9 Package, documentation

Séance 9 :
Objectif : consolidation héritage, interface, package, éventuellement documentation
fin 46, 47, 48
51, 52
En TME 49, 50 rapidement + site web

Exercice 49 – Documentation Java, package

Rappel : Java est fourni avec un ensemble de classes. Par exemple, les classes String, Math, System. Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles appelés packages. Cet exercice a pour but de vous familiariser avec la documentation fournie avec Java, ainsi qu'avec les packages.
Allez sur le site de l'UE, puis cherchez le lien vers la "Documentation Java".

Q 49.1 Recherchez la classe `Random`. Combien a-t-elle de constructeurs ? Combien a-t-elle de méthodes ? A quel package appartient cette classe `Random` ? La classe `Math` appartient-elle au même package que la classe `Random` ? Aide : les packages sont écrits tout en minuscule.

2 constructeurs, 10 méthodes + 11 méthodes héritées de la classe Object
java.util
Non, la classe `Math` appartient au package java.lang.

Q 49.2 Recherchez la classe `ArrayList`. D'après la documentation, combien a-t-elle de champs ? Combien a-t-elle de constructeurs ? Combien environ a-t-elle de méthodes ? De quelles classes hérite-t-elle ? A quel package appartient cette classe `ArrayList` ?

1 champs hérité
3 constructeurs
20 méthodes+26 méthodes héritées
Voici l'arbre d'héritage :

```
java.lang.Object
    extended by java.util.AbstractCollection<E>
          extended by java.util.AbstractList<E>
              extended by java.util.ArrayList<E>java.lang.Object

->java.util.AbstractCollection<E>

->java.util.AbstractList<E>

->java.util.Vector<E>
```

Elle appartient au package : java.util

Q 49.3 Il est possible de créer une documentation pour les classes que vous créez. Pour cela, il faut utiliser la commande javadoc. Récupérez sur le site web de l'UE le fichier `Clavier.java`. Placez ce fichier dans un répertoire vide, puis tapez la commande : javadoc `Clavier.java`, puis : `firefox index.html` Comparez les commentaires du fichier `Clavier.java` et la page web affichée.

L'idée c'est qu'ils comprennent que les commentaires du fichier java sont ajoutés dans la doc HTML.

Exercice 50 – Documentation Java, package

Rappel : pour qu'une classe appartienne à un package, il suffit de mettre l'instruction : `package nomdupackage;` *au début du fichier contenant la classe. Si l'on souhaite utiliser une classe d'un package dans une classe d'un autre package, il faut importer la classe :* `import nomdupackage.NomDeLaClasse;`

Q 50.1 Créez 3 classes A, B et C chacune dans un fichier différent. Déclarez ces classes public. Mettez la classe `A` dans le package pack1 et les classes B et C dans le package pack2. Ajoutez rapidement une méthode avec des commentaires à chaque classe (pour cela, il faut mettre les commentaires entre `/** ... */` avant le nom de la méthode ou de la classe). Générez une (et une seule) documentation pour ces 3 classes.

Il faut mettre en haut de chaque fichier :

```
package pack1; // pour A
package pack2; // pour B et C
javadoc A.java B.java C.java
```

Attention : seules les classes déclarées public sont dans la doc HTML

Q 50.2 Créez un objet de la classe `A` dans la classe B. Compilez les fichiers. Quelle instruction faut-il ajouter ?

Il faut importer la classe `A` dans la classe B. Au début de la classe B, on ajoute l'instruction : `import pack1.A;`
Il faut compiler les deux fichiers en même temps : `javac A.java B.java`

Exercice 51 – Visibilité et package

Rappel : En java, il existe 3 modificateurs de visibilité : `private`, `protected` et `public`. Lorsqu'il n'y a pas de modificateur, on dit que la visibilité est la visibilité par défaut.
Une classe est :

- soit `public` : elle est alors visible de partout.
- soit a la visibilité par défaut (sans modificateur) : elle n'est alors visible que dans son propre paquetage.

