Langage Java Chapitre 5

Les packages, les commandes javac et java du jdk, les répertoires, les options -classpath et -d.

Révisions : les classes, la surcharge des méthodes (overloading).

Présentation d'autres commandes du jdk :

- **javap le désassembleur.**
- **javadoc qui permet de créer une documentation.**
- **jar pour compresser et décompresser des fichiers. Fichiers archives d'extension .jar.**

Rappels

Le nom de la classe et le nom du fichier source doivent être les mêmes, y compris pour les majuscules et minuscules. Le fichier source doit avoir obligatoirement l'extension .java.

Pour qu'une application java soit exécutable, elle doit posséder une méthode main(), qui est le point d'entrée du programme pour son exécution.
On compile un fichier Compteur.java avec la ligne de commande :

javac Compteur.java

On obtient un fichier Compteur.class, exécutable avec la ligne de commande :

java Compteur

javac est le compilateur, java est l'interpréteur qui lance l'exécution du programme dans la JVM (Machine virtuelle Java).

1. Rappels sur le concept de Classe et Objets

Les objets rencontrés dans la vie réelle sont habituellement rangés par famille.
Par exemple :

- le terme Voiture permet d'identifier tous les véhicules permettant de transporter des gens,
- le terme Individu permet d'identifier tous les humains,
- les termes « Voiture à moteur » permettent d'identifier tous les véhicules à moteur permettant de transporter des gens,
- le terme Ordinateur identifie tous les équipements informatiques permettant un traitement des données.

Détaillons la famille « Individu ».
On caractérise un individu à l'aide de plusieurs attributs : son nom, son prénom, son âge, sa date de naissance, son lieu d'habitation, sa situation maritale, s'il est marié ou non…
Un individu n'est pas statique, il parle, mange, travaille, dort, se réveille, se déplace, se marie… Toutes les actions qu'un individu peut faire modifient son état.

La programmation Objet permet de définir une classe pour caractériser une famille d'objets.

On peut ici définir la classe Individu, cette classe contiendra des attributs qui correspondent aux paramètres nom, prénom, âge… d'un individu mais également des méthodes comme parler, travailler, dormir qui correspondent à l'activité de l'individu.

La classe Individu

Les attributs :

Nom
Prénom
Age
Lieu de résidence

Les méthodes

Manger
Dormir
Travailler
SeRéveiller

La modélisation UML propose le diagramme de classe simplifié suivant :

Individu
Nom Prénom Age LieuDeRésidence
Manger() Dormir() Travailler() SeRéveiller()

La classe est représentée par un rectangle avec 3 compartiments : le compartiment du haut contient le nom de la classe, celui du milieu les attributs, celui du bas les méthodes.

2. La classe Compteur sans constructeur défini

On présente ici une classe **Compteur** qui n'a pas de constructeur, elle contient un **attribut**, de nom valeur, (appelé aussi **donnée membre** ou **propriété**) public de type int.

```
public class Compteur  {
        private int valeur ;

        public void affiche() {
                System.out.println("Valeur du compteur = "+valeur) ;
        }

        public void incremente() {
                valeur++ ;
        }

        public void decremente() {
                if (valeur > 0) valeur-- ;
        }
```

```
	public static void main(String argv[]) {
		Compteur c1  ;
		c1 = new Compteur() ;
		c1.affiche();
		int i =0 ;
		while (i++<10) c1.incremente() ;
		System.out.println("Apres 10 incrémentations");
		c1.affiche();
		i=0;
		while (i++<20) c1.decremente() ;
		System.out.println("Apres 20 décrémentations");
		c1.affiche();
	}
}
```

Pour simplifier, la méthode main est ici définie dans la classe Compteur.

Q1 Tester le fonctionnement de la classe Compteur ci-dessus. Justifier les affichages obtenus.

Le fonctionnement de cet exemple montre une nouvelle fois qu'une classe sans constructeur se voit dotée par la JVM d'un constructeur par défaut.
L'exécution de l'instruction c1 = new Compteur() entraîne l'exécution du constructeur par défaut. Ce dernier initialise l'attribut valeur à 0.
On parfaitement imaginé que la définition de ce constructeur par défaut créé à l'occasion par la JVM est la suivante :

```
public Compteur() { }
```

Ce pseudo constructeur n'exécute aucune instruction.

