



# LIVRAISON D'UNE APPLICATION



# LIVRAISON D'UNE APPLICATION

- Une fois le développement d'une application terminé, celle-ci doit être packagée et documentée avant d'être livrée au client



# LA DOCUMENTATION



# LA DOCUMENTATION : DEUX VOILETS

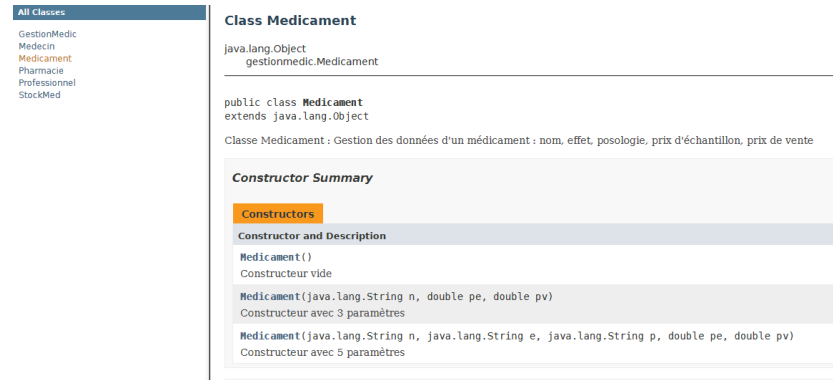
- Documentation utilisateur
- Documentation programmeur

# LA DOCUMENTATION PROGRAMMEUR

- Commentaires du code
- Description des classes, propriétés, méthodes... implémentées dans l'application

# LA JAVADOC

- Utilisation d'un outil Java pour générer la documentation des classes



The screenshot displays a Java IDE interface. On the left, a sidebar titled "All Classes" lists several classes: GestionMedic, Medecin, Medicament (highlighted in orange), Pharmacie, Professionnel, and StockMed. The main area shows the documentation for the "Class Medicament".

**Class Medicament**  
java.lang.Object  
gestionmedic.Medicament

---

```
public class Medicament
extends java.lang.Object
```

Classe Medicament : Gestion des données d'un médicament : nom, effet, posologie, prix d'échantillon, prix de vente

**Constructor Summary**

**Constructors**

**Constructor and Description**

**Medicament()**  
Constructeur vide

**Medicament(java.lang.String n, double pe, double pv)**  
Constructeur avec 3 paramètres

**Medicament(java.lang.String n, java.lang.String e, java.lang.String p, double pe, double pv)**  
Constructeur avec 5 paramètres

# GÉNÉRATION DE LA JAVADOC

## Générer la Javadoc

- Pour vérifier les éléments à commenter :  
Clic droit sur une classe → Tools → Analyze Javadoc
- Pour générer la doc du projet :  
Clic droit sur la racine du projet → Generate Javadoc

## Emplacement de la Javadoc

- Création d'un répertoire dist/javadoc dans la racine du projet
- Génération des pages html automatique
- Menu : index.html

# FORMAT DES COMMENTAIRES

- Sur la première ligne du commentaire on doit trouver les signes: /\*\*
- Et sur la dernière ligne :\*/
- Chaque ligne du commentaire commence par \*



# TAGS DANS LES COMMENTAIRES

Tags	Informations du commentaire
<b>Pour chaque classe</b>	
@author	Nom du développeur de la classe
@version	Version de la classe
<b>Pour chaque méthode</b>	
@param	Une ligne par paramètre utilisé : nom du paramètre, type et information
@return	Si une valeur est retournée, indique l'information retournée et son type

# CHOIX DES TAGS PRIS EN COMPTE

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- Run
- Application
  - Web Start
- License Headers
- Formatting
- Hints

Include Private and Package Private Members

Generate:

- Class Hierarchy Tree
- Class and Package Usage Pages
- Navigation Bar
- Index
  - Separate Index per Letter

Document Additional Tags:

- @author
- @version

Browser Window Title:

Additional Javadoc Options:   
(e.g. -overview <file> or -header "Some header")

Preview Generated Javadoc

# EXEMPLE

```
/**
 * Constructeur complet
 * @param n nom
 * @param pv point de vie
 * @param puis puissance
 * @param def bouclier
 */
public Heros(String n, int pv, int puis, int def)
{
    nom=n;
    pointsVie=pv;
    puissance=puis;
    defense=def;
}
```

## Heros

```
public Heros(java.lang.String n,
             int pv,
             int puis,
             int def)
```

### Constructeur complet

#### Parameters:

n - nom

pv - point de vie

puis - puissance

def - bouclier

# EXERCICE

- Commenter les classes Heros et Joueur
- Générer la Javadoc du projet en prenant en compte l'auteur et la version



# PACKAGER UNE APPLICATION



# LIVRAISON AU CLIENT

- Le client doit pouvoir lancer l'application à partir d'une icône
  - Sans logiciel de développement
  - Sans accès au code source

# CRÉATION D'UN EXÉCUTABLE

## Générer l'exécutable .jar

- Menu Run → Clean and Build
- Clic droit sur le projet → Clean and Build
- Création du fichier .jar dans le répertoire dist du projet

## Paramétrer l'exécution

- Exécution d'un .jar :  
`java -jar fichier.jar`
- Création d'un fichier de commande pour lancer l'exécution
- Création d'un raccourci pour lancer le fichier de commande

# UTILISATION D'UN FICHER MANIFEST

- Le fichier manifest contient des propriétés du projet
  - La description du projet
  - La version du projet...



# GÉNÉRATION DU MANIFEST

- Génération automatique lors de l'opération Build
- Paramétrage du nom du fichier dans nbproject\project.properties
- Paramétrage du contenu du fichier dans build.xml
  - Possibilité d'ajouter des propriétés dans la balise <manifest>

# FICHER PROJECT.PROPERTIES

```
javadoc.nonavbar=false
javadoc.notree=false
javadoc.private=false
javadoc.splitindex=true
javadoc.use=true
javadoc.version=false
javadoc.windowtitle=
main.class=applijeu.AppliJeu
manifest.file=manifest.mf
meta.inf.dir=${src.dir}/META-INF
mkdist.disabled=false
platform.active=default_platform
run.classpath=\
    ${javac.classpath}:\
    ${build.classes.dir}
```

Classe de démarrage

Ficher manifest

# FICHER BUILD.XML

```
<manifest file="MANIFEST.MF">  
  <attribute name="Bundle-Name" value="JeuHeros" />  
  <attribute name="Bundle-Version" value="1.0" />  
</manifest>
```

# FICHER MANIFEST GÉNÉRÉ

---

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.9.7
Created-By: 1.8.0_131-b11 (Oracle Corporation)
Bundle-Name: JeuHeros
Bundle-Version: 1.0
```

# EXERCICE

- Créer une icône de lancement du projet JeuHeros
- Paramétrer le fichier manifest pour intégrer le nom du projet et sa version

# INSTALLATION D'UNE APPLICATION

- Livraison au client des jar et de l'exécutable de lancement
- Utilisation de logiciels spécialisés créant un exécutable d'installation

# CRÉATION D'EXÉCUTABLE D'INSTALLATION

- Ce type de logiciel permet de réaliser plusieurs manipulations en même temps :
  - Installation de la sdk
  - Copie des répertoires de l'application dans Program files
  - Création des ressources de l'application, import d'une base de données par exemple
- Exemple de logiciels pour générer les installations : JSmooth, JExeCreator, Launch4j...