Correction TD n°8

Java Avancé

—M1 Apprentissage—

Generics

Varargs, generics, wildcards...

▸ Exercice 1. Maximums

On veut écrire une méthode tel que le code suivant fonctionne :

```
System.out.println(myMax(42,1664)); //1664
System.out.println(myMax(2014,86,13)); //2014
```

1. Quelle doit être la signature de la méthode ? (indice)
2. Comment faire pour qu'un appel à votre méthode sans aucun argument ne compile pas ?
3. Implémenter.
4. Modifier votre méthode pour qu'elle puisse fonctionner avec n'importe quel objet `Comparable`. Par exemple :

```
System.out.println(myMax(8.6,16.64)); //16.64
System.out.println(myMax("Denis", "Cornaz")); //Denis
System.out.println(myMax(8.6, "Denis")); //ne doit pas compiler !
```

```
//public static <E extends Comparable<E>> E myMax(E first, E...rest) {
public static <E extends Comparable<? super E>> E myMax(E first, E...rest) {
//certaines classes étendent d'une classe sans redefinir toutes les methodes, dont compare to par exemple.
//ici, sql.Time est comparable avec des Date, donc on fait comparable avec quelque chose au dessus de E.
    E max = first;
    for(E a : rest) {
      if(a.compareTo(max)>0) {
        max=a;
      }
    }
    return max;
  }
```

▸ Exercice 2. Be wild !

```java
private static void print(List<Object> list) {
  for(Object o:list)
    System.out.println(o);
}

public static void main(String[] args) {
  List<String> list=Arrays.asList("foo", "toto");
  print(list);
}
```

1. Pourquoi le code plus haut ne compile t-il pas ?
2. Comment le modifier pour qu'il compile (sachant qu'on veut pouvoir afficher également une liste d'entiers par la même méthode) ?
3. Ajouter une méthode statique prenant en argument une liste d'objets implémentant CharSequence et en affichant la longueur de chaque objet.

Le code proposé ne compile pas car le sous-typage classique ne marche que sur les types paramétrés ayant le même type argument. Il n'y a pas de sous-typage entre des types paramétrés ayant des types arguments sous-types. Il n'y a donc pas de sous-typage entre List<Object> et List<String>.

Pour résoudre le problème, il faut modifier la signature de la méthode print en utilisant un wildcard : private static void print(List¡ ?¿ list). C'est une écriture spécifique qui permet de faire du sous-typage sur les types paramétrés : List¡ ?¿ veut dire une liste de type fixé que l'on ne connait pas " ?" mais qui hérite de Object. Marche aussi avec un type paramétré, mais un peu plus lourd à coder.

```java
private static void print(List<?> list) {
    for(Object o:list)
      System.out.println(o);
  }

  private static void printLength(List<? extends CharSequence> list) {
    for(CharSequence o:list)
      System.out.println(o.length());
  }
```

▸ Exercice 3.

```java
public static List listLength(List list) {
   ArrayList length=new ArrayList();
   for(int i=0;i<list.size();i++) {
     CharSequence seq=(CharSequence)list.get(i);
```

```
    length.add(seq.length());
  }
  return length;
}
public static void main(String[] args) {
  List l=Arrays.asList("colonel", "reyel");
  System.out.println(listLength(l));
}
```

1. Rendre le code ci-dessus générique en :
   (a) Utilisant une variable de type `T`.
   (b) La notation wildcard.

```
  public static <T extends CharSequence> List<Integer> listLength(List<T> list) {
    if (list.isEmpty()) {
           // field does not provide type safety.
           //return Collections.EMPTY_LIST ;
      //on gagne une création d'objet en moins si la liste est vide...
           return Collections.<Integer>emptyList() ;
    }
       ArrayList<Integer> length = new ArrayList<Integer>() ;
       for (int i = 0 ; i < list.size() ; i++) {
               T seq = list.get(i) ;
               //ou CharSequence seq =
               length.add(seq.length()) ;
       }
       return length ;
 }

   public static List<Integer> listLength2(List< ? extends CharSequence> list) {
       ArrayList<Integer> length = new ArrayList<Integer>() ;
       for (int i = 0 ; i < list.size() ; i++) {
               CharSequence seq = list.get(i) ;
               length.add(seq.length()) ;
       }
       return length ;
 }
```

▸ Exercice 4. Fusion !

