



TP Noté 2011-2012 - Durée : 1h50

POO : Programmation Orientée Objet
Première année



Tous supports autorisés.

L'objectif du TP est de développer un micro-système capable d'évaluer des expressions algébriques sur des ensembles.

Nous allons considérer deux types d'implémentations des ensembles :

- une reposant des tableaux à taille fixe (`Object[]`);
- l'autre reposant sur des tableaux à taille variable (`java.util.ArrayList`).

Éléments fournis. Un ensemble de classes de test vous sont fournies, elles sont disponibles :

- dans le répertoire `/home/depot/1A/POO/TP_NOTE`
- sur internet, <http://www.loria.fr/~oster/poo/tp-note-2012.html>
- sur internet, <http://gerald.oster.free.fr/poo/tp-note-2012.html>

Voici la liste des classes et interfaces qui sont fournies :

- | | |
|--|---|
| – <code>set.Factory</code> | – <code>test.TestOperator</code> |
| – <code>set.SetFactory</code> | – <code>test.TestBinaryOperator1</code> |
| – <code>set.ArraySetFactory</code> | – <code>test.TestBinaryOperator2</code> |
| – <code>set.ArrayListSetFactory</code> | – <code>test.TestUnion1</code> |
| | – <code>test.TestUnion2</code> |
| – <code>test.AbstractTest</code> | – <code>test.TestIntersect1</code> |
| – <code>test.TestObjectSet</code> | – <code>test.TestIntersect2</code> |
| – <code>test.TestArraySetImpl1</code> | – <code>test.TestMinus1</code> |
| – <code>test.TestArraySetImpl2</code> | – <code>test.TestMinus2</code> |
| – <code>test.TestSetIterator</code> | – <code>test.TestArrayListSetImpl1</code> |
| – <code>test.TestSetIterable</code> | – <code>test.TestArrayListSetImpl2</code> |
| – <code>test.TestSetArraySetIterator1</code> | – <code>test.TestArrayListSetIterator1</code> |
| – <code>test.TestSetArraySetIterator2</code> | – <code>test.TestArrayListSetIterator2</code> |
| – <code>test.TestArraySetImpl3</code> | – <code>test.TestArrayListSetImpl3</code> |
| | – <code>test.TestAll</code> |
| | – <code>test.TestUnion3</code> |
| | – <code>test.TestIntersect3</code> |
| | – <code>test.TestMinus3</code> |

Éléments à rendre. Avant la fin de la séance, vous devez rendre votre travail et recopiant les classes dans le répertoire : `/home/depot/TP-NOTES/POO-<horaire>/<login>`. L'accès à ce répertoire sera automatiquement interdit à la fin de la séance. Vous déposerez également une archive (`.zip` ou `.tar.gz`) en utilisant le formulaire web disponible à l'adresse : <http://neptune.esial.uhp-nancy.fr/~oster/>.

Tests et Compilation. Les classes de test fournies vous permettront de tester les différentes classes que vous écrirez. Ces classes vous afficheront quels sont les tests qui ne s'exécutent pas correctement et vous donneront à titre indicatif un score pouvant s'apparenter à votre note de TP. **À la fin de la séance de TP, il est impératif que toutes vos classes compilent ! Un manquement à cette règle sera sévèrement sanctionné.**

Ne restez pas bloqué bêtement sur un test qui échoue. Si vraiment vous n'arrivez pas à résoudre le problème, passez aux questions suivantes. Certains tests ont un impact faible sur le reste du fonctionnement de votre programme.

Rappel de dernière minute. Il est bien sûr interdit de s'inspirer (plus ou moins) du travail d'un autre élève (voisin ou d'un autre groupe). Nous disposons d'un outil capable de détecter le plagiat. Il sera utilisé pour s'assurer que votre travail est une œuvre originale.

► **Question 1.** `set/ObjectSet.java`.

