations  strategy=InheritanceType.SINGLE\_TABLE: mother class and daughter classes are represented by a single table.  strategy=InheritanceType.JOINED: mother and daughter classes are each represented by a |

|  |  |  |
| --- | --- | --- |
|  |  | table. In the table representing the parent class, you will find the attributes common to all daughter classes. In the table representing the daughter class, we will find the information specific to the daughter class.  If the strategy is not specified, the SINGLE\_TABLE strategy is used. The annotation @Inheritance is placed on the parent class.  @Entity @Inheritance(strategy=InheritanceType. ***SINGLE\_TABLE***) **public abstract class** Utilisateur {} |
| @DiscriminatorColumn | Defines the name of the discriminating column in the table representing the parent class. In the absence of this annotation, the discriminating column is named DTYPE.  This annotation is only used when the inheritance strategy is SINGLE\_TABLE.  This column will contain the name of the child class whose object is an instance of  @Entity @Inheritance(strategy=InheritanceType. ***SINGLE\_TABLE***) @DiscriminatorColumn(name="TypeUtilisateur" ) public **abstract class Utilisateur** {} |
| @Basic | Declares in the simplest way a correspondence between a Java attribute and an eponymous column in the associated table.  @Basic  **private int** quantite; |
| @Column | Details how to create the base column  The main attributes of the @Column annotation are:   * name: name of the column in the associated table * nullable: allows to allow or forbid null in the column, to allow: nullable=true * unique: adds a uniqueness constraint on the column * length: specifies the length of the column (for attributes of type String), by default: 255   @Column(unique=true, null=false, length=250) |

|  |  |  |
| --- | --- | --- |
|  |  | **private** String name; |
| @Lob | Indicates that the column must contain long text (also allows to store binary data)  @Lob  **private** String script Hook; |
| @Temporal | Indicates that it is a temporal data (Date or Calendar) @Temporal(TemporalType.DATE) : creates a column of type date @Temporal(TemporalType.) in the table.TIME) : creates in the table a column of type time @Temporal(TemporalType.TIMESTAMP) : creates in the table a column of type datetime Without specifying @Temporal, Hibernate adds a column datetime for an attribute of type java.util.Date  @Temporal(TemporalType. ***DATE***)  **private** DateDeNaissance; |
| @OneToOne | Indicates a bijection with the other class.  Example: a mayor is the mayor of only one city. A city has only one mayor |
| @ManyToOne | Indicates that several objects of the class will be associated to one and the same object of the other class. |

|  |  |  |
| --- | --- | --- |
|  |  | @Entity  **public class** Article { |
| @Id @GeneratedValue(strategy=GenerationType. ***IDENTITY***) **private Long** id; |
| **private** String designation; **private int** nbPoints**; private int** stock**;** |
| @ManyToOne  **private** Category category;  } |
| Basically, in the Article table, a category\_id column will be added, it is a foreign key to the Category table. |
| If you forget the @ManyToOne annotation on a business attribute, you get the following error: Caused by: org.hibernate.MappingException: Could not determine type for: fr.telecom\_st\_etienne.fx.commandes\_cadeau.business.Categorie, at table: Article, for columns: [org.hibernate.mapping.Column(category)] |
| @OneToMany | Indicates that an object of the class will be associated with several objects of the other class. For the mappedBy attribute, we specify the name of the object in the other class.  In the example on the previous page: a category has a list of items. An article is associated to a category, so there is a category attribute in the class Article. It is the name of this attribute that is written in mappedBy.  @OneToMany(mappedBy="category" )  **private** List<Article> articles;  As a general rule, as soon as a class contains a list of business objects, it will be annotated @OneToMany. Same for an object of type Set. |

