Langage Java Chapitre 3

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

Présentation de la classe Compteur enregistrée dans le fichier Compteur.java.

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. 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

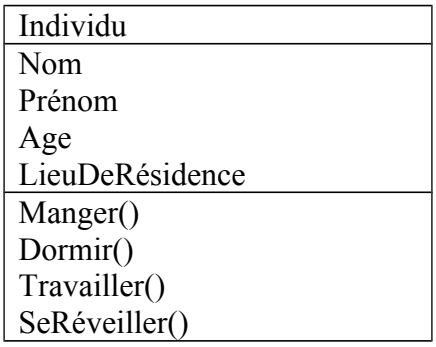
Les attributs :

Nom
Prénom
Age
Lieu de résidence

Les méthodes

Manger
Dormir
Travailler
SeRéveiller

La modélisation UML propose le diagramme de classe simplifié suivant :



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, (appelée aussi donnée membre ou propriété) public de type int.

```
public class Compteur
{
public 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();
 }
}
```

Exercice 1 : 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.

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 mais qui n'a aucun type.

```
public class Compteur
{
public 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) ;
        }
---
}
```

Exercice 2 : 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
{
public 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) 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();
    }
}
```

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. Une classe ou une méthode static s'utilise sans instancier d'objet, ce qui n'est pas possible avec des classes 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 objets **déclarés** sur la classe Compteur. Puis ils sont **instanciés** (créés) grâce à l'opérateur **new** qui appelle un **constructeur**. Dans la classe Compteur, 2 constructeurs ont été prévus, l'un sans paramètre, utilisé par l'objet c1 et qui initialise la variable d'instance valeur à 0 par défaut,
et l'autre avec un paramètre utilisé par l'objet c2 et 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 utilisées par les objets, 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, le type retourné 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) { ---  }
}
```

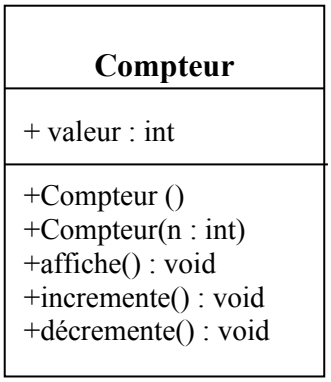
La méthode affiche() est ici surchargée 6 fois. Le compilateur sait quelle méthode il doit appeler grâce à l'argument donné.

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

Représentation UML de la classe Compteur

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 peut être placée dans une classe d'application ou de test.

Le + indique des méthodes ou attributs publics.



Exercice 3 :
Prévoir les résultats du programme de la classe Compteur modifiée ci-dessus (chapitre 4).
Tester la classe Compteur modifiée ci-dessus.

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 4 :

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.

Compteur
+ valeur : int
+Compteur () +Compteur(n :int) +affiche() : void +incremente() : void +incremente(pas :int) :void + décremente() : void +decremente(pas :int) :void

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

Un ***package*** désigne un ensemble de classes compilées (fichiers « .class »), situées dans un même répertoire. La position où se situe le fichier compilé « .class » défini le nom du package, (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**.

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 ;

On suppose l'existence d'un dossier **lib**.

Attention : le répertoire "lib" doit être créé avant de compiler.

La compilation de la 1ère classe par

javac –d lib -classpath lib NomClasse1.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 NomClasse1.class de la classe compilée.**

Remarques **:**

- **L'option classpath n'est ici à priori pas utile pour compiler la 1ère classe de l'application, mais sera nécessaire pour compiler d'autres classes utilisant cette classe.**
- **On peut remplacer –classpath par –cp.**

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

javac –d lib -classpath lib NomClasseXXX.java

qui place automatiquement les classes compilées dans le répertoire de même nom que celui du package.

Ainsi, tous les fichiers .class sont placés dans un même répertoire de même nom que celui du package.

L'option classpath

La compilation des fichiers **NomClasseXXX.java** peut échouer si les **NomClasseXXX.java** utilise la classe **NomClasse1.java,** par exemple une relation d'héritage.

Il est alors nécessaire d'indiquer au compilateur où se situent les fichiers .class dont il a besoin en utilisant l'option –classpath :

javac –d lib -classpath lib NomClasseXXX.java

Ainsi, le compilateur peut accéder aux fichiers .class situés dans lib.

Exercice 5 : Exercice sur les packages.