Écrivez une interface `set.ObjectSet` dont les profils des méthodes sont les suivants :

- une méthode `add()` prenant un paramètre de type `Object` sans type de retour. L'implémentation concrète de cette méthode dans les sous-classes permettra d'ajouter l'objet passé en paramètre à l'ensemble.
- une méthode `addAll()` prenant un paramètre de type `ObjectSet` sans type de retour. L'implémentation concrète de cette méthode dans les sous-classes permettra d'ajouter tous les objets contenus dans l'ensemble passé en paramètre à l'ensemble courant.
- une méthode `contains()` prenant un paramètre de type `Object` et dont le type de retour est `boolean`. L'implémentation concrète de cette méthode dans les sous-classes permettra de savoir si l'objet passé en paramètre appartient ou non à l'ensemble.
- une méthode `size()` sans paramètre et dont le type de retour est `int`. L'implémentation concrète de cette méthode dans les sous-classes permettra de connaître le nombre d'objets contenus dans l'ensemble.
- une méthode `copy()` sans paramètre et dont le type de retour est `ObjectSet`. L'implémentation concrète de cette méthode dans les sous-classes permettra d'obtenir un nouvel ensemble qui sera une copie de l'ensemble courant (il contiendra donc les mêmes objets que l'ensemble d'origine).

✓ **Validation 1.** Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestObjectSet`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 2.** `set/ArraySetImpl.java`.

Écrivez une classe `set.ArraySetImpl` qui réalise l'interface `set.ObjectSet`. Cette classe utilisera un tableau à taille fixe pour implémenter la structure interne permettant de conserver les objets (de type `Object`) appartenant à l'ensemble.

Cette classe doit :

- Définir un constructeur qui prend en paramètre une valeur entière (`int`) qui correspondra à la taille du tableau à taille fixe instancié pour stocker les objets de l'ensemble.
- Définir un constructeur sans paramètre qui fait appel au constructeur avec paramètre en lui passant une taille de tableau par défaut (10).
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.ObjectSet`. Ainsi, la/les méthode(s) définie(s) dans cette interface doivent toutes être implémentées.

Il faut noter que :

- Il sera sûrement nécessaire d'ajouter au moins un attribut (variable d'instance) permettant de connaître l'indice de remplissage du tableau.
- On ne cherchera pas à gérer les cas où la capacité du tableau n'est pas suffisante. (Mais si vous le souhaitez et que en vous avez le temps, vous pouvez agrandir le tableau au moment du dépassement).
- Il est tout à fait possible d'ajouter plusieurs fois le même objet à un ensemble. Par contre, l'ajout d'une référence `null` est impossible (rien n'est ajouté dans ce cas).
- La méthode `contains()` effectuera ses comparaisons en utilisant la méthode `equals()` des objets stockés dans l'ensemble.
- Pour écrire la méthode `addAll()`, il est nécessaire de parcourir la structure interne. A ce moment du tp, la seule manière de procéder est de supposer que le paramètre `ObjectSet` est en réalité un `ArraySetImpl` (en utilisant un transtypage). Cela vous permet alors de parcourir le tableau à taille fixe interne.

✓ **Validation 2.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestArraySetImpl1` et `test.TestArraySetImpl2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 3.** `set/SetIterator.java`.

Écrivez une interface `set.SetIterator` dont les profils des méthodes sont les suivants :

- une méthode `hasNext()` sans paramètre et dont le type de retour est `boolean`. L'implémentation concrète de cette méthode dans les sous-classes permettra de savoir si il reste des objets à parcourir dans l'ensemble.
- une méthode `next()` sans paramètre et dont le type de retour est `Object`. L'implémentation concrète de cette méthode dans les sous-classes permettra d'obtenir le prochain objet de l'ensemble parcouru.

✓ **Validation 3.** Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestSetIterator`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 4.** `set/SetIterable.java`.

Écrivez une interface `set.SetIterable` dont les profils des méthodes sont les suivants :

- une méthode `iterator()` sans paramètre et dont le type de retour est `SetIterator`. L'implémentation concrète de cette méthode dans les sous-classes permettra d'obtenir un itérateur afin de parcourir l'ensemble des objets.

✓ **Validation 4.** Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestSetIterable`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 5.** `set/ArraySetIterator.java`.

Écrivez une classe `set.ArraySetIterator` réalisant l'interface `set.SetIterator` qui permet d'énumérer tous les objets d'un ensemble `set.ArraySetImpl` (implantation d'un ensemble sous la forme d'un tableau à taille fixe).

