université de BORDEAUX

4TWE802U – Object Oriented Programming
Master Informatique
Pôle Universitaire Français

Final Exam, April, 27th 2019
No document except a dictionary, length 3h

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 recommended 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.

Exercice - Iterator

We recall the interface `Iterator` :

```
public interface Iterator<E> {

        /** Returns true if the iteration has more elements. */
        boolean hasNext();

        /** Returns the next element in the iteration.
        Throws NoSuchElementException if the iteration has no more elements*/
        E next();

        /** Removes from the underlying collection the last element returned by
        this iterator (optional operation). This method can be called only once
        per call to next(). It will throw an IllegalStateException if the next method
        has not yet been called, or the remove method has already been called
        after the last call to the next method.
        The behavior of an iterator is unspecified if the
        underlying collection is modified while the iteration is in progress in any
        way other than by calling this method.
        The default implementation throws an instance of
        UnsupportedOperationException and performs no other action. */
        default void remove();

        /** Performs the given action for each remaining element until all
        elements have been processed or the action throws an exception.
        The default implementation behaves as if:
        while (hasNext())
            action.accept(next());
```

```
         */
         default void forEachRemaining(Consumer<? super E> action);
}
```

Question 1 In a class `Iterators`, write a class method
`public static <E> Iterator<E> doubleIterator(Iterator<E> it1, Iterator<E> it2)`
which returns an iterator *it* which iterates first the elements returned by *it*1, then those returned by *it*2. The methode `remove` called on the iterator returned by *doubleIterator* will call the method `remove` on the last iterator *it*1 or *it*2 used. The method `forEachRemaining` will execute its default code.

Example : the following code

```
List<String> l1 = new ArrayList<>(Arrays.asList("a", "b", "c"));
List<String> l2 = new ArrayList<>(Arrays.asList("d"));
Iterator<String> it = Iterators.doubleIterator(l1.iterator(), l2.iterator());
while (it.hasNext()) {
    String s = it.next();
    System.out.print(s + " ");
    if (s.equals("b")) {
        it.remove();
    }
}
System.out.println();
it = Iterators.doubleIterator(l1.iterator(), l2.iterator());
while (it.hasNext()) {
    System.out.print(it.next() + " ");
}
System.out.println();
```

will display on the standard output

```
a b c d
a c d
```

Question 2 We want to generalize this method. For that, add in the class `Iterators` a class method `public static <E> Iterator<E> multipleIterator(Iterator<E>... iterators)` returning an iterator in which the method *next* will return one by one all the elements returned by each iterator given as parameter. As in the previous question, the method `remove` on the generated iterator will call the method `remove` on the last used iterator from the parameters. The method `forEachRemaining` will execute its default code.

Question 3 If we try to execute the following instructions :

```
Iterator<Integer> it1 = Arrays.asList(1, 2, 3).iterator();
Iterator<Double> it2 = Arrays.asList(4., 5., 6.).iterator();
Iterator<Number> it3 = doubleIterator(it1, it2);
```

the following error appears during compilation time :

```
The method doubleIterator(Iterator<E>, Iterator<E>) in the type Iterators is not
applicable for the arguments (Iterator<Integer>, Iterator<Double>)
```

Modify the signatures of the methods `doubleIterator` et `multipleIterator` to avoid this kind of problem.

Problem - Network

The objective of this problem is to create an interface which allows to optimize the choice of links between points chosen in advance. We first consider the class `Link` below :

```
public class Link {
    private Point p1, p2;

    public Link(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }

    public Point p1() {
        return p1;
    }

    public Point p2() {
        return p2;
    }

    public double length() {
        return Math.sqrt(Math.pow(p2.x - p1.x, 2) + Math.pow(p2.y - p1.y, 2));
    }
}
```

Question 1 Modify the class `Link` in such a way that the instruction

```
System.out.print(li);
```

where `li` is an instance of `Link`, write on the standard output the following result :
`Link between (`*x1*`, `*y1*`) and (`*x2*`, `*y2*`)`
where *x1*, *y1* are the coordinates of the point `p1` and *x2*, *y2* those of the point `p2` and `p1` and `p2` are the parameters of the constructor of `Link` used to create `li`.

Question 2 Modifier la classe `Link` in such a way that two instances of `Link` are equals if and only if they link two points with the same coordinates, whatever is the order of those two points. For example, after the following code :

```
Link l1 = new Link(new Point(0, 0), new Point(100, 100));
Link l2 = new Link(new Point(100, 100), new Point(0, 0));
```

`l1.equals(l2)` must return `true`.

Which other method inherited from the class `Object` is it necessary to redefine ? Give a solution (take care that the order of the two points is not used for the equality).

