

# Mémento Annotations Spring, JPA et Hibernate

Auteur: François-Xavier COTE ([fxcote@clelia.fr](mailto:fxcote@clelia.fr))

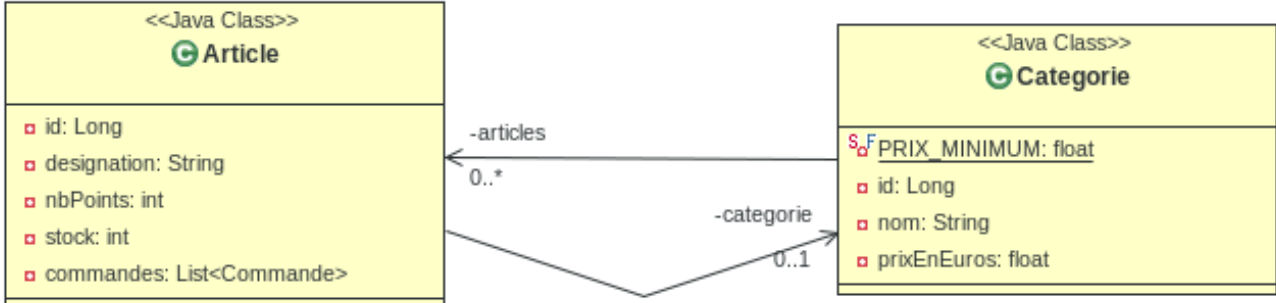
Révision: 1.17.1

JPA, Hibernate	Validation	Spring
----------------	------------	--------

Contexte	Annotation	Description
<b>JPA, Hibernate ORM</b> <i>( à utiliser dans les classes métier en important les annotations du package <code>javax.persistence</code> )</i>	@Entity	<p>Définit une classe POJO persistante</p> <pre>@Entity public class Enquete {}</pre> <p>Hibernate va créer une table Enquete. Chaque objet de type Enquete aura une correspondance avec un enregistrement de la table Enquete.</p> <p>En oubliant cette annotation sur une classe métier on obtient une exception <code>IllegalArgumentException :Not a managed type: class fr.telecom_st_etienne.fx.enquetes.business.Enquete</code></p>
	@Table	<p>Définit le nom de la table associée à la classe POJO persistante</p> <pre>@Entity</pre>

		<pre>@Table(name="utilisateurs") public class Utilisateur {}</pre>
	@NamedQuery	<p>Déclare une requête HQL qui sera accessible par toutes les classes du projet</p> <pre>@Entity @NamedQuery(name="Joueur.findAll", query="SELECT j FROM Joueur j") public class Joueur {}</pre>
	@Id	<p>Précise que l'attribut annoté va donner lieu à une colonne qui sera la clé primaire de la table associée</p> <pre>@Id private Long id;</pre>
	@GeneratedValue	<p>Indique comment générer l'identifiant. Il est possible de préciser une stratégie de génération grâce à un attribut strategy.</p> <pre>@Id @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id;</pre> <p>strategy=GenerationType.IDENTITY utilise une identité propre au SGBD (auto_increment de MySQL, sequence d'Oracle)  strategy=GenerationType.AUTO: Hibernate se charge de la génération en créant une séquence unique au schéma  strategy=GenerationType.SEQUENCE: La génération de la clé primaire se base sur une séquence  strategy=GenerationType.TABLE: La génération de la clé primaire utilise des valeurs stockées dans une table qui se nomme par défaut hibernate_sequences</p>
	@Inheritance	<p>Permet de préciser la stratégie dans la manière de créer les tables liées à la notion d'héritage</p> <p>strategy=InheritanceType.TABLE_PER_CLASS : duplique les données pour éviter les opérations de jointure</p> <p>strategy=InheritanceType.SINGLE_TABLE : classe mère et classes filles sont représentées par une table unique.</p> <p>strategy=InheritanceType.JOINED : classe mère et classes filles sont représentées chacune par une</p>

