LABO 10 – LISTES CHAINEES
Informatique 1

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

But du laboratoire (4 périodes)

1. Dans ce laboratoire vous allez réaliser l'implémentation complète de listes chaînées en *Java*. En guise de démonstrateur, vous allez réaliser une application de gestion d'itinéraire d'avion pour une fameuse compagnie aérienne, *Swixx™*. Les besoins de cette compagnie sont les suivants :
   - Pouvoir créer des vols comportant un nombre quelconque d'escales.
   - Pouvoir afficher les étapes d'un vol.
   - Pouvoir ajouter ou enlever des escales intermédiaires.
   - Vérifier si une escale particulière est prévue dans un vol.
2. La durée *estimée* pour réaliser ce laboratoire est de **quatre périodes**.
3. Vous pouvez trouver cette donnée sous forme électronique sur *http://inf1.begincoding.net* à la rubrique *Labo 10*. Vous y trouverez également le corrigé de ce labo dans deux semaines.

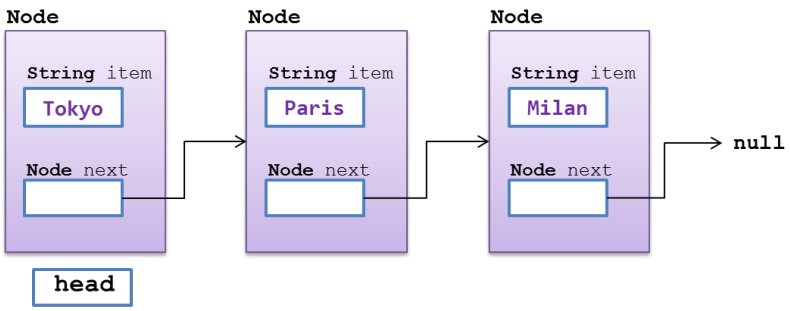
Partie 1 – Vecteurs de base, 15 minutes

Dans cette première partie vous allez travailler un peu avec les vecteurs. N'oubliez pas de rajouter la ligne `import java.util.Vector` au début de votre code.

1. On vous demande d'instancier un vecteur de `String` contenant quatre prénoms différents. Le deuxième prénom doit être « *Paul* ».
2. Affichez tous les prénoms du vecteur en utilisant la syntaxe avancée.
3. A l'aide des méthodes vues au cours sur les collections, afficher un message sur la console si « *Paul* » est dans la liste.
4. Enlevez « *Paul* » de la liste des prénoms à l'aide de la méthode adéquate.
5. Affichez tous les prénoms du vecteur et voyez comment le vecteur ne contient pas de trou. Serait-ce la même chose avec un tableau ?

Partie 2 – Listes, 2 périodes

Dans cette partie, vous allez devoir réaliser l'implémentation de listes chaînées simples. Pour ce faire, vous avez besoin de deux classes. L'une modélisera les nœuds de la liste (`Node`) et la seconde classe contiendra le premier élément (`head`) ainsi que les méthodes pour ajouter des nouveaux nœuds, afficher tous les nœuds, etc… Cette deuxième classe s'appellera `LinkedList`.



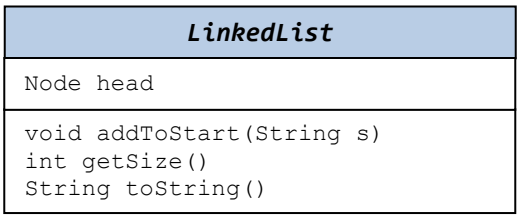
La première de ces classes, `Node`, servira à stocker les informations et à créer la chaîne de données. Cette classe `Node` doit contenir deux attributs principaux, `item` de type `String` (qui va stocker le nom de la ville) et `next` (de type `Node`) qui servira à stocker la référence sur l'élément suivant .

