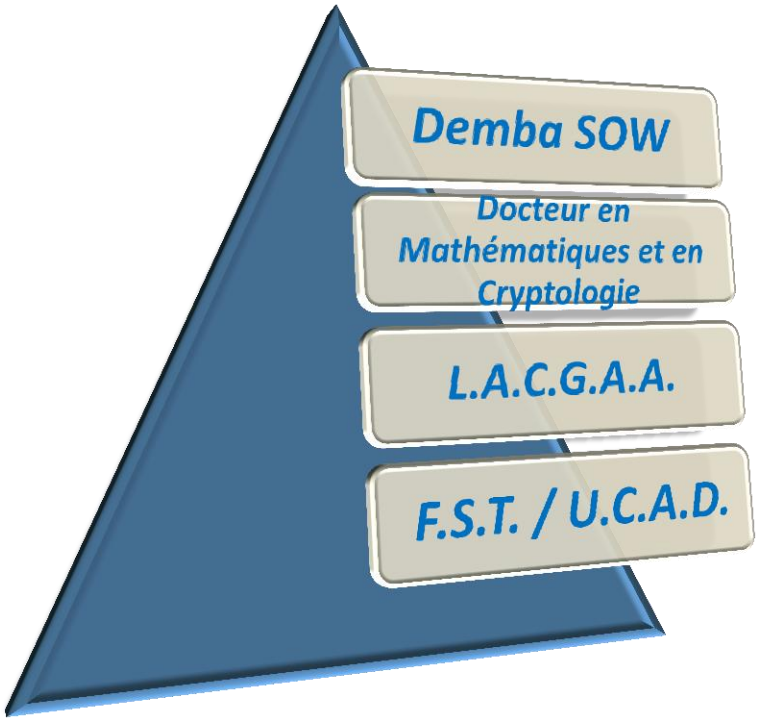
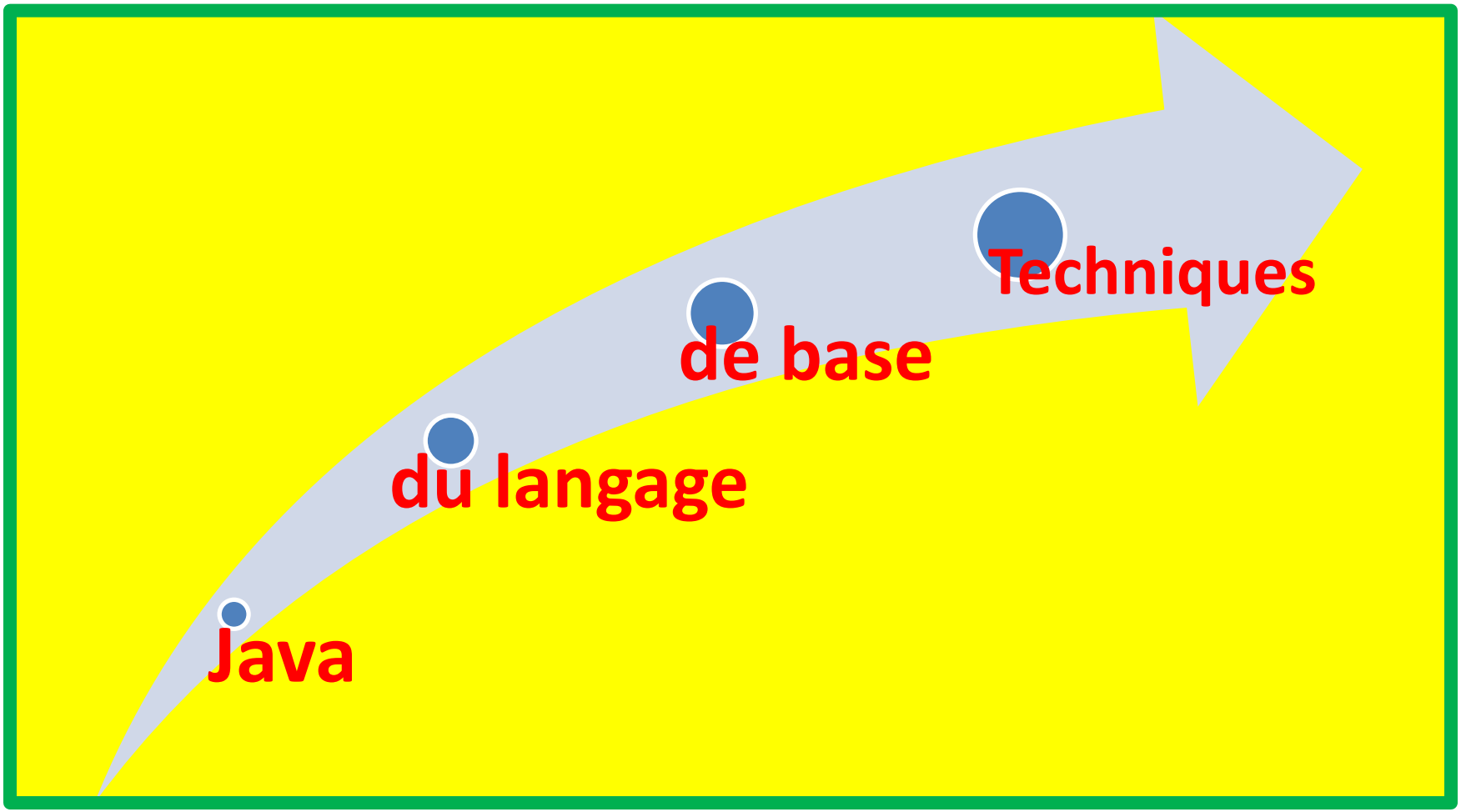
Cours de Programmation Orientée Objet JAVA