|  |  |  |
| --- | --- | --- |
|  |  | Remember to specify the type of recovery using the fetch attribute, by default the fetch is FetchType.LAZY.  Example: if you want to retrieve all the items in a category, you will choose the fetch type EAGER:  @OneToMany(mappedBy="category" , fetch=FetchType. EAGER)  **private** List<Article> articles;  When trying to retrieve the items without first using the EAGER fetch type we get the following exception:  Caused by: org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: fr.telecom\_st\_etienne.fx.belair.business.TypeAppareil.avions, could not initialize proxy - no Session  Please note: you cannot have more than one FetchType to EAGER per class. If you use the EAGER FecthType twice in the same class you get the following exception:  Caused by: org.hibernate.loader.MultipleBagFetchException: cannot simultaneously fetch multiple bags  To automatically delete all items associated with a category that needs to be deleted, the cascade attribute must be specified:  @OneToMany(mappedBy="category" , cascade = CascadeType. REMOVE)  **private** List<Article> articles; |
| @ManyToMany | Indicates a list in each class. Between the two classes there are two directed associations (going in opposite directions) with a multiplicity 0..\* for each association.  Example: a survey has several websites. One website is used by several surveys.  Basically a mapping table will be created.  The mappedBy attribute must only appear on one of the two annotations otherwise the AnnotationException is lifted with the following message: Illegal use of mappedBy on |

|  |  |  |
| --- | --- | --- |
|  |  | both sides of the relationship  In the example below, the record of an object of type Survey will "feed" well the mapping table because in the SitePartner class we have the following code:  @ManyToMany(mappedBy="sitesPartners" )  **private** Set<Internet Survey> surveys;  Hibernate will fill the sitesPartners list with what has been selected in the multiple list (select). |
| IdClass | Fill in the mapping class |
| @JoinColumn | Renames the name of the column on which there is a foreign key constraint  @ManyToOne |

|  |  |  |
| --- | --- | --- |
|  |  | @JoinColumn(name=" idEnquete")  **private** Enquete enquete; |
| @Transient | Allows to define the attribute in the class but not in the associated table.  @Transient  **private float** price incl. VAT; |
| Validation  *( to be used in business classes by importing the annotations of the javax package. validation or org.hibernate.vali dator.constraints*  *)* | @NotBlank | Guarantees that the value of the attribute (of type String) does not contain an empty string.  @NotBlank(message="Please name the survey" ) @Column(unique = true, null = **false,** length = 250) private String name; |
| @NotNull | Ensures that the value of the attribute contains a reference to another object.  @NotNull(message="Please specify the city where you live" )  **private** City; |
| @NotEmpty | Ensures that the string or list is not empty  @NotEmpty(message="The criteria list cannot be empty" )  **private** List<Criteria> criteria; |
| @Min | Guarantees that the value of the attribute is greater than or equal to a Min value.  @Min(value=100, message="The price cannot be less than 100 euros" )  **private float** price; |
| DecimalMin | Guarantees that the value of the attribute is greater than or equal to a decimal value Min.  @DecimalMin(value="100.5", message="The price cannot be less than  100.50 euros" )  **private BigDecimal** price; |
| @Max | Guarantees that the value of the attribute is less than or equal to a Max value.  @Max(value=1000, message="The price can't be more than 1000 euros" )  **private float** price; |
| DecimalMax | Guarantees that the value of the attribute is less than or equal to a decimal value Max |

|  |  |  |
| --- | --- | --- |
|  |  | @Max(value="1000.99", message="The price cannot be higher than 1000.99 euros" )  **private BigDecimal** price; |
| @CreditCardNumber | Ensures that the attribute value contains a valid credit card number. To do this Hibernate uses Luhn's algorithm: https:[//fr.wikipedia.org/wiki/Formule\_de\_Luhn](https://fr.wikipedia.org/wiki/Formule_de_Luhn)  @NotNull(message="Please enter your card number" ) @CreditCardNumber(message="The card number is not valid" ) private String numero;  Examples of valid card numbers: 371449635398431, 411111111111111111111111 |
| @SafeHtml | Ensures that the value of the attribute contains only benevolent HTML code  @SafeHtml(message="Thank you for producing kind HTML code" )  **private** String bio;  NB: Deprecated annotation: the Hibernate community would like another community to manage this annotation. |
| @URL | Ensures that the value of the attribute corresponds to a valid URL  @URL(message="Please specify a valid URL" )  **private** String url; |
| @Range | Ensures that the value of the attribute is between the min and max terminals.  @Range(min=15, max=28, message="Please specify a temperature between 15 and 28 degrees Celsius" )  **private** Float temperatureSought; |
| @Past | Guarantees that the value of the attribute (of type Date) contains a date in the past.  @Past(message="Your date of birth must be in the past" )  **private** DateDeNaissance; |

