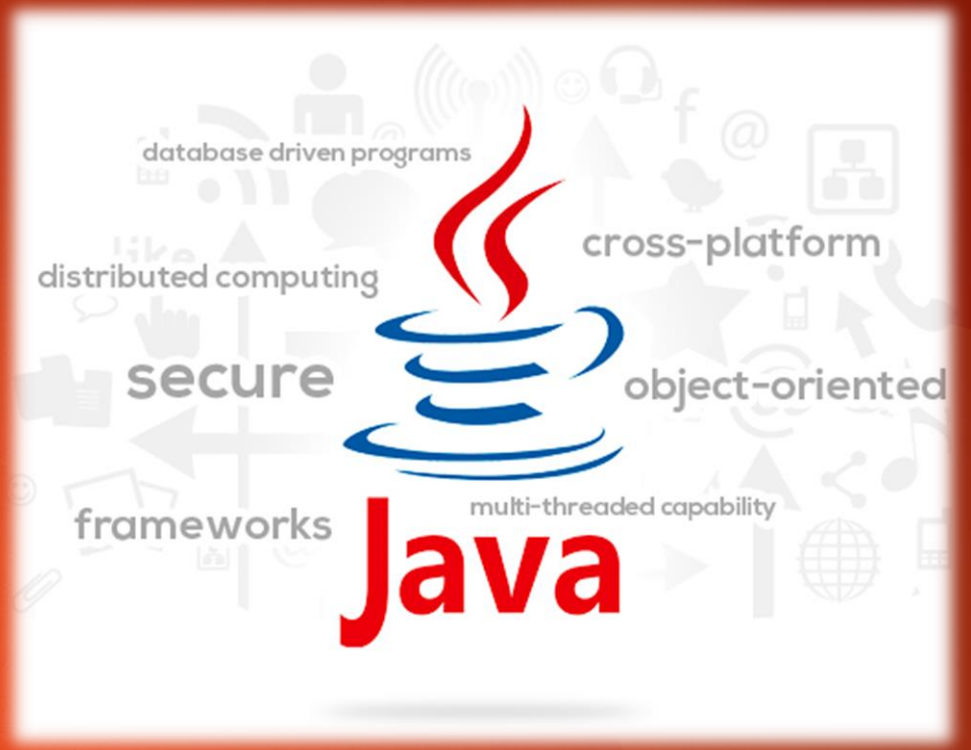

Exposition sur java

PAR: HALLOUL.TAREK

Sommaire

I. Packages java
II. Test et conditions java

Packages java

Définition : Java permet de regrouper les classes en ensembles appelés packages afin de faciliter la modularité. Parmi les paquetages de Java vous avez :

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 réseaux(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 securité.

Java.sql: Classes pour l'utilisation de JDBC.

Java.text : Classes pour la manipulation de texte, de dates et de nombres dans plusieurs langues.

Java.util : Classes d'utilitéres (vecteurs, hashtable)

Java.swing : Classes d'interface grafique.

Caractéristiques

Nom: chaque paquetage porte un nom. Ce nom est soit un simple identificateur ou une suite d'identificateurs séparés par des points. L'instruction permettant de nommer un paquetage doit figurer au début du fichier source (.java) comme suit:

```
---------- Fichier Exemple.java ----------
package nomtest; // 1ere instruction
class    MTest {}
class    Uneautre {}
public class Exemple {// blabla suite du code ...}
---------- Fin du Fichier ----------
```

Ainsi par cette opération nous venons d'assigner toutes les classes du fichier Exemple.java au paquetage nomtest.
Si l'instruction package était absente du fichier, alors c'est le paquetage par défaut qui est pris en considération. Ainsi toutes les classes du le dit fichier vont appartenir au paquetage par défaut.

Utilisation: Si vous instanciez un objet d'une classe donnée, le compilateur va chercher cette classe dans le fichier où elle a été appelée, nous dirons que cette recherche a eu lieu dans le paquetage par défaut même si ce dernier porte un nom.

Si votre appel fait référence à une classe appartenant à un autre paquetage, vous devez aider le compilateur à retrouver le chemin d'accès vers cette classe en procédant comme suit:

- En mentionnant le chemin complet d'accès à la classe : En citant les noms complets des paquetages nécessaires:

```
nomtest.Uneautre mm = new nomtest.Uneautre();
```

C'est une opération fastidieuse si vous avez affaire à une arborescence de paquetages ou bien à un nombre important de classes dans un même paquetage. Source d'erreurs donc.