Techniques
de base
du langage
Java

Techniques de base du langage

Premier programma Java.

```
package home.user.java.essai;

// un premier programme

/* la version JAVA du classique
   Hello World
*/

public  class HelloWorld {
        public static void main(String [ ] args) {
                System.out.println("Hello World !");
}
}
```

// Ceci est un commentaire finissant en fin de ligne

/* ceci est un commentaire pouvant encadrer
un nombre quelconques de caractères
sur un nombre quelconque de lignes */

Hello World !

Structure du programme (1/2)

```
package home.user.java.essai ;

public class HelloWorld
   {
      public static void main(String [ ] args)
         {
            System.out.println("Hello World !");
         }
}
```

entête de la classe

Définition de la classe avec une seule méthode(la fonction *main*)

Structure du programme (2/2)

- Le mot clé ***static*** précise que la méthode ***main*** n'est pas liée à une instance (objet) particulière de la classe.

- Le paramètre ***String[ ] args*** est un tableau de chaînes de caractères qui permet de récupérer des arguments transmis au programme au moment de son lancement. Ce paramètre est ***OBLIGATOIRE*** en Java.

- Le mot clé ***public*** dans *public class* sert à définir les **droits d'accès** des autres Classes (en fait de leurs méthodes) à la classe . **[A voir].**

- Le mot clé ***public*** dans *public static void main* est ***obligatoire*** pour que votre programme s'exécute. Il s'agit d'une convention qui permet à la machine virtuelle d'accéder à la méthode ***main*** .

Paquetage de la classe

La notion de paquetage se définit comme étant un regroupement(ensemble)de classes en une structure commune.

La classe définit ici (HelloWorld) appartient au paquetage nommé :
home.user.java.essai

La classe a un nom *simple* : HelloWorld
Le nom *complet* de la classe est : home.user.java.essai.HelloWorld

On précisera qu'une classe appartient à un paquetage en plaçant **en début de fichier** l'instruction **package nom_du_paquet ;**

Contenu du programme

Le programme est constitué d'une seule instruction :

System.out.println("Hello World !");

System.out.println(argument)

Affiche la valeur de *argument* **puis passe** à la ligne

Classe *System du* package ***java.lang***

Variable de classe (référence un objet de type ***PrintStream***)

Méthode d'instance de la classe **PrintStream** du package ***java.io***

System.out.**print (argument)**

Affiche la valeur de *argument* **sans passer** à la ligne

Exécution du programme (1/2)

La sauvegarde du programme se fait impérativement dans un fichier qui porte le nom **HelloWorld.java**

Le code source d'une classe ***publique*** doit toujours se trouver dans un fichier portant le même nom et possédant l'extension ***.java***.