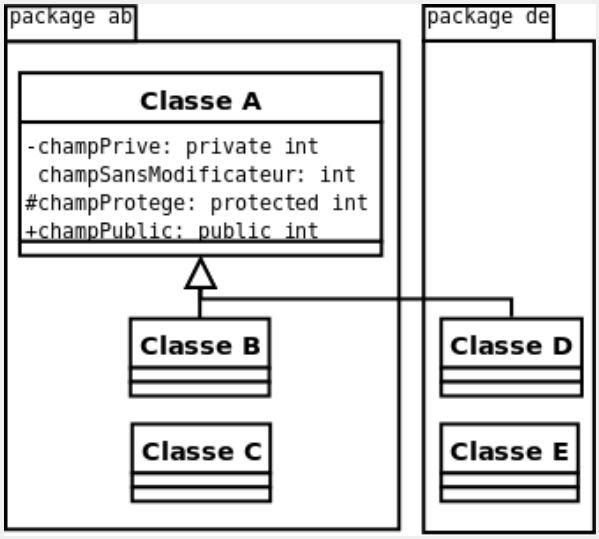
Si un champ d'une classe `A` :

- est `private`, il est accessible uniquement depuis sa propre classe ;
- est sans modificateur, il est accessible de partout dans le paquetage de `A`, mais de nulle part ailleurs ;
- est `protected`, il est accessible de partout dans le paquetage de `A` et, si `A` est publique, dans les classes héritant de `A` dans d'autres paquetages ;
- est `public`, il est accessible de partout dans le paquetage de `A` et, si `A` est publique, de partout ailleurs.

On considère les classes `A`, `B`, `C` qui sont dans le package `abc`, et les classes `D` et `E` qui sont dans le package `de`. Les classes `B` et `D` héritent de la classe `A`. On donne la classe `A` suivante :

```
1 package abc;
2 public class A {
3         private int champPrive;
4         int champSansModificateur;
5         protected int champProtected;
6         public int champPublique;
7 }
```

Q 51.1 Donner la déclaration des classes `B`, `C`, `D` et `E`, et faire un schéma.



Obligatoirement chaque classe dans un fichier différent.

```
1 package abc;
2 public class B extends A { }
3
4 package abc;
5 public class C { }
6
7 package de;
8 public class d extends A { }
9
10 package de;
11 public class E { }
```

Q 51.2 Compléter le tableau ci-dessous en cochant les cases pour lesquelles les variables d'instance de la classe `A` sont visibles.

	Classe `A`	Classe `B`	Classe `C`	Classe `D`	Classe `E`
`champPrive`					
`champSansModifieur`					
`champProtege`					
`champPublic`					

	Classe `A`	Classe `B`	Classe `C`	Classe `D`	Classe `E`
`champPrive`	X				
`champSansModifieur`	X	X	X		
`champProtege`	X	X	X	X	
`champPublic`	X	X	X	X	X

Q 51.3 Si la classe `A` n'était pas déclarée `public`, est-ce que cela change la visibilité des variables ?

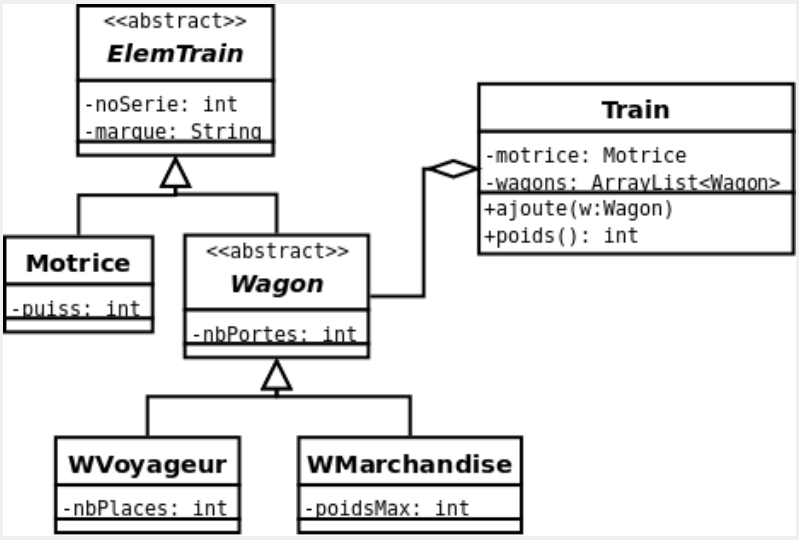
Oui, les variables ne sont visibles que dans le paquetage. C'est un cas que l'on rencontre rarement.

