Les files et piles implémentées via pointeurs

Exercices obligatoires

A Implémentation de l'interface Pile via pointeurs

A1 Complétez à la main le document *A1*.
La solution de cet exercice se trouve sur moodle dans le dossier *solutions*. Elle est là pour vérifier vos réponses après avoir terminé l'exercice ! Le document s'appelle *A1Sol*.

A2 Sur moodle, répondez au questionnaire à choix multiples *PileImplChainee*.

A3 Implémentez l'interface *Pile* :
Complétez la classe *PileImplChainee*.
Testez-la avec la classe *TestPileImplChainee*.
Cette classe de tests propose un menu.
Ce menu permet de tester chaque méthode séparément.
Ce menu propose également de tester le scénario repris dans l'exercice A1.
Testez d'abord chaque méthode séparément avant de vérifier les implications des unes sur les autres via le scénario.

B Implémentation de l'interface File via pointeurs

B1 Complétez à la main le document *B1*.
La solution de cet exercice se trouve sur moodle dans le dossier *solutions*. Le document s'appelle *B1Sol*.

B2 Sur moodle, répondez au questionnaire à choix multiples *FileImplChainee*.

B3 Implémentez l'interface *File*.
Complétez la classe *FileImplChainee* et testez-là avec la classe *TestFileImplChainee*
La classe *TestFileImplChainee* reprend des tests pour chaque méthode ainsi que les tests repris dans le scénario repris dans l'exercice B1.

C Patrouille de scouts

Une troupe de scouts est divisée en patrouilles.
Le chef de patrouille (CP) est un scout qui a la charge d'une patrouille.

Un scout sera identifié par son prénom (String).

Une patrouille possède un nom (par ex : patrouille des castors) et des scouts.
Les scouts sont placés dans une liste.
Cette liste est implémentée via pointeurs.
Elle ne sera jamais vide, car elle contient au minimum toujours le CP (chef de patrouille).
Dans la liste, on trouve **en tête le CP** et ensuite les scouts qui la composent.
Les scouts y apparaitront du **plus récent (dernier ajouté) au plus ancien**.

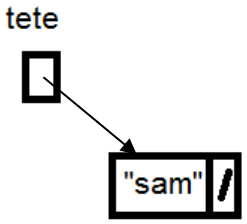
La classe *Patrouille* contient une classe interne *Nœud*.
Chaque nœud possède un « scout » et référencie le nœud suivant.

Une patrouille possède comme attribut le nom de la patrouille, le nœud de tête de la liste et le nombre de scouts.

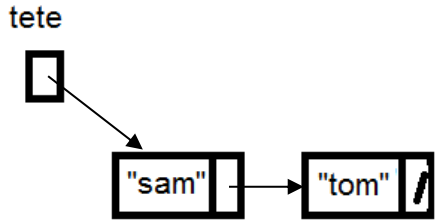
Le constructeur reçoit en paramètre le CP, il construit une liste avec un nœud qui contient ce CP.

La méthode `ajouterScout(String scout)`, ajoute le scout directement après son CP.

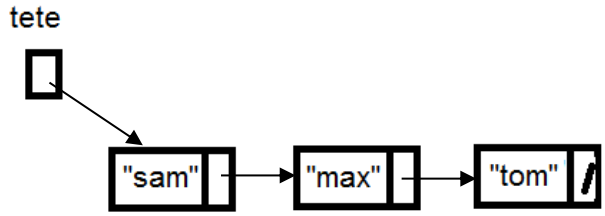
Voici le schéma d'une liste après appel du constructeur avec comme paramètre sam (qui est le CP) :



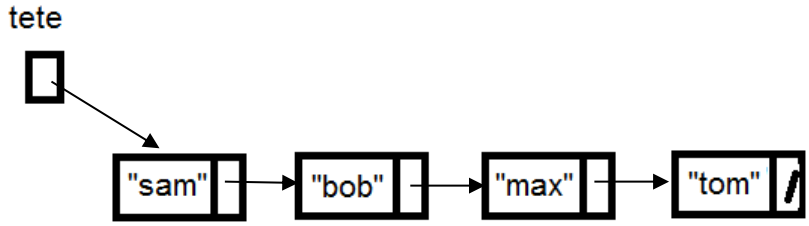
Voici le schéma de cette même liste après ajout de tom :



Voici le schéma de cette même liste après ajout de max :



Voici le schéma de cette même liste après ajout de bob :



Commencez par remplir « à la main » le document remis.

Ensuite complétez la classe *Patrouille.*
Respectez la *JavaDoc* et l'implémentation choisie.
Pour vos tests, utilisez la classe *TestPatrouille*.

