Interfaces des Programmes d'Application

INF302 - Master Informatique
Université Bordeaux 1
27 juin 2006
Aucun document autorisé, durée 3 heures

Justifier clairement et succinctement vos choix, et décrivez brièvement vos algorithmes lorsque vous le jugez nécessaire.

Toute réponse difficile à comprendre sera considérée comme fausse.

Vous pouvez rajouter du code (méthode, donnée ou classe) non demandé à chaque fois que vous le jugerez nécessaire. Ecrivez entièrement ce code, en particulier dans le cas d'exceptions, de classes abstraites ou d'interfaces.

Vous pouvez toujours considérer une classe ou une méthode demandée dans une question précédente comme disponible, même si vous n'avez pas traité cette question. Il vous est également conseillé de lire le sujet entièrement avant de commencer votre travail.

Si vous souhaitez utiliser une classe (ou une méthode particulière) de l'API Java dont vous ne vous souvenez pas du nom, donnez-lui un nom explicite et précisez-le clairement sur votre copie.

Vous pouvez à chaque fois que vous le jugez compréhensible utiliser des abréviations à la place des noms complets de classe et méthode ou encore '...' pour remplacer du code non modifié.

Les instructions `import` et `package` ne seront pas précisées.

On rappelle les interfaces `java.util.Iterator` et `java.util.Iterable`.

```
public interface Iterator<E> {
    /** Renvoie true si l'itérateur possède encore des éléments.
     */
    public boolean hasNext();
    /** Renvoie l'élément suivant de l'itérateur.
        Lève une NoSuchElementException si l'itérateur n'a plus d'élément.
     */
    public E next();
    /** Supprime de la collection sous-jacente le dernier élément
        retourné pas l'itérateur.
        Lève une IllegalStateException si la méthode next n'a pas
        encore été appelée ou si remove a déja été appelée après le
        dernier appel de next.
        Lève une UnsupportedOperationException si la méthode remove
        n'est pas supportée par l'itérateur.
     */
```

```
    public void remove();
}

public interface Iterable<T> {
    /** Renvoie un itérateur sur un ensemble d'éléments de type T.
     */
    public Iterator<T> iterator();
}
```

Partie 1 - Graphes orientés

On se donne les deux interfaces suivantes pour représenter un graphe orienté : `GrapheOriente` et `Arc`.

```
/** Graphe Oriente non modifiable.
 *  Aucun itérateur ne supporte la méthode remove. */
public interface GrapheOriente {
    /** Nombre de sommets. */
    public int n();
    /** Nombre d'arcs. */
    public int m();
    /** Presence de o comme sommet du graphe. */
    public boolean contient(Object o);
    /** Iteration sur les sommets. */
    public Iterator<Object> sommets();
    /** Iteration sur les arcs. */
    public Iterator<Arc> arcs();
    /** Degre sortant. Leve une IllegalArgumentException si s n'est pas
     *  un sommet du graphe. */
    public int degreSortant(Object s);
    /** Degre entrant. Leve une IllegalArgumentException si s n'est pas
     *  un sommet du graphe. */
    public int degreEntrant(Object s);
    /** Iteration sur les arcs sortants. Leve une IllegalArgumentException
     *  si s n'est pas un sommet du graphe. */
    public Iterator<Arc> arcsSortants(Object s);
    /** Iteration sur les arcs entrants. Leve une IllegalArgumentException
     *  si s n'est pas un sommet du graphe. */
    public Iterator<Arc> arcsEntrants(Object s);
    /** Iteration sur les successeurs. Leve une IllegalArgumentException
     *  si s n'est pas un sommet du graphe. */
    public Iterator<Object> successeurs(Object s);
    /** Iteration sur les predecesseurs. Leve une IllegalArgumentException
```

```
     *  si s n'est pas un sommet du graphe. */
    public Iterator<Object> predecesseurs(Object s);
    /** Iteration sur les arcs entre deux sommets. Leve une
     *  IllegalArgumentException si l'un des objets n'est pas un sommet du
     * graphe. */
    public Iterator<Arc> arcs(Object s1, Object s2);
}

public interface Arc {
    /** Sommet origine. */
    public Object origine();
    /** Sommet destination. */
    public Object destination();
}
```

Question 1 Ecrire une classe `ArcImpl` qui implémente `Arc`.

On rappelle que les interfaces `java.util.Set` et `java.util.Collection` étendent l'interface `Iterable`. Voici également quelques méthodes de l'interface `java.util.Map` qui pourront vous être utiles :

```
public interface Map<K,V> {
    /** true si la table contient key comme entrée. */
    public boolean containsKey(Object key);
    /** true si la table contient value comme valeur. */
    public boolean containsValue(Object value);
    /** renvoie la valeur associée à l'entrée key,
     *  null si key n'est pas une entrée. */
    V get(Object key);
    /** ensemble des entrées dans la table. */
    Set<K> keySet();
    /** associe la valeur value à l'entrée key, et renvoie
     * l'ancienne valeur associée. */
    V put(K key, V value);
    /** supprimme l'entrée key et renvoie la valeur qui lui
     *  était associée. */
    V remove(K key);
    /** nombre de clés. */
    int size();
    /** collection des valeurs dans la table. */
    Collection<V> values();
    ...
}
```

Dans la classe `GrapheOrienteImpl`, un graphe orienté est codé à l'aide de deux tables : `successions` et `precedences`.