	<p>table. Dans la table représentant la classe mère, on trouvera les attributs communs à toutes les classes filles. Dans la table représentant la classe fille, on trouvera les informations propres à la classe fille.</p> <p>Si la stratégie n'est pas précisée, la stratégie SINGLE_TABLE est utilisée.</p> <p>L'annotation @Inheritance se place sur la classe mère.</p> <pre>@Entity @Inheritance(strategy=InheritanceType.SINGLE_TABLE) public abstract class Utilisateur {}</pre>
@DiscriminatorColumn	<p>Définit le nom de la colonne discriminante dans la table représentant la classe mère.</p> <p>En l'absence de cette annotation, la colonne discriminante se nomme DTYPE.</p> <p>Cette annotation sert uniquement lorsque la stratégie d'héritage est SINGLE_TABLE.</p> <p>Cette colonne contiendra en toutes lettres le nom de la classe fille dont l'objet est une instance</p> <pre>@Entity @Inheritance(strategy=InheritanceType.SINGLE_TABLE) @DiscriminatorColumn(name="TypeUtilisateur") public abstract class Utilisateur {}</pre>
@Basic	<p>Déclare de la manière la plus simple une correspondance entre un attribut Java et une colonne éponyme dans la table associée</p> <pre>@Basic private int quantite;</pre>
@Column	<p>Détaille la manière de créer la colonne en base</p> <p>Les principaux attributs de l'annotation @Column sont:</p> <ul style="list-style-type: none"> <li>- name: nom de la colonne dans la table associée</li> <li>- nullable: permet d'autoriser ou d'interdire null dans la colonne, pour autoriser: nullable=true</li> <li>- unique: ajoute une contrainte d'unicité sur la colonne</li> <li>- length: précise la longueur de la colonne (pour les attributs de type String), par défaut: 255</li> </ul> <pre>@Column(unique=true, nullable=false, length=250)</pre>

		<code>private String nom;</code>
	@Lob	Indique que la colonne doit contenir un texte long (permet aussi de stocker des données binaires)  <code>@Lob</code> <code>private String scriptAccroche;</code>
	@Temporal	Indique qu'il s'agit d'une donnée temporelle (Date ou Calendar) @Temporal(TemporalType.DATE) : crée dans la table une colonne de type date @Temporal(TemporalType.TIME) : crée dans la table une colonne de type time @Temporal(TemporalType.TIMESTAMP) : crée dans la table une colonne de type datetime Sans préciser @Temporal, Hibernate ajoute une colonne datetime pour un attribut de type java.util.Date  <code>@Temporal(TemporalType.DATE)</code> <code>private Date dateDeNaissance;</code>
	@OneToOne	Indique une bijection avec l'autre classe. Exemple: un maire est maire d'une seule ville. Une ville a un seul maire
	@ManyToOne	Indique que plusieurs objets de la classe vont être associés à un seul et même objet de l'autre classe   <pre> classDiagram     class Article {         id: Long         designation: String         nbPoints: int         stock: int         commandes: List&lt;Commande&gt;     }     class Categorie {         id: Long         nom: String         prixEnEuros: float         PRIX_MINIMUM: float     }     Article "0..*" -- "0..1" Categorie : -articles, -categorie </pre>

		<pre> @Entity public class Article {      @Id     @GeneratedValue(strategy=GenerationType.IDENTITY)     private Long id;      private String designation;     private int nbPoints;     private int stock;      @ManyToOne     private Kategorie categorie; } </pre> <p>En base, dans la table Article, une colonne categorie_id sera ajoutée, c'est une clé étrangère vers la table Kategorie.</p> <p>En oubliant l'annotation @ManyToOne sur un attribut métier, on obtient l'erreur suivante :          Caused by: org.hibernate.MappingException: Could not determine type for: fr.telecom_st_etienne.fx.commandes_cadeau.business.Kategorie, at table: Article, for columns: [org.hibernate.mapping.Column(categorie)]</p>
	@OneToMany	<p>Indique qu'un objet de la classe va être associé à plusieurs objets de l'autre classe. Pour l'attribut mappedBy, on précise le nom de l'objet dans l'autre classe.</p> <p>Dans l'exemple de la page précédente: une catégorie comporte une liste d'articles. Un article est associé à une catégorie, il y a donc un attribut categorie dans la classe Article. C'est le nom de cet attribut que l'on écrit dans mappedBy.</p> <pre> @OneToMany(mappedBy="categorie") private List&lt;Article&gt; articles; </pre> <p>En règle générale, dès qu'une classe comporte une liste d'objets métier, elle sera annotée @OneToMany. Idem pour un objet de type Set.</p>

