Corrigé d'exercices de TD POO

Série n°2 et Série n°3 (suite) – Classes, Objets et niveaux de visibilité

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

Exercice 4 –Série n°2

Exercice 4

On considère l'entité **Intervalle** (fermé) dans l'ensemble des nombres réels. Un intervalle est défini par deux valeurs **BI** et **BS** correspondant respectivement, à la borne inférieure et la borne supérieure de l'intervalle. Donner l'implémentation en java, de la classe Intervalle comportant un constructeur avec paramètres et les méthodes :

afficher : affiche l'objet intervalle [BI, BS]

milieu : calcule le point milieu de l'intervalle

avant : vérifie si un intervalle est avant un autre sur l'ensemble des réels

disjoint : vérifie si deux intervalles sont disjoints (aucune intersection entre eux)

contenu : vérifie si un intervalle est complètement inclus dans un autre

intersection : calcule l'intervalle intersection de deux autres intervalles

fusion **:** calcule l'intervalle fusion de deux autres intervalles, si possible.

Exercice 4 –Série n°2

Exercice 4

Ecrire un programme permettant de :

-Créer un vecteur de *n* intervalles (n donné) de sorte que les intervalles soient *numérotés de manière séquentielle* (modifier la classe intervalle à cet effet).

- Détermine l'intersection de tous les intervalles et affiche le résultat.

Exercice 4 –Série n°2

```
class Intervalle

{ private float  BI, BS;
      //constructeur
public Intervalle (float bi, float bs)
{ BI= bi; BS= bs;}
                          //méthodes

public void afficher ()
  {  System.out.println ( "[" + BI + "," + BS + "]"); }

public float milieu ()
{return (BI+BS)/2; }

public boolean  avant (Intervalle I2)
{  if (this.BS < I2.BI) return true;
else return false;  }
```

/* Pour la méthode avant, on doit fixer une logique, on vérifie si l'intervalle référencé par this est avant l'intervalle I2

Il faut que la borne sup de this soit inférieure à la borne inf de I2 */

Exercice 4 –Série n°2

```
//suite des méthodes

public boolean  disjoint  (Intervalle I2)
{ if (this.avant (I2) || I2.avant (this))
            return true;
    else return false;  }

public boolean  contenu (Intervalle I2)
  { if(I2.BI >= this.BI &&  I2.BS <= this.BS)
            return true;
   else return false; }
```

/* la méthode disjoint fait appel à la méthode avant de la même classe */

On vérifie si l'intervalle I2 est contenu dans l'intervalle référencé par this
Il faut que la borne inf de I2 soit > à la borne inf de this et la borne sup de I2 soit < à la borne sup de this

Exercice 4 –Série n°2

```
//suite des méthodes

public Intervalle  intersect  (Intervalle I2)
{  float  bi, bs;
   if (this.disjoint (I2))
     return null;
     else
        { if (this.BI >= I2.BI) bi = this.BI;
            else bi = I2.BI;
         if (this.BS <= I2.BS) bs = this.BS;
            else bs = I2.BS;
          return (new Intervalle (bi, bs);) }

 }
```

/* la méthode intersect fait appel à la méthode disjoint de la même classe */

/* Pour l'intersection, on vérifie d'abord que les intervalles ne sont pas disjoints.
On prend la plus grande des bornes inf et la plus petite des bornes sup */

Exercice 4 –Série n°2

```
//suite des méthodes

public Intervalle fusion (Intervalle I2)
{  float  bi, bs;
   if (this.disjoint (I2))
     return null;
     else
        { if (this.BI <= I2.BI) bi = this.BI;
            else bi = I2.BI;
         if (this.BS >= I2.BS) bs = this.BS;
            else bs = I2.BS;
          return (new Intervalle (bi, bs);) }

 } } //fin de la classe Intervalle
```

/* la méthode fusion fait appel à la méthode disjoint de la même classe */

/* Pour la fusion, on vérifie d'abord que les intervalles ne sont pas disjoints.
On prend la plus petite des bornes inf et la plus grande des bornes sup */

Exercice 4 –Série n°2

Exercice 4

Ecrire un programme permettant de :

-Créer un vecteur de *n* intervalles (n donné) de sorte que les intervalles soient *numérotés de manière séquentielle* (modifier la classe intervalle à cet effet).