Cette classe doit :

- Définir un constructeur qui prend un tableau à taille fixe d'objets (`Object[]`) en paramètre et qui le conserve en tant qu'attribut. Ce constructeur prendra également en second paramètre une valeur entière (`int`) indiquant le nombre de valeurs dans le tableau (le tableau pouvant être partiellement rempli).
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.SetIterator`. Ainsi, les méthodes définies dans cette interface doivent être toutes implémentées. (Il sera sûrement nécessaire d'ajouter un attribut pour conserver l'indice du parcours).
- Le comportement (résultat) d'un appel à la méthode `next()` alors que la méthode `hasNext()` indique qu'il n'y a plus d'élément à parcourir n'est pas défini.

✓ **Validation 5.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestArraySetIterator1` et `test.TestArraySetIterator2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 6.** `set/ArraySetImpl.java`.

Modifiez votre classe `set.ArraySetImpl` afin qu'elle réalise l'interface `set.SetIterable`.

Modifiez la méthode `addAll()` pour qu'elle utilise un itérateur et qu'elle ne fasse plus de supposition sur le type du paramètre.

✓ **Validation 6.** Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestArraySetImpl3`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 7.** `set/Operator.java`.

Écrivez une interface `set.Operator` dont les profils des méthodes sont les suivants :

- une méthode `evaluate()` sans paramètre et dont le type de retour est `ObjectSet`.

✓ **Validation 7.** Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestOperator`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 8.** `set/BinaryOperator.java`.

Écrivez une classes abstraite `set.BinaryOperator` réalisant l'interface `set.Operator` et dont les profils des méthodes sont les suivants :

- une méthode `getFirstOperand()` sans paramètre et dont le type de retour est `ObjectSet`. Cette méthode retournera la première opérande de l'opérateur.
- une méthode `getSecondOperand()` sans paramètre et dont le type de retour est `ObjectSet`. Cette méthode retournera la seconde opérande de l'opérateur.
- une méthode abstraite `evaluate()` sans paramètre et dont le type de retour est `ObjectSet`. L'implémentation concrète de cette méthode dans les sous-classes retournera en résultat un ensemble contenant le résultat du calcul effectué par l'opérateur.

Cette classe devra conserver les deux ensembles sur lesquels elle travaille. Vous ajouterez donc le/les attributs nécessaire(s) (`op1` et `op2`) ainsi qu'un constructeur prenant ces deux ensembles en paramètre.

✓ **Validation 8.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestBinaryOperator1` et `test.TestBinaryOperator2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 9.** `set/Union.java`.

Écrivez une classe `set.Union` héritant de la classe `set.BinaryOperator` qui implémente l'opérateur d'union entre deux ensembles.

Cette classe doit :

- Définir un constructeur qui prend deux ensembles (`set.ObjectSet`) en paramètre et qui les conserve en tant qu'attribut (en faisant appel au super constructeur).
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.Operator`. Ainsi, la/les méthode(s) définie(s) dans cette interface doivent être implémentées.

Il faut noter que :

- Si vous avez besoin de créer un nouvel ensemble (par exemple pour stocker le résultat de l'évaluation de l'opérateur), vous créerez cet ensemble en utilisant l'instruction suivante :
`ObjectSet r = Factory.getInstance().create()` et non pas en appelant directement le constructeur de la classe `set.ArraySetImpl` ou `set.ArrayListSetImpl`.

✓ **Validation 9.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestUnion1` et `test.TestUnion2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 10.** `set/Intersect.java`.

Écrivez une classe `set.Intersect` héritant de la classe `set.BinaryOperator` qui implémente l'opérateur l'intersection entre deux ensembles.

Cette classe doit :

- Définir un constructeur qui prend deux ensembles (`set.ObjectSet`) en paramètre et qui les conserve en tant qu'attribut (en faisant appel au super constructeur).
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.Operator`. Ainsi, la/les méthode(s) définie(s) dans cette interface doivent être implémentées.

