Programmation Objet, UE INF906

Master Informatique 1
Université Bordeaux 1
april 2009
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();
}
```

Write a class `ConstantIterator<T>` which returns always the same value. This value will be given as parameter at the construction. The method `remove` will throw an `UnsupportedOperationException`.

Example : the following code

```
Iterator<String> it = new ConstantIterator<String>("Hello");
```

```
for (int i = 0; i < 10; i++)
    System.out.print(it.next() + " ");
System.out.println();
```

will print on the standard output

```
Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello
```

Add a second constructor to `ConstantIterator<T>` : `ConstantIterator(T value, int n)`. An instance created using this constructor will return always the same value *value*, but only n times.

Example : the following code

```
Iterator<String> it1 = new ConstantIterator<String>("Hello");
Iterator<String> it2 = new ConstantIterator<String>("World", 3);
while (it1.hasNext() && it2.hasNext())
    System.out.print(it1.next() + " " + it2.next() + " ");
System.out.println();
```

will print on the standard output

```
Hello World Hello World Hello World
```

Exercise - Point

We consider the two classes `Point2D` and `Point3D`.

```
public class Point2D {
    private int x;
    private int y;

    public Point2D(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int x() {
        return x;
    }

    public int y() {
        return y;
    }

    public double distance(Point2D p) {
```

```
        return Math.sqrt((p.x - x) * (p.x - x) + (p.y - y) * (p.y - y));
    }

    public boolean equals(Object o) {
        if (!(o instanceof Point2D))
            return false;
        Point2D p2 = (Point2D) o;
        return x == p2.x && y == p2.y;
    }
}

public class Point3D extends Point2D {
    private int z;

    public Point3D(int x, int y, int z) {
        super(x, y);
        this.z = z;
    }

    public int z() {
        return z;
    }

    public double distance(Point3D p) {
        return Math.sqrt((x() - p.x()) * (x() - p.x()) + (y() - p.y())
                         * (y() - p.y()) + (z() - p.z()) * (z() - p.z()));
    }

    public boolean equals(Object o) {
        if (!(o instanceof Point3D))
            return false;
        Point3D p2 = (Point3D) o;
        return super.equals(p2) && p2.z == z;
    }
}
```

Which rules must follow the method `equals` ?
Explain why the method `equals` from `Point3D` is not correct.
Propose a solution which respect the rules for `equals`.

Problem - Thermometer

We consider the interface `Thermometer` :

```
public interface Thermometer {
```

```
    public double getTemperature();
    public void setTemperature(double temperature);
}
```

Question 1 Add a class constant to the interface `Thermometer` which gives the value of the absolute zero in Celsius degree (= -273.15 °C).

Question 2 Propose a class `ThermometerImpl` which implements the interface `Thermometer`.

Question 3 Modify the class `ThermometerImpl` and the interface `Thermometer` in such a way that trying to set a temperature below the absolute zero will throw an exception `ImpossibleTemperatureException` .

Question 4 Give an implementation of `ImpossibleTemperatureException`.

We give here the methods of the class `Observable` you will have to use in the next questions 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 5 Write a class `ObservableThermometer` for which instances are both instances of `Observable` and `Thermometer`. An instance of `ObservableThermometer` will notify to its observers every change of temperature. The new value v for the temperature will be given as parameter to `notifyObserver`.

Note that to transform a primitive double value v into an object, you must use the expression `new Double(`v`)`, and in the other way, to get the double value of an instance of `Double` d, you must use `d.doubleValue()`.

We consider now the class `BoundedValue` :

```
public class BoundedValue extends JComponent {

    private double min;
    private double max;
```

```
    private double value;

    public BoundedValue(double min, double max, double initialValue) {
        super();
        this.min = min;
        this.max = max;
        setValue(initialValue);
    }

    public double getMin() {
        return min;
    }

    public double getMax() {
        return max;
    }

    public double getValue() {
        return value;
    }

    public void setValue(double value) {
        if (value < min)
            this.value = min;
        else if (value > max)
            this.value = max;
        else
            this.value = value;
        repaint();
    }

    public void paintComponent(Graphics g) {
        int w = getWidth();
        int h = getHeight();
        int hValue = (int) ((value - min) / (max - min) * h);
        g.setColor(Color.RED);
        g.fillRect(0, h - hValue, w, h);
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, w, h - hValue);
    }
}
```

Question 6 Explain in two or three sentences what the class `BoundedValue` is doing.

We consider now the class `ThermometerFrame` :

```
public class ThermometerFrame extends JFrame {

    private BoundedValueObserver bvo;
    private final static double INITIAL_VALUE = 10;

    public ThermometerFrame() {
        bvo = new BoundedValueObserver(-30, 50, INITIAL_VALUE);
        setContentPane(bvo);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setPreferredSize(new Dimension(50, 300));
        pack();
        setVisible(true);
    }

    public static void main(String [] args) throws ImpossibleTemperatureException,
                                                   InterruptedException {
        ThermometerFrame tf = new ThermometerFrame();
        ObservableThermometer ot = new ObservableThermometer(INITIAL_VALUE);
        ot.addObserver(tf.bvo);
        Random r = new Random();
        for (;;) {
            int i = r.nextInt(3);
            ot.setTemperature(ot.getTemperature() + i - 1);
            Thread.sleep(1000);
        }
    }
}
```

Question 7 Give an implementation for the class `BoundedValueObserver`, used in `ThermometerFrame`.