1. Ecrire une méthode de fusion de deux listes renvoyant une nouvelle liste avec alternativement un élément de chaque liste. Les deux listes devront être de la même taille, et en cas de liste vide, il ne faudra pas allouer d'objet inutilement. Le code suivant doit fonctionner :

```
List<String> l1= Arrays.asList("C", "rc");
List<StringBuilder> l2= Arrays.asList(new StringBuilder("a ma"), new StringBuilder("he!"));
List<? extends CharSequence> r1=fusion(l1,l2);
List<?> r2=fusion(l1,l2);
List<Integer> l3 = Arrays.asList(1,2);
```

```
List<Integer> l4 = Arrays.asList(10,20);
List<Integer> r3 = fusion(l3,l4);
List<?> r4 = fusion(l1,l3);
```

2. Quelle est la complexité de votre solution si une des deux listes est une `LinkedList` ? Modifier votre code pour ne parcourir qu'une seule fois chaque liste.

✂ ----------

```
1  private static <T> List<T> merge(List<? extends T> list1, List<? extends T> list2) {
2  // Dans ce cas la capture n'est pas possible et on ne peut pas récupérer le type d'éléments
3  // renvoyés par next() ...
4  // public static <T, T1 extends T, T2 extends T> List<T> merge(List<T1> list1, List<T2> list2) {
5    if(list1.size()!=list2.size()) throw new IllegalArgumentException("must be same size");
6    if(list1.isEmpty()) return Collections.<T>emptyList();
7    List<T> sol = new ArrayList<T>(list1.size());
8    Iterator<? extends T> it1 = list1.iterator();
9    //Iterator<T1> it1 = list1.iterator();
10   Iterator<? extends T> it2 = list2.iterator();
11   while(it1.hasNext()) {
12     //sol.add(l1.get(i));
13     sol.add(it1.next());
14     sol.add(it2.next());
15   }
16   return sol;
17 }
```

---------- ✂

▸ Exercice 5. Mélange !

Le but de l'exercice est la création d'une méthode mélangeant les éléments d'une liste prise en argument.

1. Ecrire une méthode prenant en arguments une liste et deux entiers, et échangeant les éléments des la liste situés aux indices correspondant aux deux entiers.
2. On souhaite que la liste prise en argument de cette méthode soit paramétré par un wildcard. Est-ce possible ?
3. Réfléchir à un algorithme permettant de mélanger les éléments d'une liste prise en argument et l'écrire. Vous pouvez utiliser la classe `Random`.
4. Quelle est la complexité de votre implémentation ? Et dans le cas de l'utilisation d'une `LinkedList` ? Peut-on mieux faire ? Faire mieux en s'inspirant de la méthode shuffle de `Collections` (`http://www.docjar.com/docs/api/java/util/Collections.html`).

✂ ----------

1. .

2. Pas directement à cause du set. On peut feinter et faire capturer en appelant la méthode avec le generics.
3. pour i de size-1 à 1, l'échanger avec un élement au hasard entre 0 et i.
4. n^2 à cause du set(i). Collections copie tout dans un tableau, fait le shuffle dans le tableau puis recopie avec un iterateur, donc $2n$.

```java
public static <E> void shuffle(List<E> l) {
  Random r = new Random();
  for(int i=l.size()-1;i>0;--i) {
    int j = r.nextInt(i);
    swap(l,i,j);
  }
}

public static void shuffleWC(List<?> l) {
  Random r = new Random();
  for(int i=l.size()-1;i>0;--i) {
    int j = r.nextInt(i);
    swapWC(l,i,j);
  }
}

private static void swapWC(List<?> l, int i, int j) {
  swap(l, i, j);
}
private static <E> void swap(List<E> l, int i, int j) {
  E tmp = l.get(i);
  //set impossible avec un wildward parceque set ne garantie pas que l'objet est du bon type
  //utilise un type parametre
  l.set(i, l.get(j));
  l.set(j, tmp);
}

/*
public static <E> void doubleShuffleE(List<E> l1, List<E> l2) {
}

public static void doubleShuffle(List<?> l1, List<?> l2) {
  doubleShuffleE(l1, l2);
  //trick pas possible ici
}
*/
```