		<p>Bien penser à préciser le type de récupération en utilisant l'attribut fetch, par défaut le fetch est FetchType.LAZY.</p> <p>Exemple: si l'on souhaite récupérer d'emblée tous les articles d'une catégorie, on choisira le type de fetch EAGER:</p> <pre>@OneToMany(mappedBy="categorie", fetch=FetchType.EAGER) private List&lt;Article&gt; articles;</pre> <p>En essayant de récupérer les articles sans avoir au préalable utiliser le fetch type EAGER on obtient l'exception suivante:  Caused by: <a href="#">org.hibernate.LazyInitializationException</a>: failed to lazily initialize a collection of role: fr.telecom_st_etienne.fx.belair.business.TypeAppareil.avions, could not initialize proxy - no Session</p> <p>A noter: on ne peut pas avoir plus d'un FetchType à EAGER par classe. En utilisant deux fois le FetchType EAGER dans une même classe on obtient l'exception suivante:  Caused by: org.hibernate.loader.MultipleBagFetchException: cannot simultaneously fetch multiple bags</p> <p>Pour effacer automatiquement toutes les articles associées à une catégorie qui doit être effacée, l'attribut cascade doit être précisé:</p> <pre>@OneToMany(mappedBy="categorie", cascade = CascadeType.REMOVE) private List&lt;Article&gt; articles;</pre>
	@ManyToMany	<p>Indique une liste dans chaque classe. Entre les deux classes il y a deux associations dirigées (allant dans des directions opposées) avec une multiplicité 0..* pour chaque association.</p> <p>Exemple: à une enquête correspond plusieurs sites internet. Un site internet est utilisé par plusieurs enquête.</p> <p>En base une table de mapping sera créée.</p> <p>L'attribut mappedBy doit figurer uniquement sur une des deux annotations sinon l'exception AnnotationException est levée avec le message suivant: Illegal use of mappedBy on</p>

		<p>both sides of the relationship</p> <p>Dans l'exemple ci-dessous, l'enregistrement d'un objet de type Enquete "alimentera" bien la table de mapping car dans la classe SitePartenaire on a le code suivant:</p> <pre>@ManyToMany(mappedBy="sitesPartenaires") <b>private</b> Set&lt;EnqueteInternet&gt; <b>enquetes</b>;</pre> <p>Hibernate va remplir la liste sitesPartenaires avec ce qui a été choisi dans la liste multiple (select)</p> <div><h3>Enquete Internet</h3><div><input type="text" value="Nom"/></div><div><input type="text" value="22/05/2019"/></div><div><input type="text" value="2000.0"/></div><div><div>Veuillez sélectionner un theme ▼</div><div>Veuillez sélectionner un ou plusieurs sites partenaires www.clelia.fr www.lemonde.fr</div></div><div><div>Enregistrer</div><div>Annuler</div></div></div>
--	--	---