Now, we consider the following class `Network` below :

```
public class Network {

    private Set<Point> points = new HashSet<>();
    private Set<Link> links = new HashSet<>();
```

```
    public Network(Point... points) {
        for (Point p : points) {
            this.points.add(p);
        }
    }

    public Set<Point> points() {
        return Collections.unmodifiableSet(points);
    }

    public void addLink(Point p1, Point p2) {
        links.add(new Link(p1, p2));
    }

    public Set<Link> links() {
        return Collections.unmodifiableSet(links);
    }

    // Returns the link betweeen the points p1 and p2 if it exists,
    // null otherwise.
    public Link link(Point p1, Point p2) {
        for (Link l : links)
            if ((l.p1().equals(p1) && l.p2().equals(p2)) ||
                    (l.p2().equals(p1) && l.p1().equals(p2)))
                return l;
        return null;
    }
}
```

Question 3 We want that it becomes impossible to change the coordinates of the points associated to a network. But it is possible in the class `Point` to change the coordinates using the method `setLocation`. Modify the class `Network` to guarantee this encapsulation.

Question 4 We want that, if we try to add a link which yet exists between two points, an exception `ExistingLinkException` is thrown. Modify the class `Network` and give also the code of the class `ExistingLinkException`.

We consider now the two classes below `Editor` and `NetworkPanel`.

```
public class Editor extends JFrame {

    public Editor(Network network) {
        super("Editor");
        JFrame.setDefaultLookAndFeelDecorated(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        NetworkPanel networkPanel = new NetworkPanel(network);
        add(networkPanel, BorderLayout.CENTER);

        MouseListener ml = new MouseAdapter() {
            Point[] points = new Point[2];
            int index = 0;

            public void mouseClicked(MouseEvent arg0) {
                Point p = networkPanel.point(arg0.getPoint());
```

```
                if (p != null) {
                    points[index++] = p;
                    if (index == 2) {
                        network.addLink(points[0], points[1]);
                        networkPanel.repaint();
                        index = 0;
                    }
                }
            }
        };

        networkPanel.addMouseListener(ml);

        setPreferredSize(new Dimension(400, 400));
        pack();
        setVisible(true);
    }

    // Exemple of use of the class Editor
    public static void main(final String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                int[] coords = { 50, 50, 100, 100, 100, 50, 200, 200, 300, 200, 200, 300};
                Point[] points = new Point[coords.length / 2];
                for (int i = 0; i < points.length; i++) {
                    points[i] = new Point(coords[2 * i], coords[2 * i + 1]);
                }
                new Editor(new Network(points));
            }
        });
    }
}

public class NetworkPanel extends JPanel {

    private static final int POINT_SIZE = 2;
    private static final int EPSILON = 3;
    private Network network;

    public NetworkPanel(Network network) {
        setOpaque(true);
        setForeground(Color.black);
        setBackground(Color.white);
        this.network = network;
    }

    public Point point(Point p) {
        for (Point p2 : network.points())
            if (p.x >= p2.x - EPSILON && p.x <= p2.x + EPSILON && p.y >= p2.y - EPSILON && p.y <
                return p2;
        return null;
    }

    public void paintComponent(Graphics g) {
```

```
        if (isOpaque()) {
            g.setColor(getBackground());
            g.fillRect(0, 0, getWidth(), getHeight());
        }
        g.setColor(getForeground());
        for (Point p : network.points()) {
            g.drawLine(p.x - POINT_SIZE, p.y, p.x + POINT_SIZE, p.y);
            g.drawLine(p.x, p.y - POINT_SIZE, p.x, p.y + POINT_SIZE);
        }
        for (Link l : network.links())
            g.drawLine(l.p1().x, l.p1().y, l.p2().x, l.p2().y);
    }
}
```

Question 5 Explain in few sentences how an instance of `Editor` works.

Question 6 Modifiy the class `Editor` in such a way that, if we try to add two times the same link, the network is not modified and the application continue to work without any change.

Question 7 We consider now the interface `WeightedGraph` below.

```
// Graph with vertices of type V.
public interface WeightedGraph<V> {

    // Set of vertices of the graph.
    public Set<V> vertices();

    // Set of neighbors of a vertex v.
    public Set<V> neighbors(V v);

    // true if and only if v1 and v2 are neighbors.
    public boolean areNeighbors(V v1, V v2);

    // Weight of the edge between the vertices v1 and v2.
    // Throws an exception IllegalArgumentException if v1 and v2 are not neighbors.
    public double weight(V v1, V v2) throws IllegalArgumentException;
}
```

Create a class `NetworkGraph` which adapt the class `Network` to the interface `WeightedGraph`. The vertices of the graph will be the points of the network, the edges its links and the weight of an edge the length of a link (given by the method `length` of the class `Link`).

Question 8 We consider now that it exists in a class `Graphs` a class method `public static <V> WeightedGraph<V> minimumWeightedTree(WeightedGraph<V> g, V v)` which computes from a vertex `v` a tree with minimum weight from the graph `g` and containing all the vertices which belong to the connected component of `v`. Modify the class `Editor` in such a way that, each time you add a link, all the links which belong to such a tree from one of the two vertices of the new added link are drawn in red. We recall that the method `g.setColor(Color c)` allows to change the color used by the `Graphics g` and the existence of the two constants `Color.black` et `Color.red`.