Interface des Programmes d'Application, UE INF302

Master Informatique 1
Université Bordeaux 1
Vendredi 18 mai 2007
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 recommended to read the whole exam 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 it an explicit name and explain clearly the goal of the class on your answer paper.

You may use, each time you think it is comprehensible, an abbreviation instead of 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 not be 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> append(Iterator<? extends E> it1, Iterator<? extends E> it2)` which returns an iterator which is the concatenation of it1 and it2. The method remove() will throw UnsupportedOperationException.

Problem - Vehicle

We consider the interface `Vehicle` :

```
public interface Vehicle {
    /** Give the speed of the vehicle in km/h */
    public double getSpeed();
    /** Give the direction of the vehicle in radians,
        relativly to the x-axis */
    public double getDirection();
    /** Give the current position of the vehicle */
    public Point2D getPosition();
    /** Runs the vehicule during a time given in hour.
        This method will change the position of the vehicule,
        following its speed and its direction. */
    public void run(double time);
}
```

and a default implementation of it `DefaultVehicle` :

```
public class DefaultVehicle implements Vehicle {

    protected double maxSpeed;
    protected double speed = 0;
    protected double direction = 0;
    protected Point2D position = new Point2D.Double(0., 0.);

    private void assertPositiveSpeed(double speed) {
        if (maxSpeed < 0) {
            throw new IllegalArgumentException("Negative Speed not available");
        }
    }

    public DefaultVehicle(double maxSpeed) {
        assertPositiveSpeed(maxSpeed);
        this.maxSpeed = maxSpeed;
    }

    public void setDirection(double direction) {
        this.direction = direction % (2 * Math.PI);
    }

    public void setSpeed(double speed) {
        assertPositiveSpeed(speed);
        this.speed = Math.min(maxSpeed, speed);
    }
```

```
    public double getDirection() {
        return direction;
    }

    public double getSpeed() {
        return speed;
    }

    public double getMaxSpeed() {
        return maxSpeed;
    }

    public Point2D getPosition() {
        return position;
    }

    public void run(double time) {
        double distance = time * speed;
        double x = Math.cos(direction) * distance;
        double y = Math.sin(direction) * distance;
        position.setLocation(position.getX() + x, position.getY() + y);
    }
}
```

You will also need to use the following methods of `java.awt.geom.Point2D` :

```
double getX();
double getY();
void setLocation(double x, double y);
```

which respectively returns the x-coordinate, the y-coordinate of a point, and changes the coordinates of a point. `Point2D.Double` is a static internal class of `Point2D` which provides an implementation of it.

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

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

where `v` is an instance of `DefaultVehicle`, will print, on the standard output, the following result :
`Vehicle (`x`, `y`), speed = `s`, direction = `d
where x and y are the coordinates of the position of the vehicle and s and d its speed and its direction.

Now, we want to have a new class `VehicleWithAccelerator` which implements `Vehicle` and contains a new method :

```
/** Change the speed of the vehicle */
void accelerate(double acceleration) { ... }
```

Question 2 Give an implementation of `VehicleWithAccelerator` using inheritance from `DefaultVehicle`. Complete the method `accelerate`. The speed will grow to the value of the parameter `acceleration`. This parameter may be positive or negative. The speed cannot be less than 0 and more than the maximum speed.

Question 3 Give another implementation of `VehicleWithAccelerator` without using inheritance from `DefaultVehicle`, but delegation. In this new implementation, it will be impossible to set the speed to a given value using `setSpeed`, the speed will change olny using `accelerate`. Complete the method `accelerate` in a similar way to that suggested in the previous question.

We now consider the interface Area

```
public interface Area {

    /** return true if the point belong to the area. */
    public boolean contains(Point2D p);

    /** return true if and only if it is possible to go
        from p1 to p2 using a straight line.
        throw an IllegalArgumentException if p1 or p2 are not
        in the area. */
    public boolean areConnected(Point2D p1, Point2D p2);
}
```

Question 4 Implement a class `RectangleArea` which implements `Area`. The area will be composed of the points of a rectangle. The top-left coordinate, the width and the height of the Rectangle will be given at the construction.

Question 5 Create a class `AreaWithVehicle` which is composed of an area and a vehicle, given at the construction. This class will contain methods which allow to move the vehicle inside the area. If the vehicle tries to go outside the area, an exception `AccidentException` will be thrown.

Question 6 Propose an implementation for the exception `AccidentException`.

The problem with the current implementation is that it is possible to move the vehicle without using methods from `AreaWithVehicle`, but directly those of the `Vehicle`. In that case, an accident may not be detected. In order to avoid this problem, we will use a new approach.

We want a vehicle to be `Observable`. Each time you call 'run' on this vehicle, the observers will be notified that the vehicle has moved.

Here, we give 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 7 Implement a class `ObservableVehicle` which extends `Observable` and implements `Vehicle`. Use the class `VehicleWithAccelerator` by delegation to implement the methods of `Vehicle`.

Each time an observable vehicle moves, its observers must be notified. Make sure that the method `setChanged()` is always called before using `notifyObservers`.

Question 8 Propose a new version of the class `AreaWithVehicle` which will be constructed using an instance of `ObservableVehicle`. The vehicle will be moved directly (using its own reference) and not through the area.