Licence 3, M&I — mardi 12 février 2019.
*Programmation par objets*

# Contrôle 1

***Durée 1h30**. Une feuille de notes manuscrite recto-verso autorisée. Tout autre document, ainsi que les calculatrices et téléphones portables ne sont pas autorisés.*

## Exercice 1 : QCM

| | Vrai | Faux |
|---|---|---|
| La syntaxe `class A extends B, C {...}` est correcte | ☐ | ☐ |
| La syntaxe `class A implements B, C {...}` est correcte | ☐ | ☐ |
| La syntaxe `interface A implements B {...}` est correcte | ☐ | ☐ |
| Le constructeur par défaut est `private` | ☐ | ☐ |
| Un attribut de classe peut être laissé non initialisé à l'issue de la construction | ☐ | ☐ |
| Par défaut, une variable locale de type `int` d'une méthode `static` est initialisée à `0` | ☐ | ☐ |
| Une méthode `static` peut seulement modifier des attributs `static` | ☐ | ☐ |
| L'affectation `a=b` a le même effet si `a` et `b` sont de type `int` ou de type `int[]` | ☐ | ☐ |
| `byte a = 257;` est correct | ☐ | ☐ |
| `class A {private int x; int f(A b){return b.x;}}` est correct | ☐ | ☐ |
| `class A {... A f(){return this;}}` est correct | ☐ | ☐ |
| `abstract class A{void f(){...}} ... A a=new A();a.f();` est correct | ☐ | ☐ |
| Le code suivant affiche `true` (Vrai) ou `false` (Faux) : `Point a=new Point(1,2);Point b=new Point(a); System.out.println(a==b);` | ☐ | ☐ |
| `Object p = new Point(1,2);` est correct | ☐ | ☐ |
| Une classe `abstract` ne peut définir aucune méthode | ☐ | ☐ |
| Une classe `abstract` ne peut pas définir de constructeur | ☐ | ☐ |
| Une classe héritant d'une classe `abstract` est elle-même `abstract` | ☐ | ☐ |
| Une classe héritant d'une classe `abstract` doit être déclarée `final` | ☐ | ☐ |
| Une classe peut hériter de plusieurs classes, à condition qu'elles soient toutes `abstract` | ☐ | ☐ |
| Une classe `abstract` ne peut pas implémenter d'interface | ☐ | ☐ |

## Exercice 2 : Questions de cours

Toutes les réponses aux questions suivantes doivent être justifiées.

1. Soit le code suivant :

```
class A {...}
class B
{
        ...
        public B(B b)
        {
                this.a = b.a;
        }
        ...
        private A a;
}
```

Quel type de copie est effectué par le constructeur de `B` ? À supposer que `A` dispose d'un constructeur `public A(A a)`, réalisant une copie profonde, quelle alternative pouvez-vous proposer pour ce constructeur de `B` ?

2. Soit le code suivant :

```
class A
{
        public void f(double a, double b){...}
}
```

Pourquoi peut-on ajouter une méthode `private void f(int a, int b){...}` à la classe `A`, mais pas une méthode `public double f(double a, double b)` ?

3. Le code suivant est-il correct ? Même question si le mot-clef `protected` est changé en `private` ?

```
class A
{
        protected A(){n=1;}
        public A(int n){this.n=n;}
        private int n;
}

class B extends A
{
        public B() { super(); }
        public B(int p)
        {
                super(p);
                this.p = p;
        }
        private int p=0;
}
...
B b=new B();
```

4. Expliquez pourquoi le code suivant n'est pas correct, et proposez une modification le corrigeant.

```
class A
{
        public void f(int a){...}
}
class B extends A
{
        public void f(double a){...}
}
...
A b=new B();
b.f(0.7);
```

5. Dans le code suivant, quelles sont les attributs et méthodes disponibles dans la classe `B` ? Pour chacune, précisez ce qui est calculé.

```
class A
{
        public int add(int b){return a+b;}
        public int a;
```

```
}

class B extends A
{
        public int add(int a){return a+b;}
        public int add(){return a+b;}
        public int add(int a, int b){return a+b;}
        public int b;
}
```

6. Expliquez pourquoi le code suivant n'est pas correct :

```
final class A
{
        public int a;
}

class B extends A
{
        public int b;
}
```

7. Expliquez pourquoi le code suivant n'est pas correct :

```
class A
{
        public A(int a){this.a = a;}
        private int a;
        private final int b;
}
```

## Exercice 3 : polymorphisme

On suppose une application manipulant des nombres qui peuvent alternativement être représentés par des `double` ou par des `long`. En particulier, on peut avoir besoin de définir des tableaux de données hétérogènes dont les éléments peuvent utiliser n'importe laquelle de ces représentations, sans effectuer de conversion.

