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

Série de TD N°4

Exercice 1 (Héritage, redéfinition, droits d'accès, mots clés super)

On dispose du code suivant :

```
class Point {
    private double x;
    private double y;
    public Point (int x, int y) { this.x = x ; this.y = y ; }
    public String toString() {return "("+this.x + "," + this.y+")";}
    public static void main(String[] args)}
```

1. Déclarez une classe `PointNom`, une sous-classe de `Point` permettant de manipuler des points définis par leurs coordonnées et un `nom` (de type `char`).
2. Déclarez un constructeur qui initialise les attributs d'un objet créé de la classe `PointNom`.
3. On veut définir une méthode `deplacer(double dx, double dy)` dans la classe `PoinNom`. Faites les modifications nécessaires pour la déclaration de cette méthode, puis, déclarez la méthode.
4. Redéfinissez la méthode `toString()`dans la classe `PointNom` pour qu'elle affiche une chaine de caractères de la forme : `nom(x,y)`. Exemple : `A(5,10).`(La méthode `toString()` redéfinie doit se servir de la méthode `toString()` de classe mère `Point`).

Exercice 2 (Abstraction, transtypage)

Soit le code suivant :

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

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

```
public class Rectangle extends Forme {
    private double longueur;
    private double largeur;
    public Rectangle(double longueur, double largeur){
    this.longueur=longueur;
    this.largeur=largeur;
}
}
```

```
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));
    }
}
```

Série de TD N°4

1. Identifiez et corrigez les erreurs du code précédent.
2. Les instructions soulignées sont correctes. Expliquez pourquoi.

Exercice 3 (Redéfinition et recherche de méthodes, Polymorphisme)

Soit le code suivant :

```
class Personne{
public void afficherClasse(){ System.out.print ("Je suis une Personne") ;}
}
```

```
class Etudiant extends Personne{
 private String filiere;
 public Etudiant(String filiere) {this.filiere=filiere;}
 public void afficherFiliere(){ System.out.print (", ma filiere est: "+this.filiere) ;}
 public void afficherClasse(){ System.out.print ("Je suis un Etudiant") ;}
}
```

```
class Employe extends Personne {}
```

```
class Delegue extends Etudiant {}
```

```
public class TestRedefinition{
 public static void main (String arg[]){
 Personne p = new Personne() ; p.afficherClasse() ; System.out.println();
 Employe e = new Employe() ; e.afficherClasse() ; System.out.println();
 Etudiant t = new Etudiant("informatique") ; t.afficherClasse() ; System.out.println();
 Delegue d = new Delegue("Math") ; d.afficherClasse() ; System.out.println();}
}
```

```
public class TestPolymorphisme{
  public static void main (String arg[]){
    Personne p = new Personne() ;
    Etudiant t = new Etudiant("Informatique") ;
    Personne p1=new Etudiant("Math");p1. afficherClasse (); p1. afficherFiliere ();
    Etudiant t1=new Personne();
    t=p;
    t=p1;
    p=t;
    Object o=p; }
}
```

1. Qu'affichera l'exécution de la classe **`TestRedefinition`**? Justifiez votre réponse.
2. Dans la classe **`TestPolymorphisme`**, les instructions soulignées sont-t-elles correctes ? Justifiez votre réponse et proposez des corrections si c'est possible.
3. Qu'affichera la classe **`TestPolymorphisme`** après correction/suppression des instructions erronées ? Justifiez votre réponse.

Série de TD N°4

Exercice 4 (TP)

- Une matière est caractérisée par :
  - L'intitule (chaine de caractères);
  - Un crédit (entier) ;
  - Un coefficient (entier);
  - une note de l'examen (réel).

- La classe Matière implémente une interface Evaluable. L'interface `Evaluable` déclare 2 méthodes abstraite `moyenne()` et `creditAcquis()`.

- La classe matière a trois classes filles : MatiereCours (Matiere avec cours uniquement), MatiereCoursTD (Matiere avec cours et TD), et MatiereCoursTDTP (Matiere avec cours, TD, et TP.
- La moyenne d'une matière avec cours uniquement est égal à la note de l'examen.
- Une matière avec cours et TD se caractérise par une note de TD (réel), et une constant coefTD=0.33.
- La moyenne d'une matière avec cour et TD= note de cours*(1-coefTD)+ note de TD*coefTD.
- Une matière avec cours, TD, et TP se caractérise par une note de TD et une note TP (réels), une constant coefTD=0.2, et une constante coefTP=0.2.
- La moyenne d'une matière avec cour TD et TP= note de cours*(1-coefTD-coefTP)+ note de TD*coefTD+ note de TP*coefTP.
- Le crédit acquis d'une matière égal à son crédit si la moyenne est >=10, et à 0 sinon.

Travail demandé :

1. Implémenter les classes et les interfaces nécessaires.
2. Pour chaque classe, déclarer un constructeur avec paramètres, et les accesseurs de lecture et de modification pour chaque attribut.
3. Implémenter une méthode `toString()` dans la classe mère Matière, et dans chaqu'une des classes filles. La méthode `toString()` doit renvoyer une chaine de caractères de la forme :

```
-------------------------------------
Matiere: Programmation orientee objet
 Credit :5
 Coeficient :3
 Note de l'examen : 10
 Note TD : 10
 Note TP : 10
 Moyenne: 10
 Credit acquis: 5
```

4. Ecrire une classe `MainProgram`, qui ne possède pas d'attributs et possède une unique méthode main(), dans lequel :

- Déclarer une liste d'objet de la classe matière (de type ArrayList),
- Remplissez la liste avec les matières du semestre 3 de la deuxième année informatique Licence.
- Affichez les éléments de la liste (appels des méthodes toString()).

Série de TD N°4

Exercice 5 (TD) (Héritage, Enumérations, Interface, ArrayList)

On s'intéresse à la réalisation d'un programme pour la gestion des emprunts au sein d'une bibliothèque. Pour ce faire, nous allons déclarer 3 classes (Document, Livre, et Adhérant), et une interface (Pretable).

Un document qui correspond à un document d'une bibliothèque est caractérisé par un titre, un nom d'auteur, une année d'édition et un code de type entier (affecté de façon auto-incrementale). La classe Document ne pourra pas être instanciée.

Un livre est un document caractérisé par un domaine (informatique, mathématique, physique, ou chimie). Un livre peut être disponible ou emprunté par un adhérent de la bibliothèque.

Un adhérant se caractérise par un numéro (affecté de façon auto-incrementale), un nom, un prénom, et une liste de livres empruntés.

La classe Livre implémente l'interface Pretable. L'interface Pretable déclare deux méthodes statique :

```
String preter(Adherant ad)
void restituer()
```

1. Déclarer les 3 classes. Pour chaque classe, déclarer un constructeur avec paramètres, et une méthode toString(). Pour chaque attribut, déclarer un accesseur de lecture et un accesseur de modification.
2. Déclarer une classe Main comportant une unique méthode `main((String[] args)` qui permettra à l'utilisateur de :
   1. Créer la liste des livres ;
   2. Créer la liste des adhérant ;
   3. Afficher la liste des livres et celles des adhérents ;
   4. Vérifier la disponibilité d'un livre ;
   5. Emprunter un livre ;
   6. Restituer un livre.