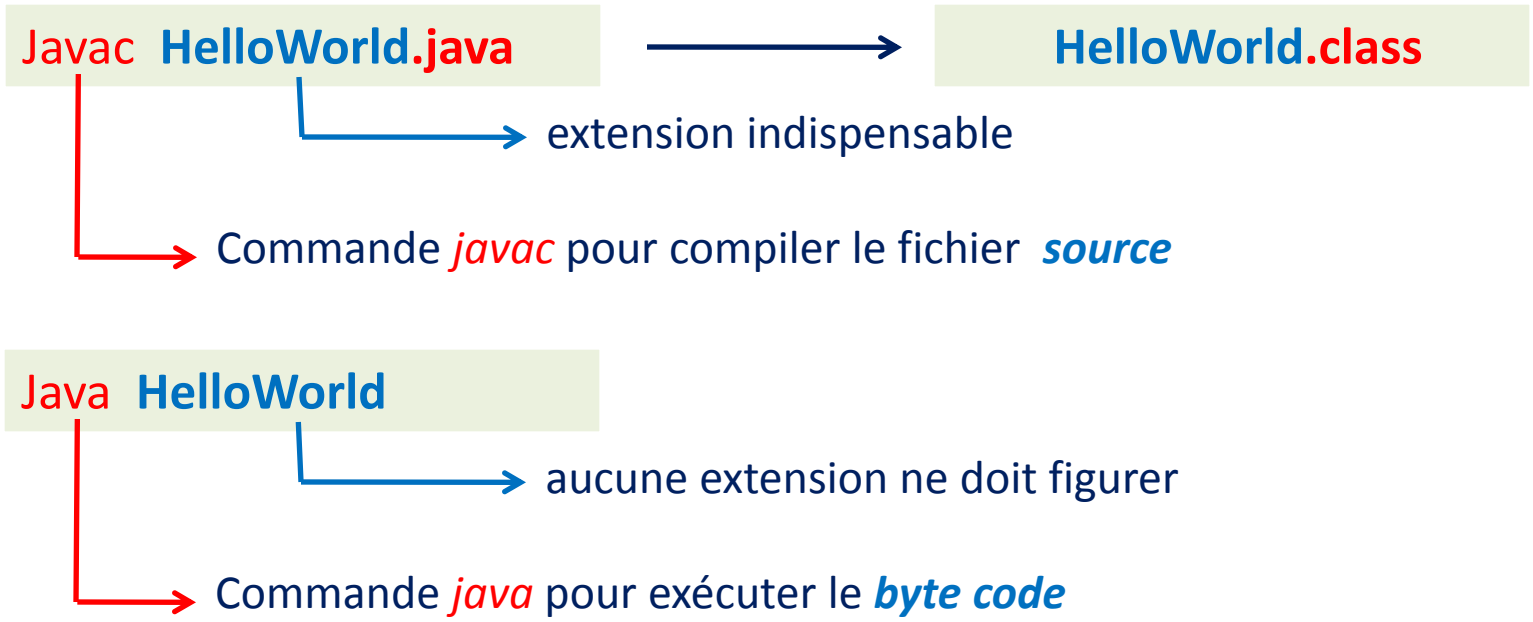
La classe contenant la méthode **main** est appelée la **classe principale** du programme. C'est cette classe qu'il faut exécuter.

EN FAIT ON EXECUTE QUE LES INSTRUCTIONS DE LA METHODE **main**.

Exécution du programme (2/2)

On procède à la COMPILATION de ce programme pour la génération du ***byte code***.

Si elle se passe bien(sans erreurs) on obtient un fichier d'extension **.class** . Ici, il s'agit **de HelloWorld.class** .



Exemple pratique de compilation et d'exécution d'une classe (1/2)

(ici dans le paquet essai, il y a une seule classe : LectureClavier)

On dispose de la classe essai.LectureClavier se trouvant dans le répertoire C:\demba\src\essai\LectureClavier.

