M3103 – Algorithmique avancée

TP 1.2 *(associé au Cours 1)*

Listes chaînées – deuxième partie

(1 séance encadrée, 1 séance AA)

Pour ce TP

- créer un projet **M3103 – ListeChaineeTriee** dans NetBeans (cf. Aide NetBeans).
- régler la propriété de syntaxe du projet C++11.
- copier les fichiers à compléter disponibles dans /users/info/pub/2a/M3103/tp12 dans le répertoire créé par NetBeans (cf. Aide NetBeans).
- importer les fichiers copiés (cf. Aide NetBeans).

Partie 1 : La liste chaînée triée de base

On propose la classe abstraite modèle **ListeTrieeInterface** (fichier `ListeTrieeInterface.h`) qui réalise le Type Abstrait de Données **ListeChaineeTriee** vue en TD ; **la lire, elle est documentée !**

On propose une implantation partielle de la classe modèle **ListeChaineeTriéee** (fichier `ListeChaineeTriee.h`) qui réalise le TAD Liste Triée sous forme d'une liste triée de **Cellules**.

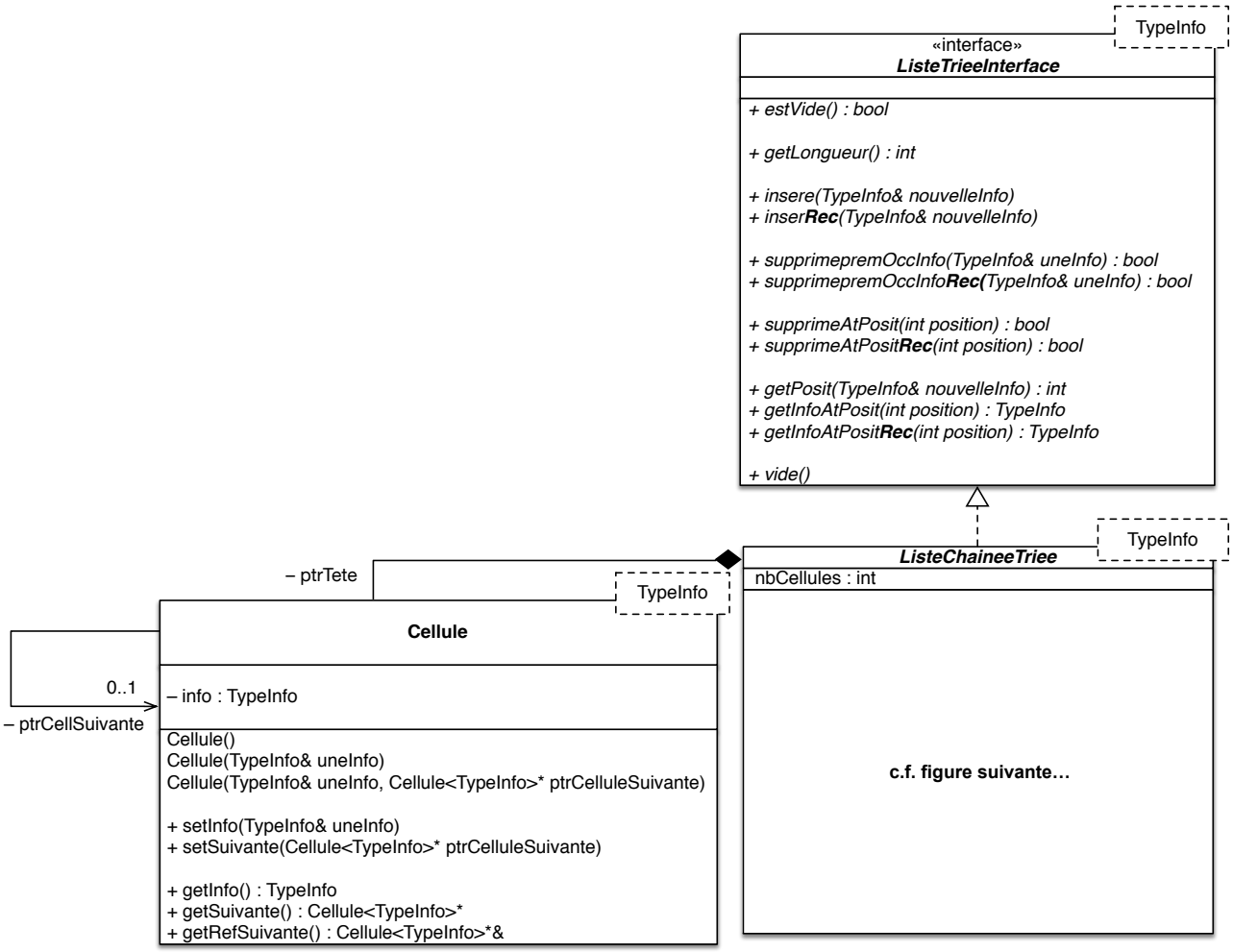

Figure 1 : Spécification UML des classes ListeTriee et ListeChaineTriee

La classe **ListeChaineeTriee** proposée est fournie Figure 2. Certaines méthodes seront implantées avec différents algorithmes d'où des noms de méthodes explicites sur le type d'algorithme.

