



Exercices Java

Les collections



1) **Objectifs** : Découvrir et expérimenter les tableaux et les API de collections en **Java**. Mettre en oeuvre les différents types de collections. Assimiler la notion d'**itérateur** sur les collections. Utiliser les classes utilitaires pour effectuer des opérations sur les collections.

2) **Vocabulaire utilisé** : tableau, index, *Arrays*, *Collection*, *List*, *ArrayList*, *HashMap*, *Iterator*, généricité, *Comparator* ....

3) **Environnement technique** :

J2SE, JDK. Un EDI ( type *Eclipse/NetBeans* ).

La documentation **Java Oracle**:

<https://docs.oracle.com/javase/tutorial/collections/index.html>

### Exo 1: tri des arguments d'un tableau:

En utilisant la méthode `sort` de la classe `Array` coder un tri par ordre alphabétique des strings appartenant au tableau `args`.

Ce tableau est fourni par l'appel de la fonction `main`.

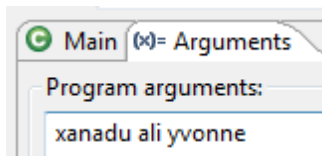
Les arguments saisis sur la ligne de commande sont rangées dans ce tableau.

Pour communiquer des arguments à votre application:

2 méthodes avec / sans I.D.E (type eclipse)

- Avec I.D.E:

- ouvrir le volet Arguments
- dans Eclipse: menu Run / Run Configurations



utiliser une boucle `for each` pour afficher les éléments du tableau:


- Sans I.D.E (avec l'interpréteur de commande `cmd.exe`)

exportation du fichier `.jar`

(voire support de cours Demarrer avec Java → 7.Créer une stand alone application page 25)

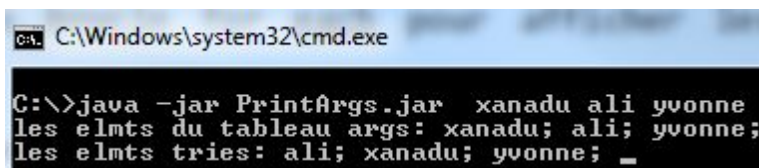
écrire un fichier `.bat` pour exécuter la commande

`java -jar PrintArgs.jar`, en ajoutant à la suite les arguments qui seront lus par l'application

 `commandebat.bat` ✕

```
1 java -jar PrintArgs.jar xanadu ali yvonne
```

*ajouter des arguments sur la ligne de commande*



```
C:\Windows\system32\cmd.exe

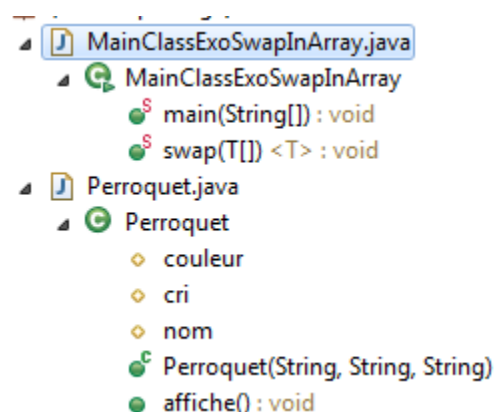
C:\>java -jar PrintArgs.jar xanadu ali yvonne
les elmts du tableau args: xanadu; ali; yvonne;
les elmts tries: ali; xanadu; yvonne; _
```