Tâche 1

1) Implémentez la classe `Node` comportant les attributs ci-dessus .
2) Redéfinissez le constructeur pour qu'il accepte directement comme paramètre un `String` contenant la donnée à stocker ainsi qu'une référence sur l'élément suivant de la liste.
3) Comme première liste, créez les trois nœuds dessinés ci-dessus et connectez-les comme dans le dessin simplement à l'aide des références.

Tâche 2

Vous devez maintenant définir la classe *LinkedList* qui contiendra le premier nœud ainsi que les différentes méthodes qui sont applicables à une liste. Sa description UML est la suivante :



1) Implémentez la classe en suivant la description UML donnée en suivant les indications suivantes :
   a) Commencez par implémenter un constructeur sans paramètre qui met le nœud `head` à `null`, valeur qui nous indiquera que notre liste est vide. La valeur `null` est une valeur particulière, que l'on retrouve également dans d'autres langages de programmation comme le C++, et qui sert à indiquer qu'une valeur est vide ou invalide.
   b) La méthode `addToStart(String s)` permet d'ajouter un élément au début de la liste.
   c) La méthode `getSize()` retourne le nombre total de nœuds dans la liste.
   d) La méthode `toString()` retourne la représentation textuelle de la liste(voir ci-dessous).

2) Testez la classe en ajoutant une méthode `main` dans la classe *LinkedList* et testez la classe comme ceci :

```
public static void main(String[] args){
	LinkedList flightList = new LinkedList();
	System.out.println(flightList);
	flightList.addToStart("Rome");
	System.out.println(flightList);
	flightList.addToStart("Paris");
	System.out.println(flightList);
	flightList.addToStart("Tokyo");
	System.out.println(flightList);
}
```

Et vous obtiendrez :

```
List content (size 0) : null
List content (size 1) : Rome -> null
List content (size 2) : Paris -> Rome -> null
List content (size 3) : Tokyo -> Paris -> Rome -> null
```

Partie 3 – Méthodes de liste, 2 périodes

Tâche 3

Une fois que le fonctionnement de base de votre liste est correct, vous devez rajouter les méthodes suivantes :

1) Méthode `removeFirstElement()` qui enlève le premier élément de la liste.
2) Méthode `Node getLastElement()` qui retourne le dernier élément de la liste (faites attention si la liste est vide).
3) Méthode `addToEnd(String element)` permettant d'ajouter un élément à la fin de la liste. Utiliser pour cette méthode la méthode `getLastElement()`.
4) Méthode `boolean isPresent(String e)` qui indique si `e` est présent dans la liste.
5) Testez le bon fonctionnement de vos méthodes.

Tâche 4

1) Méthode `Node findElement(String s)` qui retourne le premier nœud correspondant au `String` donné en argument.
2) Méthode `swapElements(String e1, String e2)` qui échange le contenu (et uniquement le contenu, pas les nœuds) des deux nœuds correspondant aux `String` donnés en argument. Utilisez la méthode `findElement()` ci-dessus.
3) Méthode `removeLastElement()` qui enlève le dernier élément de la liste.
4) Méthode `removeElement(String e)` qui enlève le nœud correspondant au `String` donné en argument. Attention, cette méthode comporte plusieurs difficultés.
5) Méthode `insertAfter(String before, String after)` qui crée un nouveau nœud et qui l'insère après le nœud correspondant au nœud `before`, si celui-ci existe.

Partie 4 - Test unitaires

Vous devez maintenant tester le bon fonctionnement de vos classes. Pour ce faire, vous allez mettre en œuvre des tests unitaires. Ces tests permettent de vérifier le bon fonctionnement d'une partie d'un programme ou d'une classe sans avoir à faire d'opération manuelle qui peut potentiellement laisser passer des erreurs. Cette méthode de programmation permet donc d'automatiser l'exécution de tests et éviter les régressions, c'est-à-dire que soudainement un code qui fonctionnait auparavant ne fonctionne plus en raison d'un changement.

