Université de Mila
Institut des sciences et technologie
2ème année Licence Informatique
Matière : Programmation Orientée Objet
Responsable de la matière : Dr. Sadek BENHAMMADA

Corrigé du TD N°4

Exercice 1

```
//1,2,3,4
public class PointNom extends Point
{
    private char nom;
    public PointNom(char nom,double x, double y)
    {
       super(x,y);
       this.nom=nom;
    }
    public void deplacer(double dx, double dy){
        x=x+dx;
        y=y+dy;
    }
    public String toString() {return this.nom+super.toString();}

}
```

4. Pour implementer la méthode ***deplacer()***, on a besoin d'accéder aux attributs ***x*** et ***y*** de la classe mère ***Point***, donc, il est nécessaire de changer la visibilité des attributs ***x*** et ***y*** de ***private*** à ***protected*** pour qu'ils soit accessible dans les classes dérivées de de classe ***Point*** :

```
class Point {
    protected double x;
    protected double y;
    ...
}
```

Exercice 2

1.

```
public abstract class Forme {
    public abstract double getSurface();
    public boolean plusEtendueQue(Forme f){
     return this.getSurface()>f.getSurface();
}
```

- La classe Forme possède une méthode abstraite (**`abstract double getSurface()`**), donc, elle doit être déclarée classe abstraite.

```
public abstract class FormeNom extends Forme{
    private char nom;
    public FormeNom(char nom){this.nom=nom;}
}
```

- Dans `FormeNom`, Il n'est pas possible d'implémenter la méthode abstraite `getSurface()` de la classe mère `Forme` (chaque forme géométrique à sa propre méthode pour calculer sa surface), donc, elle doit être déclarée classe abstraite

```
public class Rectangle extends Forme {
    private double longueur;
    private double largeur;

    public Rectangle(double longueur, double largeur){
       this.longueur=longueur;
       this.largeur=largeur;
    }
   public double getSurface() { return longueur*largeur;}
}
```

/*La classe ***Rectangle*** est classe fille de la classe abstraite ***Forme***, ainsi, elle doit implémenter la méthode ***getSurface()*** ou se déclarer abstraite. Il est possible de d'écrire une méthode pour calculer la surface d'un rectangle, donc, on a opté pour l'implémentation de ***getSurface()*** */

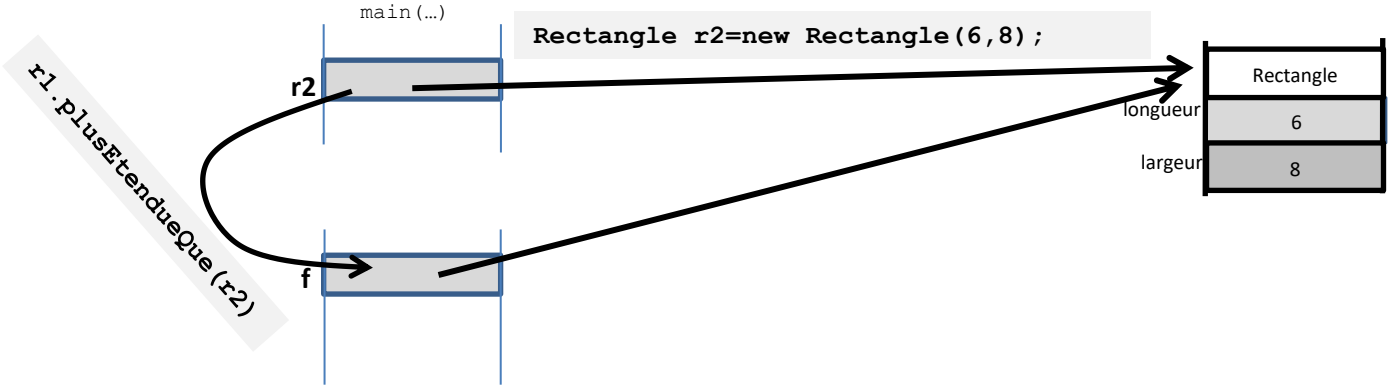
-

```
public class TestForme {
   public static void main(String[] args)
    {
        Forme f=new Forme();
        FormeNom fn=new FormeNom('A');
        Rectangle r1=new Rectangle(5,10);
        Rectangle r2=new Rectangle(6,8);
        System.out.println(r1.plusEtendueQue(r2));
    }
}
```

/*Les classes ***Forme*** et ***FormeNom*** sont déclaré abstraites, donc il n'est pas possible de les instancier. */

2.

