Wildcard, iterateur, enumeration, list, map

Exercice 1 - 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.

Exercice 2 - 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.
Que peut-on faire pour accélérer les accès à la table de hachage ?

Exercice 3 - Pris la main dedans

Il est classique dans plusieurs algorithmes d'utiliser une structure de données appelée `Bag`. Celle-ci permet de stocker des objets un certain nombre de fois, la structure garde en mémoire le nombre de fois qu'un même objet (au sens de `equals`) est stocké.

1 Écrire l'interface `fr.umlv.util.bag.Bag` possédant les méthodes `add`, `remove` et `count` qui respectivement ajoute un objet, retire un objet et renvoie le nombre d'occurences d'un objet.

2 Commenter l'interface écrite.

3 Fournir une implantation de cette interface permettant d'ajouter et de retirer des élements en temps constant moyen.

4 Ajouter une méthode `Iterator<Map.Entry<T,Integer>> iterator()` qui renvoie un itérateur sur les couples (objet, nombre d'occurences).

5 Faire en sorte que l'on puisse choisir l'ordre des objets lors de l'itération par une constante lors de la construction du `Bag` :

```
     Bag<String> b1=new BagImpl<String>(ANY_ORDER);     //
n'importe quel ordre
     Bag<String> b2=new BagImpl<String>(INSERT_ORDER);  // ordre
d'insertion
     Bag<String> b3=new BagImpl<String>(NATURAL_ORDER); // ordre
naturel de T
```

Quelles sont les contraintes sur `T` en fonction de la collection utilisée ?

6 Remplacer les constantes passées lors de la construction par une énumération (`enum`).

7 Discuter de l'utilité d'une enumération abstraite ici. Implanter la solution proposée.

8 On souhaite que le code suivant marche :

```
          Bag<String> bag=new Bag<String>(ANY_ORDER);
          for(Map.Entry<String,Integer> entry:bag)
System.out.println(entry.getKey()+'='+entry.getValue());
```

Que doit-on faire ?

9 Pourquoi le code suivant ne marche pas ?

```
     Bag<?> bag=new BagImp<String>(ANY_ORDER);
     Iterator<Map.Entry<?,Integer>> it=bag.iterator();
```

Comment changer le code pour qu'il marche ?

10 De quelle interface des collections en Java doit hériter l'interface `Bag`.
Quelles sont les problèmes que cela pose par rapport au code existant ?

11 Faire hériter `Bag` de cette interface et écrire les méthodes manquantes.