Correction : `ExoPrintsArgInRandomOrder`

**Exo 2: Ecrire une méthode generic swap pour swapper les positions de 2 éléments appartenant aux tableaux `tabStringSansNew` de type `String` et `Perroquet` de type `perroquet`**

```
public static void main(String[] args) {  
    /** test de la fonction swap avec un tableau de string*****/  
    String[] tabStringSansNew = {"String1", "String2"};  
  
    swap(tabStringSansNew);  
    // swap n'est pas une méthode de la classe Perroquet  
  
    /** test de la fonction swap avec un tableau d'objet*****/  
    Perroquet Jacko = new Perroquet("Jacko", "rouge", "koco");  
    Perroquet Pico = new Perroquet("Pico", "vert", "coco");  
    Perroquet Voiliere [] = { Jacko, Pico};  
    swap(Voiliere);  
}
```

```
before permut: String1 String2  
after permut : String2 String1  
avant permutation des perroquets dans la voiliere:  
nom: Jacko cri: koko coul: rouge  
nom: Pico cri: coco coul: vert  
apres permutation des perroquets dans la voiliere:  
nom: Pico cri: coco coul: vert  
nom: Jacko cri: koko coul: rouge
```



méthodologie:

créer 2 méthodes **swap**

une pour le type `Perroquet` et une pour le type `String`

pour au final avoir une seule méthode **swap** pour les 2 types

### Exo 3 Write a generic method to count even integers in a collection

#### objectif:

Un programme Java compte le nombre d'éléments pairs d'un tableau d'entier, nous voulons élargir ce programme au type 'character'.

Rappel:

chaque caractère a une correspondance numérique dans la table ascii

#### extrait de code:

```
public static void main(String[] args) {  
  
    int tabInt[]={1,2, 3, 4, 5, 6, 7, 8, 10, 12, 18};  
    .....  
    for(int i : tabInt)  
        if(testParite(i))  
            .....  
    Character tabCarac[]={ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' };  
    .....  
}  
public static <T> boolean testParite(T obj) {  
    if (obj instanceof Integer){  
        return (Integer).....;  
    }  
    .....  
}
```

#### Resultat:

Nombre d'entier pair = 7

Nombre de caracter pair = 4

correction :ExoGenericCountEven

#### Exo 4: écrire la fonction statique de tri → triTabSurX

fonction permettant de trier par ordre croissant des points sur la coordonnée x (transmise en argument).

**Algorithme de tri → tri à bulle (exemple à adapter pour des points)**

```
static void triTabSurX(int T[]){
    int SAV = 0;
    for (int i = 0; i < T.length - 1; i++) {
        for (int j = i + 1; j < T.length; j++) {
            if (T[i] > T[j]) {
                SAV = T[i];
                T[i] = T[j];
                T[j] = SAV;
            }
        }
    }
}
```

Les points seront des point2D, pixel, point3D.....

```
tabPts [0] = new Point ('A',3, 2) ;
tabPts [1] = new Pixel ('B',10, 8, Color.black) ;
tabPts [2] = new Point3D ('C',2, 8, 9) ;
```