SYNTAXE de la commande javac (compilation) :

javac -d <répertoire où doivent être mis les fichiers générés>
-classpath <adresse des classes déjà compilées nécessaires à la compilation>
<adresse du(des) fichiers à compiler>

l'option d permet de préciser le répertoire de base des fichiers .class générés par la compilation .

Pour ce premier exemple, aucune classe n'est nécessaire lors de la compilation, l'option classpath est donc absente.

```
C:\> javac -d  D:\tpJava\mescompils C:\demba\src\essai\LectureClavier.java
```

SYNTAXE de la commande java (exécution) :

java -classpath <adresse des classes utilisées lors de l'exécution>
<nom complet de la classe principale>

```
C:\> java -classpath D:\tpJava\mescompils essai.LectureClavier
```

Exemple pratique de compilation et d'exécution d'une classe (2/2)

(ici, dans le paquet essai, il y a maintenant la classe LectureClavier et la classe UtiliseLectureClavier qui utilise les méthodes de cette dernière)

On veut maintenant compiler la classe essai.UtiliseLectureClavier se trouvant dans le répertoire C:\demba.

```
C:\> javac  -d  D:\tpJava\mescompils
            -classpath   C:\demba\classes
             C:\demba\src\essai\UtiliseLectureClavier.java
```

```
C:\> java     -classpath      D:\tpJava\mescompils  UtiliseLectureClavier
```

Lorsqu'il y a plusieurs adresses à préciser pour une même option,
séparer les adresses
par des ; (sous Windows) ou : (sous Linux))

compilation simultanée de plusieurs fichiers

Compilation de deux fichiers: essai.HelloWorld et essai.Compte.

```
javac    -d  D:\tpJava\mescompils
         -classpath  D:\tpJava\ltdsi.jar
          C:\demba\src\essai\HelloWorld.java
          C:\demba\src\essai\Compte.java
```

Compilation de tous les fichiers d'un répertoire (on compile toutes les classes du package essai):

```
javac    -d  D:\tpJava\mescompils
         -classpath  D:\tpJava\ltdsi.jar
          C:\demba\src\essai\*.java
```

La variable d'environnement classpath

- **l'option *classpath* permet de préciser où trouver les classes**
  - **utilisées lors de l'exécution du programme pour la commande java**
  - **nécessaires à la compilation du programme pour la commande javac**
- **A cette option peut correspondre une ou plusieurs valeurs, chacune d'elle pouvant être :**
  - **l'adresse (relative ou absolue) d'un fichier jar.**
  - **l'adresse (relative ou absolue) d'un répertoire de base de classes**
- ***Remarque : les classes de l' A.P.I. ne sont pas concernées par cette option***

Si plusieurs valeurs sont associées à une option classpath, elles doivent être séparées par des ; (sous Windows) ou des : (sous linux).
La valeur par défaut de cette option est le répertoire courant (désigné par un ".")

ATTENTION : le répertoire de base d'une classe est le répertoire contenant le répertoire racine du paquetage.
Exemple : le répertoire de base de la classe essai.test.demba.hello.HelloWorld est le répertoire contenant le dossier essai (qui lui même contient le dossier test etc...)

Création et utilisation de fichiers jar

(lors de la compilation d'une classe, celle-ci peut nécessiter l'utilisation d'autres classes rassemblées au sein d'une archive jar)

On peut créer un fichier jar correspondant au répertoire C:\demba\classes.

Pour compresser le répertoire courant

1. **Placez vous dans le répertoire de base des classes**
2. **Exécutez la commande**

   jar cf <adr. du fichier jar à créer> . (n'oubliez pas le point)

   jar xf <nom du fichier jar crée> pour extraire le contenu et pour ajouter

   jar cvmf META-INF\MANIFEST.MF fichier.jar fichier.class

Exemple : cd C:\demba\classes

(création) jar cf D:\tpJava\mescompils\mesprogs.jar .

