Programmation Objet, UE INF906

Corrected copy

Master Informatique 1
Université Bordeaux 1
april 2008

Exercise - Iterator

In a class `Iterators`, write a static method `public static <E> Iterator<E> iteratorWithRepeat(Iterator<? extends E> it, int [] repeat)` which returns an iterator it'. it' will return the i^{th} element of `it`, repeated `repeat[`i`]` times. The method remove() will throw `UnsupportedOperationException`.

```
public static <T> Iterator<T> iteratorWithRepeat(final Iterator<T> it,
        final int[] repeat) {

        return new Iterator<T>() {
                int index = 0;
                int r = 0;
                T next = null;

                public boolean hasNext() {
                        return r == 0 ? it.hasNext() : true;
                }

                public T next() {
                        if (r == 0) {
                                next = it.next();
                                r = repeat[index++];
                        }
                        r--;
                        return next;
                }

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

Problem - Graphs

Question 1 Complete the methods `order()` and `size()` in the class `DefaultSimpleGraph`.

```
public int order() {
        return neighborhoods.size();
}

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

Question 2 Modify the class `DefaultSimpleGraph` in such a way that the instruction `System.out.println(g)`, where `g` is a graph with n vertices and m edges, will print on the standard output :
`Graph with` n `vertices and` m `edges.`

```
public String toString() {
        return "Graph with " + order() + " vertices and " + size() + " edges.";
}
```

Question 3 Modify the interface `SimpleGraph` and the class `DefaultSimpleGraph` in the following way :

- if we try to add a loop on a vertex v, by invoking `addEdge(v, v)`, the method will throw an exception `LoopException`;
- if we try to add an edge between two vertices $v1$ and $v2$,
  or test if the two vertices $v1$ and $v2$ are neighbors,
  or try to get an iterator on the neighbors of a vertex v,
  and v or $v1$ or $v2$ are not vertices of the graph,
  then the method `addEdge` or `areNeighbors` or `neighbors` will throw an exception `NoSuchVertexException`.

```
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 NoSuchVertexException {
        if (!containsVertex(vertex1))
                throw new NoSuchVertexException(vertex1);
        if (!containsVertex(vertex2))
                throw new NoSuchVertexException(vertex2);
        return neighborhoods.get(vertex1).contains(vertex2);
}

public Iterator<Object> neighbors(Object vertex) throws NoSuchVertexException {
        if (!containsVertex(vertex))
                throw new NoSuchVertexException(vertex);
        return neighborhoods.get(vertex).iterator();
}
```

Question 4 Give an implementation of the class `LoopException`.

```
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 Modify the interface `SimpleGraph` and the class `DefaultSimpleGraph` in such a way that the type of the vertices becomes generic.

```
public interface SimpleGraph<V> {
        public int nVertices();
        public int nEdges();
        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() {
                ...
                for (Iterator<?> itv = vertices(); itv.hasNext();)
                ...
        }

        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 {...}
        ...
}
```

We now consider the following class `Browser` which implements a very simple web browser.

Question 6 What is the problem encountered if we want directly to have an observable browser ?

In Java, multi-inheritance is forbidden. Thus, you cannot extends both the classes `Browser` *and* `Observable`*. Because an observable browser is mainly a browser, it seems more important to inherits from* `Browser`*. Unfortunatly, it will be necessary to invoke the method* `setChanged()` *from* `Observable` *which is protected.*

Question 7 To avoid this problem, we propose the following solution :

- create a class `ObservableBrowser` which extends Browser ;
- create in `ObservableBrowser` an inner class which extends `Observable` ;
- add to `ObservableBrowser` a method `Observable getObservable()` which returns an instance (always the same) of this inner class.

Implements the class `ObservableBrowser`.

```
public class ObservableBrowser extends Browser {

        private final InnerObservable observable = new InnerObservable();

        private class InnerObservable extends Observable {
                void notify(Object arg)  {
                        setChanged();
                        notifyObservers(arg);
                }
        }

        public ObservableBrowser(String initialPage) {
                super(initialPage);
        }

        public void setPage(String url) {
                super.setPage(url);
                observable.notify(url);
        }

        public final Observable getObservable() {
                return observable;
        }
}
```

Question 8 Propose a class `ObserverGraph` which allows to transform an instance of `SimpleGraph` to an observer of an instance of `ObservableBrowser`.

```
public class ObserverGraph implements Observer {

        private SimpleGraph<String> g;
        private ObservableBrowser browser;
        private String currentPage = null;

        public ObserverGraph(SimpleGraph<String> g, ObservableBrowser browser) {
                this.g = g;
                this.browser = browser;
                browser.getObservable().addObserver(this);
        }

        public void update(Observable o, Object arg) {
                if (!o.equals(browser))
                        return;
                String url = (String) arg;
                g.addVertex(url);
                if (currentPage != null)
                try {
                        g.addEdge(currentPage, url);
                } catch (NoSuchVertexException e) {
                        // This exception must not happen.
                        throw new Error(e.getMessage());
                } catch (LoopException e) {
                        // nothing to do.
                }
                currentPage = url;
        }
}
```