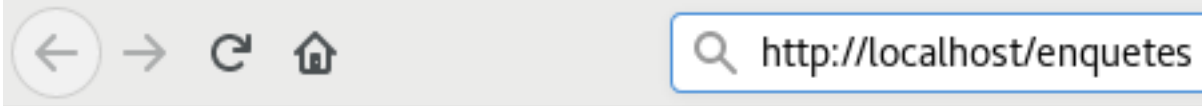
		<pre>@JoinColumn(name="idEnquete") private Enquete enquete;</pre>
	@Transient	<p>Permet de définir l'attribut dans la classe mais pas dans la table associée</p> <pre>@Transient private float prixTTC;</pre>
<b>Validation</b> ( à utiliser dans les classes métier en important les annotations du package <code>javax.validation</code> ou <code>org.hibernate.validator.constraints</code> )	@NotBlank	<p>Garantit que la valeur de l'attribut (de type String) ne contient pas une chaîne vide</p> <pre>@NotBlank(message="Merci de donner un nom à l'enquête") @Column(unique = true, nullable = false, length = 250) private String name;</pre>
	@NotNull	<p>Garantit que la valeur de l'attribut contient bien une référence vers un autre objet</p> <pre>@NotNull(message="Merci de préciser la ville où vous résidez") private Ville ville;</pre>
	@NotEmpty	<p>Garantit que la chaîne de caractère ou la liste n'est pas vide</p> <pre>@NotEmpty(message="La liste de critères ne peut être vide") private List&lt;Critere&gt; criteres;</pre>
	@Min	<p>Garantit que la valeur de l'attribut est supérieure ou égale à une valeur Min</p> <pre>@Min(value=100, message="Le prix ne peut pas être inférieur à 100 euros") private float prix;</pre>
	@DecimalMin	<p>Garantit que la valeur de l'attribut est supérieure ou égale à une valeur décimale Min</p> <pre>@DecimalMin(value="100.5", message="Le prix ne peut pas être inférieur à 100.50 euros") private BigDecimal prix;</pre>
	@Max	<p>Garantit que la valeur de l'attribut est inférieure ou égale à une valeur Max</p> <pre>@Max(value=1000, message="Le prix ne peut pas être supérieur à 1000 euros") private float prix;</pre>
	@DecimalMax	<p>Garantit que la valeur de l'attribut est inférieure ou égale à une valeur décimale Max</p>



		<pre>@Max(value="1000.99", message="Le prix ne peut pas être supérieur à 1000.99 euros") private BigDecimal prix;</pre>
@CreditCardNumber	<p>Garantit que la valeur de l'attribut contient bien un numéro de carte de crédit valide. Pour ce faire Hibernate utilise l'algorithme de Luhn : <a href="https://fr.wikipedia.org/wiki/Formule_de_Luhn">https://fr.wikipedia.org/wiki/Formule_de_Luhn</a></p> <pre>@NotNull(message="Merci de renseigner votre numéro de carte") @CreditCardNumber(message="Le numéro de la carte n'est pas valide") private String numero;</pre> <p>Exemples de numéro de carte valide : 371449635398431, 4111111111111111</p>	
@SafeHtml	<p>Garantit que la valeur de l'attribut contient uniquement du code HTML bienveillant</p> <pre>@SafeHtml(message="Merci de produire un code HTML bienveillant") private String bio;</pre> <p>NB: annotation dépréciée: la communauté Hibernate souhaiterait qu'une autre communauté gère cette annotation</p>	
@URL	<p>Garantit que la valeur de l'attribut correspond à une URL valide</p> <pre>@URL(message="Merci de préciser une URL valide") private String url;</pre>	
@Range	<p>Garantit que la valeur de l'attribut est comprise entre la borne min et la borne max</p> <pre>@Range(min=15, max=28, message="Merci de préciser une température comprise entre 15 et 28 degrés celsius") private Float temperatureSouhaitee;</pre>	
@Past	<p>Garantit que la valeur de l'attribut (de type Date) contient une date dans le passé</p> <pre>@Past(message="Votre date de naissance doit être dans le passé") private Date dateDeNaissance;</pre>	