- En important les paquetages utiles : Vous devez utiliser pour cela l'instruction import comme suit:

```
------------ Fichier Test.java ------------
        import nomtest.Uneautre;
        public class Test {
        // balblabla ...
        Uneautre mm = new Uneautre();
        } /
        / blablabla ....
----------------Fin Fichier-----------------
```

C'est bien beau! Mais si vous avez besoin d'utiliser aussi une autre classe se trouvant dans le paquetage nomtest. Vous pouvez le faire soit en important nomtest comme dans le cas de la classe Uneautre ou bien écrire ce qui suit:

```
import nomtest.*;
```

Par cette instruction vous avez demandé d'importer toutes les classes se trouvant dans le paquetage nomtest.

Remarques:

-1- Le paquetage java.lang est importé automatiquement par le compilateur.

-2- import nomtest.;*

Cette instruction ne va pas importer de manière récursive les classes se trouvant dans nomtest et dans ses sous paquetages. Elle va uniquement se contenter de faire un balayage d'un SEUL niveau. Elle va importer donc que les classes du paquetage nomtest.

```
import java.awt.*;
import java.awt.event.*;
```

La première instruction importe toutes les classes se trouvant dans le paquetage awt. Elle ne va pas importer par exemple les classes se trouvant dans le sous paquetage event. Si vous avez besoin d'utiliser les classes de event, vous devez les importer aussi.

-3- Si deux paquetages contiennent le même nom de classe il y a problème! Par exemple la classe List des paquetages java.awt et java.util ou bien la classe Date des paquetages java.util et java.sql etc.

Pour corriger ce problème vous devez impérativement spécifier le nom exact du paquetage à importer (ou à utiliser).

```
java.awt.List (ou) java.util.List ; java.util.Date (ou) java.sql.Date
import java.awt.*; // ici on importe toutes les classes dans awt
import java. util.*; // ici aussi
import java.util.List; // ici on précise la classe List à utiliser
```

-4- Vous pouvez regrouper toute une hiérarchie de classes (ou un seul répertoire) dans un fichier ZIP ou JAR (grosso modo contient les mêmes options que la commande unix: tar)

```
zip montest.zip montest/*
jar -cvf montest.jar montest/*
```

c: pour créer ; v: pour un affichage détaillé (verbose) ; f: pour préciser le nom de fichier
Par la suite vous pouvez préserver dans un endroit unique par exemple le répertoire lib tous les zip ou jar. Si vous procédez ainsi, vous devez indiquer au compilateur comment accéder au répertoire lib en précisant son chemin exact comme suit:

- ***À travers la ligne de commande en utilisant l'option -classpath***

```
javac -classpath chemin1 nom_fic.java
```

- ***En définissant la variable d'environnement CLASSPATH:***

Dans le fichier .cshrc:

```
setenv CLASSPATH .
setenv CLASSPATH ${CLASSPATH}:UnChemin/lib/montest.zip
```

Dans le fichier Autoexec.bat (dans Windows, sinon définir une variable système si vous utilisez xp ou 2000). Attention dans Windows les ":" sont remplacées par ";".

```
set CLASSPATH=.
set CLASSPATH %CLASSPATH%;UnChemin/lib/montest.zip
```

Exemple Pratique:

Étudier les paquetages fournis dans le fichier TestPackage.zip

Test et conditions java

I – Le test de base:

I – 1 – Le test simple:

Un test permet de savoir si la valeur d'une variable est égale à un nombre particulier, et d'exécuter un morceau de code spécifique. Le programme suivant, par exemple, teste si la variable ma_variable est égale à 30, et affiche un message si le test est positif. Essayez ce programme dans les 2 cas suivants :

- La variable ma_variable est initialisée à la valeur 30 sur la ligne
- La variable ma_variable est initialisée à une valeur autre que 30 sur la ligne

```
1   Class test {
2         public static void main (String[] args) {

3         /*déclaration la variable ma_variable de type int: */
4         int ma_variable;
5         /*initialisation de la variable ma_variable: */
6         ma_variable=30;
7         /*test si la variable ma_variable est égal est égal à 30*/
8         if (ma_variable==30)
9            {
10               System.out.println("ma_variable vaut trente")
11           }
12      System.out.println("Fin du programme");
13    }
14  }
```

