Commandes en Java - JavaDoc, Annotations générales, Annotations spécifiques

ATTENTION :

|  |  |  |
| --- | --- | --- |
| *Commentaires :*  //Commentaire sur 1 ligne  /\* Commentaire sur plusieurs lignes \*/ | *Instructions JavaDoc :*  /\*\* instruction  \* dites JavaDoc  \*@return int ma valeur  \* qui commencent par une minuscules après @  \*/ | *Annotation Java :*  @Return |

JavaDoc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation |  |
| Règles générales : | La première phrase doit être une courte description de la classe/méthode/variable/...  Utiliser des phrases courtes  Détail concis de l'algorythme/du traitement  Les tags @param et @return obligatoires pour méthodes avec param et non void  **Tout en anglais !**  Utilisation possible des balises HTML (sans en abuser !)  <b>, <i>, <ul> et <li>, <p> | | La JavaDoc sert à générer et maintenir facilement la documentation liée au code  Elle doit être maintenue au même titre que le reste du code.  Chaque Classe, chaque Interface, chaque méthode, chaque variable doit être précédé de sa JavaDoc (plus ou moins longue selon la complexité de l'élément décrit).  Rq: la JavaDoc appliquée à une Interface, ou une classe Mère est héritée dans la classe fille => inutile de la réécrire sur les éléments en communs ! | |
|  |  |  |  |  |
| @param | paramètres de la méthode | @param monParamettreA  @param monParamettreB |  |  |
| @return | objet retourné par la méthode | @return Un objet de type T qui représente un truc. |  |  |
| @throws | indique la présence d'une exception qui risque d'être propagée | @throws MonException Si division par zéro |  |  |
| @see | fait référence à une autre interface, classe, méthode | @see MonAutreClasse  @see MonAutreClasse#maMethode(int, String)  @see CetteClasse#monParametre |  |  |
| @since | permet de dater la présence d'une classe, méthode, paramètre, ... | @since 3.0.2 |  |  |
| @serial | ne peut s'appliquer que sur une Classe ou une Interface |  |  |  |
| @deprecated | décrit la version depuis laquelle cette méthode, classe, ... est dépréciée, ainsi que ce qu'il faut utiliser à sa place | @deprecated Depuis Java 1.4, remplacée par monAutreMethode() |  |  |
|  | Annotation ne pouvant être utilisées que sur des Interfaces et Classes | |  |  |
| @author | indique le nom de l'auteur de la classe | @author NomAuteur |  |  |
| @version | indique le numéro de version de la classe | @version 3.0.2 |  |  |
|  |  |  |  |  |

