Liste générique, auto-boxing et génerics

Exercice 1 - List, LinkedList et ArrayList

Expliquer ce que fait le code suivant :

```
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;
...
  public static void main(String[] args) {
    List list;
    if (args.length==0)
      list=new ArrayList();
    else
      list=new LinkedList();

    for(String s:args)
      list.add(s);
  }
...
```

1 Quel différence y a t'il entre `LinkedList` et `ArrayList` ?

2 A quoi sert l'interface `List` ?

Le code précédent compile avec un `warning`, que doit-on faire pour supprimer celui-ci ?

Ré-écrire le code du main en conséquence

De la même façon, transformer le code suivant :

```
 public static void main(String[] args) {
   List list=new ArrayList();
   Collections.addAll(list,args);

   for(int i=0;i<list.size();i++) {
     String s=(String)list.get(i);
     System.out.printf("%s:%d\n"s,s.length());
   }
 }
```

Quel est l'intérêt des `generics` en Java ?

Exercice 2 - Auto-boxing et auto-unboxing

Nous allons maintenant voir comment marche le mécanisme qui transforme tout type primitf en objet et vice-versa.

1 Que fait le code suivant :

```
        Integer value=777;
        int i=value;
```

Dans ce code, quelle instruction peut effectuer une allocation ?

PS: utiliser `javap` pour vous aider.

2 Qu'affiche le code ci-desssous ?

```
            public static boolean isEquals(Integer i,Integer j) {
              return i==j;
            }
            public static void main(String[] args) {
```

```
        System.out.println(isEquals(3,3));
        System.out.println(isEquals(128,128));
    }
```

Expliquer pourquoi ?

Nous allons maintenant utiliser le mécanisme de boxing avec la liste chaînée générique.

1. Le code suivant ne compile pas que doit-on faire ?
    1. Sans ajouter de `generics`
    2. En ajoutant des `generics`

```
LinkedList list=new LinkedList();
list.add(3);
int i=list.get(0);
```

Exercice 3 - Generification

Ré-écrire les classes `fr.umlv.datas.LinkedLing` et `fr.umlv.datas.Link` en utilisant des types paramétrés.

Exercice 4 - Stack générique

Ecrire une classe `Stack` générique permettant de gérer une pile de taille fixe indiquée à la construction.

La pile devra avoir les opérations ayant les opérations suivantes :

1. `push()` qui ajoute un élément au sommet de la pile.
   Attention, que doit-on faire si la pile est pleine.
2. `isEmpty()` qui indique si la pile est vide.
3. `pop()` qui retire le sommet de la pile.
   Attention, que doit-on faire si la pile est vide.