TP Listes et Ensembles

Il est fortement conseillé de lire la totalité du sujet avant de se lancer dans la réalisation.

1 Introduction

Dans ce TP, nous explorons la conception et l'implémentation d'une structure de données spécialisée pour la gestion d'ensembles dits "creux", c'est-à-dire des ensembles dont les éléments sont potentiellement nombreux, mais dont la densité d'éléments effectivement présents est faible. Ce type de représentation est particulièrement utile dans des contextes comme la manipulation de grands espaces d'états, d'ensembles d'identifiants, ou de bits, contextes où des structures classiques comme `HashSet` ou `ArrayList` peuvent s'avérer peu efficaces en termes de performances ou de mémoire. **L'implémentation de cette structure de données prendra la forme d'un ensemble personnalisé basé sur une liste doublement chainée.**

1.1 Objectifs

L'objectif principal est de mettre en œuvre une structure modulaire, permettant :
— un accès rapide aux éléments,
— une navigation itérative contrôlée (**SANS parcours inutile des données**),
— une prise en charge des opérations ensemblistes classiques (union, intersection, différence...),
— une représentation compacte.

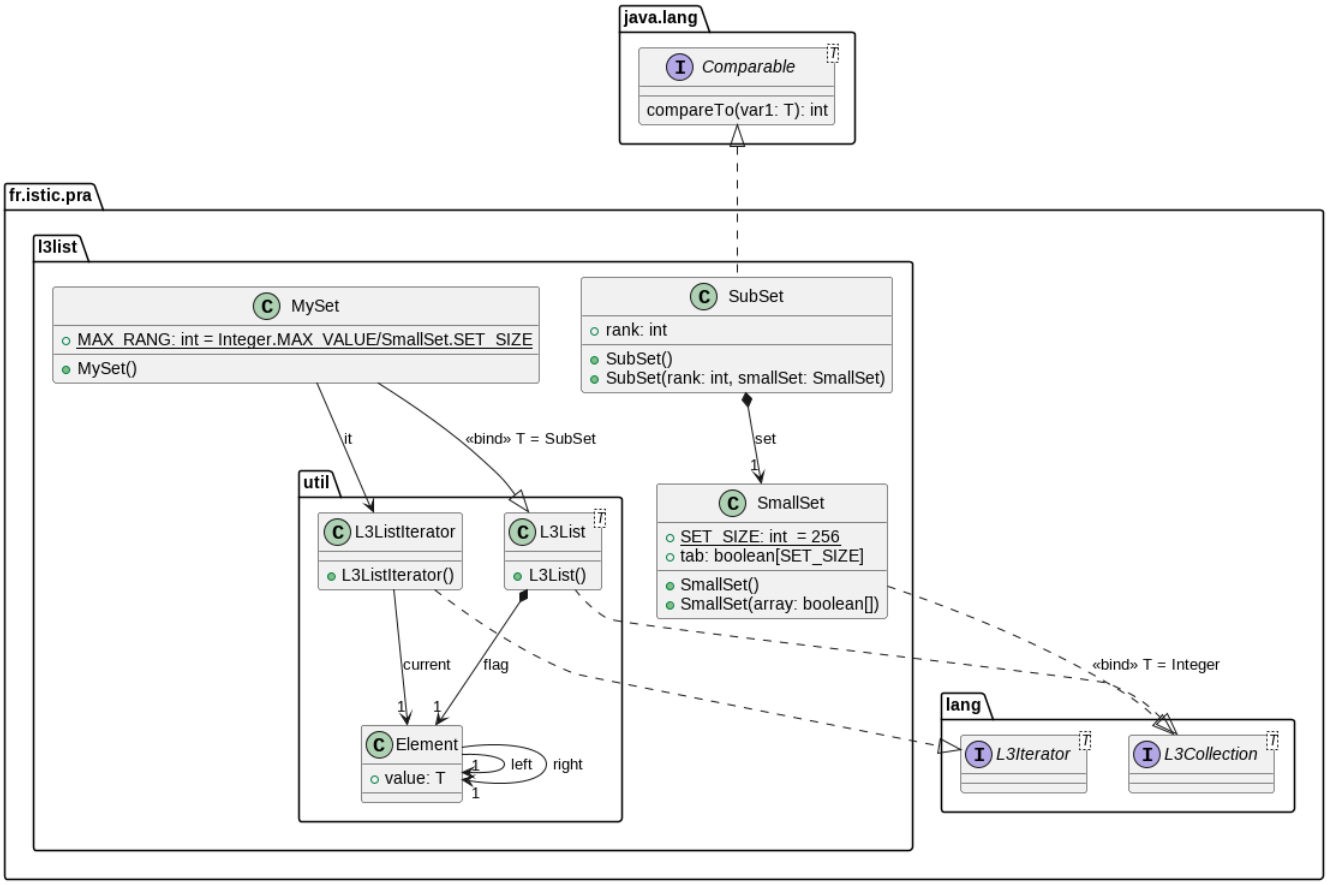
1.2 Architecture logicielle

Pour cela, nous vous fournissons un squelette de projet composé de trois grands sous-ensembles :

1. **Les contrats de conception** (package `fr.istic.pra.lang`) qui se compose des interfaces :
   — `L3Collection<T>` : interface définissant toutes les actions possibles sur une collection manipulable,
   — `L3Iterator<T>` : interface de parcours d'une liste doublement chainée.
