جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Corrigé d'exercices de TD POO

Série n°4 – Héritage et Polymorphisme

S. BOUKHEDOUMA

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

Exercice 4 –Série n°4

Exercice 4

On considère les classes **Point** et **PointEspace** définies comme suit :

Point : caractérisée par deux attributs x et y de type double, des constructeurs et les méthodes : String *toString()*,
void *deplacer(double d)*, on déplace des deux coordonnées
void *deplacer (Point p)* (On inclut dans la méthode, le message « déplacer de Point »)
Point *symetrie()*.(On inclut dans la méthode, le message « Symétrie de Point »)

PointEspace : dérivée de la classe Point, à laquelle on ajoute un attribut *z*, des constructeurs et les méthodes :

String *toString()*,
void *deplacer (double d)*, on déplace les trois coordonnées
void *deplacer (PointEspace p)* (On inclut dans la méthode, le message « déplacer de PointEspace »)
PointEspace *symetrie()*. (On inclut dans la méthode, le message « Symétrie de PointEspace »)

Exercice 4 –Série n°4

Exercice 4 (suite)

Questions :

1. Implémenter les classes **Point** et **PointEspace**
2. Déterminer les méthodes surchargées et les méthodes redéfinies.
3. Les instructions suivantes sont-elles correctes :

```
PointEspace P5 = new Point (10.0, 15.0) ;
Point p1 = new Point(1,4) ;PointEspace p5 = (PointEspace) p1;
PointEspace p2 = new PointEspace (3,6, -1) ;
((PointEspace) p1). deplacer (p2) ;
```

Exercice 4 –Série n°4

```
public class Point
{private double x;
  private double y;

//constructeurs
 public Point (double x, double y)
   { this.x = x; thix.y = y;}
public Point ()  {}

  // méthodes
public String toString()
 {return (”(”+ x + ”,”+y + ”)”); }

public void déplacer (double d)
 { x = x+d; y = y+d;}
```

Exercice 4 –Série n°4

```
//suite de la classe Point

public void déplacer (Point p)
 {   System.out.println("Déplacer de Point");
     x = p.x; y = p.y;}

Point  symétrie ()
{ System.out.println(" Symétrie de Point");
   return (new Point (-x, -y));}

        //accesseurs

public double getX() {return x;}          public double getY() {return y;}
public void setX(double d) {x=d;}          public void setY(double d) {y=d;}

} // fin de la classe Point
```

Exercice 4 –Série n°4

```
public class PointEspace  extends  Point
{
   private double z;    //attribut supplémentaire

                                        //constructeurs
 public PointEspace (double x, double y, double z)
  { super(x, y);
     this.z = z;}
public PointEspace()  {super(); }

                                        // méthodes
public String toString()
 {return ("("+ getX() + ","+ getY() +","+ z +")"); }      //ou super.getX()

public void déplacer (double d)
 { super.déplacer(d); z = z+d;}
```

Exercice 4 –Série n°4

```
//suite de la classe PointEspace

public void déplacer (PointEspace p)
 {System.out.println(" Déplacer de PointEspace");
   super.setX(p.x); super.setY(p.y); z = p.z;}

PointEspace  symétrie ()
{System.out.println("Symétrie de PointEspace");
   return (new PointEspace (- getX(), - getY(), -z));}

      //accesseurs

public double getZ() {return z;}
public void setZ(double d) {z=d;}

} // fin de la classe PointEspace
```

Exercice 4 –Série n°4

Remarque

Dans la sous-classe **PointEspace**, on aurait pu accéder directement aux attributs ***x*** et ***y*** de la superclasse, si on les avait déclarés **« protected ».**

Mais comme ils sont **« private »,** il faut y accéder via les **accesseurs** (get et set)

Exercice 4 –Série n°4

Méthodes surchargées

Dans la classe Point:

Le constructeur

La méthode déplacer: déplacer (double d) et déplacer (Point p)

Dans la classe PointEspace:

Le constructeur

La méthode déplacer : déplacer (double d) et déplacer (PointEspace p) et déplacer (Point p)

Méthodes redéfinies

Dans la classe Point:

toString ()

Dans la classe PointEspace:

toString() **déplacer (double d)** **PointEspace Symétrie ()**

Exercice 4 –Série n°4

Exercice 4 (suite)

Questions :

1. Implémenter les classes **Point** et **PointEspace**
2. Déterminer les méthodes surchargées et les méthodes redéfinies.