3. La classe Compteur avec 1 constructeur

On ajoute un constructeur sans argument.

Un constructeur est une méthode «spéciale» qui porte le même nom que la classe, qui peut recevoir des arguments mais qui n'a aucun type.

```
public class Compteur
{
	private int valeur ;

	/* Constructeur de la classe  Compteur */
	public Compteur() {
		valeur = 0 ;
		System.out.println("Je suis le constructeur sans argument") ;
		System.out.println("La valeur du compteur ="+valeur) ;
	}
	---
}
```

Q2 Tester le fonctionnement de la classe Compteur ci-dessus. Justifier les affichages obtenus.

4. La classe Compteur avec 2 constructeurs, la surcharge des méthodes

On crée un 2ème constructeur de la classe Compteur mais celui-ci reçoit un argument de type int qui sert à initialiser l'attribut valeur.

```
public class Compteur
{
        private int valeur ;

        /* 1er constructeur de la classe  Compteur */
        public Compteur() {
                valeur = 0 ;
                System.out.println("Je suis le constructeur sans argument") ;
                System.out.println("La valeur du compteur = "+valeur) ;
        }

        /* 2ème constructeur de la classe  Compteur */
        public Compteur(int n) {
                valeur = n ;
                System.out.println("Je suis le constructeur avec un argument") ;
                System.out.println("La valeur du compteur = "+valeur) ;
        }

        public void affiche() {
                System.out.println("Valeur du compteur = "+valeur) ;
        }

        public void incremente() {
                valeur++ ;
        }

        public void decremente() {
                if (valeur > 0) valeur-- ;
        }

        public static void main(String argv[])
        {
                Compteur c1 , c2 ;
                c1 = new Compteur() ;
                c1.affiche();
                c2 = new Compteur(15);
                c2.affiche() ;
                int i =0 ;
                while (i++<10) c2.incremente() ;
                System.out.println("Apres 10 incrémentations");
                c2.affiche();
                i=0;
                while (i++<20) c2.decremente() ;
                System.out.println("Apres 20 décrémentations");
                c2.affiche();
```

```
        }
}
```

Les méthodes public sont :
Les 2 **constructeurs** Compteur() et Compteur(int n)
void affiche()
void incremente()
void decremente()

Un constructeur est une méthode qui porte le même nom que sa classe, qui peut recevoir ou non des paramètres, qui ne retourne aucune valeur, pas même void, qui ne peut pas être appelé directement, mais qui est appelé automatiquement (implicitement) quand un objet est instancié sur la classe.

Une méthode peut recevoir ou pas des paramètres, retourner ou pas des valeurs. Les 3 méthodes **void affiche() void incremente() void decremente()**, ne retournent aucune valeur - elles sont de type void - et ne reçoivent aucun paramètre -parenthèses vides -

La méthode main() permet de rendre exécutable l'application. Il n'y a qu'une méthode main() par programme, elle sert de point d'entrée.
Pour une application complexe qui nécessite plusieurs classes, on écrit le plus souvent une classe par fichier, et on écrit la méthode main() dans un ficher de lancement de l'application, qui utilisera les classes qui auront été définies. Les diverses classes d'une telle application sont généralement placées dans un **package**.

La méthode main() peut recevoir des paramètres, qui sont des arguments passés à la ligne de commande de l'interpréteur java.
static void main(String argv[]) : les arguments sont passés sous la forme d'un tableau de String. Dans l'exemple, il n'y a pas de paramètre à passer.

La méthode main() doit être déclarée static, car on n'instancie pas d'objet pour l'utiliser. Un attribut ou une méthode static s'utilise sans instancier d'objet, ce qui n'est pas possible avec des attributs et des méthodes non static.

Instanciation d'objets ou création d'objets.

