Programmation Objet, UE INF906

Master Informatique 1
Université Bordeaux 1
april 2008

No document, except a dictionary, length 3 hours

Any answer difficult to understand will be considered as false.

You may add any code (method, data, class) not requested each time you think it is necessary.

A class or a method requested may be considered available in the follow-up of the problem, even if you have not treated the question. It is recommanded to read all the subject before starting your work.

If you want to use a class (or a given method) from the Java API, and you don't remember its name, give to it an explicit name and explain clearly the goal of the class on your copy.

You may use, each time you think it is comprehensible, an abbreviation instead the complete name of a class or a method, and also use '...' to replace an unmodified part of the code.

The instructions `import` *and* `package` *will be not given.*

Exercise - Iterator

We recall the interface `Iterator`

```
public interface Iterator<E> {
    /** returns true if the iteration has more elements. */
    public boolean hasNext();
    /* returns the next element of the iteration. Throws
       NoSuchElementException if the iteration has no more element. */
    public E next();
    /* Removes from the underlying collection the last element
       returned by the iterator. Throws UnsupportedOperationException
       if the remove  operation is not supported by this Iterator. */
    public void remove();
}
```

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 `repetitions[`i`]` times. The method remove() will throw `UnsupportedOperationException`.

Example : the following code

```
Integer[] ti = { 1, 2, 3 };
int[] tr = { 2, 1, 2 };
Iterator<Integer> it = Iterators.iteratorWithRepeat(
    java.util.Arrays.asList(ti).iterator(), tr);
while (it.hasNext())
    System.out.print(it.next() + " ");
System.out.println();
```

will print on the standard output

```
1 1 2 3 3
```

Problem - Graphs

We recall the two interfaces `Set`

```
public interface Set<E> {
    // Adds the specified element to this set if it is not already present.
    boolean add(E e);
    // Adds all of the elements in the specified collection to this set if
    // they're not already present.
    boolean addAll(Collection<? extends E> c);
    // Removes all of the elements from this set.
    void clear();
    // Returns true if this set contains the specified element.
    boolean contains(Object o);
    // Returns true if this set contains all of the elements of the specified
    // collection.
    boolean containsAll(Collection<?> c);
    // Compares the specified object with this set for equality.
    boolean equals(Object o);
    // Returns the hash code value for this set.
    int hashCode();
    // Returns true if this set contains no elements.
    boolean isEmpty();
    // Returns an iterator over the elements in this set.
    Iterator<E> iterator();
    // Removes the specified element from this set if it is present.
    boolean remove(Object o);
    // Removes from this set all of its elements that are contained in the
    // specified collection.
    boolean removeAll(Collection<?> c);
    // Retains only the elements in this set that are contained in the
    // specified collection.
```

```
    boolean retainAll(Collection<?> c);
    // Returns the number of elements in this set (it s cardinality).
    int size();
    // Returns an array containing all of the elements in this set.
    Object[] toArray();
    // Returns an array containing all of the elements in this set;
    // the runtime type of the returned array is that of the specified array.
    <T> T[] toArray(T[] a);
}
```

and `Map`

```
public interface Map<K, V> {

    // A map entry (key-value pair).
    interface Entry<K,V> {
        // Compares the specified object with this entry for equality.
        boolean equals(Object o);
        // Returns the key corresponding to this entry.
        K getKey();
        // Returns the value corresponding to this entry.
        V getValue();
        // Returns the hash code value for this map entry.
        int hashCode();
        // Replaces the value corresponding to this entry with the
        // specified value.
        V setValue(V value);
    }

    // Removes all of the mappings from this map.
    void clear();
    // Returns true if this map contains a mapping for the specified key.
    boolean containsKey(Object key);
    // Returns true if this map maps one or more keys to the specified value.
    boolean containsValue(Object value);
    // Returns a Set view of the mappings contained in this map.
    Set<Map.Entry<K,V>> entrySet();
    // Compares the specified object with this map for equality.
    boolean equals(Object o);
    // Returns the value to which the specified key is mapped, or null if
    // this map contains no mapping for the key.
    V get(Object key);
    // Returns the hash code value for this map.
    int hashCode();
    // Returns true if this map contains no key-value mappings.
    boolean isEmpty();
```

```
    // Returns a Set view of the keys contained in this map.
    Set<K> keySet();
    // Associates the specified value with the specified key in this map.
    V put(K key, V value);
    // Copies all of the mappings from the specified map to this map.
    void putAll(Map<? extends K,? extends V> m);
    // Removes the mapping for a key from this map if it is present.
    V remove(Object key);
    // Returns the number of key-value mappings in this map.
    int size();
    // Returns a Collection view of the values contained in this map.
    Collection<V> values();
}
```