Voici ce que doit afficher la classe *TestPatrouille* :

```
appel du constructeur avec sam comme CP
Apres appel du constructeur, le nombre de scouts est 1
Voici la patrouille obtenue : [sam]

ajout de tom
Apres ajout, le nombre de scouts est 2
Voici la patrouille obtenue : [sam, tom]

ajout de max
Apres ajout, le nombre de scouts est 3
Voici la patrouille obtenue : [sam, max, tom]

ajout de bob
Apres ajout, le nombre de scouts est 4
Voici la patrouille obtenue : [sam, bob, max, tom]
```

D Les drapeaux : le retour !

D1 La classe *DrapeauBelge* que vous allez compléter contient une liste chaînée de nœuds de la classe interne *NoeudCouleur*.
Cette liste va contenir les couleurs du drapeau belge.

Une couleur est représentée par un caractère :

Noir → n
Jaune → j
Rouge → r

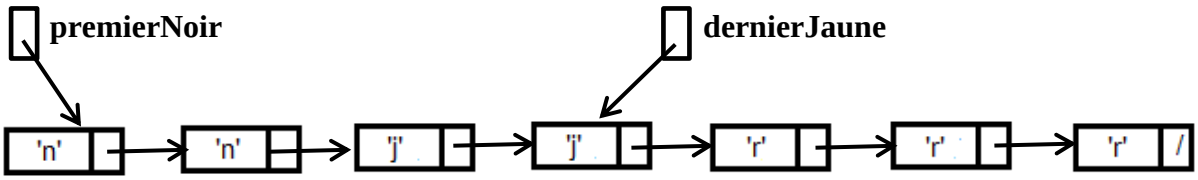
Dans la liste, les couleurs doivent apparaître triées selon les couleurs du drapeau belge : noir-jaune-rouge.

On vous demande d'écrire :

- Le constructeur
- La méthode `ajouter(char couleur)`

Pour éviter de nombreux cas particuliers à la méthode `ajouter(char couleur)`, le constructeur va initialiser la liste avec 3 nœuds : un noir, un jaune et un rouge.

Afin de rendre la méthode `ajouter(char couleur)` la plus efficace possible, 2 nœuds sont retenus : le premier nœud qui est de couleur noire (`premierNoir`) et le dernier nœud de couleur jaune (`dernierJaune`).



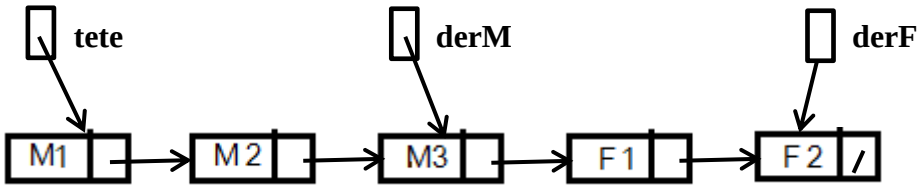
Avant d'écrire la méthode `ajouter()`, faites différents schémas (3) et réfléchissez aux cas particuliers à prévoir.

Testez votre classe grâce à la classe *TestDrapeauBelge.*

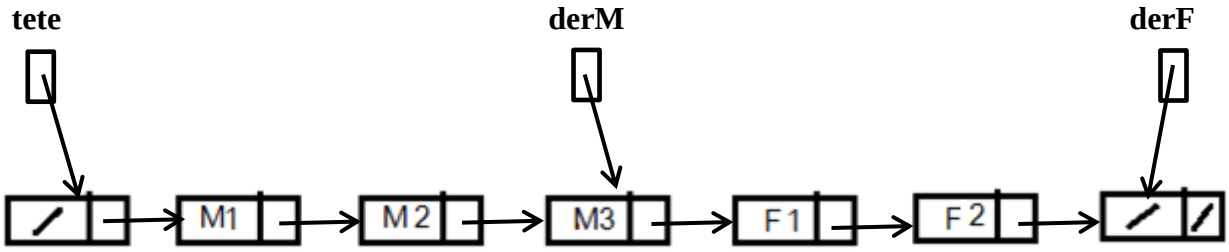
E BAL

Vous allez écrire 2 implémentations différentes de l'interface *Bal*.
Il s'agit d'écrire des méthodes qui permettent de gérer le bal de fin d'année.
Les étudiants inscrits au bal de fin d'année sont placés dans une liste.
On y trouve d'abord les hommes et ensuite les femmes.
Pour ces deux sous-listes, l'ordre dans lequel les étudiants vont apparaître doit respecter l'ordre d'encodage (du plus ancien au plus récent).
La classe *TestBal* permet de tester ces 2 implémentations.

E1 Implémentation de la classe Bal en utilisant une **liste chaînée avec sentinelles**
La classe *Bal1* contient la liste des étudiants inscrits au bal de fin d'année.
La liste est triée par sexe.
Pour permettre des ajouts sans parcours de liste, 3 nœuds sont retenus :
Le nœud de tête (`tete`), le nœud contenant le dernier homme (`derM`) et le nœud contenant la dernière femme (`derF`).