Dans la méthode main() , on peut lire les lignes suivantes :

```
Compteur c1 , c2 ;
c1 = new Compteur() ;
c2 = new Compteur(15);
```

c1 et c2 sont 2 **variables références** sur la classe Compteur. Puis 2 objets (ou instances) sont créés grâce à l'opérateur **new** ce qui entraîne l'exécution d'un **constructeur**. Dans la classe Compteur, 2 constructeurs ont été prévus, l'un sans paramètre, utilisé pour l'objet c1 et qui initialise la variable d'instance valeur à 0 par défaut,
et l'autre avec un paramètre utilisé pour l'objet c2 qui initialise la variable d'instance valeur avec l'entier passé en paramètre, soit 15 dans l'exemple.

Ensuite, les méthodes de la classe Compteur sont appelées pour l'objet référencé par c2, pour incrémenter, décrémenter et afficher des valeurs.

Le langage Java permet la surcharge des méthodes (et donc des constructeurs) : des méthodes d'une classe portent le même nom mais ont des arguments différents (type ou nombre d'arguments différents), le type de la valeur de retour n'a pas d'importance. On dit que les méthodes surchargées ont une signature différente (grâce aux arguments différents).

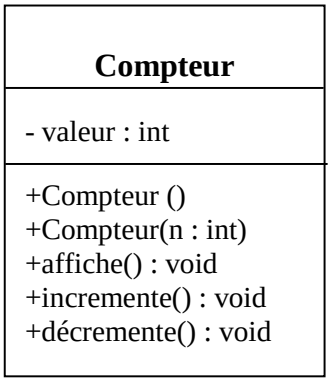
Exemple :

```
public class Exemple {
        public int valeur ;
        public Exemple () { --- }
        public Exemple (int n) { --- }                  // 2ème constructeur
        public void affiche() { --- }
        public void affiche(int i) { ---  }
        public void affiche(char c) { ---  }
        public void affiche(String c) { ---  }          // 4ème méthode affiche
        public void affiche(char []t , int i)  { ---  }
        public boolean affiche(char []t , int i , int j) { ---  }
}
```

La méthode affiche() est ici surchargée 6 fois. Le compilateur sait quelle méthode il doit utiliser grâce au(x) paramètre(s) transmis lors de l'appel.

```
Exemple  ex = new Exemple(2) ;          // appel du 2ème constructeur
ex.affiche("Bonjour") ;                 // exécution de la 4ème méthode
```

Représentation UML de la classe Compteur



Le symbole + indique des méthodes ou attributs publics.

Le symbole - indique qu'un membre est privé.

La méthode main() est volontairement enlevée du diagramme UML de la classe Compteur car elle contient les instructions qui créent et utilisent des objets Compteur : elle doit être placée dans une classe d'application ou de test.

Attention

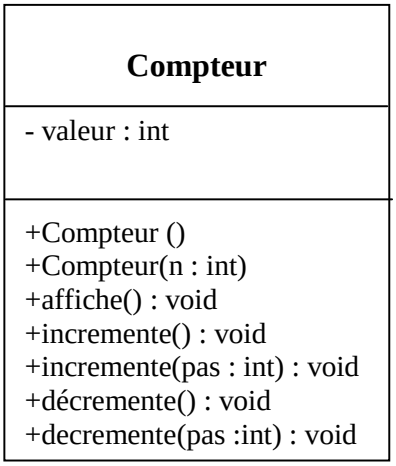
Si une application java en mode console est bloquée, on peut reprendre le contrôle sur la ligne de commande en terminant l'application par les touches **Ctrl C** .

Exercice 1 :

On surcharge les méthodes incremente() et decremente().

incremente(int pas) : augmente le compteur de la valeur passée en argument.
decremente(int pas) : diminue le compteur de la valeur passée en argument.

Ecrire le code de ces méthodes et tester le fonctionnement.



5. Les packages, les options -classpath et -d

Un ***package*** désigne un ensemble de classes compilées (fichiers « .class ») qui seront situées dans un même répertoire. Le nom du package définit le nom du répertoire où se situent les fichiers compilés « .class », (en remplaçant le séparateur / de répertoire par un point).

Le répertoire et le package portent le même nom, par exemple :

- Un **package lib.compt** correspondrait au répertoire **\lib\compt** sous Windows.
- Ce package contiendrait des **fichiers .class** qui seraient donc placés dans le répertoire **\lib\compt**.
- Le fichier **compt.Compteur.class** correspond au fichier **Compteur.class** dans le package **compt**.