	@Future	<p>Garantit que l'attribut (de type Date) contient une date dans le futur</p> <pre>@Future(message="La date de planification doit être dans le futur") @DateTimeFormat(pattern = "dd/MM/yyyy") private Date dateEnquete;</pre>
	@Pattern	<p>Garantit que la valeur de l'attribut de type String respecte une expression régulière précisée dans l'attribut regexp</p> <pre>@Pattern(regexp="^[A-Za-z]+\$", message="La référence doit contenir uniquement des lettres") private String reference;</pre> <p>Pour vous aider à écrire vos regex: <a href="http://jkorpela.fi/perl/regexp.html">http://jkorpela.fi/perl/regexp.html</a> et <a href="https://regex101.com/">https://regex101.com/</a></p>
	@Email	<p>Garantit que la valeur de l'attribut de type String contient une adresse email valide</p> <pre>@Email(message="L'adresse email renseignée n'est pas valide") private String email;</pre>
	@Size	<p>Garantit que le nombre de caractères de l'attribut respecte les contraintes données en paramètre</p> <pre>@Size(min=5, message="Le mot de passe doit contenir au minimum 5 caractères") private String motDePasse;</pre>
	@Valid	<p>Demande la validation des données de l'objet vis-à-vis des contraintes exprimées dans la classe de l'objet.</p> <p>Exemple : on demande au moment de l'invocation de la méthode la validation des données de l'objet enquete :</p> <pre>public ModelAndView enregistrerEnquetePost(@Valid @ModelAttribute("enquete") Enquete enquete, BindingResult result) {}</pre>
Spring	@Autowired	<p>Demande à Spring d'injecter automatiquement un objet dans l'objet de la classe considérée. Cette annotation est de moins en moins utilisée car la communauté Spring suggère de demander l'injection de dépendances dans le constructeur de la classe</p>

		<pre>@Autowired private EnqueteDao enqueteDao;</pre>
	@Controller	<p>Déclare une classe qui va traiter les requêtes HTTP (cf couche coordination)</p> <pre>@Controller public class EnqueteController {}</pre>
	@RestController	<p>Déclare une classe qui va traiter des requêtes de type REST</p> <pre>@RestController public class EnglishBattleControllerWS {}</pre>
	@Service	<p>Déclare une classe de service</p> <pre>@Service public class EnqueteServiceImpl implements EnqueteService {}</pre> <p>En oubliant l'annotation @Service sur la classe de service on obtient l'erreur :</p> <pre>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.</pre>
	@Repository	<p>Déclare une classe DAO (Data Access Object : classe capable de communiquer avec la base de données)</p> <pre>@Repository public class JoueurDaoImpl implements JoueurDao {}</pre> <p>Grâce à Spring Data, cette annotation est de moins en moins utilisée car pour mettre en œuvre une DAO il suffit de déclarer une interface qui hérite de JpaRepository. Spring Data implémente les DAO à notre place (voir SimpleJpaRepositoryImpl)</p>
	@Query	<p>Déclare la requête HQL associée à la méthode de l'interface de DAO</p>

	<pre>@Query("from Enquete where prix&gt;9000") public List&lt;Enquete&gt; findMostExpensiveSurveys();</pre> <p>A noter: en utilisant Spring Data, cette annotation ne sert que très rarement, voir:  <a href="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</a></p>
@Param	<p>Déclare les paramètres de la méthode Java devant être utilisés comme des paramètres HQL</p> <pre>@Query("from Question where enquete.idEnquete=:eid") public List&lt;Question&gt; findByIdEnquete(@Param("eid") Long idEnquete);</pre>
@Transactional	<p>Définit un contexte transactionnel (typiquement sur les méthodes des classes de service)  Pour définir une transaction en lecture seule: @Transactional(readOnly = true)</p> <pre>@Service @Transactional public class EnqueteServiceImpl implements EnqueteService {}</pre>
@RequestMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur. En d'autres termes, cette annotation déclare la correspondance entre une méthode du contrôleur et la ou les URL qu'elle prend en charge</p> <pre>@RequestMapping(value = { "/index", "/"}, method = RequestMethod.GET ) public ModelAndView accueil(@RequestParam Map&lt;String, String&gt; map) {}</pre>
@GetMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode HTTP est de type Get.</p> <p>La méthode enqueteGet ci-dessous sera invoquée lorsqu'un navigateur se rend sur l'URL <a href="http://localhost/enquetes">http://localhost/enquetes</a> ou lorsqu'un hyperlien avec un attribut href égal à "http://localhost/enquetes" est cliqué</p> 