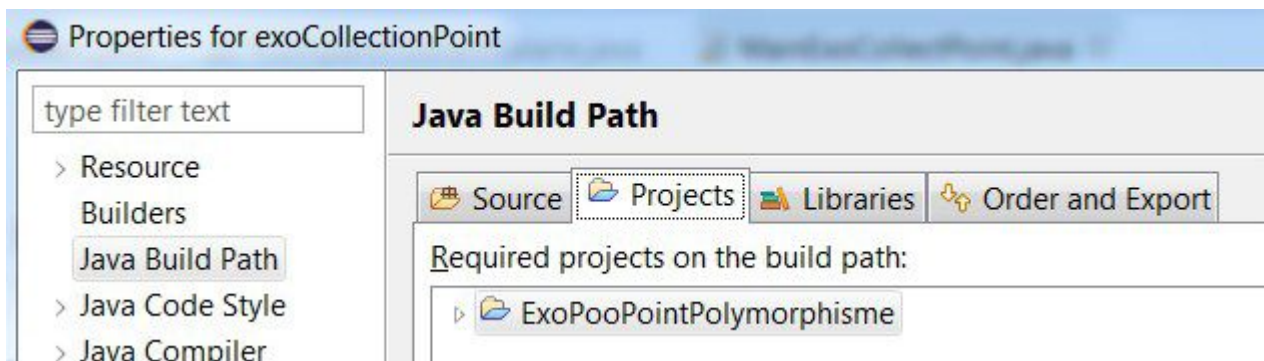
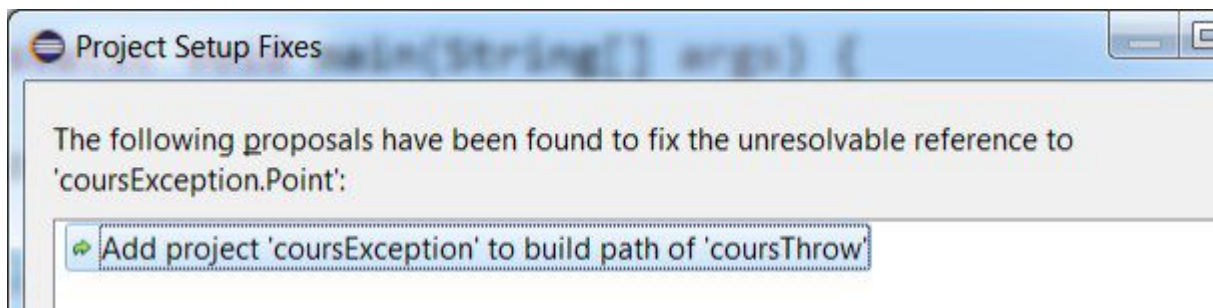
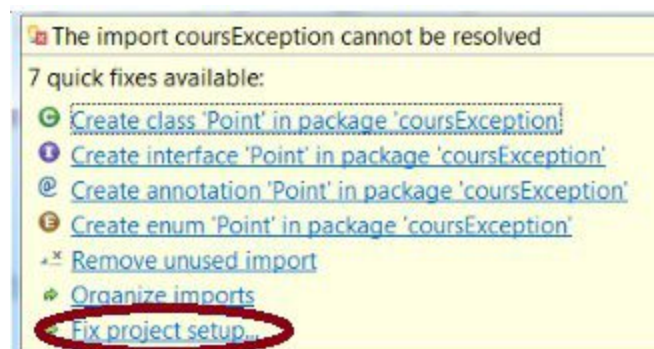
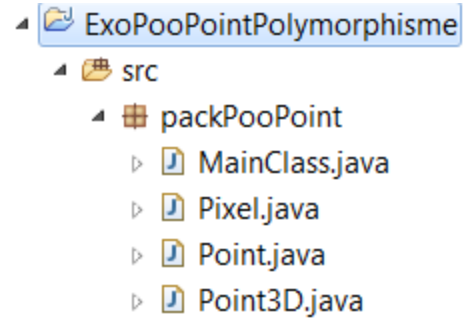
```
Console
tableau de point avant le tri
A 3 2
B 10 8
ma couleur est : java.awt.Color[r=0,g=0,b=0]
C 2 8
ma coordonne z est: 9
tableau de point apres le tri
C 2 8
ma coordonne z est: 9
A 3 2
B 10 8
ma couleur est : java.awt.Color[r=0,g=0,b=0]
```

méthodologie : commencer par tester 2 points dans votre méthode **triTab**

```
static void triTabTest(Point T[]){
    Point SAV = null;
    if (T[0].getX() > T[1].getX()) { //test de l'abscisse
        SAV = T[0]; // debut permutation
        T[0] = T[1]; // permutation en cours
        T[1] = SAV; // fin permutation
    }
    Point [] tabPtsTest = {new Point ('A',2, 0),new Point ('B',1, 0) };
```

vous pouvez judicieusement importer le package contenant ces classes. (le package doit être visible dans votre workspace)

```
import packPooPoint.Pixel;  
import packPooPoint.Point;  
import packPooPoint.Point3D;
```



correction : **exoTriABulleTabPt**

## Exo 5: tri sur une collection de point

liste des points par ordre croissant sur l'abscisse x  
**a) avec l'interface comparable**