Annotations Java générales

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | | |  | |
| Annotations marqueurs | = annotations standards | java.lang.annotation |  | | |  | |
| @Override | indique que la méthode est redéfinie via le principe d'héritage | @Override  public String toString() {  } |  | | | ATTENTION : doit être utilisée UNIQUEMENT sur des méthodes héritées et redéfinies | |
| @Deprecated | indique que l'élément ne devrait plus être utilisé  si l'élément est utilisé, Eclipse le signale par un warning | @Deprecated  public void maMethode() { |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Méta-annotation | servent à annoter nos annotations personnalisées |  |  | | |  | |
| @Documented | inclus l'annotation dans la JavaDoc lors de sa génération | @Documented  public @interface MonAnnotation {  } |  | | |  | |
| @Inherit | les classes dérivant d'une classe mère contenant des annotations hériteront également des annotations qui seront anotée par @Inherit | @Inherited  public @interface MonAnnotation {  } | @MonAnnotation  public class ClasseMere {  }  public class ClasseFille extends ClasseMere {  } | | | cette classe fille héritera de l'annotation car elle est annotée @Inherited | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Annotations paramétrées |  |  |  | | |  | |
| @SupressWarning( ) | indique au compilateur d'ignorer les éléments générant le type de warning indiqué en paramètre  ignore :  tous les warning  annotation dépréciée  élément déprécié  oubli d'un break dans bloc switch  bloc finally douteux  serialVersionUID manquant  redéfinition de cstr ss super()  code inutilisé | @SupressWarning("all")  all  boxing  cast  dep-ann  deprecation  fallthrough  finally  serial  super  unused | @SupressWarning("cast", "deprecation") | | | ATTENTION :   * à utiliser avec parcimonie pour ne pas ignorer des erreurs importantes * le compilateur Java ne prend pas les mêmes paramètres que le compilateur d'Eclipse | |
|  |  |  |  | | |  | |
| @Retention( ) | définit une durée de vie pour l'annotation | @Retention(RetentionPolicy.CLASS)  public @interface MonAnnotation {  }  @Retention(RetentionPolicy.RUNTIME)  @Retention(RetentionPolicy.SOURCE) | défaut, annotation enregistrée dans le fichier .class, mais non utilisable par la JVM  enregistrée dans le fichier .class et utilisable lors de l'exécution du prog grâce à l'introspection  non enregistrée dans le fichier .class, utilisable uniquement par les outils sachant lire les fichiers sources (utilitaire JavaDoc, compilateur, …) | | |  | |
| @Target | permet de cibler sur quel type d'élément(s) l'annotation peut être utilisée  tous type de base ou personnalisé  champs d'une classe (= variables)  autres annotations | @Target(ElementType.PACKAGE)  public @interface MonAnnotation {  }  PACKAGE  TYPE  CONSTRUCTOR  METHOD  PARAMETER  FIELD  LOCAL\_VARIABLE  ANNOTATION\_TYPE | *Avec plusieurs paramètres :*  @Target(ElementType.TYPE, ElementType.FIELD)  public @interface MonAnnotation {  }  *Dans le fichier package-info.java :*  @PackAnnotationA  @PackAnnotationB  package mon.package; | | | ATTENTION :  Les annotations sur le Package se font dans un fichier spécifique nommé ***package-info.java*** à placer dans le package concerné | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Annotations multiparamétrées |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Annotation Personnalisée |  |  |  | | |  | |
| @interface | créer une annotation personnalisée | public @interface MonAnnotation {  } | public @interface MonAnnotation {  String monChampA();  String monChampB() default "valeur par défaut";  NIVEAU niveau() default NIVEAU.MINEUR;  int[] tableauEntier();  char[] tableauCaractere() default {'A', 'D'};  MonAnnotationB annotation();  Class maClasse() default Double.class;  } | | | seuls les léléments suivant peuvent être utilisés dans les annotations :   * primitifs (int, double, boolean, char, …) * String * enum * Class * autres annotations | |
|  |  |  | @MonAnnotation(  monChampA = "valeur champA",  annotation = @MonAnnotationB,  tableauEntier = { 0 }  )  public void maMethode() { } | | | ATTENTION : lors de l'utilisation de l'annotation, il faut, à minima, attribuer des valeurs aux éléments qui n'en ont pas par défaut | |
|  | annotation monoparamétrée personnalisée | public @interface MonAnnotation1Param {  Type value();  } | @MonAnnotation1Param(valeurParametre); | | | Rq: si l'annotation ne comporte qu'un paramètre, en le nommant value(), on évite d'avoir à préciser son nom lors de l'appel à l'annotation | |
|  | Exemple concret de définition d'une annotation et de son traitement | *Définition de l'annotation @Rule*  *(Rule.java)*  @Retention(RUNTIME)  @Target({ FIELD, LOCAL\_VARIABLE })  public **@interface Rule {**  int min() default 0;  int max() default Integer.MAX\_VALUE;  **}**  *Utilisation de l'annotation @Rule*  *(pizza.java)*  @Entity  @Table(name="PIZZA")  public class Pizza {  @Id  @GeneratedValue(strategy=GenerationType.IDENTITY)  private int id;  @Column(name="LIBELLE")  @ToString(uppercase = true)  private String libelle;  @Column(name="PRIX")  @ToString  **@Rule(min = 12)**  private double prix;  public Pizza() {  }  public Pizza(String libelle, double prix) throws PizzaException {  this.libelle = libelle;  this.prix = prix;  **Validator.validate(this);**  }  } | | | *Récupération des annotations et traitements selon leurs valeurs*  *(Validator.java)*  **public class Validator {**  public static void validate(Pizza pizza) throws PizzaException {    for(Field field : pizza.getClass().getDeclaredFields()) {  field.setAccessible(true);  if(field.isAnnotationPresent(**Rule.class**)) {  **Rule ruleAnnot = field.getAnnotation(Rule.class);**  try {  Double value = field.getDouble(pizza);    if(value < **ruleAnnot.min()**) {  throw new RulePizzaException("La valeur de '" + **field.getName()** + "' doit être supérieure ou égale à " + ruleAnnot.min());  }  if(value > **ruleAnnot.max()**) {  throw new RulePizzaException("La valeur de '" + **field.getName()** + "' doit être inferieure ou égale à " + ruleAnnot.max());  }    } catch (IllegalArgumentException e) {  e.printStackTrace();  } catch (IllegalAccessException e) {  e.printStackTrace();  }    }  }  }  **}** | | |
|  |  |  | |  | | |  |
|  |  |  | |  | | |  |
|  |  |  | |  | | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation |  |
|  |  |  |  |  |
|  |  |  |  |  |