TD n°1 - Correction

Premières classes

Exercice 1 Qu'affiche le programme suivant ?

```
public class Ex1 {
    int a = 0 ;
    public void f() {
        a++ ;
    }
    public void affiche() {
        System.out.println(a);
    }
    public static void main(String [] args) {
        Ex1 p = new Ex1();
        Ex1 q = new Ex1();
        p.affiche();
        q.affiche();
        p.f();
        p.affiche();
        q.affiche();
        p = q;
        p.f();
        p.affiche();
        q.affiche();
    }
}
```

Correction :

```
0
0
1
0
1
1
```

On rappelle que `p` et `q` sont des references sur des objets instances de la classe Ex1.

Exercice 2 Qu'affiche le programme suivant ?

```
public class Ex2 {
    int a = 0 ;
    public static void f(Ex2 o) {
        o.a++ ;
        o.affiche();
    }
    public void affiche() {
        System.out.println(a);
    }
    public static void main(String[] args) {
        Ex2 obj = new Ex2();
        obj.affiche();
        f(obj) ;
        obj.affiche();
    }
}
```

Correction :

```
0
1
1
```

Exercice 3 Qu'affiche le programme suivant ?

```
public class Ex3 {
    int a = 0 ;
    public void f(int a) {
        System.out.println(a) ;
    }
    public static void main(String[] args) {
        Ex3 p = new Ex3();
        p.f(12);
    }
}
```

Dans la fonction `f`, comment faire si on veut également afficher la valeur de l'attribut `a` ?

Correction :

```
12
```

Pour afficher la valeur de l'attribut `a` dans la méthode `f` qui a un parametre formel `a`, on doit utiliser le mot clé `this` qui désigne une référence sur l'objet qui appelle la méthode. La nouvelle méthode f est donc :

```
public void f(int a) {
    System.out.println(a);
    System.out.println(this.a);
}
```

Exercice 4 Soit le programme suivant :

```
// Fichier Point.java
class Point {
    // deux attributs de type int
    int abscisse;
    int ordonnee;

    // constructeur
    Point(){
        abscisse = 0 ;
        ordonnee = 0 ;
    }

    // méthode permettant de changer les coordonnées d'un point
    void set( int u , int v ){
        abscisse = u ;
        ordonnee = v ;
    }

    // méthode permettant de translater un point
    void translate( int u , int v ){
        abscisse = abscisse + u ;
        ordonnee = ordonnee + v ;
    }
}
```

1. Ajouter à la classe `Point` la méthode `affiche`, de type de retour `void`, de sorte que `p.affiche()` affiche à l'ecran l'abscisse et l'ordonnée de `p`.
   Correction :
   ```
   void affiche(){
       System.out.println( "abscisse = " + abscisse +
                                   " ordonnee = " + ordonnee ) ;
   }
   ```

2. Ajouter à la classe `Point` la méthode `origine`, de type de retour `boolean` qui teste si les coordonnées du point sont nulles. Ajouter également une méthode `egale` telle que `p.egale(q)` renvoie `true` si et seulement si les abscisses et ordonnées des points `p` et `q` sont égaux.
   Correction :
   ```
   boolean origine(){
       return ( ( abscisse == 0 ) && ( ordonnee == 0 ) ) ;
   }

   boolean egale( Point q ){
      return ( ( abscisse == q.abscisse ) && ( ordonnee == q.ordonnee ) ) ;
   }
   ```

3. Écrire un deuxième constructeur de la classe `Point`, dont le prototype est `Point( int u , int v )` qui permet d'initialiser l'abscisse et l'ordonnée avec `u` et `v`. Écrire une seconde méthode `set`, prenant en argument un objet de la classe `Point`, et qui recopie les attributs de ces arguments.

Correction :

```
Point( int u , int v ){
    abscisse = u ;
    ordonnee = v ;
}
void set( Point q ){
    abscisse = q.abscisse ;
    ordonnee = q.ordonnee ;
}
```

4. Ajouter à la classe `Point` une méthode `symetrie` telle que `p.symetrie()` renvoie un nouvel objet `Point` qui représente le symétrique du point `p`, dans une symétrie centrale par rapport à l'origine du repère.