- La constructeur de la classe ***Rectangle*** est correcte : La classe ***Forme*** ne déclare pas un constructeur, donc, il n'est pas nécessaire d'appeler le constructeur par défaut avec ***super()***, cependant, le faire n'est pas une erreur.
- La méthode ***plusEtendueQue(Forme f)*** peut être appelée avec comme paramètre effectif, un objet d'une classe dérivée (fille) de la classe ***Forme***. C'est ce qu'on appelle le transtypage implicite : Utilisation d'une référence d'une super-classe (mère) pour désigner un objet d'une classe dérivée. ***r2*** est un objet de ***Rectangle***, classe fille de ***Forme***, donc, ***r1.plusEtendueQue(r2)*** est correct.



Exercice 3

1.

Résultat affiché :

Je suis une Personne Je suis une Personne Je suis un Etudiant Je suis un Etudiant

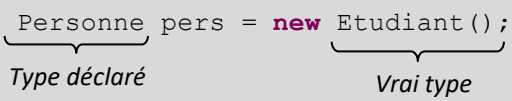
Explication :

Lors de l'appel de la méthode ***afficherClasse(),*** on recherche tout d'abord la méthode dans la classe de l'objet appelant. Si aucune méthode n'est trouvée, on poursuit la recherche dans la classe supérieure et ainsi de suite jusqu'à ce que la méthode soit trouvée, sinon, on obtient une erreur de compilation.

2.

Instruction	Correcte ?	Explication	Correction
Personne p1=new Etudiant("Math")	**Oui**	Transtypage implicite : utilisation de la référence d'une super-classe pour désigner un objet d'une classe dérivée	/
p1.afficherClasse ()	**Oui**	Le type déclaré de ***p1*** est Personne, la méthode ***afficherClasse ()*** est déclarée dans Personne.	/
p1.afficherFiliere ()	**Non**	Le type déclaré de ***p1*** est Personne, la méthode ***afficherFiliere ()*** n'est pas déclarée dans Personne.	Transtypage explicite : ***((Etudiant)p1).afficherFiliere();***
Etudiant t1=new Personne()	**Non**	Il n'est pas possible d'utiliser une référence d'une classe dérivée pour un objet d'une s'per-classe	à supprimer
t=p	**Non**	Il n'est pas possible d'utiliser une référence d'une classe dérivée pour un objet d'une super-classe. Le transtypage explicite ***t=(Etudiant)p***; est correcte lors de la compilation, cependant, une exception est soulevé lors de l'exécution, car le vrai type de *p1* est `Personne`.	à supprimer
t=p1	**Non**	**p1** référence un objet de type **Etudiant**, mais son type déclaré est **Personne**, l'instruction soulève une erreur lors de la compilation (le transtypage implicite sens mère→ fille est interdit.	***t=(Etudiant)p1 ;*** Correct lors de la compilation (transtypage explicite sens mère→fille) Correct lors de l'exécution : Le vrai type de ***p1*** est **Etudiant**, donc ***t*** référence un objet de **Etudiant.**
p=t	**Oui**	Transtypage implicite sens fille→mère	/
Object o=p	**Oui**	***Object*** est la classe super-classe de toutes les classes, donc, il est possible d'utiliser une référence de ***Object*** pour un objet de la classe ***Personne***.	/

Remarques :



Un objet a deux types :
1. Le type déclaré : vérifié lors de compilation par le compilateur javac ;
2. Le vrai type vérifié est au moment de l'exécution par la JVM.

Le transtypage implicite(fille→mère) est toujours possible

Le transtypage explicite (mère→fille) est possible lors de la compilation, et interdit lors de l'exécution.

3. Le programme affichera :

```
Je suis un Etudiant, ma filière est : Math
```

Le vrai type de ***p1*** est ***Etudiant***, donc, lors de l'exécution de ***p1***. ***afficherClasse(),*** c'est la méthode ***afficherClasse ()*** de la classe ***Etudiant*** qui sera appelé.