2. **La structure de données** (package `fr.istic.pra.l3list.util`), qui se compose des classes :
   — `L3List<T>` (vu en TD) : une structure de liste doublement chaînée avec un élément fictif appelé "flag" qui n'appartient pas aux données représentées,
   — `L3ListIterator<T>` (vu en TD) : l'itérateur personnalisé associé à `L3List<T>`.
3. **L'implémentation concrète** (package `fr.istic.pra.l3list`), qui se compose des classes :
   — `SmallSet` : une structure compacte en tableau de booléens de taille fixe (`SET_SIZE`, ici fixé à 256), représentant la présence ou l'absence d'éléments,

— `SubSet` : qui contient un `SmallSet` et y ajoute un rang (c'est donc un *wrapper* de `SmallSet`), pour représenter un sous-ensemble identifié et comparable,
— `MySet` : l'implémentation principale d'un ensemble creux, par une liste chainée de `SubSet`.

Les classes sont résumées dans ce diagramme UML simplifié :
(Le modèle complet est en annexe de ce TP)



Donc, dans un `MySet`, un entier x est représenté par son rank $r = x/$`SET_SIZE` et par son modulo $m = x\%$`SET_SIZE`. Ainsi x est représenté dans le `SubSet` de rang r avec m appartenant à l'ensemble `set` associé, où `set` est une instance de la classe `SmallSet` permettant de manipuler des ensembles d'entiers variant de 0 à `SET_SIZE` $- 1$. Ce `SubSet` est alors un élément de l'ensemble `MySet` qui représente l'ensemble des entiers d'un utilisateur.

Par conséquent, un entier x appartient à l'ensemble défini par `MySet` si et seulement si l'ensemble `MySet` contient un élément `SubSet` tel que :

— la valeur du `rank` vaut $x/$`SET_SIZE`,
— l'élément $x\%$`SET_SIZE` appartient au champ `set` de cet élément.

Propriétés à respecter :