- Détermine l'intersection de tous les intervalles et affiche le résultat.

Exercice 4 –Série n°2

```
class Intervalle

{ private float  BI, BS;  private int numéro;
   private static int cpt;


      //constructeur

public Intervalle (float bi, float bs)
{ cpt ++; numéro = cpt;
   BI= bi; BS= bs;}
                        //méthodes

… }
```

/* Pour numéroter les intervalles de manière séquentielle, on utilise un compteur **cpt static** qui sera incrémenté à chaque création d'un nouvel objet Intervalle */

/* chaque objet Intervalle aura son propre numéro différent des autres*/

Exercice 4 –Série n°2

```
class ProgIntervalle

{  public static void main ( String args[])
     {Scanner  e = new Scanner (System.in);
       System.out.println ("donner  le nombre d'intervalles à créer");
       int n = e.nextInt();  float bi, bs;

   // création du vecteur de références d'objets
  Intervalle  V [ ] = new Intervalle [n];  //initialisé à null

// création des objets Intervalle
  for (int i = 0; i< n; i++)
          {System.out.println ("donner les bornes de l'intervalle");
           bi = e.nextFloat(); bs = e.nextFloat();
           V[i] = new Intervalle (bi, bs);
           }
```

Exercice 4 –Série n°2

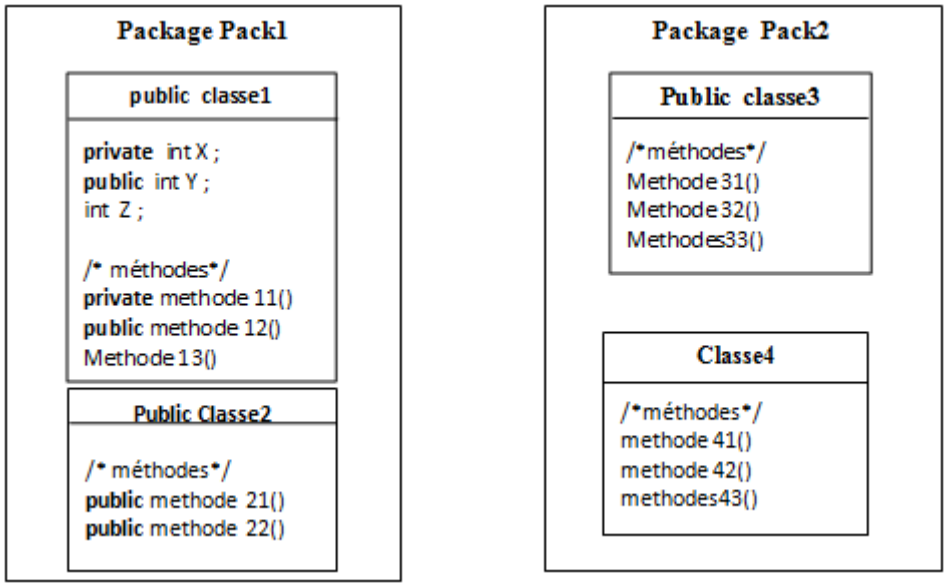
```
// intersection de tous les objets Intervalle
Intervalle R;  // une référence de type Intervalle initialisé à null

R =  V[0];   //affectation de référence

for (int i = 1; i< n && R!= null ; i++)
        { R = R.intersect (V[i]); }        // dès que l'intersection donne un
                                           // résultat null, on arrête le calcul

 if (R != null)  { System.out.println ("l'intervalle résultat est:")
                        R.afficher();}
   else System.out.println ("l'intersection est vide");
    } } // fin de la classe ProgIntervalle
```

Exercice 2 –Série n°3



1. La classe Classe4 est-elle accessible dans le paquetage Pack2 ? dans le paquetage Pack1 ? Sinon comment la rendre accessible ?