(ajout) jar cvmf META-INF\MANIFEST.MF mesprogs.jar essai\Hello.class

```
javac   -d D:\tpJava\mesclasses
        -classpath D:\tpJava\mescompils\mesprogs.jar
         C:\demba\src\essai\UtiliseLectureClavier.java
```

```
Java  -classpath D:\tpJava\mesclasses essai.UtiliseLectureClavier
```

jar exécutable

- **On peut exécuter une classe à partir d'un fichier jar, il suffit simplement d'éditer le fichier MANIFEST.MF contenu dans le répertoire META-INF de l'archive.**
- **Avec n'import quel éditeur de texte, ouvrez le fichier Manifest (il ne contient que deux lignes);**

  conserver la ligne (c'est la première):

  Manifest-Version: 1.0

  et remplacer la deuxième ligne c-à-d:

  CreatedBy: 1.4.1_05 (Sun Microsystems Inc.)

  par:

  Main-Class: nom_complet_de_la_classe_principale
- **Par exemple, pour la classe LectureClavier, le fichier manifest sera:**

  Manifest-Version: 1.0

  Main-Class: essai.LectureClavier

```
// pour exécuter le programme à partir du jar
C:\> java –jar D:\tpJava\mescompils\mesprogs.jar
```

Les commandes de base du langage

- ***javac*** : pour la compilation (générer le .class).
- ***java*** : pour l'exécution (du main).
- ***appletviewer*** : pour exécuter une applet.
- ***javadoc*** : pour générer une documentation automatique.

Règles d'écriture en Java (1/3)

Les différentes entités utilisées dans un programme (méthodes, variables, classes, objets, ….) sont manipulées au travers d'*identificateurs* .

Un ***identificateur*** est formé *de* ***lettres*** ou de ***chiffres***, le premier caractère *étant* **obligatoirement** une lettre. Les lettres comprennent les majuscules **A-Z** et les minuscules **a-z**, ainsi que le caractère « souligné »(**_**) et le caractère **$**

Exemple :

ligne *Clavier* *valeur_5* *_total* *_56* *$total* ~~*2nombre*~~

Remarque très importante :

Java est très sensible à la **casse** : **ligne** ≠ **Ligne** .

Règles d'écriture en Java (2/3)

Un identificateur ne peut pas appartenir à la liste des mots réservés du langage Java :

abstract	**assert**	**boolean**	**break**	**byte**
case	**catch**	**char**	**class**	**const**
continue	**default**	**do**	**double**	**else**
extends	**false**	**final**	**finally**	**float**
for	**if**	**implements**	**import**	**instanceof**
int	**interface**	**long**	**native**	**new**
null	**package**	**private**	**protected**	**public**
return	**short s**	**tatic**	**super**	**switch**
synchronized	**this**	**throw**	**throws**	**transient**
true	**try**	**void**	**volatile**	

Règles d'écriture en Java (3/3)

Voici quelques conventions de codage en java

```
public class MaPremiereClasse
{ public void affiche( int argument )
    {  int nombreEntier =12 , i =1;
       final float NOMBRE =100;
          while ( i >100)
             { System.out.println (" i = "+ i ) ;
             }
    }
}
```

Nom de ***classe*** commence par une **majuscule.**

Nom de ***méthode*** , de ***variables*** ou ***d'attributs*** commence par une **minuscule**.

Nom de ***constante*** écrit **tout** en **majuscule**.

structures de contrôle: mettre des accolades

Indenter votre programme pour plus de lisibilité

La documentation des programmes

Pour pouvoir utiliser une classe,
il faut et il suffit de connaître son interface
(la déclaration commentée de tous les membres publics)
consultable grâce à la documentation en ligne

La documentation en ligne peut être générée à partir des fichiers sources par l'utilitaire javadoc.

Cette documentation est organisée et générée de la même manière pour toutes les classes que ce soit les classes de l'API ou les classes que nous définissons nous même.