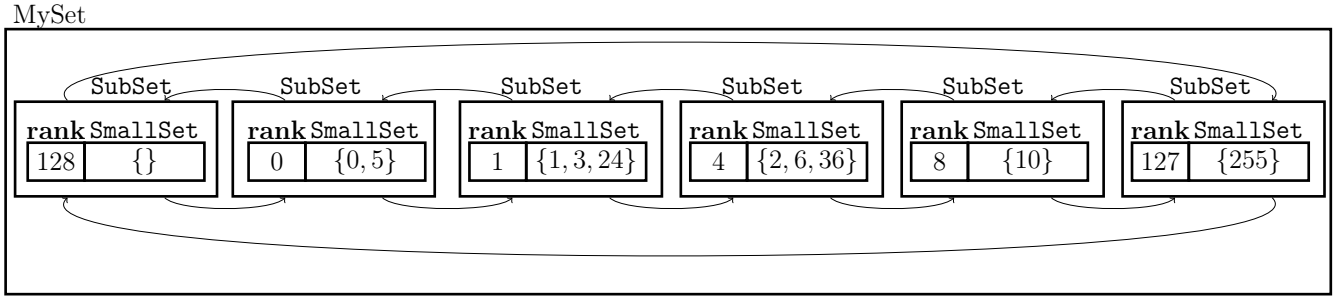
— Les éléments de l'ensemble `MySet` sont triés par ordre croissant selon le rang.
— Dans l'ensemble `MySet`, on ne fait figurer que des éléments dont le champ `set` est **non vide**.

1.3 Exemple d'un ensemble représenté avec un `MySet`

On souhaite représenter l'ensemble suivant avec un `MySet` :

$$\{0, 5, 257, 259, 280, 1026, 1030, 1060, 2058, 32767\}$$

Voici une représentation de cet ensemble dans un `MySet` :



