generics

Depuis Java 5, on peut déclarer une liste de commandes par : List**<Commande>**. Une telle liste est initialisée comme suit :

```
liste = new ArrayList<Commande>();
```

De même pour une Map, on peut déclarer Map<Article, Integer> articles initialisé par :

```
articles = new HashMap<Article, Integer>();
```

La type paramétré précisé dans **<>** doit être un type de référence (classe ou interface) et non un type primitif.

On peut également utiliser une déclaration générique pour un Iterator :

```
Iterator<Commande> it = liste.iterator();
```

Lorsqu'on utilise un tel itérateur, un cast n'est plus nécessaire :

```
while (it.hasNext())
      it.next().compléter();
      // et non pas ((Commande) it.next()).compléter();
```

Il est possible de définir une sous-classe d'un type générique :

```
class MyIterator extends Iterator<Commande> {
```

Une méthode peut renvoyer un type générique:

```
List<Commande> getListeCommandes();
```

On peut même se passer d'`Iterator` en parcourant les collections et les tableaux en utilisant un for/in :

```
for (Commande c : liste)
      System.out.println(c.getDate());

double[] table …
for (double x : table)
      System.out.println(x);
```

En réalité ceci utilise de manière cachée un `Iterator`

Par contre, un tel parcours, ne permet pas de modification de la collection ou du tableau

```
for (double x : table)
      x++;
```

Ne modifie pas le contenu de la table.

De même, on ne pourra pas supprimer dans une `Collection` durant son parcours, comme on peut le faire en appliquant la méthode `remove` sur un `Iterator`. Les objets parcourus sont toutefois modifiables mais pas la `Collection` elle-même.