|  |  |  |
| --- | --- | --- |
|  | @Future | Guarantees that the attribute (of type Date) contains a date in the future.  @Future(message="The planning date must be in the future" ) @DateTimeFormat(pattern = "dd/MM/yyyyy" )  **private** Date dateEnquiry; |
| @Pattern | Guarantees that the value of the attribute of type String respects a regular expression specified in the regexp attribute.  @Pattern(regexp="^[A-Za-z]+$" , message="The reference must contain only letters" )  **private** String reference;  To help you write your regex: <http://jkorpela.fi/perl/regexp.html> and <https://regex101.com/> |
| @Email | Guarantees that the value of the attribute of type String contains a valid email address.  @Email(message="The email address entered is not valid" )  **private** String email; |
| @Size | Guarantees that the number of characters of the attribute respects the constraints given in parameter  @Size(min=5, message="The password must contain at least 5 characters" )  **private** String motDePasse; |
| Valid | Requires the validation of the object's data against the constraints expressed in the object's class.  Example: when the method is invoked, the validation of the data of the object under investigation is requested:  **public** ModelAndView enregistrerEnquetePost(@Valid @ModelAttribute("enquete" ) Enquete enquete, BindingResult result) {} |
| Spring | @Autowired | Asks Spring to automatically inject an object in the object of the class. This annotation is less and less used because the Spring community suggests to ask Spring to inject dependencies in the class constructor. |

|  |  |  |
| --- | --- | --- |
|  |  | @Autowired  **private** EnqueteDao enqueteDao; |
| @Controller | Declares a class that will process HTTP requests (see coordination layer)  @Controller  **public class** EnqueteController {} |
| @RestController | Declares a class that will handle REST requests.  @RestController  **public class** EnglishBattleControllerWS {} |
| @Service | Declares a class of service  @Service  **public class** EnqueteServiceImpl **implements** EnqueteService {}  Forgetting the @Service annotation on the service class results in the error :  Parameter 2 of constructor in fr.telecom\_st\_etienne.fx.enquete.controller.EnqueteController required a bean of type 'fr.telecom\_st\_etienne.fx.enquetes.service.EnqueteService' that could not be found. |
| @Repository | Declares a CAD class (Data Access Object: class able to communicate with the database)  @Repository  **public class** JoueurDaoImpl **implements** JoueurDao {}  Thanks to Spring Data, this annotation is less and less used because to implement a CAD you just have to declare an interface that inherits from JpaRepository.  Spring Data implements CAD for us (see SimpleJpaRepositoryImpl) |
| @Query | Declares the HQL request associated with the CAD interface method. |

|  |  |  |
| --- | --- | --- |
|  |  | @Query("from Enquete where prix>9000" )  **public** List<Enquete> findMostExpensiveSurveys();  Note: using Spring Data, this annotation is only rarely used, see: [https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query- creation](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation) |
| @Param | Declares Java method parameters to be used as HQL parameters  @Query("from Question where enquete.idEnquete=:eid" )  **public** List<Question> findByIdEnquete(@Param("eid" ) Long idEnquete); |
| Transactional | Defines a transactional context (typically on service class methods) To define a read-only transaction: @Transactional(readOnly = true)  @Service @Transactional  **public class** EnqueteServiceImpl **implements** EnqueteService {} |
| @RequestMapping | Defines a correspondence between one (or more) URL(s) and a method of the controller. In other words, this annotation declares the correspondence between a controller method and the URL(s) it supports.  @RequestMapping(value = { "/index" , "/" }, method = RequestMethod. ***GET*** )  **public** ModelAndView home(@RequestParam Map<String, String> map) {} |
| @GetMapping | Defines a correlation between one (or more) URLs and a method in the Spring controller. This mapping applies only if the HTTP method is of type Get.  The following EnqueteGet method will be invoked when a browser goes to the URL [http://localhost/enquetes](http://localhost/enquete) or when a hyperlink with an href attribute equal to ["http://localhost/enquetes"](http://localhost/enquetes) is clicked |

