

**Programmation Objet Avancée,**  
**UE INF460**  
*Corrigé*  
**Master Informatique 1 - Master Miage 1 - Master**  
**BioInformatique 2**  
**Olivier Baudon, Université Bordeaux 1**  
**Jeudi 19 juin 2008, 14h**

**Exercice - Itération**

Écrire dans une classe `Iterations` une méthode de classe  
`public static <T> Iterator<T> iterationAvecRepetition(Iterator<T> it,  
int[] repetitions).`

```
public class Iterations {
    public static <T> Iterator<T> iterationAvecRepetition(Iterator<T> it,
                                                         int[] repetitions) {

        final Iterator<T> itf = it;
        final int[] rf = repetitions;

        return new Iterator<T>() {
            int indice = 0;
            int repetition = 0;
            T suivant = null;

            public boolean hasNext() {
                return suivant == null ? itf.hasNext() : true;
            }

            public T next() {
                if (suivant == null) {
                    suivant = itf.next();
                    repetition = rf[indice++];
                }
                T resultat = suivant;
                if (repetition == 1)
                    suivant = null;
                else
                    repetition--;
                return resultat;
            }
        }
    }
}
```

```

        public void remove() {
            throw new UnsupportedOperationException();
        }
    };
}
}

```

## Problème - Graphe et Navigateur

**Question 1** Compléter les méthodes `nVertices()` et `nEdges()` dans la classe `DefaultSimpleGraph`.

**Question 2** Modifier la classe `DefaultSimpleGraph` de telle façon que l'instruction `System.out.println(g)`

où `g` est un graphe à  $n$  sommets et  $m$  arêtes, affiche sur la sortie standard `Graph with  $n$  vertices and  $m$  edges.`

```

public class DefaultSimpleGraph implements SimpleGraph {
    ...
    public int nVertices() {
        return neighborhoods.size();
    }

    public int nEdges() {
        int m = 0;
        for (Iterator<?> itv = vertices(); itv.hasNext();)
            m += neighborhoods.get(itv.next()).size();
        return m;
    }
    ...
    public String toString() {
        return "Graph with " + nVertices() + " vertices and " + nEdges()
            + " edges.";
    }
}

```

**Question 3** Modifier l'interface `SimpleGraph` et la classe `DefaultSimpleGraph` de telle façon que :

- si on essaye d'ajouter une boucle sur un sommet  $v$ , en invoquant `addEdge(v, v)`, cette méthode retournera une exception `LoopException`;
- si on essaye d'ajouter une arête entre deux sommets  $v1$  et  $v2$ , ou tester si deux sommets  $v1$  and  $v2$  sont voisins, ou demander un itérateur sur les voisins d'un sommet  $v$ , et que  $v$ ,  $v1$  ou  $v2$  ne sont pas des sommets du graphe, alors la méthode `addEdge` ou `areNeighbors` ou `neighbors` lèvera une exception `NoSuchVertexException`.

```

public interface SimpleGraph {

    public int nVertices();

    public int nEdges();

    boolean containsVertex(Object vertex);

    boolean addVertex(Object vertex);

    boolean addEdge(Object vertex1, Object vertex2)
        throws NoSuchVertexException, LoopException;

    boolean areNeighbors(Object vertex1, Object vertex2)
        throws NoSuchVertexException;

    Iterator<Object> vertices();

    Iterator<Object> neighbors(Object vertex) throws NoSuchVertexException;
}

public class DefaultSimpleGraph implements SimpleGraph {
    ...
    public boolean addEdge(Object vertex1, Object vertex2)
        throws NoSuchVertexException, LoopException {
        if (!containsVertex(vertex1))
            throw new NoSuchVertexException(vertex1);
        if (!containsVertex(vertex2))
            throw new NoSuchVertexException(vertex2);
        if (vertex1.equals(vertex2))
            throw new LoopException(vertex1);
        if (neighborhoods.get(vertex1).contains(vertex2))
            return false;
        neighborhoods.get(vertex1).add(vertex2);
        neighborhoods.get(vertex2).add(vertex1);
        return true;
    }
}

```

```

    }

    public boolean areNeighbors(Object vertex1, Object vertex2)
        throws NoSuchElementException {
        if (!containsVertex(vertex1))
            throw new NoSuchElementException(vertex1);
        if (!containsVertex(vertex2))
            throw new NoSuchElementException(vertex2);
        return neighborhoods.get(vertex1).contains(vertex2);
    }
    ...
    public Iterator<Object> neighbors(Object vertex)
        throws NoSuchElementException {
        if (!containsVertex(vertex))
            throw new NoSuchElementException(vertex);
        return neighborhoods.get(vertex).iterator();
    }
    ...
}