   Correction :

   ```
   Point symetrie(){
       Point q = new Point(0-abscisse,0-ordonnee) ;
       return q ;
   }
   ```

5. On veut numéroter les points au fur et à mesure de leur création. On ajoute donc les variables suivantes :
   `static int nombre ;`
   `int numero ;`
   où l'attribut `numero` indique le numero du point et où la variable de classe `nombre` indique combien d'objets ont été créés.
   Réécrire les constructeurs `Point` en conséquence. Réécrire également la méthode `affiche` pour observer la valeur de ces nouveaux attributs.
   Correction :

   ```
   Point(){
       abscisse = 0 ;
       ordonnee = 0 ;
       nombre++ ;
       numero = nombre ;
   }
   Point( int u , int v ){
       abscisse = u ;
       ordonnee = v ;
       nombre++ ;
       numero = nombre ;
   }
   void affiche(){
       System.out.println( "abscisse = " + abscisse +
                                   " ordonnee = " + ordonnee +
                                   " nombre de points = " + nombre +
                                   " numero = " + numero) ;
   }
   ```

Exercice 5 Ecrire une classe implémentant une *paire* d'entier :

1. Définir une classe `Paire` dont le constructeur initialise les attributs privés de la paire. Définir une méthode `affiche` et une fonction `main` pour tester cette classe.
2. Définir un deuxième constructeur, qui initialisera à 0 les composants de la paire.
3. Définir un troisième constructeur, qui initialisera une paire à l'aide d'une autre paire.
4. Définir des fonctions permettant d'accéder et de modifier chaque élément de la paire.

Correction :

```
public class Paire{
    private int first ;
    private int second ;

    public Paire( int first , int second ){
        this.first = first ;
        this.second = second ;
    }
    public Paire(){
        this.first = 0 ;
        this.second = 0 ;
    }
    public Paire( Paire p ){
        this.first = p.first ;
        this.second = p.second ;
    }
    public int getFirst(){
        return first ;
    }
    public int getSecond(){
        return second ;
    }
    public void setFirst( int x ){
        first = x ;
    }
    public void setSecond( int y ){
        second = y ;
    }
    public void affiche(){
        System.out.print( "("+ first + ","+ second +")" ) ;
    }
    public static void main( String[] args ){
        Paire p1 = new Paire( 1 , 2 ) ;
        Paire p2 = new Paire() ;
        Paire p3 = new Paire( p1 ) ;

        p1.affiche() ;
        p2.affiche() ;
        p3.affiche() ;
        p3.setFirst(6) ;
        System.out.println( "(" + p3.getFirst() + ","+ p3.getSecond() + ")" ) ;
    }
}
```

Exercice 6 Algorithmes de tri naïfs, sur des Paires d'entiers :

1. Enrichir la classe `Paire` d'une méthode définissant quand une paire est inférieure à une autre selon la règle lexicographique suivante :
   $(x1, y1) < (x2, y2)$ *ssi* $(x1 < x2)$ *ou* $(x1 = x2$ *et* $y1 < y2)$.
   Correction :

```
public boolean strictInf( Paire q ){
    return ( (this.first < q.first) || (this.first == q.first && this.second < q.second) ) ;
}
```

2. Le principe du tri bulle est de parcourir un tableau jusqu'à ce qu'il soit trié. A chaque parcours, on compare deux à deux les éléments consécutifs et on les permute s'ils ne sont pas dans l'ordre. Si lors d'un parcours on n'a permuté aucun élément, le tableau est trié. Créer une classe `Tri`, et une methode statique `Tribulle` qui trie un tableau de paires.
3. Le principe du tri par insertion consiste à considérer une à une les valeurs du tableau, et à les insérer au bon endroit dans le tableau constitué des valeurs précédemment considérées et triées. Le tableau est donc parcouru de droite à gauche, dans l'ordre décroissant. Les éléments sont donc décalés vers la droite tant que l'élément à insérer est plus petit qu'eux. Dès que l'élément à insérer est plus grand qu'un des éléments du tableau trié, il n'y a plus de décalage et on insère l'élément dans la case laissée vacante. Créer une méthode statique `TriInsertion` qui trie un tableau de paires.
4. Le principe du tri fusion suit la stratégie *diviser pour régner* : on divise le tableau à trier en deux parties, on trie (récursivement) chacune d'entre elle, et on fusionne le résultat. Créer une méthode statique `TriFusion` qui trie un tableau de paires.
5. Pour les trois algorithmes de tri que vous avez implémenté, et pour les deux tableaux suivants :
   – $\{(3,3),(2,2),(1,1),(0,0)\}$
   – $\{(0,0),(1,1),(2,2),(3,3)\}$
   Combien de comparaisons de paire ont été effectuées ? Combien de références vers des objets de la classe `Paire` ont été déclarées lors du tri ?