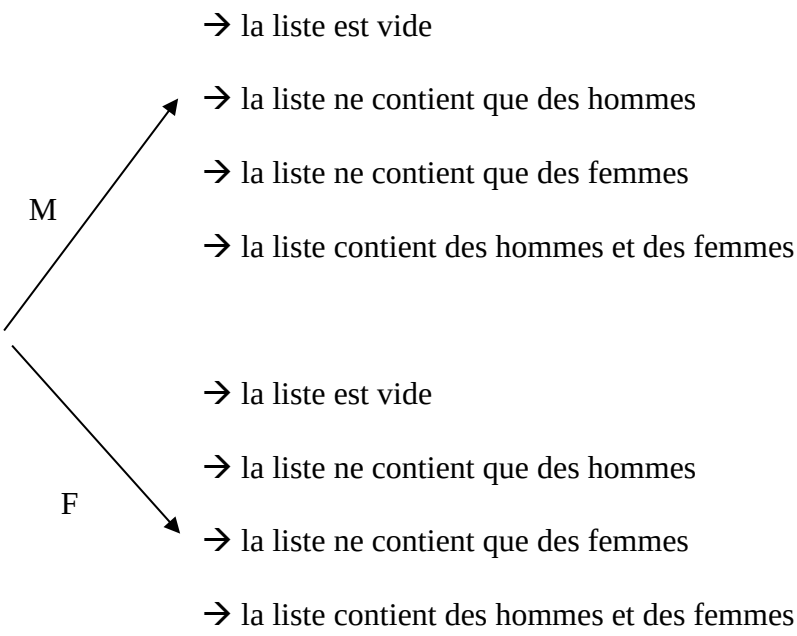


Afin d'éviter de nombreux cas particuliers, souvent on ajoute dans une liste des nœuds « bidon ». On dit que la liste contient des sentinelles.
On place 2 sentinelles : la sentinelle de tête (tete) et la sentinelle de queue (derF).



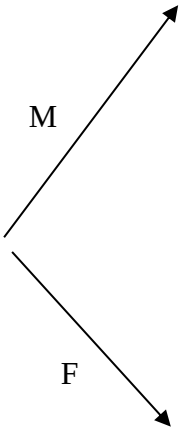
Sans sentinelle :

De nombreux cas devraient être envisagés pour la méthode `ajouterEtudiant()` :



Avec sentinelles :

Grâce à la présence des 2 nœuds bidons, il ne reste plus que deux cas à envisager pour la méthode `ajouterEtudiant()` :

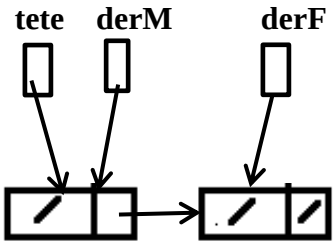


Pensez à faire deux schémas.

Complétez la classe *Bal1*

Cette classe possède 3 attributs : le nœud de tête, le dernier nœud contenant un homme et le dernier nœud contenant une femme.
Le constructeur crée deux nœuds « bidon ». (Mettez les éléments de ces nœuds à *null*). Il crée une liste de départ en enchaînant ces 2 nœuds.

Au départ :



La méthode `toString()` ne renvoie pas d'étudiants « *null* » !

E2 ❤️
Pourquoi ne pas faire simple et utiliser l'existant.
La classe *Bal2* possédera 2 files : 2 objets de type *ArrayDeque.*
Les ajouts des étudiants se feront dans l'une ou dans l'autre file selon le sexe.
La méthode `toString()` renverra bien une seule liste, mais fera la concaténation des 2 files.

F *Deque*

Un *deque* (*double ended queue*) est une structure de données dans laquelle les ajouts et les retraits peuvent se faire aux 2 extrémités.

F1 Comme la file et la pile, le *deque* peut être implémenté via une structure chaînée.
Comment aller vous chaîner les différents nœuds ? Quels nœuds allez-vous retenir ?
Faites une représentation schématique.
💣Mise en garde :
Le chaînage qui va garantir un maximum d'efficacité n'est pas évident.
Faites **valider** votre représentation schématique par un professeur !

F2 Implémentez l'interface *Deque.*
Complétez la classe *DequeImplAS.*
AS → Avec Sentinelles : une sentinelle en début de liste et une sentinelle en fin de liste
*T*estez-là avec la classe *TestDequeImplAS.*

Exercices supplémentaires

D2 La classe *DrapeauBelgeBis* est similaire à la classe *DrapeauBelge* mais elle retient 3 nœuds : le premier nœud de couleur noire, le premier nœud de couleur jaune et le premier nœud de couleur rouge.

F3
Complétez la classe *DequeImplSS.*
SS → Sans Sentinelle
*T*estez-là avec la classe *TestDequeImplSS.*