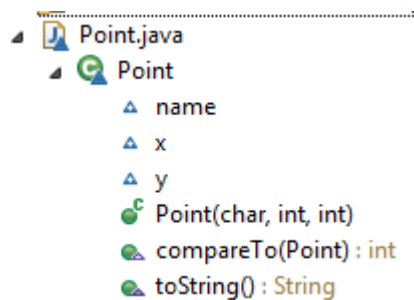
votre classe point permettre la comparaison

comme elle doit être modifiée, nous en créons une nouvelle dans le package :

**class** Point **implements** Comparable<Point> {

elle contient

- la redéfinition de la classe compareTo



utilisation de cette classe:

```
ArrayList<Point> listePts = new ArrayList<Point>();
```

```
Point Pt1 = new Point ('A',13, 2);
```

```
Point Pt2 = new Point ('B',11,4);
```

```
Point Pt3 = new Point ('C',34,45);
```

```
Point Pt4 = new Point ('D',0, 2);
```

```
//ajouter les pts a la liste
```

```
Collections.sort(listePts ); //tri la liste
```

```
affichage des éléments
```

```
(A,13,2)
```

```
(B,11,4)
```

```
(C,34,45)
```

```
(D,0,2)
```

```
affichage des éléments triés
```

```
(D,0,2)
```

```
(B,11,4)
```

```
(A,13,2)
```

```
(C,34,45)
```

avec l'interface comparable une seule comparaison est possible

si nous voulons permettre au choix la comparaison

sur x ; sur y ; sur les distances entres les points il nous faut utiliser **comparator**



## Exo 5: tri sur une collection de point (suite)

### b) avec l'interface **comparator** création de classes de comparaison:

Puisque nous ne modifions pas la classe Point, importons les classes Point et ses dérivées

```
import packPooPoint.Pixel;
import packPooPoint.Point;
import packPooPoint.Point3D;
```

- **Coder le tri sur l'abscisse x:**

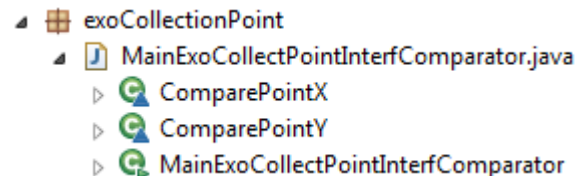
```
ArrayList<Point> listePts = new ArrayList<Point>();
Point Pt1 = new Point ('A',13, 2) ;
Pixel Pt2 = new Pixel ('B',25, 2, Color.black) ;
Point3D Pt3 = new Point3D ('C',1, 5,3) ;
Point Pt4 = new Point ('D',0, 0) ;

//ajouter les pts a la liste
//tri de la liste
Collections.sort(listePts, new ComparePointX() );
```

```
class ComparePointX implements Comparator<Point>
{
    @Override
    public int compare(Point e1, Point e2)
    {
        return (e1.getX() - e2.getX());
        .....
    }
}
```

- **Coder le tri sur l'ordonnée y**

```
(13,1)
(25,2)
(1,5)
(0,0)
liste des points triés sur coord x:
(0,0)
(1,5)
(13,1)
(25,2)
liste des points triés sur coord y:
(0,0)
(13,1)
(25,2)
(1,5)
```



A noter les classes ComparePointX, ComparePointY seront avantageusement remplacés par des classe anonyme

ci dessous l'**expression Lambda** remplace la classe ComparePointX

```
Collections.sort ( listePts, (Point e1, Point e2) → { return (e1.getX() - e2.getX());} );
```

## Exo 6: écrire et utiliser une méthode max générique

- Afficher l'élément maximal pour les 3 premiers éléments d'une liste de type integer.