Pour point de départ, on dispose des classes suivantes :

```
class Ent
{
        public Ent(long val){this.val = val;}
        public long retVal(){return val;}
        public void afficheVal(){System.out.print(val);}
        private long val;
}
class Flo
{
        public Flo(double val){this.val = val;}
        public double retVal(){return val;}
        public void afficheVal(){System.out.print(val);}
        private double val;
}
```

**Q1 :** Proposez une solution permettant d'utiliser des tableaux hétérogènes d'`Ent` et de `Flo`, et fournissez une implémentation.

**Q2 :** Spécifiez et implémentez une méthode prenant en argument un tableau de données hérérogènes et qui en affiche le contenu. Quel possibilité, `static` ou non, vous paraît la plus pertinente ? À quelle classe pensez vous rattacher cette méthode ?

On souhaite maintenant être capable de calculer la somme coordonnée par coordonnée de deux tableaux hétérogènes, sans faire aucune conversion de type. C'est à dire qu'étant donnés trois tableaux `A`,`B`,`C` remplis de la façon suivante : `A = {Ent, Flo}`, `B = {Ent, Flo}`, `C = {Flo, Flo}`, la somme de `A` et `B` est bien définie et doit retourner un tableau rempli comme `{Ent, Flo}`, tandis que la somme de `A` et `C` n'est pas définie.

**Q3 :** Proposez une modification de vos classes permettant de résoudre ce problème, et fournissez une illustration de leur utilisation en écrivant une méthode d'addition de tableaux hétérogènes. Vous pouvez supposer dans un premier temps que l'addition sera toujours bien définie.

**Q4 :** Écrivez une méthode `main` qui définit deux tableaux hétérogènes de votre choix dont la somme est bien définie ; calcule leur somme ; affiche le résultat.

**Q5 (*bonus*) :** Expliquez comment modifier votre méthode d'addition pour prendre en compte des arguments pour lesquels le résultat n'est pas défini.

## Exercice 4 : polynômes

On dispose des interfaces suivantes, pour définir des opérations sur des groupes et anneaux dont les éléments peuvent être représentés par des entiers de type `int` :

```
interface Group
{
        public abstract int add(int a, int b);
        public abstract int neg(int a);
        public abstract int sub(int a, int b);
        public abstract boolean is_g_neutral_element(int a);
}

interface Ring extends Group
{
        public abstract int mult(int a, int b);
        public abstract int div(int a, int b);
        public abstract int inv(int a);
        public abstract boolean is_invertible(int a);
        public abstract boolean is_r_neutral_element(int a);
}
```

**Q1 :** Définissez une classe `Polynome` permettant d'effectuer

On souhaite représenter des objets géométriques du plan par une structuration orientée objet. On utilisera, la classe `Point` vue en cours et rappelée ci-dessous :

```
class Point
{ Point (double x, double y) {this.x=x; this.y=y;}
 double getX(){return x;}
 double getY(){return y;}
 double setX(double y){this.x=x;}
 double setY(double y){this.y=y;}
 private double x;
```

```
  private double y;
}
```

On souhaite représenter les formes suivantes :

- une classe `FormePlane` qui ne peut être instanciée mais de laquelle sera dérivée toute les autres classes (directement ou indirectement).
- une classe `Carré` qui ne peut être dérivée
- des classes `Parallelogramme`, `Rectangle`, `Cercle`, `Triangle`.

Toute forme possède les méthodes suivantes :

`void translate(double x, double y)` qui déplace la forme selon une translation de coordonnées $(x, y)$

`void affiche()` qui décrit l'objet, en précisant sa nature, ainsi que les informations sur ses attributs (coordonnées des points, valeur du rayon, ...)

`double aire()` qui retourne la valeur de l'aire de la forme considérée

1. Proposer une hiérarchie de classes pour représenter ces formes
2. Écrire le code Java correspondant
3. On souhaite tester l'égalité de deux formes. Celle-ci peut être au sens faible, le fait que deux formes sont superposables (et donc non nécessairement situées au même endroit), ou au sens fort que ces deux formes soit exactement égales, c'est à dire qu'elles appartiennent **à la même classe** et possèdent les mêmes caractéristiques. Ajouter les méthodes `estSuperposable` et `estEgal` correspondantes à votre code.