Dans ces deux tables, les entrées sont tous les sommets du graphe. A chaque sommet `s` est associé une table `successions.get(s)` et une table `precedences.get(s)`, éventuellement vides si `s` n'a pas de successeur ou de prédécesseur.

Dans `successions.get(s)` (resp. `precedences.get(s)`), les entrées sont les successeurs (resp. prédécesseurs) de `s`. Si `t` est un successeur (resp. un prédécesseur) de `s`, alors `successions.get(s).get(t)` (resp. `precdences.get(s).get(t)`) est l'ensemble (`Set`) des arcs de `s` vers `t` (resp. de `t` vers `s`).

Voici le début de la classe `GrapheOrienteImpl` :

```
public class GrapheOrienteImpl implements GrapheOriente {

    private int m = 0;

    private Map<Object, Map<Object, Set<Arc>>> successions =
        new HashMap<Object, Map<Object, Set<Arc>>>();

    private Map<Object, Map<Object, Set<Arc>>> precedences =
        new HashMap<Object, Map<Object, Set<Arc>>>();

    protected boolean ajouterSommet(Object s) {
        if (successions.containsKey(s)) { // s est deja un sommet du graphe
            return false;
        }
        successions.put(s, new HashMap<Object, Set<Arc>>());
        precedences.put(s, new HashMap<Object, Set<Arc>>());
        return true;
    }

    protected boolean ajouterArc(Arc a) {
        Object o = a.origine();
        Object d = a.destination();

        if (!contient(o)) {
            ajouterSommet(o);
        }
        if (!contient(d)) {
            ajouterSommet(d);
        }
        Set<Arc> succ = successions.get(o).get(d);
        Set<Arc> prec = precedences.get(d).get(o);
        if (succ == null) { // Pas d'arc existant de o vers d
            succ = new HashSet<Arc>();
            successions.get(o).put(d, succ);
            prec = new HashSet<Arc>();
            precedences.get(d).put(o, prec);
```

```
        }
        // prec.add(a) et succ.add(a) sont vrais ssi a n'est pas deja presente.
        if (prec.add(a)) {
            m++;
        }
        return succ.add(a);
    }

    /**
     * Cree un graphe oriente dont les sommets sont les elements de sommets
     * ainsi que les extremites des arcs non inclues dans sommets, et les arcs
     * les elements de arcs.
     */
    public GrapheOrienteImpl(Object[] sommets, Arc[] arcs) {
        for (Object s : sommets) {
            ajouterSommet(s);
        }
        for (Arc a : arcs) {
            ajouterArc(a);
        }
    }

    ...
}
```

Question 2 Compléter la classe `GrapheOrienteImpl`. On terminera par la méthode `arcs()` plus complexe.

Question 3 Modifier les interfaces `GrapheOriente` et `Arc` pour paramétrer le type des sommets en utilisant la généricité (classes génériques).

Expliquez brièvement (deux ou trois lignes) les modifications à apporter à `GrapheOrienteImpl`. *Ne pas donner le code entier de la nouvelle version de GrapheOrienteImpl.*

Partie 2 - Graphes non orientés

Question 1 Proposer deux interfaces `GrapheNonOriente` et `Arete` pour représenter un graphe non orienté. On rappelle que l'on parle d'arête dans le cas non orienté et d'arc dans le cas orienté.

Question 2 Donnez une implémentation de ces deux interfaces. Pour implémenter `GrapheNonOriente`, on utilisera une instance de `GrapheOrienteImpl` dont les arcs seront des instances de la classe ArcArete décrite ci-dessous :

```
class ArcArete<T> extends ArcImpl<T> {
    private Arete<T> a;

    ArcArete(Arete<T> a) {
        super(a.sommet1(), a.sommet2());
        this.a = a;
    }

    Arete<T> arete() {
        return a;
    }
}
```

Attention à ne pas renvoyer plusieurs fois le même voisin ou la même arête, en particulier dans le cas d'une boucle.

Partie 3 - Dessin de graphes

Dans cette partie, on se contentera de parler des graphes non orientés.

On souhaite maintenant pouvoir dessiner les graphes. Pour cela, on dispose de dessins pour représenter les sommets et de lignes pour représenter les arcs.

```
public interface Dessin {
    public Point coordonnees();
    public void dessiner(Graphics g);
}
public interface Ligne {
    public void dessiner(Graphics g, Point p1, Point p2);
}
```

Un graphe dessinable sera un graphe possèdant une méthode `public void dessiner(Graphics g)`.

Question 1 Proposer une solution utilisant l'héritage. Les sommets seront des instances de `Dessin` et les arêtes des instances à la fois de `Ligne` et d'`Arete`.

Question 2 Proposer une solution utilisant la décoration. Les instances de `Dessin` et de `Ligne` seront fournies à l'instanciation du décorateur. Quelle est l'avantage de cette solution par rapport à la précédente ?