En utilisant la méthode max ci dessous qui récupère la plus grande valeur de la collection

```
private static Integer max(List<? extends Integer> list, int begin, int end)
{
    Integer maxElem = list.get(begin);

    for (int i = begin; i < end; i++)
        if (maxElem.compareTo(list.get(i)) < 0)
            maxElem = list.get(i);
    return maxElem;
}
```

- Récrire max

1) pour afficher l'élément maximal pour les 3 premiers éléments d'une liste de type double et integer.

```
private static Object max(List<? extends Object> list, int begin, int end) {
    Integer maxElem = null;
    Double maxElemDble = null;

    if (list.get(begin) instanceof Integer ){
        .....
        return maxElem;
    }

    if (list.get(begin) instanceof ..... ){
        .....
        return maxElemDble;
    }
}
```

2) pour afficher l'élément maximal pour les 3 premiers éléments d'une liste de n'importe quel type.

La nouvelle méthode maxi

```
static <T extends Comparable<? super T> > T maxi(List<? extends T> list, int begin, int end) {
    T maxElem = list.get(begin);
    .....
}
```

### Dans le main appel de la méthode maxi()

```
Character tabCarac[]={'a','b','z','d','e','f','g','h'};
List<Character> myLstCarac = new ArrayList<Character>(Arrays.asList(tabCarac));
System.out.println(maxi(myLstCarac, 0, 8));
→ affiche z    correction: exoGenericMethodMax
```

## Exo 7 Glossaire Interface Map

**Objectifs** : Assimiler les notions liées aux collections de type **Map** en Java.

**Déroulement** :

créer une collection de type *Map* (donc tableau associatif) permettant de modéliser un **glossaire** .

Cette collection aura pour but de contenir des expressions liés à la P.O.O. en **Java** et d'associer pour chacun de ces mots/clé la définition correspondante .

On pourra donner plusieurs fois la même définition pour des mots distincts .  
En revanche - principe des clés - , chaque mot/clé dans le glossaire est **unique**.

Clé	Valeur
sous classe	Classe héritant d'une autre classe.
méthode	une fonction destinatrice d'un message
classe dérivée	Classe héritant d'une autre classe.
polymorphisme	Un comportement différent pour un même message
collection	Instance d'une classe gérant un ensemble d'éléments.
bonheur	Maîtriser Java et ses subtilités ...
interface	Classe dont toutes les méthodes sont abstraites
instance	Objet créé à partir d'une classe.

- Créez le tableau associatif suivant :

```
HashMap<String,String> glossaire = new HashMap<.....>() ;
```

Vous remarquerez que la **généricité** nous permet ici de typer la clé : *String* et la valeur : de type *String* également.



- Remplissez le glossaire avec des clés et leur définition associée ( comme le suggère la figure précédente) .
- Affichez le contenu clé du glossaire

**pas besoin d'itérateur pour parcourir la collection**

```
for (Map.Entry<String, String> entry : glossaire.entrySet())  
{  
    System.out.println(entry.getKey() );  
}
```

## Exo 7 Glossaire Interface Map (suite)

- Dans une boucle inviter l'utilisateur à saisir un mot pour en connaître sa définition

 Console 

```
-bonheur
-instance
-sousclasse
-classe dérivée
-collection
-interface
-polymorphisme
saisir un des mot présents dans le glossaire pour afficher sa définition?
votre saisie?:
polymorphisme
Un comportement différent pour un même message
voulez vous continuer:(o/n)?
o
saisir un des mot présents dans le glossaire pour afficher sa définition?
votre saisie?:
instance
Objet créer à partir d'une classe
voulez vous continuer:(o/n)?
n
merci pour votre participation
```

correction: `exoMapTpGlossaire`

## **exo 8 : TP\_Comparator**

voir support de cours ExoCollectionAFPA.pdf page 21: TP 8: TP\_Comparator

**exercice testé et approuvé!!!**

correction: `exoListHeteroIComparator`