2 Manipulation du projet

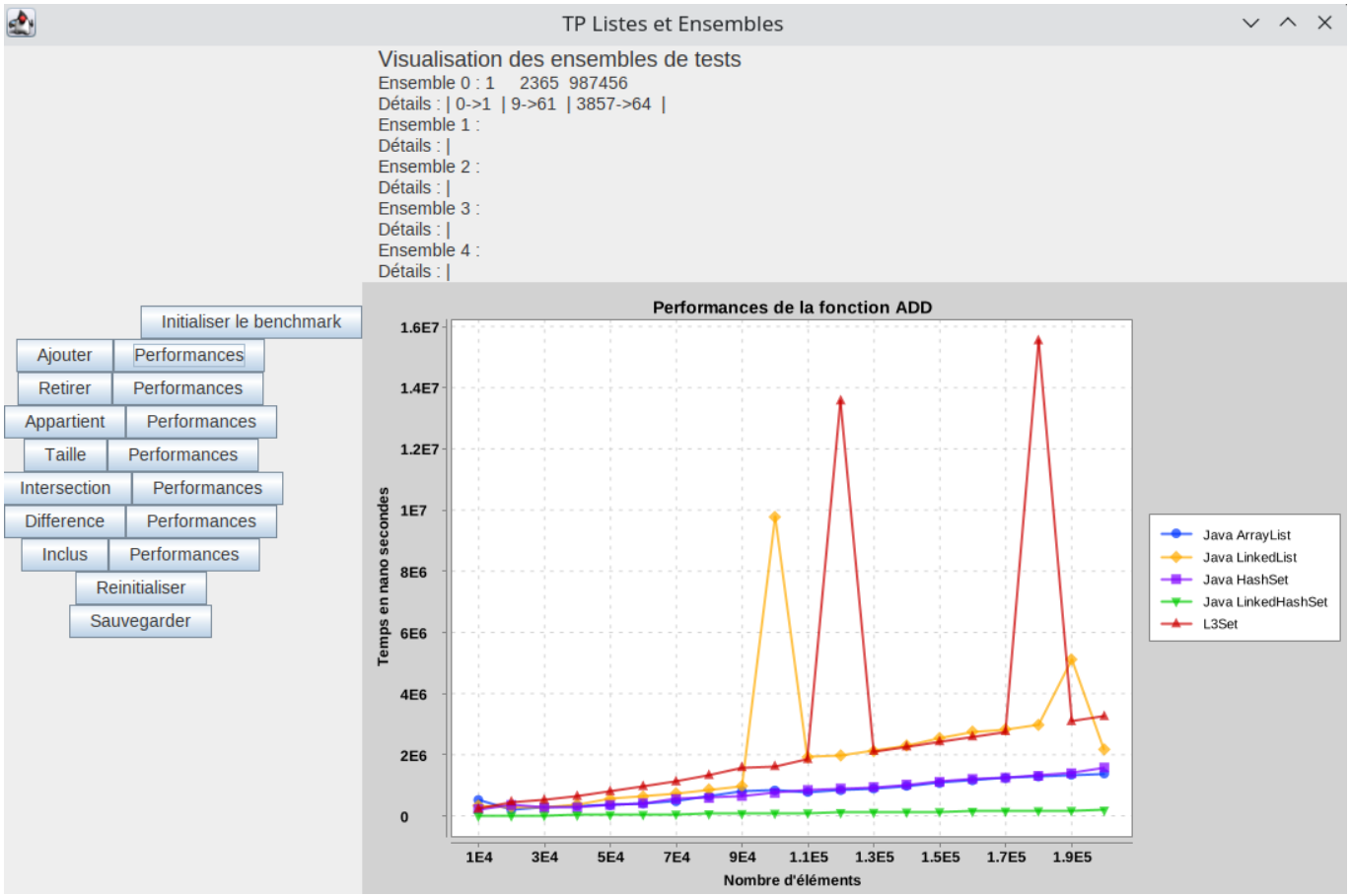
2.1 Structure du projet

Vous devez récupérer le squelette du projet maven sur le git du cours à l'adresse `https://gitlab2.istic.univ-rennes1.fr/pra/tp-listes.git` et l'ouvrir avec l'IDE de votre choix (plus de détails sur comment faire dans le `README` du projet). Nous vous y fournissons les squelettes des classes `MySet`, `L3List` et `SmallSet` ainsi que la classe `SubSet` complète. Elles sont accompagnées d'une librairie `pra-list-2.0.jar` qui contient les interfaces ainsi que le code compilé des classes `L3List` et `SmallSet`.

2.2 Tester le projet

Il y a deux façons de tester le projet :

— *Tests unitaires* : Vous disposez d'un jeu de tests dans la classe `TestMySet`.
— *Test en mode interactif* : Vous disposez d'une interface graphique qui permet d'appeler les différentes méthodes d'instances de la classe `MySet` dans la classe `App.java`. Elle permet de manipuler jusqu'à cinq ensembles et de tracer les performances de chaque méthode en les comparant à une méthode équivalente côté Java.



Attention : Le tracé de la courbe pour "intersection" peut prendre plus de 10 min !

3 Travail à réaliser

Pour commencer, voici un travail préliminaire pour bien maitriser la structure que l'on vous demande d'implémenter. Soit, les deux ensembles suivants :

$$n1 : \{10, 21, 79, 121, 350, 826, 930, 931, 4259, 30201\}$$

$$n2 : \{10, 21, 121, 360, 420, 777, 806, 2001, 3058, 32767\}$$

Donnez la représentation des ensembles $n1$ et $n2$ tel qu'ils sont définis par `MySet`.

Donnez la représentation des listes résultantes des opérations suivantes :
(a) $n1 \cap n2$ (intersection)

(d) $n1 - n2$ (différence)

3.1 Implémentation de `MySet`

Dans un premier temps, seule la classe `MySet` sera à implémenter. À ce stade, elle utilise le code compilé des classes `L3List` et `SmallSet` pour fonctionner. Pensez à tester votre implémentation avec l'interface graphique et les tests unitaires.

Attention : Vous ne pourrez rien tester tant que la fonction `add(value)` ne sera pas implémenter.

3.2 Implémentation de `L3List`

Ensuite, vous devrez implémenter la structure de liste chainée dans la classe `L3List`. AVANT DE COMMENCER À CODER VOTRE L3LIST, pour que ce soit bien votre implémentation de `L3List` qui soit utilisée et non celle de la librairie, vous devez changer l'import de cette classe dans `MySet` `SubSet`, `TestMySet` et dans `App`. Re-testez `MySet` de la même façon qu'auparavant.

3.3 Implémentation de `SmallSet` (bonus)

Enfin, en bonus, vous pouvez implémenter votre propre `SmallSet` en changeant les imports dans les classes `MySet`, `SubSet` et `TestMySet`.

4 Annexe

4.1 Model UML complet