A propos des commentaires

Commenter *toujours* les entêtes de fonctions

- **Un bon commentaire permet de pouvoir utiliser la fonction sans consulter le code.**
  - il indique à l'aide d'une phrase le rôle de la fonction en faisant intervenir le nom de tous les paramètres
  - il précise le rôle de chaque paramètre
  - il indique la signification du résultat retourné
  - il indique les restrictions sur la valeur des paramètres

Commenter *si nécessaire* des fragments de codes difficiles (un bon programme en contient généralement peu)

Éviter les commentaires inutiles
A =5; /* a prend la valeur 5 */

Types de commentaires en Java

```
package  essai;
/**
* @param autor Demba
* @since 1.0
*/
// un premier programme
/* la version JAVA du classique
Hello World
*/
public class HelloWorld {
    public static void main(String [ ] args) {
        System.out.println("Hello World !");
    }
}
```

/* ceci est un commentaire de documentation automatique javadoc */

// Ceci est un commentaire sur une seule ligne

/* ceci est un commentaire pouvant encadrer un nombre quelconques de caractères sur un nombre quelconque de lignes */

Les commentaires JAVADOC

Les commentaires javadoc / … */ sont des commentaires spéciaux permettant la production automatique de documentation au format html.**
Ils sont placés juste avant ce qu'ils commentent.

balises de commentaires JAVADOC de classe

@see <une autre classe>
@author <nom de l'auteur>
@version <n° de version>

balises de commentaires JAVADOC de fonction

@param <nom paramètre> <description>
@return <description>
@exception <nom exception> <description>
@since <n° de version>
@deprecated

Contenu d'une documentation javadoc

Description générale de la classe

Description des attributs (FIELD)

Description des constructeurs (CONSTRUCTOR)

Description des méthodes (METHOD)

- La description des ***attributs***, des ***constructeurs*** et des ***méthodes*** publics est donnée
  - **brièvement en début de document**
  - **en détail dans la suite du document.**

La commande javadoc

GENERATION DE LA DOCUMENTATION

javadoc -d <répertoire où doivent être mis les fichiers générés>
-sourcepath <répertoire(s) de base des fichiers sources>
<nom du paquetage>

-sourcepath
le(s) répertoire(s) de base des sources
(s'il y en a plusieurs, séparer par des ; (Windows) ou : (linux))

La documentation est ensuite consultable à partir du fichier index.html du répertoire mentionné avec l'option -d.

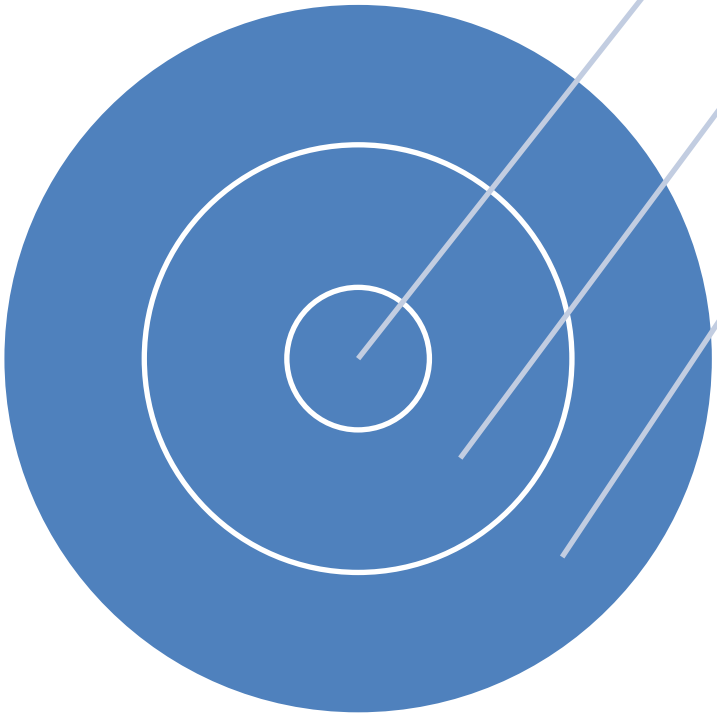
La commande javadoc: exemple

- Pour générer la documentation des classes du paquetage **essai**

```
javadoc     -d  D:\tpJava\javaprog\doc
            -sourcepath  C:\demba\src
            essai
```

- Pour générer la documentation des classes des paquetages **essai** (dans C:\demba) et **exemple.test** (dans D:\tpJava)

```
javadoc     -d  D:\tpJava\javaprog\doc
            -sourcepath  C:\demba\src;D:\tpJava\src
            essai exemple.test
```



Fin

du

chapitre