3. Les instructions suivantes sont-elles correctes :

```
PointEspace P5 = new Point (10.0, 15.0) ;
Point p1 = new Point(1,4) ;
PointEspace p5 = (PointEspace) p1;
PointEspace p2 = new PointEspace (3,6, -1) ;
((PointEspace) p1). deplacer (p2) ;
Point p3 = p2;
((PointEspace) p3). deplacer (2.0) ;
```

Exercice 4 –Série n°4

PointEspace P5 = new Point (10.0, 15.0) ;

Erreur: une référence de type sous-classe ne peut pas être utilisé pour créer un objet de type superclasse

Point p1 = new Point(1,4) ;

Correcte

PointEspace p5 = (PointEspace) p1;

Erreur: un objet de type superclasse (Point) ne peut pas être converti (cast) en un objet de type sous-classe car p1 est créé de type Point.

Exercice 4 –Série n°4

PointEspace p2 = new PointEspace (3,6, -1) ;

Correcte

((PointEspace) p1). deplacer (p2) ;

Erreur: un objet de type superclasse (Point) ne peut pas être converti (cast) en un objet de type sous-classe car l'objet p1 est créé de type Point.

Point p3 = p2;

Correcte: une référence de type superclasse peut référencer un objet de type sous-classe

((PointEspace) p3). deplacer (2.0) ;

Correcte: la méthode déplacer(d) de la sous-classe va s'appliquer sur l'objet p2 (référencé aussi par p3) car on a fait une affectation de références plus haut.

Exercice 4 –Série n°4

Soit le code suivant :

Point **p1** = new Point(1,4) ; Point **p2** = new Point(1,4) ;

PoinEspace **p3** = new PointEspace (5,4,3) ;

Point **p4** = new PointEspace (1,1,1) ;

Question: *remplir le tableau*

-Un objet de type **Point** a accès à la méthode **déplacer(Point) de la superclasse**

- Un objet de type **PointEspace** a accès à la méthode **déplacer(Point)** de la superclasse et la méthode **déplacer(PointEspace)** de la sous-classe

Exercice 4 –Série n°4

Instruction	Valide ?	Méthode redéfinie	Liaison dynamique	Polymorphisme appliqué	Classe de la méthode	Commentaire
p1.deplacer(p2)	Oui	Non	Non	Non	Point	P1 est de type Point
p1.deplacer(p3)	Oui	Non	Non	Oui (sur p3, cast implicite)	Point	P1 est de type Point
p3.deplacer(p1)	Oui	Non	Oui (dépend du paramètre)	Non	Point	Le paramètre est de type Point
p3.deplacer(p4)	Oui	Non	Oui (dépend du paramètre)	Oui (sur p4)	Point	Le paramètre est de type Point
p4.deplacer(2)	Oui	Oui	Oui (dépend de l'objet)	Oui (sur p4)	PointEspace	Le type réel de P4 est un PointEspace
P4.deplacer(p1)	Oui	Non	Oui (dépend du paramètre)	Oui (sur p4)	Point	Le paramètre est de type Point
P4.deplacer(p3)	Oui	Non	Oui (dépend du paramètre)	Oui (sur p4)	PointEspace	Le paramètre est de type PointEspace
P1.symétrie()	Oui	Non (dans la classe Point)	Non	Non	Point	P1 est de type Point
P3.symétrie()	Oui	Oui	Oui (dépend de l'objet)	Non	PointEspace	P3 est de type PointEspace
P4.symétrie	Oui	Oui	Oui (dépend de l'objet)	Oui (sur p4)	PointEspace	Le type réel de P4 est PointEspace