Code source 1

Remarques à propos du Code source 1 :

- *En Java, l'instruction permettant de faire un test est le if*
- *Un test d'égalité est fait par l'opérateur == (deux signes « égale » à la suite)*
- *Dans un if, la condition du test doit être mise entre parenthèses : `(ma_variable == 30)` sur la ligne 8.*
- *Si la condition est vraie, alors le bloc d'instruction suivant le if est exécuté. Ce bloc doit être encadré par des accolades (commençant ligne 9 et finissant ligne 11 dans le Code source1), et peut contenir plusieurs lignes à exécuter.*

I – 2 – Le test complet:

Dans le Code source 2 suivant, un message différent est affiché, et ce, que la variable soit égale ou différente à 30 :

```
1   Class  test  {
2            public static void main (String[] args) {
3                     int  ma_variable;
4                     ma_variable=30;

5           if  (ma_variable==30)
6              {
7                System.out.println(“ma_variable vaut trente”);
8              }
9          else
```

```
10            {
11        System.out.println(”ma_variable ne vaut pas trente”);
12            }
13      }
14  }
```

---

code source 2

Remarques à propos du Code source 2 :

- La condition placée dans le if sur la ligne 5 est « la variable ma_variable est égale à 30 ».
- Si cette condition est VRAIE, alors le bloc placé entre accolade commençant à la ligne 6 et finissant à la ligne 8 est exécuté.
- Si cette condition est FAUSSE, alors le bloc placé entre accolade commençant à la ligne 10 et finissant à la ligne 12 est exécuté.
- L'instruction else, placée sur la ligne 9, veut dire « sinon », et désigne le bloc à exécuter si la condition du if est FAUSSE.
- Pour bien observer le fonctionnement et le déroulement du Code source 2, on pourra l'exécuter pas à pas, à l'aide du déboguer de JBuilder.

II – Les différents tests possibles:

Nous venons de voir, à travers les Code source 1 et Code source 2 comment effectuer un test d'égalité : il faut utiliser l'opérateur ==. Java permet de faire d'autres test, en utilisant les opérateurs de comparaison mentionnés dans le tableau suivant :

Les opérateurs de comparaison en Java

Opérateur	Signification	Exemple
==	égal à	`if (i == 5)`
!=	différent de	`if (i != 5)`
<	inférieur à	`if (i < 5)`
<=	inférieur ou égal à	`if (i <= 5)`
>	supérieur à	`if (i > 5)`
>=	supérieur ou égal à	`if (i >= 5)`

Le *Code source 3* suivant teste la variable i par rapport à 10, en utilisant chacun des 6 opérateurs de comparaison, et affiche un message spécifique en fonction des résultats des tests :

```
 1  class test {
 2       public static void main(String[] args) {
 3            int i;
 4            i=8;

 5            if (i == 10) { System.out.println("i égal à 10"); }
 6            else { System.out.println("i pas égal à 10"); }

 7            if (i != 10) { System.out.println("i différent de 10"); }
 8            else { System.out.println("i pas différent de 10"); }

 9            if (i < 10) { System.out.println("i inférieur à 10"); }
10            else { System.out.println("i pas inférieur à 10"); }

11            if (i <= 10) { System.out.println("i inf. ou égal à 10"); }
12            else { System.out.println("i pas inf. ou égal à 10"); }

13            if (i > 10) { System.out.println("i supérieur à 10"); }
14            else { System.out.println("i pas supérieur à 10"); }

15            if (i >= 10) { System.out.println("i sup. ou égal à 10"); }
16            else { System.out.println("i pas sup. ou égal à 10"); }

17       }
18  }
```

Code source 3

Remarques à propos du *Code source 3* :

- Dans le *Code source 3*, les bloc associés aux différents **if** et aux différents **else** ont été écrits sur une seule ligne : ces blocs sont toujours encadrés par des accolades
- Cette manière d'écrire le code source est pratique pour diminuer le nombre de lignes du programme, mais réduit considérablement sa lisibilité