Les classes Java sont regroupées par Oracle dans des packages, en voici quelques-uns :

`java.applet` Classes de base pour les applets
`java.awt` Classes d'interface graphique AWT
`java.io` Classes d entrées/sorties (flux, fichiers)
`java.lang` Classes de support du langage
`java.math` Classes permettant la gestion de grands nombres.
`java.net` Classes de support réseau (URL, sockets)
`java.rmi` Classes pour les méthodes invoquées à partir de machines virtuelles non locales.
`java.security` Classes et interfaces pour la gestion de la sécurité.
`java.sql` Classes pour l'utilisation de JDBC.
`java.text` Classes pour la manipulation de textes, de dates et de nombres dans plusieurs langages.
`java.util` Classes d'utilitaires (vecteurs, hashtable)
`javax.swing` Classes d'interface graphique

Java permet de regrouper les classes des applications créées par les développeurs dans des packages afin de faciliter la modularité.
En effet, une application complexe nécessite souvent la création de plusieurs classes. Il est alors conseillé de les regrouper dans un package en écrivant la ligne suivante au début de chaque fichier **.java** :

package NomDuPackage ;

par exemple :

package compt ;
// La classe définie dans le fichier correspondant est placé dans le package
// nommé "compt"

On suppose l'existence d'un dossier **lib** (dans le répertoire courant).
Attention : le répertoire "lib" doit être créé avant de compiler.
On suppose l'instruction package compt ; placée en 1er dans le fichier Compteur.java.

La compilation de la 1ère classe par

javac –d lib Compteur.java

entraîne automatiquement :

- **grâce à l'option –d la création d'un répertoire "compt" portant le nom du package dans le répertoire "lib",**
- **le remplissage de ce répertoire avec le fichier Compteur.class de la classe compilée.**

L'option -d (d comme destination) utilisée avec javac permet d'indiquer au compilateur le répertoire de destination des fichiers .class à générer.

Admettons qu'ensuite, on souhaite compiler une autre classe nommée **CompteurBis** qui **utilise la classe Compteur.** La compilation de la classe **CompteurBis** doit être réalisée par :

javac –d lib -classpath lib CompteurBis.java

L'option -classpath utilisée avec javac ou java permet d'indiquer le (ou les) répertoire(s) où se situent les fichiers .class dont javac ou java ont besoin.

Remarques **:**

- **On peut remplacer –classpath par –cp.**

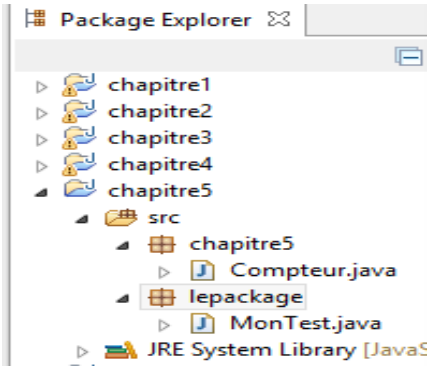
Puis pour les classes suivantes, on utilisera toujours la même commande :

javac –d lib -cp lib NomClasseXXX.java

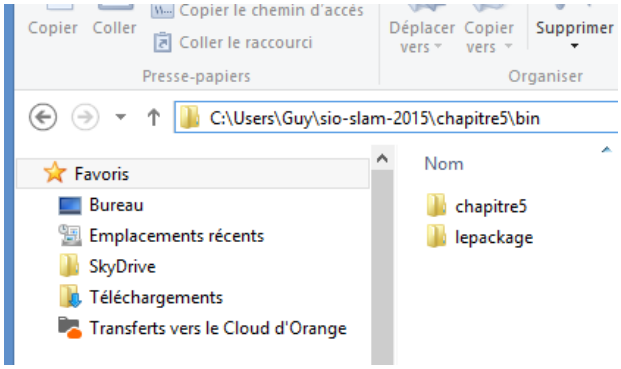
qui place automatiquement les classes compilées dans le répertoire de même nom que celui du package à l'intérieur du répertoire "lib".