Exercice 52 – Compagnie de chemin de fer (ArrayList, instanceof)

Remarque : Exercice utilisant la classe **ArrayList, instanceof**

Une compagnie de chemin de fer veut gérer la formation de ses trains, à partir de la description suivante. Un train est formé d'éléments de train. Un élément de train possède un numéro de série et une marque. Un élément de train est soit une motrice, soit un wagon. Une motrice a une puissance. Un wagon a un nombre de portes. Un wagon peut être soit un wagon voyageurs, auquel cas il possède un nombre de places, soit un wagon de marchandise, auquel cas il possède un poids maximum représentant la charge maximale qu'il peut transporter.

Q 52.1 Dessiner la hiérarchie des classes `Train`, `ElemTrain`, `Motrice`, `Wagon`, `WVoyageur` et `WMarchandise`.



Q 52.2 Ecrire les classes `ElemTrain` (abstraite), `Wagon` (abstraite), `WVoyageur` et `WMarchandise` avec au moins un constructeur avec paramètres et une redéfinition de la méthode `public String toString()` qui retourne pour un élément son type et son numéro de série, par exemple : « Wagon Marchandise 10236 ».

```
1 public abstract class ElemTrain {
2     private int noSerie;
3     private String marque;
4     public ElemTrain(int no, String m) {
5         noSerie=no;
6         marque=m;
7     }
8     public String toString() { return ""+ noSerie; }
9 }
10 public class Motrice extends ElemTrain {
11     private int puiss;
12     public Motrice(int no, String m, int p) {
13         super(no, m);
14         puiss=p;
15     }
16     public String toString() {
17         return "Motrice␣"+super.toString();
```

```
18      }
19  }
20  public abstract class Wagon extends ElemTrain{
21      private int nbPortes;
22      public Wagon(int no, String m, int nbPo) {
23          super(no, m);
24          nbPortes=nbPo;
25      }
26  }
27
28  public class WVoyageur extends Wagon {
29    private int nbPlaces;
30    public WVoyageur(int no, String m, int nbPo, int nbPl) {
31        super(no,m,nbPo);
32        nbPlaces=nbPl;
33    }
34    public String toString() {
35        return "Wagon␣Voyageur␣"+super.toString();
36    }
37  }
38
39  class WMarchandise extends Wagon {
40    private int pdsMax;
41    public WMarchandise(int no, String m, int nbPo, int pds) {
42        super(no, m, nbPo);
43        pdsMax=pds;
44    }
45    public int getPdsMax() {
46        return pdsMax;
47    }
48  }
```

Q 52.3 Un Train possède une motrice et une une suite de wagons (on gèrera cette suite obligatoirement par la classe `ArrayList` (voir la documentation page 136). Ecrire la classe `Train` avec au minimum un constructeur a un paramètre de type `Motrice` qui construit un train réduit à cette motrice, et ayant donc un ensemble vide de wagons.

```
1  import java.util.ArrayList;
2  public class Train {
3    public Motrice motrice;
4    public ArrayList <Wagon> wagons;
5    public Train(Motrice m) {
6        motrice=m;
7        wagons=new ArrayList<Wagon>();
8    }
9    public void ajoute(Wagon w) {
10       wagons.add(w);
11   }
12
13   public String toString() {
14       return motrice+"␣"+wagons;
15   }
16   public int poids() {
17       Object o;
18       int p=0; // accumulateur
19       int c=wagons.size();
20       for (int i=0; i<c; i++) {
```

```
21            o=wagons.get(i);
22            if (o instanceof WMarchandise)
23                p+=((WMarchandise)o).getPdsMax();
24        }
25        return p;
26    }
27 }
```

Après avoir expliquer le fonctionnement de `instanceof`, il est utile d'insister sur la laideur du code produit :) En général, on préfère des fonctions abstraite de haut niveau pour gérer ce type de problème. Avec instanceof, le code n'est pas évolutif, si on ajoute un nouveau type de wagon, il faudra faire une modification ici aussi : c'est la caractéristique d'un *mauvais code*.

Q 52.4 Ajouter une méthode `void ajoute(Wagon w)` qui ajoute un wagon au vecteur de wagons du train.

Q 52.5 Redéfinir la méthode `public String toString()` qui retourne la composition de ce train.