2. Dresser un tableau qui montre l'accessibilité des attributs et méthodes de Classe1 dans les autres classes du schéma.

Exercice 2 –Série n°3

La classe Classe4 est accessible dans Pack 2, elle a une visibilité package (par défaut)
Classe4 n'est pas accessible dans Pack 1 (package externe), pour la rendre accessible il faut la déclarer "**public**"

Attrribut/méthode de Classe 1	Classe 2	Classe 3	Classe 4
private int X ;	NON	NON	NON
public int Y ;	OUI	OUI	OUI
int Z ;	OUI	NON	NON
Private methode11()	NON	NON	NON
public methode 12 ()	OUI	OUI	OUI
methode 13()	OUI	NON	NON

/* les mêmes règles d'accessibilité sont appliquées aux attributs et aux méthodes*/

Rappel: l'accessibilité d'un attribut/méthode "public" à l'extérieur du package nécessite que la classe qui le contient soit déclarée "**public**"

Exercice 2 –Série n°3

3. Soit la portion de code suivante, quelles erreurs pouvez-vous recenser dedans

```
package Pack1 ;
public class Classe2
        {public void methode21()
                { Classe1 c1 = new Classe1() ;
                   System.out.printn( ″c1.X = ″+ c1.X) ;
                   System.out.printn( ″c1.Y = ″+ c1.Y) ;
                   System.out.printn( ″c1.Z = ″+ c1.Z) ;
                   c1. methode11() ;
                   c1. methode12() ;
                   c1. Methode13() ;
                   Classe3 c3 = new Classe3();
                   c3. methode11() ;
                   c3. methode31();
                   Classe4 c4 = new Classe4() ;
                   c4.methode42() ;}
}
```

Exercice 2 –Série n°3

Erreur	Commentaires et Corrections
System.out.println("c1.X = "+ c1.X) ;	X est un attribut privé non accessible dans Classe 2. Il faut écrire **un accesseur en lecture** dans Classe 1 Public int getX() {return X ;} System.out.println("c1.X = "+ **c1.getX()**) ;
c1. methode11() ;	**methode11()** est une méthode **privée** dans Classe 1, non accessible dans Classe 2. Il faut que la méthode soit « **public** » ou au moins avec une visibilité par défaut.
Classe3 c3 = new Classe3();	Il faut importer Classe 3 dans package Pack1 **Import Pack2.Classe3** ;
c3. methode11() ;	**methode11()** est une méthode **privée** dans Classe 1 methode11() ne peut pas être appliquée à **un objet de Classe 3.**
c3. methode31();	**methode31()** est une méthode qui a la visibilité par défaut dans Pack2 il faut la rendre « **public** » pour qu'elle soit accessible dans Pack1
Classe4 c4 = new Classe4() ;	**Classe 4** n'est pas accessible dans Pack1, elle a la visibilité Package dans Pack2 Il faut la rendre « **public** » et l'importer **Import Pack2.Classe4** ;
c4.methode42() ;	**methode42()** est une méthode qui a la visibilité par défaut dans Pack2 il faut la rendre « **public** » pour l'utiliser dans une classe de Pack1.

Les modificateurs de visibilité

Règles générales - Rappel

Il faut noter que

1. les attributs doivent être déclarés **“private” (ou protected – avec l’héritage)** pour respecter **l’encapsulation**

2. les méthodes doivent être déclarées **“public” (ou protected)** pour favoriser l’**utilisation/réutilisation**

3. Les classes doivent être déclarées **“public”** pour favoriser la **réutilisation**