Les packages avec Eclipse

Eclipse crée automatiquement un dossier du nom du nouveau projet créé.
Eclipse crée alors dans ce dossier 2 nouveaux dossiers **src** et **bin**.
Les dossiers **src** et **bin** contiennent exactement la même arborescence : un dossier de nom du projet qui est le package par défaut créé, et d'éventuels dossiers correspondants aux divers packages créés en plus par le programmeur.
Les dossiers **src** et **bin** sont organisés de la même manière, mais le dossier **src** contient les fichiers sources (.java) et le dossier **bin** les classes compilées (.class).



La vue package Explorer sous Eclipse : **un projet chapitre5**, le package de même nom chapitre5 avec un package supplémentaire lepackage.



L'explorateur sous Windows : le dossier chapitre5\bin créé par défaut avec les 2 dossiers correspondants aux 2 packages. Le dossier chapitre5 créé par défaut, le dossier le package correspondant au package lepackage.

Exercice 2 : Exercice sur les packages en ligne de commande.

Créer le répertoire **lib** dans le répertoire courant.

Séparer le fichier Compteur.java en 2 fichiers qui seront dans le répertoire courant.

- **Compteur1.java** qui ne contiendra que la définition de la classe Compteur1.
- **Test1.java** qui ne contiendra qu'une méthode main() pour lancer l'application .

Attention : on doit toujours déclarer une classe en java, ce qui signifie que le **fichier Test1.java contiendra une classe Test1, dans laquelle on aura comme seule méthode : la méthode main()**.

Au début du fichier Compteur1.java, écrire la ligne suivante qui créera un package pour le fichier Compteur1.class : **package compt ;**

Au **début du fichier Test1.java**, écrire la ligne suivante qui importera la classe Compteur1 dans le fichier Test1.class : **import compt.Compteur1 ;**

Compiler en premier le fichier Compteur1.java :

javac –d lib Compteur1.java

Cette commande crée le fichier **Compteur1.class** dans un répertoire .**\lib\compt** car **:**
- dans le code de Compteur1.java on a spécifié «**compt** » comme nom du package ce qui signifie que la classe Compteur1 est placée dans le package compt;
- l'option "**–d lib**" permet d'indiquer au compilateur qu'il doit placer "ce qu'il produit" dans le répertoire "lib". Ainsi le compilateur crée le répertoire "compt" (correspondant au nom du package) dans le répertoire "lib" et place dans "compt" le fichier Compteur1.class.
La création du répertoire "compt" est réalisée automatiquement par javac et il n'est pas nécessaire de le créer avant.

Regarder le contenu du répertoire lib, retrouver le répertoire correspondant au package compt et regarder à l'intérieur de ce répertoire.
Dans cet exemple, un seul fichier, Compteur1.class, est dans le package compt.

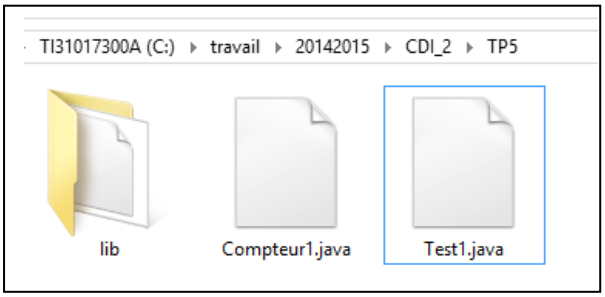
Compiler en second le fichier Test1.java.

javac -d lib -classpath lib Test1.java

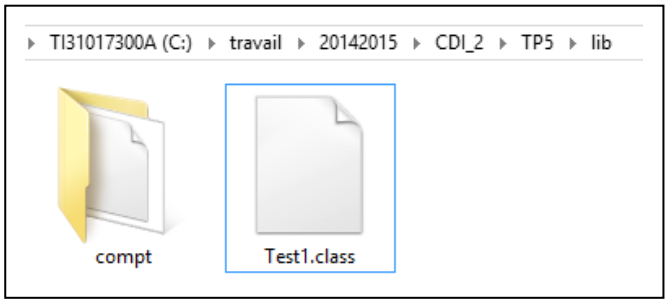
L'option –classpath lib indique au compilateur de rechercher les fichiers .class dont il a besoin dans le répertoire lib (Ici, il a besoin du fichier Compteur1.class car la classe Compteur1 est instanciée dans la méthode main() de la classe Test1).

