Map, list, capture, iterable iterateur

Exercice 1 - Prendre de la hauteur

Soit la classe `fr.umlv.td.author.Author` :

```
public class Author {
  public Author(String firstName,String lastName) {
    this.firstName=firstName;
    this.lastName=lastName;
  }
  public @Override String toString() {
    return firstName+' '+lastName;
  }
  private final String firstName;
  private final String lastName;
}
```

Expliquer ce que fait le code ci-dessous :

```
Map<Author,String> tels=new HashMap<Author,String>();
Author danBrown=new Author("Dan","Brown");
tels.put(danBrown,"001-745-897");

System.out.println(tels.get(danBrown));
System.out.println(tels.get(new Author("Dan","Brown")));
```

Faites les changements qui s'imposent pour que le code marche de façon plus logique. Notez que `Author` est non mutable, comment peut-on faire pour accélérer les accès à la table de hachage sachant cela ?

Exercice 2 - Capture de carte sauvage

On souhaite implanter un algorithme permettant de répartir une liste dans un ordre aléatoire.

1 Indiquer un algorithme possible pour cela. Quel est la complexité de celui-ci ?

2 Implanter une méthode `swap` suivant la documentation suivante :

```
/** Swaps the elements at the specified positions in the
specified list.
  * (If the specified positions are equal, invoking this
method leaves the list unchanged.)
  *
  * @param list the list in which to swap elements.
  * @param i the index of one element to be swapped.
  * @param the index of the other element to be swapped.
  * @throws IndexOutOfBoundsException if either i or j is
out of range.
 */
```

3 Que doit-on changer pour que la méthode `swap` ait la signature suivante :

```
void swap(List<?> list,int i,int j);
```

4 Ecrire une méthode `shuffle` utilisant la méthode `swap` pour permuter les valeurs. La classe `java.util.Random` est un générateur pseudo-aléatoire.

5 En supposant que l'on veuille répartir les élements de façon aléatoire entre deux listes, expliquer pourquoi la capture ne marche pas avec une méthode ayant la signature

suivante :

```
void shuffle(List<?> list1,List<?> list2);
```

Exercice 3 - Interval

On souhaite pouvoir écrire le code suivant :

```
Iterator<Integer> it=rangeIterator(1,5);
for(;it.hasNext();)
  System.out.println(it.next()); // affiche 1 2 3 4 5
```

1 Quel est le profil de la méthode `rangeIterator` ?
2 Implanter de cette méthode en utilisant une classe anonyme.

On souhaite maintenant pouvoir écrire le code suivant :

```
for(int i:range(1,5))
  System.out.println(i); // affiche 1 2 3 4 5
```

1 Quel est le profil de la méthode `range` ?
2 Écrire une implantation utilisant la méthode `rangeIterator` ainsi qu'une version en une seule méthode.
3 Combien le code donné en exemple effectue-t-il d'allocations ?
Et le code ci-dessous ?

```
for(int i:range(190,200))
  System.out.println(i);
```

Exercice 4 - J'en veux plus

On cherche à implanter la méthode `twice` qui prend une liste d'entiers et qui renvoie une nouvelle liste contenant les valeurs de la liste multipliées par deux.

1 Implanter la méthode `twice`.
2 Changer l'implantation pour allouer moins d'éléments en utilisant le concept de vue.
Pour l'implantation, regarder du côté de `java.util.AbstractList`.
3 Quel est le problème de votre implantation dans le cas d'un appel à `twice` avec une `LinkedList` ?
Changer l'implantation de `twice` en fonction du fait que la liste prise en paramètre implante ou non `java.util.RandomAccess`.