Réalisation d'un ensemble d'entiers

L'objectif de ces exercices est d'écrire plusieurs réalisations d'un ensemble d'entiers. La programmation par contrat est utilisée pour spécifier l'interface mais aussi les réalisations proposées. Les contrats sont exprimés avec JML (Java Modeling Language). Les outils associés permettent d'engendrer la documentation avec la spécification et de produire un exécutable avec les contrats instrumentés et donc vérifiés lors de l'exécution du programme.

Exercice 1 : Utiliser les classes fournies
La classe `Crible` implante le crible d'Ératosthène en s'appuyant sur l'interface `Ensemble`. `EnsembleTab` est une réalisation de `Ensemble` qui stocke les éléments de l'ensemble dans un tableau.

La classe `AfficheurNombresPremiers` est le programme principal qui calcule les nombres premiers de 2 à MAX grâce au crible et à un ensemble. Il prend comme argument, sur la ligne de commande, le nom de la classe à utiliser en guise d'ensemble et la valeur de MAX. Pour afficher les nombres premiers de 2 à 100 en utilisant la classe `EnsembleTab`, il suffit de faire :

```
java AfficheurNombresPremiers EnsembleTab 100
```

Il est inutile de regarder le code de cette classe. **Dans ce sujet, l'interface `Ensemble` et les classes `Crible` et `AfficheurNombresPremiers` n'ont pas à être modifiées.**

1.1 *Exécuter sans instrumentation les contrats.* Compiler avec javac et exécuter avec java la classe `AfficheurNombresPremiers` (voir ci-dessus). Est-il facile de savoir si l'exécution s'est déroulée normalement, d'identifier les erreurs, leur origine et ce qui doit être corrigé ?

1.2 *Exécuter avec instrumentation des contrats grâce à JML.* Compiler en utilisant `jmlc` et exécuter avec `jmlrac` pour exécuter. Que constatez-vous ?

```
# Pour compiler tous les fichiers .java du répertoire courant avec
# instrumentation des contrats :
licorne> jmlc .

# Pour exécuter un programme instrumenté :
licorne> jmlrac AfficheurNombresPremiers EnsembleTab 100
```

Exercice 2 : Comprendre l'interface Ensemble
L'interface Java de l'ensemble d'entiers (`Ensemble`) est fournie avec les contrats exprimés en JML. En s'appuyant sur les contrats, répondre aux questions suivantes.

2.1 Peut-on ajouter un élément dans un ensemble s'il est déjà présent ?

2.2 Peut-on toujours ajouter un élément dans l'ensemble ?

2.3 Si on ajoute trois fois l'élément 2 dans l'ensemble, puis si on le supprime une fois, l'élément est-il encore présent dans l'ensemble ?

Exercice 3 : Comprendre le premier choix de réalisation de l'ensemble

`EnsembleTab` est une réalisation de l'interface `Ensemble` qui s'appuie sur un tableau d'entiers. On considère que les nouveaux éléments ajoutés à l'ensemble sont ajoutés à la fin du tableau.

3.1 Est-ce que la manière d'utiliser le tableau est bien spécifiée par les invariants de `EnsembleTab` ? On se contentera d'une explication intuitive, et donc informelle.

3.2 Pourquoi les invariants ajoutés dans `EnsembleTab` sont-ils déclarés **`private`** ?

3.3 Que deviennent les invariants définis dans `Ensemble` ?

3.4 Quels sont les contrats des méthodes `supprimer(`**`int`**`)` et `ajouter(`**`int`**`)` ?

Exercice 4 : Compléter la première réalisation des ensembles d'entiers

On souhaite écrire la classe `EnsembleTab` en respectant les choix faits dans l'exercice 3.

4.1 Compléter et/ou corriger le corps des méthodes de la classe `EnsembleTab`.

4.2 Comparer le temps de calcul nécessaire pour afficher les nombres premiers entre 2 et 1000 avec puis sans instrumentation du code. Pour la version instrumentée, il faut compiler avec jmlc. Pour la version sans instrumentation, il faut *tout* recompiler avec javac.

4.3 Compiler et exécuter la classe de test EnsembleTest. On doit lui passer en premier argument de la ligne de commande le nom de la réalisation de l'ensemble à utiliser, donc EnsembleTab.

Objectifs de ce sujet :

- Intégrer la spécification / programmation par contrat dans le développement ;
- Utiliser les contrats comme une aide à la mise au point ;
- Définir des contrats significatifs ;
- Faire la différence entre invariants de spécification et invariants d'implantation ;
- Comprendre l'intérêt d'une interface Java pour définir plusieurs réalisations !