III – Utilisation des opérateurs logiques dans les conditions de test :

Pour formuler des conditions de test complètes, précises, et de complexité quelconques, Java propose l'emploi des opérateurs logiques. Ces opérateurs sont :

- Le **ET** logique
- Le **OU** logique
- Le **NON** logique

Les opérateurs logiques en Java		
Opérateur	**Signification**	**Exemple**
`&&`	ET logique	`if ((i==5) && (j==8))`
`||`	OU logique	`if ((i>10) || (i<2))`
`!`	NON logique	`if !(i==9)`

Rappel des tables de vérité des opérateurs logiques de base :

Opérateur ET		
a	**b**	**a && b**
faux	faux	**FAUX**
faux	vrai	**FAUX**
vrai	faux	**FAUX**
vrai	vrai	**VRAI**

Opérateur OU		
a	**b**	**a || b**
faux	faux	**FAUX**
faux	vrai	**VRAI**
vrai	faux	**VRAI**
vrai	vrai	**VRAI**

Opérateur NON	
a	**!a**
faux	**VRAI**
vrai	**FAUX**

A retenir :

- Le résultat d'un **ET** est **VRAI** si et seulement si les deux conditions liant le **ET** sont **VRAIES** toutes les deux
- Le résultat d'un **OU** est **FAUX** si et seulement si les deux conditions liant le **OU** sont **FAUSSES** toutes les deux

Exemple de programme utilisant les opérateurs logiques pour formuler des conditions dans les tests :

```
1  class logique {
2       public static void main(String[] args) {
3            int a,b;
4            a=5;
             b=8;
5
6            if ((a == 10) && (b == 10))
7                 {
8                  System.out.println("a et b sont tous les deux égaux à dix");
9                 }
10
11           if ((a == 5) && !(b == 5))
12                {
13                 System.out.println("a est égal à 5 et b est différent de 5");
14                }

             if ((a > 2) && (a < 14))
                  {
                   System.out.println("a est compris dans l'intervalle ]2;14[");
                  }

             if !((b >= 10) && (b <= 20))
                  {
                   System.out.println("b n'est pas dans l'intervalle [10;20]");
                  }

       }
}
```

Code source 4

Pour exécuter le *Code source 4*, donnez des valeurs aux variable a et b afin que chacun des 4 tests soit vrai à tour de rôle.

IV – PROPRIÉTÉS LOGIQUES DES CONDITIONS

Pour tester une condition, il y a souvent plusieurs manière d'écrire ce test. Par exemple, pour tester si une variable est différentes d'une valeur, on peut soit utilisé l'opérateur de comparaison « différent de », soit utiliser l'opérateur de comparaison « égal à » associé à l'opérateur logique NON, ce qui donne les deux tests suivants qui sont strictement équivalents :

```
/* teste la différence : */
if (a != 5)
     {
       System.out.println("a est différent de 5");
     }

/* teste la non égalité : */
if !(a == 5)
     {
       System.out.println("a n'est pas égal à 5");
     }
```

Le tableau suivant montre des exemples de condition égales deux à deux. Remarquez que le théorème de De Morgan permet d'expliquer les 4 dernières lignes :

Conditions équivalentes	
La condition	**est strictement équivalente à**
`a==5`	`!(a!=5)`
`a!=5`	`!(a==5)`
`(a!=5) && (b!=8)`	`!((a==5) || (b==8))`
`(a!=5) || (b!=8)`	`!((a==5) && (b==8))`
`!((a!=5) && (b!=8))`	`(a==5) || (b==8)`
`!((a!=5) || (b!=8))`	`(a==5) && (b==8)`

Le tableau suivant montre des exemples de condition inverses :

Conditions inverses	
Le contraire de	**est**
`a==5`	`a!=5`
`a!=5`	`a==5`
`a > 5`	`a <= 5`
`a >= 5`	`a < 5`
`a < 5`	`a >= 5`
`a <= 5`	`a >5`
`(a==5) && (b==8)`	`(a!=5) || (b!=8)`
`(a==5) || (b==8)`	`(a!=5) && (b!=8)`

La maîtrise de l'algèbre de Boole permet généralement de simplifier l'écriture des conditions logiques, rendant ainsi le programme bien plus lisible, et sans en changer son fonctionnement.