Il faut noter que :

- Si vous avez besoin de créer un nouvel ensemble (par exemple pour stocker le résultat de l'évaluation de l'opérateur), vous créerez cet ensemble en utilisant l'instruction suivante :
`ObjectSet r = Factory.getInstance().create()` et non pas en appelant directement le constructeur de la classe `set.ArraySetImpl` ou `set.ArrayListSetImpl`.

✓ **Validation 10.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestIntersect1` et `test.TestIntersect2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 11.** `set/Minus.java`.

Écrivez une classe `set.Minus` héritant de la classe `set.BinaryOperator` qui implémente l'opérateur la différence entre deux ensembles.

Cette classe doit :

- Définir un constructeur qui prend deux ensembles (`set.ObjectSet`) en paramètre et qui les conserve en tant qu'attribut (en faisant appel au super constructeur).
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.Operator`. Ainsi, la/les méthode(s) définie(s) dans cette interface doivent être implémentées.

Il faut noter que :

- Si vous avez besoin de créer un nouvel ensemble (par exemple pour stocker le résultat de l'évaluation de l'opérateur), vous créez cet ensemble en utilisant l'instruction suivante :
`ObjectSet r = Factory.getInstance().create()` et non pas en appelant directement le constructeur de la classe `set.ArraySetImpl` ou `set.ArrayListSetImpl`.

✓ **Validation 11.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestMinus1` et `test.TestMinus2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 12.** `set/ArrayListSetImpl.java`.

Écrivez une classe `set.ArrayListSetImpl` qui réalise l'interface `set.ObjectSet`. Cette classe utilisera la classe `java.util.ArrayList` pour implémenter la structure interne pour conserver les objets appartenant à l'ensemble.

Cette classe doit :

- Définir un constructeur sans paramètre.
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.ObjectSet`. Ainsi, la/les méthode(s) définie(s) dans cette interface doivent toutes être implémentées.

Il faut noter que :

- On ne cherchera pas à gérer les cas où la capacité du tableau n'est pas suffisante. (Mais si vous le souhaitez et que vous en avez le temps, vous pouvez agrandir le tableau au moment du dépassement).
- Il est tout à fait possible d'ajouter plusieurs fois le même objet à un ensemble. Par contre, l'ajout d'une référence `null` est impossible (rien n'est ajouté dans ce cas).
- La méthode `contains()` effectuera ses comparaisons en utilisant la méthode `equals()` des objets stockés dans l'ensemble.

✓ **Validation 12.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestArrayListSetImpl1` et `test.TestArrayListSetImpl2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 13.** `set/ArrayListSetIterator.java`.

Écrivez une classe `set.ArrayListSetIterator` réalisant l'interface `set.SetIterator` qui permet d'énumérer tous les objets d'un ensemble `set.ArrayListSet` (implantation d'un ensemble sous la forme d'un tableau à taille variable `java.util.ArrayList`).

Cette classe doit :

- Définir un constructeur qui prend un tableau à taille variable d'objets (`java.util.ArrayList`) en paramètre et qui le conserve en tant qu'attribut.
- Par héritage, cette classe implémente le contrat défini dans l'interface `set.SetIterator`. Ainsi, les méthodes définies dans cette interface doivent être toutes implémentées. (Vous pourrez conserver et utiliser l'itérateur construit par un appel à la méthode `iterator()` de la classe `java.util.ArrayList`).
- Le comportement (résultat) d'un appel à la méthode `next()` alors que la méthode `hasNext()` indique qu'il n'y a plus d'élément à parcourir n'est pas défini.

✓ **Validation 13.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestArrayListSetIterator1` et `test.TestArrayListSetIterator2`. Si nécessaire, corrigez les différentes erreurs dans votre code.

► **Question 14.** `set/ArrayListSetImpl.java`.

Modifiez votre classe `set.ArrayListSetImpl` afin qu'elle réalise l'interface `set.SetIterable`.

✓ **Validation 14.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestArrayListSetImpl3` et `test.TestUnion3`, `test.TestIntersect3`, `test.TestMinus3`. Si nécessaire, corrigez les différentes erreurs dans votre code.

✓ **Validation 15.** Testez votre implémentation complète.

Afin de relancer l'ensemble des tests fournis, vous pouvez compiler et exécuter la classe de test nommée `set.TestAll`.

Bonne chance ! ;-)