		<pre>@GetMapping("/enquetes") public ModelAndView enquetesGet() {}</pre>
	@PostMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode HTTP est de type Post.</p> <pre>@PostMapping("/filtrerLesEnquetes") public ModelAndView filtrerEnquetes(@RequestParam Map&lt;String, String&gt; map) {}</pre> <p>A noter: La méthode ci-dessous est invoquée lorsque le bouton submit du formulaire ci-dessous est cliqué:</p> <pre>&lt;form action="/filtrerLesEnquetes" method="post"&gt;   &lt;input type="text" name="NOM"&gt;   &lt;input type="submit" value="Filtrer"&gt; &lt;/form&gt;</pre>
	@PutMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode HTTP est de type Put.</p> <pre>@PutMapping("/mettreAJourEnquete", produces="application/json") public Enquete majEnquete(@RequestParam Map&lt;String, String&gt; map) {}</pre>
	@DeleteMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode HTTP est de type Delete.</p> <pre>@PutMapping("/supprimerEnquete", produces="application/json") public boolean supprimerEnquetes(@RequestParam Map&lt;String, String&gt; map) {}</pre>
	@RequestParam	<p>Indique que le paramètre de la méthode (du contrôleur) provient de l'objet request. Spring convertira l'objet dans le type attendu</p> <pre>@PostMapping(value = "/question") public ModelAndView questionPost(@RequestParam(name="ID_ENQUETE") Long idEnquete, @RequestParam(name="LIBELLE") String libelle) {}</pre>
	@ModelAttribute	<p>Annotation utilisée sur un paramètre d'une méthode du contrôleur. Elle précise un objet d'une classe métier retourné par une vue.</p>

		<pre> @PostMapping("/inscription") public ModelAndView inscriptionPost(@Valid @ModelAttribute("personne") Personne personne, BindingResult result) {      if (result.hasErrors()) {         ModelAndView mav = inscriptionGet();         mav.addObject("personne", personne);         return mav;     } else {         personneService.enregistrerPersonne(personne);         ModelAndView mav = new ModelAndView("merciInscription");         return mav;     } } </pre> <p>A noter: la vue (le fichier inscription.jsp) doit contenir une balise form:form comme suit:  <code>&lt;form:form action="inscription" method="post" modelAttribute="personne"&gt;</code></p> <p>Pour rappel: l'annotation @Valid permet de déléguer à Spring le travail de validation sur l'objet annoté. En d'autres termes, toutes les contraintes de validation exprimées dans les classes métier (présentes dans le package business) seront vérifiées par Spring.</p>
	@SessionAttributes	<p>Demande à Spring de stocker en session les attributs d'un objet envoyé à un formulaire HTML</p> <pre>@SessionAttributes("taches")</pre>
	@DateTimeFormat (pattern = "dd/MM/yyyy")	<p>Permet de définir un attribut de type Date dans une classe métier avec un format précisé en attribut</p> <pre>@DateTimeFormat(pattern = "dd/MM/yyyy") private Date dateEnquete;</pre>
	@PageableDefault	<p>Définit les paramètres de pagination et de tri par défaut</p> <pre>public ModelAndView enquetesGet(@PageableDefault(value = 10, sort = "dateCommande") Pageable pageable) {}</pre> <p>A noter : dans les services et les DAO, toutes les méthodes ayant un paramètre de type Pageable</p>