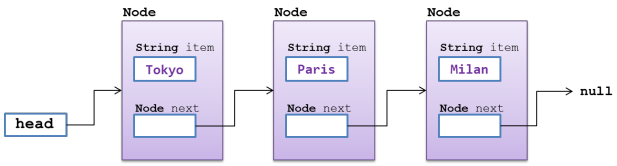
La bonne nouvelle, c'est que vous n'avez pas besoin d'écrire les tests pour la classe `LinkedList` car nous les avons déjà implémentés pour vous. Pour les utiliser :

1) Téléchargez et copiez le fichier `LinkedListTest.java` dans le répertoire src du projet.
2) Sélectionnez le projet dans Eclipse, puis appuyez F5 pour rafraîchir.
3) Ouvrez le fichier `LinkedListTest.java`
4) Mettez le pointeur sur `org.junit` (souligné en rouge sur la première ligne)
5) Sélectionnez l'option *"Fix project setup..."*
6) Lorsque *"Add JUnit 5 library to the build path"* s'affiche, sélectionnez OK.

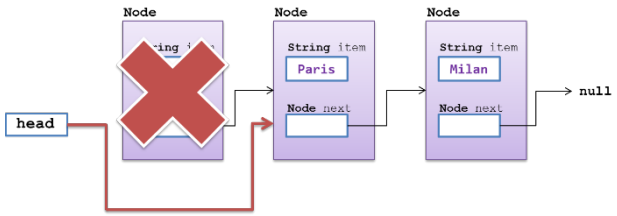
Une fois cela fait, pour exécuter un test :

1. Sélectionner le fichier de test
2. *CTRL+F11* ou alors le bouton "run" pour exécuter le test
3. L'affichage JUnit apparaît avec :
   - En haut un résumé des tests, contenant le nom du test, le résultat, ...
     - Les tests réussis sont accompagnés d'un "vu" sur fond vert.
     - Les tests qui ont échoués sont signalés par une croix sur fond bleu et sont à corriger.
     - Les tests exceptions inattendues sont signalées par une croix sur fond rouge, (généralement un NullPointerException dans ce laboratoire) et sont aussi à corriger.
   - En bas, la "*Failure Trace*", avec sur la première ligne l'erreur rencontrée. En cliquant sur la deuxième ligne il est possible d'accéder directement à la ligne du test qui a échoué.

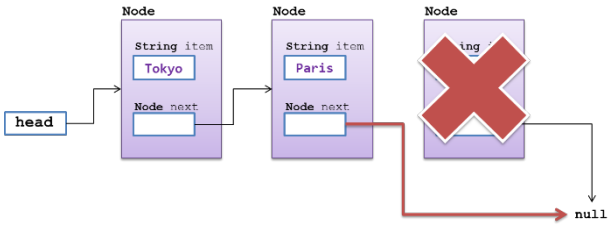
Pour vous aider, voici une description schématique de comment les opérations d'insertion et d'effacement fonctionnent sur la liste ci-dessous :



Effacement premier élément :

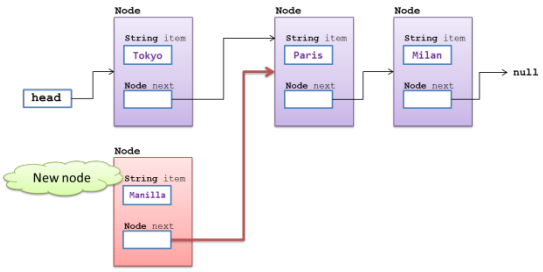


Effacement dernier élément :

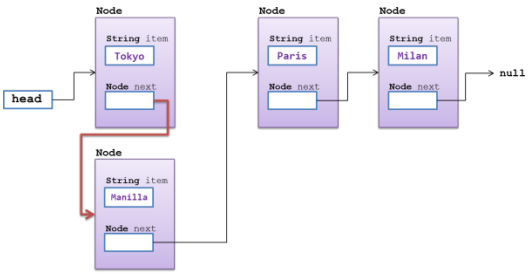


Insertion entre 2 noeuds:

a) Création du nouveau nœud :



b) Insertion du lien de l'ancien nœud :



c) Situation finale :