Correction :

```
public class Tri{
/*****************************************************************************/
//  Tri bulle

    public static void triBulle( Paire [] tabPaire ){
        // booleen qui sera vrai a la fin d'une boucle seulement si on n'a
        // pas trouve de permutation a faire c'est a dire si le tableau est trie
        boolean triFini ;
        // dernier indice du tableau
        int borne = tabPaire.length -1;
        // tampon pour la permutation de deux paires su tableau
        Paire temp ;

        do{
            triFini = true ;
            for ( int i = 0 ; i < borne ; i++ )
                // si deux elements successifs ne sont pas dans l'ordre
                // croissant
                if ( ( tabPaire[i+1] ).strictInf( tabPaire[i] ) ){
                    // le tableau n'est pas dans l'ordre
                    triFini = false  ;
                    // on permute ces deux éléments dans le tableau
```

```
                    temp = tabPaire[i] ;
                    tabPaire[i] = tabPaire[i+1] ;
                    tabPaire[i+1] = temp ;
                }
        // on sort de la boucle si le tableau est dans l'ordre croissant
        }while( !triFini ) ;
    }

/*****************************************************************************/
// Tri Insertion

   public static void triInsertion( Paire [] tabPaire ){
        int longueur = tabPaire.length;

        // La partie gauche du tableau (jusqu'a i-1) est triee
        for ( int i = 1 ; i < longueur ; i++ ){
            // on cherche a placer aPlacer dans la partie triee
            Paire aPlacer = tabPaire[i];
            // parcours de droite a gauche de la partie triee
            int parcoursTrie = i-1;
            // drapeau indiquant si on a trouve la place de aPlacer
            boolean placeNonTrouvee;

            // boucle de parcours de la partie triee droite a gauche
            // on s'arrete quand on a trouve le premier element inferieur
            // ou egal a aPlacer. Tant qu'on ne le trouve pas, on decale
            // les elements de la partie triee vers la droite pour lui
            // faire une place.
            do{
                placeNonTrouvee = false ;
                if ( aPlacer.strictInf( tabPaire[parcoursTrie] ) ){
                    // decalage de l'element courant vers la droite
                    tabPaire[parcoursTrie+1]=tabPaire[parcoursTrie];
                    parcoursTrie--;
                    placeNonTrouvee = true ;
                }
                // on est au debut de la partie triee
                if ( parcoursTrie < 0 )
                    placeNonTrouvee = false ;
            } while ( placeNonTrouvee ) ;
            tabPaire[parcoursTrie+1] = aPlacer ;
        }
    }
/*****************************************************************************/
//Tri Fusion

    public static void triFusion ( Paire tableau[]){
        int longueur = tableau.length;
        if ( longueur > 0 ){
            triFusionPartiel( tableau , 0 , longueur-1 );
        }
    }

    // tri de la partie du tableau de l'indice debut a l'indice fin
    private static void triFusionPartiel ( Paire tableau[] , int debut , int fin ){
```

```
        if ( debut != fin ){
            int milieu = ( fin + debut ) / 2;
            // tri de la moitie gauche
            triFusionPartiel( tableau , debut , milieu );
            // tri de la moitie droite
            triFusionPartiel( tableau , milieu + 1 , fin );
            // fusion des deux parties triees
            fusion( tableau , debut , milieu , fin );
        }
    }
    // fusion en partie triee des parties de debut a milieu et de milieu
    // a fin du tableau
    private static void fusion ( Paire tableau[] , int debut , int milieu , int fin ){

        //on recopie les éléments de la premiere partie a trier dans
        // un nouveau tableau pour ne pas ecraser de valeur originale
        Paire table1[] = new Paire [milieu - debut + 1];
        for ( int i = debut ; i <= milieu ; i++ ){
            table1[i-debut] = tableau[i];
        }

        // compteurs pour les deplacements dans les deux parties triées
        int compteur1 = debut ;
        int compteur2 = milieu+1 ;
        // on parcours la partie a fusionner pour y placer les elements
        //  des deux sous parties triees si compteur1 atteint milieu,
        // c'est qu'on a place tous les elements de la premiere partie
        // et les suivants sont deja dans l'ordre dans le tableau
        for ( int i = debut ; ( ( i <= fin ) && ( compteur1 <= milieu ) ) ; i++ ){
            // la deuxieme partie du tableau est placee, on fini
            // de placer les elements de la premiere
            if ( compteur2 == ( fin + 1 ) ){
                tableau[i] = table1[compteur1-debut];
                compteur1++;
            }
            // l'element courant dans la premiere partie du tableau doit
            // etre place avant l'element courant de la deuxieme partie
            else if ( ( table1[compteur1-debut] ).strictInf( tableau[compteur2] ) ){
                tableau[i] = table1[compteur1-debut];
                compteur1++ ;
            }
            // l'element courant dans la deuxieme partie du tableau doit
            // etre place avant l'element courant de la premiere partie
            else{
                tableau[i] = tableau[compteur2];
                compteur2++;
            }
        }
    }
}
```