Comparaisons, collections et classes anonymes

Exercice 1 - Ensemble et Bag

1 Écrire un programme indiquant quels sont les mots qui se trouvent sur la ligne de commande, on affichera les doublons une unique fois.

```
java Exo2 toto tutu toto titi tutu
```

Affiche "toto", "tutu" "titi".

2 Écrire un second programme pour qu'il compte le nombre de fois qu'un mot apparait sur sa ligne de commande.
N.B : penser à faire des fonction assez générales.

Exercice 2 - List et Wildcard

1 Générifier le code ci-dessous, en transformant `Pair` en `Pair<U,V>`.

```
class Pair {
  private final Object first;
  private final Object second;

  public Pair(Object first,Object second) {
    this.first=first;
    this.second=second;
  }
  public Object getFirst() {
    return first;
  }
  public Object getSecond() {
            return second;
  }
  public static void main(String[] args) {
    Pair p1=new Pair("toto","titi");
    Pair p2=new Pair(2,p1);

    Pair p3=(Pair)p2.getSecond();
  }
}
```

2 Générifier ensuite le code suivant :

```
  public static void main(String[] args) {
    Pair p1=new Pair("toto","titi");
    Pair p2=new Pair(2,p1);

    Pair p4;
    if (args.length==0)
      p4=p1;
    else
      p4=p2;
  }
```

3 Ajouter dans la classe `Pair` une méthode `contains(List)` (bien typé) prenant en paramètre une liste et renvoyant vrai si la paire courante est stockée dans la liste.

Exercice 3 - Performance sur les listes

Le but de cet exercice est de tester les différences de performance entre les classes `ArrayList` et `LinkedList` sur différents algorithmes.

1 Nous allons dans un premier temps chronométrer le temps d'un parcours d'une `ArrayList` contenant un million `(1 000 000)` d'entiers en utilisant un `Iterator>Integer<` (pour le parcours).
Utilisez la méthode `System.nanoTime()` pour effectuer une mesure de temps.

2 Modifier le code pour pouvoir facilement chronométrer le parcours dans le cas d'une `ArrayList` ou d'une `LinkedList`.

3 Dans le but de faire d'autres tests de performance, imaginer un patron de conception (**design pattern**) permettant d'effectuer plusieurs tests sur plusieurs types de `List`.
On appelle patron de conception, une façon d'arranger les objets dans un but précis, ici pour effectuer des tests sur des listes.

4 Refactoriser le code existant en utilisant le patron de conception ainsi défini.

Utiliser le patron de conception pour effectuer les tests suivants sur les deux implémentations de `List` :

- parcours de la liste d'un million d'entiers par un itérateur (dans les deux sens).
- parcours de la liste d''un million d'entiers par un index (dans les deux sens).
- en insérant un milier d'entiers en première position dans une liste vide au départ (comme pour une file).

Exercice 4 - Vue d'une liste

Qu'affiche le code ci-dessous :

```
String[] array=new String[]{"toto","titi","tutu"};
List<String> list=Arrays.asList(array);
Collections.sort(list);

System.out.println(array[0]);
```

Expliquer le concept de vue.

Exercice 5 - Comparaison

Ecrire un programme qui prend les arguments sur la ligne de commande et les affiche :

1 Dans l'ordre lexicographique (ordre du dictionnaire).

2 Dans l'ordre militaire (on compare d'abord la taille des deux chaines avant d'utiliser l'ordre lexicographique ; par exemple, `tb`, `tau` et `tata` sont dans l'ordre militaire).

3 Dans l'ordre inverse de l'ordre lexicographique.

Une fois les trois ordres implantés, on souhaite modifier le programme pour qu'il prenne en premier argument une chaîne de caractères identifiant l'affichage à choisir.

```
java Order inverse titi toto tutu
```

Ici, on demande l'affichage en ordre inverse (**inverse**) . Les deux autres chaînes possibles sont **lexicographique** et **militaire**.

PS: éviter d'utiliser un `switch` pour implanter le choix de l'affichage.

Exercice 6 - Liste et Collection de coordonnées

Le but de cette exercice est d'implanter différentes vues sur des collections.

1 Ecrire une méthode `xCoordinates` prenant en paramètre une liste de points; utilisez pour cela la classe `java.awt.Point`; et renvoyant une liste permettant de voir uniquement les abscisses de chacun des points.
Voici un exemple d'utilisation :

```
ArrayList list=new ArrayList();
list.add(new Point(1,2));

List view=Coordinates.xCoordinates(list);
System.out.println(view.get(0)); // affiche 1

list.add(new Point(3,4));
System.out.println(view.get(1)); // affiche 3
```

Créer pour cela une classe interne. Regarder la documentation de la classe `AbstractList`, celle-ci pourra vous aider.

2 Changer le code précédent en replaçant la classe interne par une classe anonyme.

3 Ecrire la même méthode mais avec comme paramètre et comme type de retour une `Collection`.