Programmation Objet, UE INF906

Corrected copy

Master Informatique 1
Université Bordeaux 1
april 2009

Exercise - Iterator

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`.

```
public class ConstantIterator<T> implements Iterator<T> {
    private T value;

    public ConstantIterator(T value) {
        this.value = value;
    }

    public boolean hasNext() {
        return true;
    }

    public T next() {
        return value;
    }

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

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.

```
public class ConstantIterator<T> implements Iterator<T> {
    private int times;
    private boolean infinite;
    private T value;
```

```
    public ConstantIterator(T value) {
        this.value = value;
        infinite = true;
    }

    public ConstantIterator(T value, int times) {
        this.value = value;
        infinite = false;
        this.times = times;
    }

    public boolean hasNext() {
        return infinite || times > 0;
    }

    public T next() {
        if (infinite || times-- > 0)
            return value;
        throw new NoSuchElementException();
    }

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

Exercise - Point

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

Which rules must follow the method `equals` ?

`equals` *must be :*

- *reflexive :* `o.equals(o)` *is always true ;*
- *symmetric :* `o1.equals(o2)` *and* `o2.equals(o1)` *must have the same value ;*
- *transitive : if* `o1.equals(o2)` *and* `o2.equals(o3)` *are true, then* `o1.equals(o3)` *must be true ;*
- *consistant : if* `o1.equals(o2)` *is true at some moment m, and neither* `o1` *or* `o2` *have changed between m and a next moment m', then* `o1.equals(o2)` *remains true.*

Explain why the method `equals` from `Point3D` is not correct.

The method `equals` *from* `Point3D` *is not symmetric. For example, the following code :*

```
Point2D p2d = new Point2D(10, 10);
Point3D p3d = new Point3D(10, 10, 10);
System.out.println(p2d.equals(p3d) + " " + p3d.equals(p2d));
```

will write on the standard output

```
true false
```

Propose a solution which respect the rules for `equals`.

The class `Point3D` *must use delegation instead of inheritance to reuse the code of* `Point2D`*. The right code for* `Point3D` *is the following :*

```
public class Point3D {
    private Point2D delegate;
    private int z;

    public Point3D(int x, int y, int z) {
        delegate = new Point2D(x, y);
        this.z = z;
    }

    public int x() {
        return delegate.x();
    }

    public int y() {
        return delegate.y();
    }

    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 delegate.equals(p2.delegate) && p2.z == z;
```

```
    }
}
```

Problem - Thermometer

We consider the interface `Thermometer`

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

```
public interface Thermometer {
    /** Absolute zero in Celsius degree */
    public static final double ABSOLUTE_ZERO = -273.15;
    ...
}
```

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

```
public class ThermometerImpl implements Thermometer {
    double temperature;

    public ThermometerImpl(double initialTemperature) {
        setTemperature(initialTemperature);
    }

    public double getTemperature() {
        return temperature;
    }

    public void setTemperature(double temperature) {
        this.temperature = temperature;
    }
}
```

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` .

```
public interface Thermometer {
    ...
    public void setTemperature(double temperature)
    throws ImpossibleTemperatureException;
}

public class ThermometerImpl implements Thermometer {
    ...
    public ThermometerImpl(double initialTemperature)
    throws ImpossibleTemperatureException {
    setTemperature(initialTemperature);
    }
    ...
    public void setTemperature(double temperature)
    throws ImpossibleTemperatureException {
    if (temperature < Thermometer.ABSOLUTE_ZERO)
        throw new ImpossibleTemperatureException(temperature);
    this.temperature = temperature;
    }
}
```

Question 4 Give an implementation of `ImpossibleTemperatureException`.

```
public class ImpossibleTemperatureException extends Exception {
    private double temperature;

    public ImpossibleTemperatureException(double temperature) {
        this.temperature = temperature;
    }

    public double temperature() {
        return temperature;
    }

    public String getMessage() {
        return "temperature " + temperature + " impossible";
    }
}
```

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`.

```
public class ObservableThermometer extends Observable implements Thermometer {
    private final Thermometer thermometer;

    public ObservableThermometer(double initialTemperature)
            throws ImpossibleTemperatureException {
        super();
        thermometer = new ThermometerImpl(initialTemperature);
    }

    public double getTemperature() {
        return thermometer.getTemperature();
    }

    public void setTemperature(double temperature)
            throws ImpossibleTemperatureException {
        thermometer.setTemperature(temperature);
        setChanged();
        notifyObservers(temperature);
    }
}
```

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

`BoundedValue` *draw a rectangle with two parts : one in white (top) and one in red (bottom). The red part of the rectangle is proportional to the total height of the rectangle with same ratio than* `(value - min)/(max - min)`.

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

```
public class BoundedValueObserver extends BoundedValue implements Observer {
    public BoundedValueObserver(double min, double max, double initialValue) {
        super(min, max, initialValue);
    }

    public void update(Observable o, Object arg) {
        setValue((Double) arg);
    }
}
```