

# Travaux Dirigés n°6

## Java Avancé

—M1—

---

### Iterables

Lists, anonymes, iterateurs, iterable...

---

- Pour ce TD, créez un nouveau projet Maven dans votre repertoire `ja` (voir `td00`) avec les `groupId` et `artifactId` suivants.
  - `groupId` `fr.dauphine.ja.nomprenom.iterables`
  - `artifactId` `iterables`
- N'oubliez pas de commiter régulièrement et de charger les dernières versions de votre dépôt avec `git push`!

Instructions pour utiliser les tests fournis dans ce TD :

- Vous aurez besoin de la version 4.0 de JUnit. Éditez le fichier `pom.xml` de votre projet et mettez à jour la version minimale requise de JUnit depuis 3.8.1 vers 4.0.
- Pour ajouter un des fichiers test fournis à votre projet, il faut charger le fichier test dans le répertoire `src/test/java/fr/dauphine/ja/nomprenom/iterable/` et de modifier la ligne `package` du fichier test pour qu'elle corresponde à la ligne suivante.

```
1 package fr.dauphine.ja.nomprenom.iterable;
```

#### ► Exercice 1. De A à Z

On cherche à écrire une classe `Panel` qui représentera un intervalle de nombres entiers. Il faudra faire en sorte que `Panel` soit `Iterable`, c'est-à-dire qu'il soit possible de parcourir toutes les valeurs de l'intervalle avec une boucle `for`.

1. On veut que le code suivant fonctionne :

```
1 Iterator<Integer> it=panel1(1,5);
2 for(;it.hasNext();)
3     System.out.println(it.next()); // affiche 1 2 3 4 5
```

Pour que ce code fonctionne, la fonction statique `panel1()` doit retourner un objet issu de la classe `Iterator<>` correctement implémenté. Consultez la documentation de `Iterator` et implémentez la fonction `panel1()` pour faire en sorte que le code précédent fonctionne. Attention, vous ne devez pas utiliser de liste pour stocker les éléments!

2. On souhaite simplifier la code précédent en faisant usage des *classes anonymes*. Consultez la documentation des classes anonymes ici, puis modifiez le code de la fonction `panel1()` pour retourner un itérateur implémenté à l'aide d'une classe anonyme.
3. On veut maintenant faciliter le parcours de l'intervalle en supportant la syntaxe *foreach* de Java comme dans l'exemple suivant :

```
1 for(int i:panel2(1,5))
2   System.out.println(i); // affiche 1 2 3 4 5
```

Quelle interface doit-on implémenter pour pouvoir itérer de la sorte? Implémentez la fonction statique `panel2()` pour faire en sorte que cela soit possible. (On a le droit de réutiliser du code!)

4. On veut aller plus loin et faire en sorte que `Panel` puisse être traité comme n'importe quel objet du type `List`. Regardez la documentation de `AbstractList`. À quel problème répond cette classe? Quelles méthodes (abstraites) doivent être implémentées? Pourquoi ces méthodes?
5. En utilisant `AbstractList` écrivez une méthode `panel()` qui retourne un objet du type `List` qui « contient » tous les éléments de l'intervalle. **Attention**, on ne veut pas **stocker réellement** tous les éléments de la liste, mais seulement les retrouver à l'aide des deux bornes données en argument).

```
1 List<Integer> l = panel(3,6);
2 for(int i:l) {
3   System.out.println(i);
4   //affiche 3 4 5 6
5 }
6 System.out.println(l.get(1)); //affiche 4
```

6. Testez avec le fichier `PanelTest`