Université Paris Diderot **Programmation Orientée Objet**
L3 **Année 2012-2013**

Projet

Représentations de données

Le but de ce projet est de développer une bibliothèque qui gère différents types de données. Toutes ces choses étant classiques, vous ne pourrez EN AUCUN CAS utiliser des librairies existantes du type `LinkedList`, `ArrayList`, `Map`, etc. Pour être plus précis tout ce qui implémente `Collection` est interdit.

Le projet est à faire en binôme impérativement. Les soutenances se feront à deux, mais la note pourra être individualisée si le travail a été trop inégalement réparti ; naturellement chacun doit être capable de répondre à toutes les questions.

Vous fournirez un rapport le plus clair possible pour expliquer vos choix, difficultés, et illustrerez le bon fonctionnement de votre projet sur des jeux de tests significatifs. Votre code devra être soigneusement commenté.

A la date fixée, vous déposerez sur Didel une archive de votre travail, et c'est cette version qui fera foi pour les soutenances.

`Important` : le travail que vous rendez `doit` être le travail du binôme. Toute copie (sur le web, sur les copains, ...) est proscrite.

1 Présentation

On s'intéresse à des entiers, regroupés soit dans des ensembles, des multi-ensembles, ou des listes triées, sur lesquels nous définissons des opérations classiques.

Nous vous demanderons d'implémenter et d'organiser au mieux des classes pour chacune des structures citées, ce qui vous amènera à introduire des interfaces et/ou des classes abstraites et, bien sûr, à justifier vos choix.

Nous attirons votre attention sur la nécessité de bien réflêchir à la hiérarchie des classes et interfaces avant de programmer.

1.1 Les structures de données

Nous appelons `multi-ensembles` une famille d'éléments au sens le plus général possible. Des éléments de même valeur peuvent apparaître, et il n'y a pas de notion d'ordre dans l'accès à cette structure. Par exemple, le multi-ensemble $\{1; 2; 4; 1; 1; 5\}$ est égal au multi-ensemble $\{1; 1; 1; 2; 4; 5\}$

Les `ensembles` sont des multi-ensembles, où l'on interdit les doublons dans les contenus. Ce sont les ensembles "classiques".

Pour les `multi-ensembles` et les `ensembles` nous vous demandons deux implémentations concrètes `signi cativement di erentes` : l'une utilisant des tableaux, et l'autre des listes chaînées.

Les `listes triees` sont des structures pour lesquelles les insertions se font "à la bonne place", c'est-à-dire en maintenant l'ordre. On peut accéder à n'importe quel élément par son numéro d'ordre. Une seule implémentation vous est demandé (listes chaînées), mais toutefois on distinguera deux cas : selon qu'on autorise ou pas les doublons. Ces deux cas devront être traités dans des classes différentes.

1.2 Les opérations

Chaque spécicifité des structures nous amène à préciser ce que nous entendons pour les opérations qui parfois sembleront ambigües. Ainsi dans le cas des multi-ensembles, la suppression d'un élément peut se comprendre comme la suppression de tous les éléments de même valeur, ou bien comme la suppression d'une occurrence. Ces deux interprétations donneront lieu à deux méthodes. Il peut arriver que vous soyez face à des interprétation différentes. Si c'est le cas, identifiez les clairement, et proposez une solution.

Voici la liste des opérations que l'on souhaite voir implémentées :

- constructions des structures vides
- ajout d'un élément
- suppression d'un élément. (voir commentaire ci-dessus)
- produire un énumerateur qui respecte l'ordre (ou l'absence d'ordre) associé à la structure. Ainsi les structures non ordonnées doivent donner des énumérateurs aléatoirement différents à chaque appel. (On pourra cependant dans un premier temps faire un énumérateur qui donne les élément toujours dans le même ordre.)
- redéfinition de la méthode `toString()`
- union et intersection pour des structures identiques. Une version statique des ces méthodes produira un nouvel objet, une version dynamique modifiera l'objet d'appel sans modifier l'argument. A noter :
  - l'intersection prend en compte les multiplicités dans les structures où il peut y en avoir.

1.3 Conversions - Héritages

Certaines conversions peuvent être laissées au mécanisme de l'héritage, d'autres devront être explicites. Ecrivez les méthodes de conversions nécessaires.

Puisque nous vous avons demandé une version statique et dynamique pour l'union et l'intersection, dans vos jeux de tests vous préciserez les cas intéressant de liaison.

2 Interface utilisateur

Dans un premier temps vous écrirez une interface textuelle où l'on pourra faire les opérations selon un menu. Puis vous développerez une interface graphique. Vous pourrez, par exemple, produire une fenêtre composée de deux grandes lignes. La première se divisera en 3 larges colonnes et la seconde prendra toute la largeur de la fenêtre. Dans les deux premières colonnes, vous afficherez dans chacune une structure (avec son type). La seconde ligne servira de dialogue pour les opérations (boutons d'intersection, union, création, remise à zéro, d'ajouts rapide, suppression, de transfert d'une colonne à l'autre, etc). Enfin la troisième colonne affichera les résultats.

3 S'il vous reste du temps

Vous pouvez

1. proposer une version générique de ses structures, pour la totalité de la hiérarchie ou juste une partie.
2. ajouter la structure de `files` à la hiérarchies : les `files` sont des structures ou l'ordre d'insertion et de suppression correspond à ce qu'on observe dans la gestion des files d'attentes. Cette structure sera implémentée en utilisant une (ou des) classes bien choisies de la hiérarchie de `Collection`.