Résultats :

- le contenu du répertoire de travail :



- le répertoire lib :



Le fichier Test1.class est dans le répertoire lib : l'usage veut que les classes de test ne soient pas mises dans les packages.

Exécuter ensuite Test1.class avec l'interpréteur java, en écrivant la ligne de commande :

java -classpath lib Test1

L'option –classpath indique le répertoire où sont placés les fichiers .class (dont l'interpréteur java a besoin), ici le répertoire lib pour la classe Test1 et la classe Compteur1 (située dans le package compt).
Il est possible de préciser plusieurs répertoires en séparant leurs noms par le caractère **;** .
Par exemple : java -classpath "lib;lib1;c:/applijava/lib2" test

Modifier la ligne **import compt.Compteur1 ;** par **import compt.* ;** dans le fichier Test1.java. **import compt.*** permet d'importer toutes les classes du package compt.
Vérifier le bon fonctionnement.

Exercice 3

Créer une classe CompteurBorne, dans laquelle le compteur devra rester entre une valeur supérieure et une valeur inférieure :

- après plusieurs incrémentations successives la valeur du compteur atteint la valeur maximale et il est bloqué à cette valeur supérieure et n'accepte plus d'être incrémenté,
- après plusieurs décrémentations successives la valeur du compteur atteint la valeur minimale et il est bloqué à cette valeur inférieure et n'accepte plus d'être décrémenté.

Séparer l'application en 2 fichiers et placer le fichier CompteurBorne.class dans le package "compt".

Proposer une représentation UML pour la classe CompteurBorne.

Exercice 4

Créer une classe CompteurCyclique, dans laquelle le compteur ne pourra pas dépasser une valeur supérieure. De plus lorsqu"il atteint la valeur maximale, une incrémentation de 1 lui fait prendre alors la valeur inférieure. Le compteur ne pourra pas non plus dépasser une valeur inférieure, et prendra alors la valeur supérieure après une décrémentation de 1.

Séparer l'application en 2 fichiers et placer le fichier CompteurCyclique.class dans le package "compt".

Proposer une représentation UML pour la classe CompteurCyclique.

Exercice 5 Documentez vos classes avec la commande javadoc du jdk

La commande **javadoc**, en ligne de commande, permet de créer une documentation du style javadoc, dans des fichiers html, sur les classes que l'on écrit.

►Réaliser ceci pour la classe Compteur1 :

javadoc Compteur1.java

►Ouvrir le fichier Compteur.html qui documente la classe Compteur1.

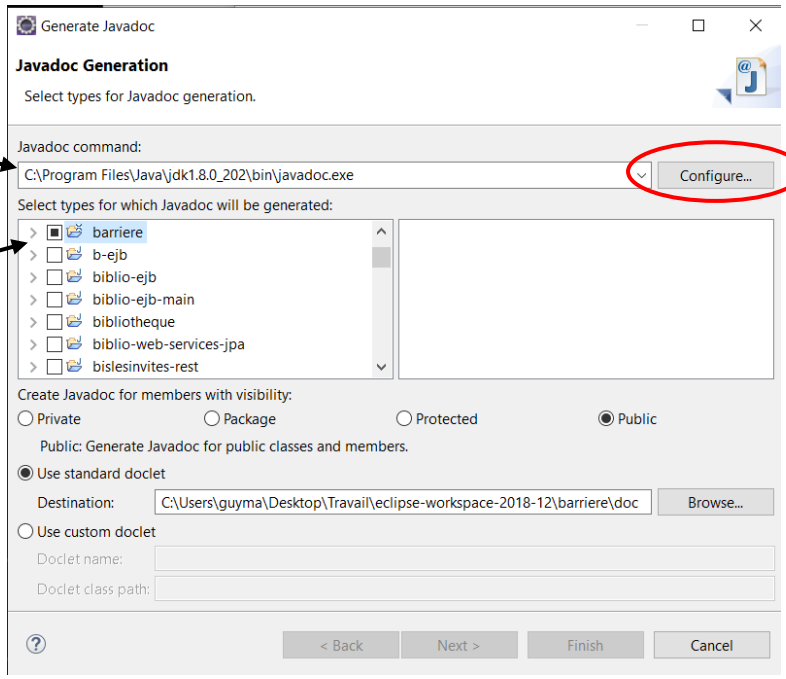
►Utiliser la commande d'Eclipse pour générer la documentation HTML d'une classe :