		doivent renvoyer une page
	@InitBinder	<p>Annotation d'une méthode "montrant" à Spring comment obtenir un objet métier à partir de son id ou comment transformer un objet en un autre objet. La conversion de données (en anglais binding) est réalisée par Spring grâce aux méthodes annotées @InitBinder</p> <pre> @InitBinder public void initBinder(WebDataBinder binder) {      // Apprend à Spring à convertir un String en Date     SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");     dateFormat.setLenient(false);     binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true));      // Apprend à Spring à convertir un id de Ville en objet de type Ville     binder.registerCustomEditor(Ville.class, "ville", new PropertyEditorSupport() {         @Override         public void setAsText(String id) {             setValue((id.equals("")) ? null :                 villeService.recupererVille(Long.parseLong((String) id)));         }     });      // Apprend à Spring à convertir une liste d'id en liste d'intérêts     binder.registerCustomEditor(List.class, "interets", new         CustomCollectionEditor(List.class) {             @Override             public Object convertElement(Object objet) {                 Long id = Long.parseLong((String) objet);                 return interetService.recupererInteret(id);             }         }); } </pre> <p>Pour la classe PropertyEditorSupport, se référer à la javadoc :  <a href="https://docs.oracle.com/javase/7/docs/api/java/beans/PropertyEditorSupport.html">https://docs.oracle.com/javase/7/docs/api/java/beans/PropertyEditorSupport.html</a></p>
	@PostConstruct	Annotation sur une méthode qui sera invoquée automatiquement lorsque Spring a injecté tous les

	objets annotés @Autowired  <code>@PostConstruct public void ajouterDonneesInitiales() {}</code>
@PathVariable	Désigne une variable qui se trouve dans l'url. Le code ci-dessous récupère l'id de l'enquête à partir de l'url. Exemple d'url: <a href="http://localhost:8080/enquete/4">http://localhost:8080/enquete/4</a> (idEnquete aura dans ce cas la valeur 4)  <code>@GetMapping(value="/enquete/{idEnquete}", produces="application/json") public Enquete enqueteGet(@PathVariable Long idEnquete) {}</code>
@Bean	Déclare une méthode dont l'objet retourné sera géré par le conteneur Spring  <code>@Bean public EmbeddedServletContainerFactory servletContainer() {}</code>
@Configuration	Déclare une classe de configuration (un objet de cette classe remplace le fichier xml de configuration de Spring (souvent appelé spring-servlet.xml)  <code>@Configuration public class KanbanConfiguration {}</code>
@Value	Récupère la valeur d'une variable déclarée dans le fichier application.properties de Spring Boot
@ResponseBody	Précise que le retour de la méthode correspond à ce qui va être envoyé au client HTTP  <code>@GetMapping(value="/fichierExcel", produces="application/vnd.ms-excel") public @ResponseBody byte[] fichierExcelGet(@RequestParam(name="ID") Long idFichierExcel) throws IOException {}</code>
@Secured	Restreint l'accès aux utilisateurs ayant le rôle précisé en paramètre  <code>@Secured("ROLE_ADMIN") @GetMapping("/enquetes") public ModelAndView enquetesGet() {}</code>
@Scheduled	Programme l'invocation de la méthode de manière automatique. Cette annotation s'inspire de cron. L'exemple ci-dessous invoque la méthode programmerEmails() tous les jeudis à 17h:



		<pre>@Scheduled("00 00 17 * * THU") public void programmerEmails() {}</pre>
	@Primary	<p>Indique la classe d'implémentation que Spring devra utiliser face à un objet déclaré avec une interface (à utiliser lorsque plusieurs classes implémentent une même interface)</p> <pre>@Primary public class EnqueteServiceImpl implements EnqueteService {}</pre>
	@Qualifier	<p>Permet de distinguer deux paramètres de même type dans une méthode</p> <pre>@RequestMapping(value = { "/index", "/"}, method = RequestMethod.GET ) public ModelAndView accueil(     @Qualifier("aerodrome") @PageableDefault(value = 10, sort = "nom")     Pageable pageableAerodrome,     @Qualifier("vol") @PageableDefault(value = 4, sort = "prixEnEuros")     Pageable pageableVol) {      ModelAndView mav = new ModelAndView("index");     mav.addObject("pageDAerodromes", aerodromeService.recupererAerodromes(pageableAerodrome));     mav.addObject("pageDeVols", volService.recupererVols(pageableVol));     return mav; }</pre> <p>Exemple d'URL: <a href="http://localhost:8080/index?vol_page=4&amp;aerodrome_page=1">http://localhost:8080/index?vol_page=4&amp;aerodrome_page=1</a></p>