OU4 - Extra Exercise 4

ID1018

October 22, 2014

Interfaces and type independent programming

An abstract model of a polyline

A planar polyline

A polyline is a geometrical figure consisting of a series of connected line segments (edges). The endpoints of these line segments are the vertices of the polyline. A polyline is defined by its vertices, its colour, and its width. An empty polyline has no edges.

The vertices, colour, width and length of a polyline can be obtained. Colour and width can be modified. The shape of a polyline changes when its sequence of vertices is modified. One can add a new vertex to the polyline — either at the end or in front of a named vertex. It is also possible to remove a named vertex.

One can iterate over a polyline; the vertices can be visited and used in sequence.

An abstract model of a polyline is to be created; an interface class called Polyline.

A model of a polyline

It is assumed that there exists a class `Point`, which in a suitable way represents a planar point. Instances of class `Point` are to be used to represent the vertices of the polyline.

```
public interface Polyline extends java.lang.Iterable<Point>
{
    Point[]     getVertices();

    String      getColour();

    int         getWidth();

    double      length();

    void setColour (String colour);

    void setWidth (int width);

    void add (Point vertex);

    void insertBefore (Point vertex, String vertexName);

    void remove (String vertexName);

    java.util.Iterator<Point> iterator();
}
```

Exercises on polylines

1. Create a class `VPolyline` that represents a planar polyline and implements the `Polyline` interface. In addition to the methods specified in the interface, the method `toString` (that returns the string representation of the line) is to be implemented. The vertices in the polyline are to be stored in a vector of the built-in type.

2. Create a class `NPolyline` that represents a planar polyline and implements the `Polyline` interface. In addition to the methods specified in the interface, the method `toString` (that returns the string representation of the line) is to be implemented. The vertices in the polyline are to be stored in a sequence of linked nodes.

The class `NPolyline` is to begin like this:

```
public class NPolyline implements Polyline
{
  private static class Node
  {
    public Point vertex;
    public Node  nextNode;

    public Node (Point vertex)
    {
      this.vertex = vertex;
      nextNode = null;
    }
  }

  private Node   vertices;
  private String colour = "black";
  private int    width = 1; // pixels

  public NPolyline ()
  {
    this.vertices = null;
  }

  public NPolyline (Point [] vertices)
  {
    if (vertices.length > 0)
    {
      Node node = new Node (new Point (vertices[0]));
      this.vertices = node;
      int pos = 1;
      while (pos < vertices.length)
      {
        node.nextNode = new Node (new Point (vertices[pos++]));
        node = node.nextNode;
      }
    }
  }

  // *** MORE CODE HERE ***
}
```

3. Draw an object of type `NPolyline`. The object's sequence of nodes (with the corresponding vertices) is to be included in the drawing.

4. Create a common test program for the classes `VPolyline` and `NPolyline`. A reference to the interface `Polyline` is to be used to refer to instances of the implemented classes and to call their methods. One can use the following strategy:

```
Polyline polyline = null;
polyline = new VPolyline (); // (1)
// polyline = new NPolyline (); // (2)
```

Depending on the class to be tested line (1) or (2) are commented out accordingly.

5. You can iterate over a polyline like this:

```
for (Point vertex : polyline)
  System.out.println (vertex);
```

Why is this possible?

6. Create a static method in a separate class called `Polylines`, that accepts a vector of `Polyline`, and returns the shortest of the polylines in the vector that are yellow.

Use this method three times: with a vector of `VPolyline`, with a vector of `NPolyline`, and with a vector containing both types.

How can a vector contain objects of different types? How can one and the same method accept instances of different types?