Figure 2 : Spécification UML de la classe ListeChaineeTriee (zoom)

Afin de faciliter la lecture de la Figure 2,

- les **procédures** et **fonctions privées**, qui font le travail, sont définies sur la ligne qui suit la spécification des méthodes publiques qui les utilisent.
- les **`méthodes`** dont l'**`implantation`** est **`fournie`** sont notées en **`Courrier`**.
- les méthodes dont l'implantation est demandée sont notées en Arial.
- les méthodes dont il n'est pas question ici sont notées en Helvetica new light.

<u>À faire :</u> Préparer la configuration **TesteListeTriee** : il faut exclure **`testeListeTieeExtentions.cpp`** de la configuration **TesteListeTriee**. De fait, le point d'entrée de **TesteListeTriee** devient **`testeListeTriee.cpp`**.

Dans la suite du TP, on vous demande de compléter, dans un certain ordre, les méthodes de la classe **ListeChaineeTriee**.

Note importante :

Veiller toujours à faire apparaître l'algorithme sous forme de commentaire après l'entête de la méthode à compléter.

I. Affichage dans l'ordre croissant (récursif et itératif)

- Implémenter les workers des méthodes **`afficheCroissantRec()`** et **`afficheCroissantIter()`** de la classe **ListeChaineeTriee**.

II. Insertion associative (récursif)

- Observer la méthode **`insereRec(TypeInfo& nouvelleInfo)`** de **ListeChaineeTriee**,
- Implanter la méthode

```
insereRecWorker(Cellule<TypeInfo>*& ptrCetteListe,
                TypeInfo& nouvelleInfo)
```

et la tester avec **`testeInsereRec()`** dans **`TesteListeChaineeTriee.cpp`**.

III. Position d'une information (Itératif)

- Implanter la méthode **`getPosit(TypeInfo& nouvelleInfo)`** de **ListeChaineeTriee**, et la tester avec **`testeGetPosit()`** dans **`TesteListeChaineeTriee.cpp`**.

IV. Suppression associative (récursif)

- Observer la méthode **`supprimePremOccInfoRec(TypeInfo& nouvelleInfo)`** de **ListeChaineeTriee**,
- Implanter la méthode

```
supprimePremOccInfoRecWorker(Cellule<TypeInfo>*& ptrCetteListe,
                             TypeInfo& nouvelleInfo)
```

et la tester avec **`testeSupprimeAtPositRec()`** dans **`TesteListeChaineeTriee.cpp`**.

V. Suppression par position (récursif)

- Observer la méthode **`supprimeAtPosition(int position)`** de **ListeChaineeTriee**,
- Implanter la méthode

```
supprimeAtPositionRec(Cellule<TypeInfo>*& ptrCetteListe,
                      int position)
```

et la tester avec **`testeSupprimeRec()`** dans **`TesteListeChaineeTriee.cpp`**.

Partie 2 : La liste chaînée triée, quelques extensions

À faire – Préparer la configuration **TesteListeTrieeExtentions** : il faut exclure **`testeListeTiee.cpp`** de la configuration **TesteListeTrieeExtentions**. De fait, le point d'entrée de **TesteListeTrieeExtentions** devient **`testeListeTrieeExtentions.cpp`**.

Dans la suite du TP, on vous demande d'implanter, dans un certain ordre, les méthodes de la classe **ListeChaineeTriee**.

Note importante :

Veiller toujours à faire apparaître l'algorithme sous forme de commentaire après l'entête de la fonction, procédure ou méthode à compléter.

I. Nombre d'occurrences d'une information (itératif)

- Ajouter, en l'implantant avec un worker itératif, la méthode suivante :

```
int compteNbOccInfoIter(TypeInfo& uneInfo);
// retourne le nombre d'occurrences de uneInfo dans cette ListeChaineeTriee
```

II. Suis-je un ensemble ?

- Ajouter, en l'implantant avec un worker itératif le plus efficace possible, la méthode suivante :

```
bool estEnsemble();
// retourne faux si au moins une information est présente plusieurs fois
// dans cette ListeChaineeTriee ; vrai sinon {c'est un ensemble trié}
```

III. Suppression des occurrences multiples d'une valeur

- Ajouter, en l'implantant avec un worker itératif, la méthode suivante :

```
void supprimeDuplicationsInfo(TypeInfo& uneInfo);
// cette liste ne contient plus qu'une seule occurrence de uneInfo si
// elle en contenait plus d'une à l'origine
```

IV. Suppression de toutes les occurrences multiples

- Ajouter, en l'implantant, la méthode suivante :

```
ListeChaineeTriee<TypeInfo>* supprimeToutesDuplications();
// retourne une ListeChaineeTriee qui ne contient plus qu'une seule
// occurrence des informations présentes dans cette ListeChaineeTriee
// le résultat peut être considéré comme un ensemble trié
```

V. Intersection de deux ensembles triés

- Ajouter, en l'implantant, la méthode suivante :

```
ListeChaineeTriee<TypeInfo>*
   intersectionAvec(ListeChaineeTriee<TypeInfo>* ensembleB);
// retourne une ListeChaineeTriee qui est un ensemble correspondant à
// l'intersection de cet ensemble et de ensembleB
```