Generics, wildcard, iterable, iterateur

Exercice 1 - Carte sauvage

1 Pourquoi le code ci-dessous ne compile-t-il pas ?

```
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(args);
  print(list)
}
```

Que doit-on changer pour qu'il compile ?

2 On veut écrire la méthode `printLength` prenant en paramètre une liste d'objet implantant l'interface `CharSequence` et affichant la longueur des chaînes de caractères de la liste.

Exercice 2 - Généréfication

1 Générifier le code ci-dessous :

```
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(args);
  System.out.println(listLength(l));
}
```

1 En utilisant une variable de type `T`

2 En utilisant la notation wildcard.

2 À quoi sert la constante `Collections.EMPTY_LIST` ?

Comment peut-on l'utiliser dans l'implantation de la méthode `listLength()` ?

3 Changer l'implantation de la méthode `listLength()` pour utiliser la méthode `emptyList` de la classe `java.util.Collections`.

Exercice 3 - C'est loin la merge

On souhaite écrire une méthode permettant de fusionner deux listes `List` pour obtenir une liste contenant alternativement un élement de chaque liste.
La méthode devra s'assurer que les deux listes ont la même taille.

Quel est le profil de la méthode `merge` (le plus générique possible) sachant que le code suivant est valide.

```
List<String> list1=...
List<StringBuilder> list2=...
```

```
List<? extends CharSequence> result1=merge(list1,list2);
List<?> result2=merge(list1,list2);
```

Noter qu'il y a un bug dans eclipse :)

Implanter la méthode `merge` créant une nouvelle liste. La signature de la méthode devra être la plus générique possible.

Indiquer sans implanter de nouveau code quelle est la complexité de la solution choisie si une des liste est une`LinkedList` ?

Exercice 4 - Pile suite suite

1. Expliquer la différence entre `java.util.ArrayList` et `java.util.LinkedList`.
2. Ajouter une nouvelle implantation de `Stack` utilisant une `java.util.ArrayList`.
3. Rappeler pourquoi il est essentiel de ne pas avoir le même code écrit plusieurs fois. Plusieurs réponses sont possibles.
4. On rappelle que `java.util.ArrayList` et `java.util.LinkedList` implante la même interface `List`, il est donc possible de partager du code entre les deux implantation de `Stack`.

   Où doit-on mettre le code commun ?
5. Changer le code pour qu'il soit partagé au maximum.

   Attention on veux garder une complexité optimale des opérations et éviter les casts inutiles.