menu Project/Generate Javadoc…

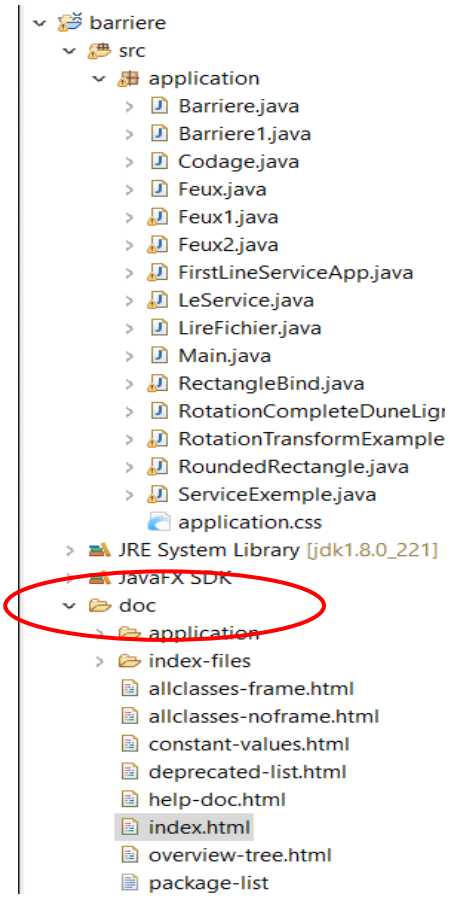
Le formulaire suivant est affiché :

1) Ce champ peut être vide ou faux, cliquer sur Configure pour rechercher la commande javadoc.exe dans le dossier **bin** du JDK

2) Sélectionner/Ouvrir le projet contenant les classes à documenter.
3) Ouvrir le package utile, les classes s'affichent dans la fenêtre de droite.
4) Sélectionner les classes à documenter.
5) Cliquer sur Finish.



Un dossier doc contenant les fichiers HTML est créé dans le dossier du projet.



6) Clic droit sur **index.hlml**
puis **Open With**
puis **Web Browser**

Le navigateur intégré à Eclipse affiche alors le document HTML.

7) Rechercher sur Internet comment ajouter des documenter les attributs et les méthodes de la classe Compteur en utilisant javadoc.

Exercice 6 Le désassembleur javap.

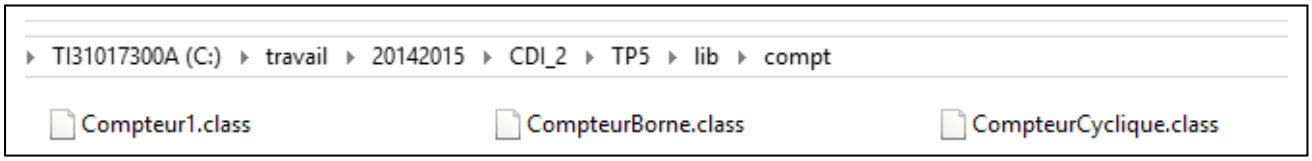
javap -c lib\compt\Compteur1

En écrivant la ligne de commande ci-dessus, on obtient un désassemblage du code. Le résultat est affiché à l'écran et mérite beaucoup d'attention pour être compris.

Exercice 7 Créez vos archives avec la commande jar du jdk.

La commande **jar** permet de créer des fichiers archives compressés, d'extension **.jar**, et de les décompresser.
On veut archiver dans un fichier DesCompteurs.jar les fichiers Compteur1.class, CompteurBorne.class et CompteurCyclique.class.

