Objectifs du TD

Ce TD a pour but de présenter les structures de données génériques en Java, notamment via l'écriture d'une classe `ListeChainee`.

Exercice 1 : La classe ListeChainee

Le but de cet exercice est d'écrire une classe `ListeChainee` permettant la manipulation de listes d'éléments, et légèrement différente de celle vue en cours. Cette classe contient une valeur de type `Object` et la *référence* de l'élément suivant dans la liste (lui-même de type `ListeChainee`).

1. Ecrire la classe `ListeChainee` avec ses attributs `valeur` et `suivant`.
2. Ecrire un constructeur pour cette classe, prenant un argument de type `Object` et un argument de type `ListeChainee`.
3. Ecrire la méthode `affiche(ListeChainee laliste)` qui parcours la liste chainée jusqu'au dernier élément (`suivant` vaut alors `null`) et affiche la valeur de chaque élément traversé.
   Remarque : on affiche un objet de type `Object` via la méthode `toString()`.
4. Ecrire la méthode `tab2Liste(Object[] liste)` qui convertit un tableau d'objets en une liste chainée.
   Exemple :
   ```
   String[] tab = {"Pierre","Paul","Jacques"};
   ```
   donne lieu à la liste chaînée suivante (un objet `String` est aussi un objet `Object`) :
   ```
   laliste.valeur = "Pierre"
   laliste.suivant.valeur = "Paul"
   laliste.suivant.suivant.valeur = "Jacques"
   laliste.suivant.suivant.suivant = null
   ```
5. Définir la méthode `longueur` retournant la longueur de la liste chainée (faire un parcours *itératif* ou *récursif* de la liste).
6. Enfin, écrire une classe `UseListe` contenant une méthode `main()` créant la liste "Pierre"-"Paul"-"Jacques".

Exercice 2 : extension de la classe ListeChainee

A présent nous allons augmenter les fonctionnalités offerte par la classe `ListeChainee` en ajoutant de nouvelles méthodes :

1. Définir une méthode itérative `renverse(ListeChainee laliste)` qui renverse l'ordre des éléments d'une liste chaînée en créant une nouvelle liste chaînée.
2. Proposer une méthode récursive `renverseEnPlace()` qui effectue un renversement "en place", c'est-à-dire sans créer de nouvelle liste ni modifier un champs `valeur` d'une cellule d'une liste. Cet algorithme est particulièrement utile lorsque le champs `valeur` contient un objet volumineux.

Exercice 3 : Application - les ensembles d'entiers

Une fois la classe `ListeChainee` écrite, nous pouvons l'utiliser pour manipuler des ensembles d'entiers (dans les ensembles d'entiers, il n'y a pas de doublons). On peut représenter un tel ensemble par une liste chainée *ordonnée* d'entiers.

1. Définir la méthode `insere(Integer i, ListeChainee laliste)` qui insère un nouvel élément dans une liste chaînée *triée*.
2. Définir la méthode `tri(ListeChainee laliste)`, cette dernière va construire une nouvelle liste chainée contenant les éléments de `laliste` triés par ordre croissant. Pour cela, `tri` crée une liste chaînée vide, et appelle `insere` pour la remplir au moyen des éléments de `laliste`.
3. Ecrire les méthodes `union(ListeChainee l1, ListeChainee l2)` et `intersection(ListeChainee l1, ListeChainee l2)`.
4. Mettre à jour la méthode `main()` de la classe `UseListe` pour tester ces nouvelles méthodes.