|  |  |  |
| --- | --- | --- |
|  |  | @GetMapping("/enquetes" )  **public** ModelAndView surveysGet() {} |
| @PostMapping | Defines a correlation between one (or more) URLs and a method in the Spring controller. This correspondence applies only if the HTTP method is of type Post.  @PostMapping("/filterTheInvestigations" )  **public** ModelAndView filter Enquetes(@RequestParam Map<String, String> map) {}  Note: The method below is invoked when the submit button of the form below is clicked:  < form action="*filterTheInvestigations" method="post"* >  < input type="*text" name="NAME" >*  < input type="*submit" value="Filter*" >  </form> |
| @PutMapping | Defines a correlation between one (or more) URLs and a method in the Spring controller. This mapping applies only if the HTTP method is of type Put.  @PutMapping("/updateinquiry" , produces="application/json" )  **public** Enquete majEnquete(@RequestParam Map<String, String> map) {} |
| @DeleteMapping | Defines a correlation between one (or more) URLs and a method in the Spring controller. This mapping applies only if the HTTP method is of type Delete.  @PutMapping("/deleteEnquete" , produces="application/json" )  **public** boolean supprimerEnquetes(@RequestParam Map<String, String> map) {} |
| @RequestParam | Indicates that the method (controller) parameter comes from the request object. Spring will convert the object to the expected type.  @PostMapping(value = "/question" )  **public** ModelAndView questionPost(@RequestParam(name="ID\_ENQUETE" ) Long idEnquete, @RequestParam(name="LIBELLE" ) String libelle) {} |
| @ModelAttribute | Annotation used on a parameter of a controller method. It specifies an object of a business class returned by a view. |

|  |  |  |
| --- | --- | --- |
|  |  | @PostMapping("/registration" )  **public** ModelAndView inscriptionPost(@Valid @ModelAttribute("personne" ) Personne personne, BindingResult result) {  **if** (result.hasErrors()) {  ModelAndView mav = inscriptionGet(); mav.addObject("person" , person); return mav**;**  } **else** {  personneService.enregistrerPersonne(personne); ModelAndView mav = **new** ModelAndView("merciInscription" ); return mav**;**  }  }  Note: the view (the registration.jsp file) must contain a form:form tag as follows:  < form:form action="*inscription" method="post"* modelAttribute="personne" >  As a reminder: @Valid annotation allows you to delegate to Spring the validation work on the annotated object. In other words, all validation constraints expressed in the business classes (present in the business package) will be checked by Spring. |
| @SessionAttributes | Ask Spring to store in session the attributes of an object sent to an HTML form.  @SessionAttributes( tasks ) |
| DateTimeFormat (pattern  = "dd/MM/yyyy") | Allows you to define a Date type attribute in a business class with a format specified in the attribute.  DateTimeFormat(pattern = "dd/MM/yyyyyy" )  **private** Date dateEnquiry; |
| @PageableDefault | Sets the default pagination and sort settings  **public** ModelAndView surveysGet(@PageableDefault(value = 10, sort = "dateOrder" ) Pageable pageable) {}  Please note: in services and CAD, all methods with a Pageable parameter type |