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

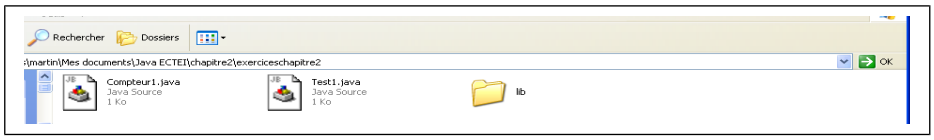
Séparer le fichier Compteur.java en 2 fichiers qui seront dans le répertoire courant.

- **Compteur1.java** qui ne contiendra que la 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 -classpath lib Compteur1.java

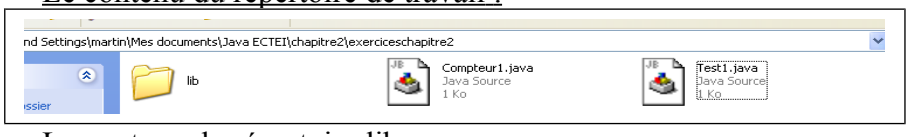
La compilation **javac –d lib** crée le fichier **Compteur1.class** dans un répertoire .**\lib\compt**, étant donné que dans le code de Compteur1.java on a spécifié «**compt** » comme nom du package, et que dans la ligne de commande de compilation **–d lib ,** le lib signifie qu'il faut créer le répertoire compt dans le répertoire lib lui-même placé dans le répertoire courant.
La création du répertoire compt est réalisée automatiquement par javac et il n'est pas nécessaire de le créer avant.
L'option –classpath lib est ici inutile car cette classe n'est utilisée par aucune autre classe et le package ne contient qu'un seul fichier.

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.

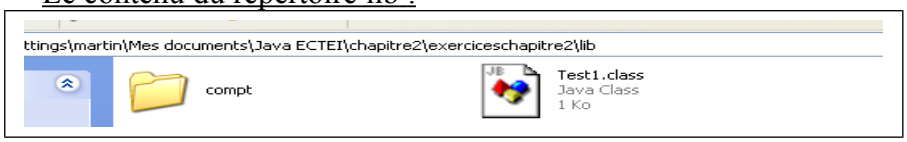
Compiler en second le fichier Test1.java.

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.
Le contenu du répertoire de travail :



Le contenu du 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 les répertoires où sont placés les fichiers .class, ici le répertoire lib pour la classe Compteur1 et 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 6

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 créer un package.

Proposer une représentation UML pour la classe CompteurBorne.

Exercice 7

Créer une classe CompteurCyclique, dans laquelle le compteur ne pourra pas dépasser une valeur supérieure, atteignant 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 créer un package.

Proposer une représentation UML pour la classe CompteurCyclique.

Exercice 8 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 Compteur :

javadoc Compteur.java

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

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

Exercice 9 Le désassembleur javap.

javap -c Compteur

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 10 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 Compteur.class, Test1.class et lib/compt/Compteur1.class.

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

jar cvf DesCompteurs.jar Compteur.class Test1.class lib\compt\Compteur1.class

```
F:\Java ECTEI\chapitre2\exerciceschapitre2>jar cvf Descompteurs.jar Compteur.class Test1.class compt\Compteur1.class
manifest ajoutÚ
ajout : Compteur.class(entrÚe = 1171) (sortie = 666)(43% compressÚs)
ajout : Test1.class(entrÚe = 685) (sortie = 467)(31% compressÚs)
ajout : compt/Compteur1.class(entrÚe = 866) (sortie = 487)(43% compressÚs)

F:\Java ECTEI\chapitre2\exerciceschapitre2>dir
 Le volume dans le lecteur F n'a pas de nom.
 Le numéro de série du volume est 70E7-2C12

 Répertoire de F:\Java ECTEI\chapitre2\exerciceschapitre2

09/10/2006  16:38    <REP>          .
09/10/2006  16:38    <REP>          ..
10/10/2006  09:33               507 Test1.java
10/10/2006  09:34               350 Compteur1.java
10/10/2006  09:35    <REP>          compt
10/10/2006  09:35               685 Test1.class
10/10/2006  09:38             1 171 Compteur.class
10/10/2006  09:46               891 Compteur.java
10/10/2006  10:18             2 333 Descompteurs.jar
               6 fichier(s)          5 937 octets
               3 Rép(s)     553 025 536 octets libres
```

Les taux de compression sont mentionnés.

Décompression :

Supprimer les fichiers .class et le répertoire compt du répertoire courant, puis les reconstituer depuis le fichier DesCompteurs.jar avec la commande :

jar xf DesCompteurs.jar

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