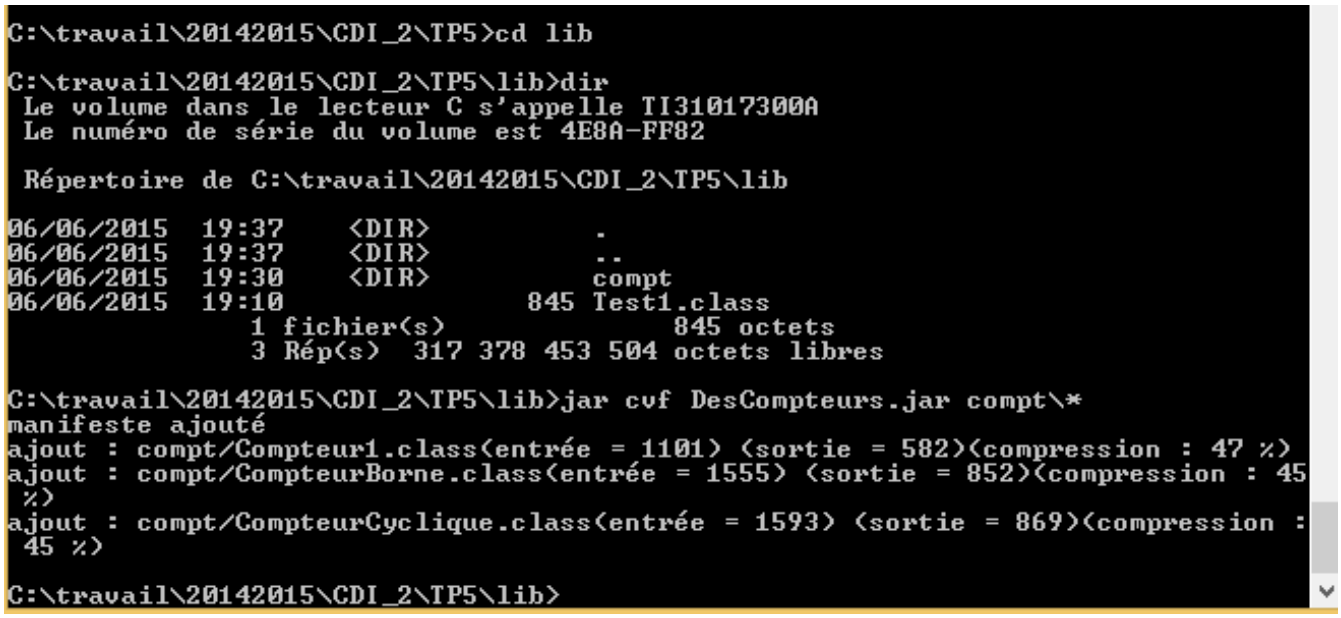
A ce point du TP, ces 3 fichiers doivent être situés dans "lib\compt" dans votre répertoire de travail (chez moi, le répertoire de travail est le répertoire TP5) :



1 . Placer vous dans le répertoire lib avec la commande :
cd lib

2. Compression : Le fichier DesCompteurs.jar est créé par la commande :

jar cvf DesCompteurs.jar compt*



Les taux de compression sont mentionnés.

3. Supprimer les fichiers .class des compteurs et le répertoire "compt".

4. Retourner dans votre répertoire de travail :

cd ..

4. Exécuter ensuite **Test1.class** avec l'interpréteur java, en écrivant la ligne de commande :

java -cp lib Test1

Quel est le problème ?
Il faut indiquer à l'interpréteur java que les classes des compteurs se trouvent dans l'archive DesCompteurs.jar grâce à l'option -cp (-classpath) :

java -cp lib;lib\DesCompteurs.jar Test1

Remarquez l'utilisation du point virgule pour l'option -cp :

- Test1.class se trouve dans lib
- et les .class des compteurs dans l'archive .jar

Pour le classpath, il faut donc indiquer le répertoire "lib" et le chemin d'accès à l'archive DesCompteurs.jar

5. Décompression :

Retourner dans le répertoire lib et décompresser DesCompteurs.jar avec la commande :

cd lib
jar xf DesCompteurs.jar

Le répertoire "compt" et son contenu sont alors reconstitués.

Les commandes javadoc et jar seront utiles pour des projets plus importants.