```

**Question 4** Donner une implémentation de la classe `LoopException`. La classe `NoSuchVertexException` n'est pas demandée.

```

public class LoopException extends Exception {
    private Object v;

    public LoopException(Object v) {
        this.v = v;
    }

    public Object getVertex() {
        return v;
    }

    public String getMessage() {
        return "Loop not allowed";
    }
}

```

**Question 5** Modifier l'interface `SimpleGraph` et la classe `DefaultSimpleGraph` de façon à ce que le type des sommets devienne générique.

```

public interface SimpleGraph<V> {

```

```

...
boolean containsVertex(V vertex);

boolean addVertex(V vertex);

boolean addEdge(V vertex1, V vertex2) throws NoSuchVertexException,
    LoopException;

boolean areNeighbors(V vertex1, V vertex2) throws NoSuchVertexException;

Iterator<V> vertices();

Iterator<V> neighbors(V vertex) throws NoSuchVertexException;
}

public class DefaultSimpleGraph<V> implements SimpleGraph<V> {

    private Map<V, Set<V>> neighborhoods = new HashMap<V, Set<V>>();

    ...
    public int nEdges() {
        int m = 0;
        for (Iterator<?> itv = vertices(); itv.hasNext();)
            m += neighborhoods.get(itv.next()).size();
        return m;
    }

    public boolean containsVertex(V vertex) {...}

    public boolean addVertex(V vertex) {
        ...
        neighborhoods.put(vertex, new HashSet<V>());
        ...
    }

    public boolean addEdge(V vertex1, V vertex2) throws NoSuchVertexException,
        LoopException {...}

    public boolean areNeighbors(V vertex1, V vertex2)
        throws NoSuchVertexException {...}

    public Iterator<V> vertices() {...}

    public Iterator<V> neighbors(V vertex) throws NoSuchVertexException {...}
}

```

```
    ...  
}
```

On considère maintenant la classe `Browser`.

**Question 6** Adapter la classe `Browser` à l'aide d'un adaptateur de classe `BrowserGraph` qui étend `Browser` et implémente `SimpleGraph`.

```
public class BrowserGraph extends Browser implements SimpleGraph<String> {  
  
    private String currentPage = null;  
  
    private SimpleGraph<String> g = new DefaultSimpleGraph<String>();  
  
    public BrowserGraph(String initialPage) {  
        super(initialPage);  
        g.addVertex(initialPage);  
        currentPage = initialPage;  
    }  
  
    protected void setPage(String url) {  
        super.setPage(url);  
        if (g != null) {  
            g.addVertex(url);  
            if (currentPage != null && !currentPage.equals(url))  
                try {  
                    g.addEdge(currentPage, url);  
                } catch (Exception e) {  
                    System.err.println(e.getMessage());  
                    throw new InternalError();  
                }  
        }  
    }  
  
    public int nVertices() {  
        return g.nVertices();  
    }  
  
    public int nEdges() {  
        return g.nEdges();  
    }  
  
    public boolean addEdge(String vertex1, String vertex2) {  
        throw new UnsupportedOperationException();  
    }  
}
```

```

    }

    public boolean addVertex(String vertex) {
        throw new UnsupportedOperationException();
    }

    public boolean areNeighbors(String vertex1, String vertex2)
        throws NoSuchVertexException {
        return g.areNeighbors(vertex1, vertex2);
    }

    public boolean containsVertex(String vertex) {
        return g.containsVertex(vertex);
    }

    public Iterator<String> vertices() {
        return g.vertices();
    }

    public Iterator<String> neighbors(String vertex)
        throws NoSuchVertexException {
        return g.neighbors(vertex);
    }
}

```

**Question 7** En utilisant la classe `java.lang.Observable` et l'interface `java.lang.Observer`, créer une classe `BrowserObservable` et une classe `GraphDrawingObserver` qui permettent d'attacher à un navigateur un ou plusieurs dessins de graphes permettant de visualiser la navigation.

*La solution proposée ici a l'avantage d'être très simple, mais elle a aussi des limites.*

```

public class BrowserObservable extends Observable {
    private BrowserGraph browser;

    public BrowserObservable(String initialPage) {
        this.browser = new BrowserGraph(initialPage) {
            protected void setPage(String url) {
                super.setPage(url);
                setChanged();
                notifyObservers();
            }
        };
    }
}

```

```
    public SimpleGraph<String> getGraph() {
        return browser;
    }
}

public class GraphDrawingObserver implements Observer {

    private GraphDrawing<String> drawing;

    public GraphDrawingObserver(BrowserObservable browser) {
        drawing = new GraphDrawing<String>(browser.getGraph());
        browser.addObserver(this);
    }

    public void update(Observable o, Object arg) {
        drawing.draw();
    }
}
```