Q 52.6 Ecrire une méthode `poids()` qui retourne le poids maximum de marchandise que peut transporter le train. *Indication* : On peut utiliser l'opérateur `instanceof` qui rend vrai si et seulement si un objet est instance d'une classe. Exemple d'utilisation : `if (a instanceof A)...`

Q 52.7 Ecrire la méthode principale `public static void main(String[] args)` dans une classe `MainTrain`. Cette méthode crée une motrice, des wagons de voyageur et des wagons de marchandise, crée un train formé de ces éléments, affiche la composition de ce train ainsi que le poids transporté.

```
1 class MainTrain {
2   public static void main(String[] args) {
3     Motrice m=new Motrice(5634,"MMM",2000); // no serie, marque, puiss
4     // no serie, marque, nbPortes, nb voyageurs :
5     WVoyageur wv=new WVoyageur(7845,"WWW",6,100);
6     // no serie, marque, nbPortes, poids max :
7     WMarchandise wm=new WMarchandise(9997, "WWW",2,1000);
8     WMarchandise wm2=new WMarchandise(3087, "WWW2",3,2000);
9     WMarchandise wm3=new WMarchandise(3114, "WWW3",3,1500);
10    Train t=new Train(m);
11    t.ajoute(wv);
12    t.ajoute(wm);
13    t.ajoute(wm2);
14    t.ajoute(wm3);
15    System.out.println("composition du train :");
16    System.out.println(t.toString());
17    System.out.println("poids maximum charge par le train :"+t.poids());
18  }
19 }
20
21 /* EXECUTION :
22 composition du train :
23 Motrice 5634 [Wagon Voyageur 7845, Wagon Marchandise 9997,
24 Wagon Marchandise 3087, Wagon Marchandise 3114]
25 poids maximum charge par le train : 4500
26 Press any key to continue...
27 */
```

Quizz 13 – ArrayList

Soient les classes suivantes :

```
1 import java.util.ArrayList;
2
3 public abstract class A {
4         public abstract void afficher();
5 }
6 public class B extends A {
7         public void afficher() {
8                 System.out.println("je suis un B");
9         }
10        public void methodeDeB() {
11                System.out.println("Methode de B");
12        }
13 }
14 public class C extends A {
15        public void afficher() {
16                System.out.println("je suis un C");
17        }
18        public void methodeDeC() {
19                System.out.println("Methode de C");
20        }
21 }
```

On souhaite créer une classe qui gère une liste d'objets dont la classe mère est `A`.

QZ 13.1 Expliquez la ligne 1.

> Java est fourni avec un ensemble de classes. Par exemple, les classes String, Math, System. Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles appelés packages. L'instruction `import` permet d'utiliser les classes d'un certain package. L'instruction `import java.util.ArrayList;` permet d'utiliser la classe `ArrayList` du package `java.util`.

QZ 13.2 Ecrire la classe `ListeDeA` qui possède une seule variable d'instance appelée `liste` qui est de type `ArrayList` de `A` (voir la documentation de la classe `ArrayList` à la page 136). Ajoutez-y un constructeur qui prend en paramètre le nombre `n` d'objets à créer à l'initialisation de la liste. Ce constructeur crée aléatoirement 50% d'objets de type `B` et 50% d'objets de type `C` et les ajoute à la liste.

```
1 public class ListeDeA {
2         private ArrayList<A> liste;
3         public ListeDeA (int n) {
4                 liste=new ArrayList<A>();
5                 for(int i=0;i<n;i++) {
6                         if (Math.random()<0.5) {
7                                 liste.add(new B());
8                         } else {
9                                 liste.add(new C());
10                        }
11                }
12        }
13
14
15 public class TestArrayList {
16        public static void main(String [] args) {
17                ListeDeA l=new ListeDeA(8);
18                l.afficherListe();
19                l.afficherMethode();
20        }
21 }
```

QZ 13.3 Ajoutez à la classe `ListeDeA` une méthode `afficherListe()` qui appelle la méthode `afficher()` de chacun des objets de la liste. Utilisez cette méthode dans une méthode `main`.

```
1       public void afficherListe() {
2               for(A a: liste) {
3                       a.afficher();
4               }
5       }
```

QZ 13.4 Ajoutez à la classe `ListeDeA` une méthode `afficherMethode()` qui pour chaque objet de la liste appelle la méthode `methodeB()` si cet objet est un objet de type `B`, et appelle la méthode `methodeC()` si cet objet est un objet de type `C`. Utilisez cette méthode dans une méthode `main`.

```
1       public void afficherMethode() {
2               for(int i=0;i<liste.size();i++) {
3                       A a=liste.get(i);
4                       if (a instanceof B) {
5                               B b=(B)a;
6                               b.methodeDeB();
7                       } else if (a instanceof C) {
8                               C c=(C)a;
9                               c.methodeDeC();
10                      } else {
11                              System.out.println("erreur ni B n C");
12                      }
13              }
14      }
```