# TD et TP n° 10 : Généricité (suite) et Patron État

## I) Généricité (suite)

**Exercice 1 :**

1. Y a-t-il une différence entre les signatures des deux méthodes suivantes ? (laquelle ?)
   - `static void dupliquePremierA(List<?> l){}`
   - et `static <T> void dupliquePremierB(List<T> l){}`
2. Écrivez des programmes appelant ces deux versions de méthode sur des arguments `l` différents. Existe-t-il des cas où l'une des signatures fonctionne mais pas l'autre ?
3. Programmez ces deux méthodes. Ce qu'elles doivent faire : si la liste `l` est non vide, insérer le premier élément une seconde fois dans la liste.

   Est-ce que vous y arrivez dans les deux cas ? Pourquoi ?
4. Implémentez la méthode que vous n'avez pas réussi à implémenter à la question précédente par un appel à l'autre méthode. Cette fois-ci, est-ce que ça marche ?

## II) Patron État (State pattern)

Le patron État est utilisé (entre autres) quand on voudrait pouvoir changer la classe d'un objet (ce qui est impossible).
Nous prendrons comme exemple Clark Kent qui de temps en temps se transforme en Superman et peut alors faire usage de sa capacité à voler :

### a) Première version

Regardez le schéma sur wikipedia : `https://en.wikipedia.org/wiki/State_pattern`.
Dans notre cas, nous allons avoir une classe abstraite `EtatPersonnage` et deux classes concrètes qui l'étendent : `EtatSuperMan` et `EtatHumain`. La classe `EtatPersonnage` aura comme méthodes `toString()` qu'on ne présente plus et une méthode `public boolean peutVoler()` qui sera à redéfinir au besoin.
Par ailleurs, on définira une classe `Personnage` qui aura un attribut `nom` et un attribut `etat` `protected` de type `EtatPersonnage`. Elle aura les méthodes `toString()` et `peutVoler()`
La classe `ClarkKent` qui étend `Personnage` aura en plus une méthode `public void changeEtat(boolean versSuperman)` et sa méthode `tostring` sera redéfinie de telle façon que si Clark Kent est en état SuperMan elle affiche "SuperMan".

### b) Amélioration

**Cacher l'attribut etat :** Pour cacher complètement l'attribut `etat`, il faut le rendre privé ; pour que cela n'empêche pas que la méthode `changeEtat` soit programmable, la solution est que la classe `ClarkKent` soit une classe interne statique de `Personnage`. Pour accéder à `etat`, il faudra alors utiliser `super` sinon, le compilateur va considérer que vous vouler voir état en tant que classe interne statique (ce qui n'est pas possible pour un attribut non statique) et non en tant que sous-classe.

**Déclarer changeEtat dans Personnage :** une option est de déclarer la méthode `changeEtat` aussi dans `Personnage` pour éviter les transtypages (cast), il faudra alors qu'au niveau de `Personnage` la méthode envoie une exception.

**Singletons** Il est beaucoup plus logique que les classes `EtatSuperMan` et `EtatHumain` soient des singletons. Faites les modifications qui s'imposent.