Exercice 04

```
public interface Evaluable {
      double moyenne();
      int creditAcquis();
}
```

```
public abstract class Matiere implements Evaluable
{
       protected String intitule;
       protected int credit;
       protected int coeficient;
       protected double noteExamen;


       public Matiere(String in,int credit,int coeficient, double noteExamen){
             this.intitule=in;
             this.credit=credit;
             this.coeficient=coeficient;
             this.noteExamen=noteExamen;
       }
       public Matiere(String in,int credit,int coeficient){
             this.intitule=in;
             this.credit=credit;
             this.coeficient=coeficient;

       }
       //Accessseurs de lecture
       public String getIntitule(){ return this.intitule;}
       public double getnoteExamen(){ return this.noteExamen;  }
       public int getCoeficient(){ return this.coeficient;  }
       public int getCredit()  { return this.credit;  }


        //Accessseurs de modification
       public void setnoteExamen(double n) {this.noteExamen=n;   }
       public void setIntitule(String in) {   this.intitule=in;  }
       public void setCoeficient(int coeficient) {this.coeficient=coeficient;}
       public void setCredit(int credit)  {   this.credit=credit;}

       public int creditAcquis(){
            if (moyenne()>=10)
                  return credit;
            else
                  return 0;
       }

       public String toString(){
       return "\n Matiere: " +this.intitule +
                  "\n Credit :" + this.credit +
                  "\n Coeficient :" + this.coeficient;
       }

}
```

```java
public class MatiereCours extends Matiere{
	public MatiereCours(String in, int credit, int coeficient, double noteExamen) {
		super(in, credit, coeficient,noteExamen);
	}

	public MatiereCours(String in, int credit, int coeficient) {
		super(in, credit, coeficient);
	}

	public  double moyenne() {return noteExamen;}

	public String toString(){
	  return super.toString()+ "\n Note Examen: " + this.noteExamen+ "\n Moyenne: "
			+ this.moyenne()+ "\n Credit acquis: " + this.creditAcquis();
     }
}
```

```java
public class MatiereCoursTD extends Matiere{
	private double noteTD;
	static final double coefTD=0.33;

	public MatiereCoursTD(String in, int credit, int coeficient, double noteExamen,
double noteTD) {
		super(in, credit, coeficient,noteExamen);
		this.noteTD=noteTD;
	}

	public MatiereCoursTD(String in, int credit, int coeficient) {
		super(in, credit, coeficient);
	}

	public double getNoteTD(){ return  noteTD;}
	public void setNoteTD(double noteTD){ this.noteTD=noteTD;}

	public double moyenne() {
		return this.noteExamen*(1-coefTD)+this.noteTD*coefTD;
	}

	public String toString(){
	  return super.toString()+ "\n Note Examen: " + this.noteExamen+ "\n Note TD: "
+ this.noteTD+ "\n Moyenne: " + this.moyenne()+ "\n Credit acquis: " +
this.creditAcquis();
    }
}
```

```
public class MatiereCoursTDTP extends Matiere{
	private double noteTD;
	private double noteTP;
	static final double coefTD=0.2;
	static final double coefTP=0.2;
	public MatiereCoursTDTP(String in, int credit, int coeficient, double noteExamen, double
noteTD, double noteTP) {
		super(in, credit, coeficient,noteExamen);
		this.noteTD=noteTD;
		this.noteTP=noteTP;
	}
	public MatiereCoursTDTP(String in, int credit, int coeficient) {
		super(in, credit, coeficient);
	}

	public double getNoteTD(){ return	noteTD;}
	public void setNoteTD(double noteTD){ this.noteTD=noteTD;}
	public double getNoteTP(){ return	noteTP;}
	public void setNoteTP(double noteTP){ this.noteTD=noteTP;}
	public double moyenne() {
			return noteExamen*(1-coefTD-coefTP)+this.noteTD*coefTD+this.noteTP*coefTD;
	}
	public String toString(){
	  return super.toString()+ "\n Note Examen: " + this.noteExamen+ "\n Note TD: " +
this.noteTD+ "\n Note TP: " + this.noteTP+ "\n Moyenne: " + this.moyenne()+ "\n Credit acquis:
" + this.creditAcquis();
    }
}
```

```
import java.util.ArrayList;
public class MainProgram {
	public static void main(String [] args){
	Matiere m;
	ArrayList<Matiere> listematieres=new ArrayList<Matiere>();

	m=new MatiereCoursTD("Architecture des ordinateurs",5,2,10,10);
	listematieres.add(m);
	m=new MatiereCoursTDTP("Algorithmique et structures de données",6,3,10,10,10);
	listematieres.add(m);
	m=new MatiereCoursTD("Logique mathématique",4,2,10,10);
	listematieres.add(m);
	//....
	for (int i=0;<=listematieres.size();i++)
		System.out.println (listematieres.get(i).toString)

	}
}
```