A *simple graph* is a graph without loops or multiple edges.
We propose the interface `SimpleGraph`

```
public interface SimpleGraph {
    // Returns the number of vertices
    int order();
    // Returns the number of edges
    int size();
    // Returns true if the graph contains the specified vertex.
    boolean containsVertex(Object vertex);
    // Adds the specified vertex to this graph if it is not already present.
    boolean addVertex(Object vertex);
    // Adds an edge between the two vertices, if they are not yet neighbors.
    // Loops (when vertex1 and vertex2 are equals) are forbidden.
    boolean addEdge(Object vertex1, Object vertex2);
    // Returns true if an edge have been added between the two specified
    // vertices.
    boolean areNeighbors(Object vertex1, Object vertex2);
    // Returns an iterator on the vertices of the graph.
    Iterator<Object> vertices();
    // Returns an iterator on the neighbors of the specified vertex.
    Iterator<Object> neighbors(Object vertex);
}
```

and its implementation `DefaultSimpleGraph`

```
public class DefaultSimpleGraph implements SimpleGraph {

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

    public int order() {
```

```
        // code omitted
    }

    public int size() {
        // code omitted
    }

    public boolean containsVertex(Object vertex) {
        return neighborhoods.containsKey(vertex);
    }

    public boolean addVertex(Object vertex) {
        if (neighborhoods.containsKey(vertex))
            return false;
        neighborhoods.put(vertex, new HashSet<Object>());
        return true;
    }

    public boolean addEdge(Object vertex1, Object vertex2) {
        if (!containsVertex(vertex1) || !containsVertex(vertex2)
            || vertex1.equals(vertex2)
            || 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) {
        return containsVertex(vertex1) && containsVertex(vertex2)
            && neighborhoods.get(vertex1).contains(vertex2);
    }

    public Iterator<Object> vertices() {
        return neighborhoods.keySet().iterator();
    }

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

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

Question 2 Modify the class `DefaultSimpleGraph` in such a way that the following 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.`

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`.

Question 4 Give an implementation of the class `LoopException`. *The class* `NoSuchVertexException` *is not requested.*

Question 5 Modify the interface `SimpleGraph` and the class `DefaultSimpleGraph` in such a way that the type of the vertices become generic.

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

```
public class Browser extends JFrame {
    private JEditorPane pane;

    /**
     * The constructor runs the browser. It displays the main frame with the
     * fetched initialPage
     *
     * @param initialPage
     *            the first page to show
     */
    public Browser(String initialPage) {
        pane = new JEditorPane();
        pane.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(pane);
        pane.addHyperlinkListener(new HyperlinkListener() {
                public void hyperlinkUpdate(HyperlinkEvent evt) {
```

```
                    if (evt.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
                        setPage(evt.getURL().toString());
                    }
                }
            });
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setContentPane(scrollPane);
        pack();
        setSize(960, 720);
        setVisible(true);
        setPage(initialPage);
    }

    protected void setPage(String url) {
        try {
            pane.setPage(url);
        } catch (Exception e) {
        }
    }

    /** Create a Browser object. Use the command-line url if given */
    public static void main(String[] args) {
        String initialPage = "http://www.google.fr";
        if (args.length > 0)
            initialPage = args[0];
        new Browser(initialPage);
    }
}
```

.

We want to be able to create one or more instances of graphs corresponding to the navigation. Each page loaded during the navigation will be represented by a vertex of the graph, and two vertices will be neighbors if we have used a link from one of the corresponding page to the other.

To obtain this result, we want use the model Observable/Observer.

We give here the methods of the class `Observable` you will have to use and the interface `Observer`.

```
public class Observable {
...
    public void addObserver(Observer o);
    public void notifyObservers(Object arg);
    protected void setChanged();
}
```

```
public interface Observer {
   public void update(Observable o, Object arg);
}
```

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

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`.

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