|  |  |  |
| --- | --- | --- |
|  |  | must return a page |
| @InitBinder | Annotation of a method "showing" Spring how to get a business object from its id or how to transform an object into another object. The data conversion (in English binding) is done by Spring thanks to the @InitBinder annotated methods.  @InitBinder  **public void** initBinder(WebDataBinder binder) {  // Teach Spring to convert a String to Date SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyyy-MM-dd" ); dateFormat.setLenient(false);  binder.registerCustomEditor(Date. **class**, **new** CustomDateEditor(dateFormat, **true**));  // Teach Spring to convert a city id to a city object binder.registerCustomEditor(Ville. **class**, "ville" , new **PropertyEditorSupport**() {  @Override  **public void** setAsText(String id) { setValue((id.equals("" )) ? null :  cityService.recuperateCity(*ParseLong*((String) id)));  }  });  // Teach Spring to convert a list of ids into a list of interests binder.registerCustomEditor(List. **class**, "interests" , new  CustomCollectionEditor(List. **class)** { @Override  **public** Object convertElement(Object objet) { Long id = Long. *parseLong*((String) objet); **return** interetService.recupererInteret(id);  }  });  }  For the PropertyEditorSupport class, refer to the javadoc: https:[//docs.oracle.com/javase/7/docs/api/java/beans/PropertyEditorSupport.html](https://docs.oracle.com/javase/7/docs/api/java/beans/PropertyEditorSupport.html) |
| @PostConstruct | Annotation on a method that will be invoked automatically when Spring has injected all the |

|  |  |  |
| --- | --- | --- |
|  |  | annotated objects @Autowired  @PostConstruct  **public void** addInitialData() {} |
| @PathVariable | Refers to a variable found in the url. The code below retrieves the survey id from the url. Example of url: http://localhost:8080/enquete/4 (in this case the survey id will have the value 4)  @GetMapping(value="/enquete/{idEnquete}" , produces="application/json" )  **public** Survey SurveyGet(@PathVariable Long idEn survey) {} |
| Bean | Declares a method whose returned object will be managed by the Spring container.  Bean  **public** EmbeddedServletContainerFactory servletContainer() {} |
| @Configuration | Declares a configuration class (an object of this class replaces the Spring configuration xml file (often called spring-servlet.xml))  @Configuration  **public class** KanbanConfiguration {} |
| @Value | Retrieves the value of a variable declared in the Spring Boot application.properties file. |
| @ResponseBody | Specifies that the method return corresponds to what will be sent to the HTTP client.  @GetMapping(value="/fichierExcel" , produces="application/vnd.ms-excel" ) public @ResponseBody **byte[] fichierExcelGet(**@RequestParam(name="ID" ) Long idFichierExcel) throws IOException {} |
| Secure | Restricts access to users with the role specified in parameter  @Secured("ROLE\_ADMIN" ) @GetMapping("/enquetes" )  **public** ModelAndView surveysGet() {} |
| @Scheduled | Programs the invocation of the method automatically. This annotation is inspired by cron. The example below invokes the method programEmails() every Thursday at 5pm: |

|  |  |  |
| --- | --- | --- |
|  |  | @Scheduled ( 00 00 17 \* \* THU )  **public** void programmerEmails() {} |
| @Primary | Indicates the implementation class that Spring should use when dealing with an object declared with an interface (to be used when several classes implement the same interface).  @Primary  **public class** EnqueteServiceImpl **implements** EnqueteService {} |
| @Qualify | Allows to distinguish two parameters of the same type in a method  @RequestMapping(value = { "/index" , "/" }, method = RequestMethod. ***GET*** )  **public** ModelAndView home(  @Qualifier("aerodrome" ) @PageableDefault(value = 10, sort = "name" ) PageableAerodrome,  @Qualifier("flight" ) @PageableDefault(value = 4, sort = "priceInEuros" ) Pageable pageableVol) {  ModelAndView mav = **new** ModelAndView("index" ); mav.addObject("pageDAerodromes" ,  aerodromeService.recuperateAerodromes(pageableAerodrome)); mav.addObject("pageOfFlights" , volService.recuperateFlights(pageableFlight)); return mav**;**  }  Sample URL: http://localhost:8080/index?vol\_page=4&aerodrome\_page=1 |