OU5 - Mandatory Exercise 5

ID1018

December 11, 2014

Create new object types

A model of a polyline — vertices stored in a vector

A) A model of a planar point

A planar point (a point on a two-dimensional plane) has a name and coordinates. The coordinates are integer numbers.

A point can be defined by a name and two coordinates. It is also possible to define a point relative another point — a copy can be made.

It is also possible to create a character string that represents a point. This string could be on the form `(A 3 4)`. The name and coordinates of a point can be obtained. The coordinates can be modified, one coordinate at the time. The distance between two points can be computed. Two points can be compared for equality.

A model of a planar point is to be created; you are to create a definition class called `Point`.

A simple test program for class Point

```
import java.io.*;    // PrintWriter

class PointTest
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        // test a constructor and a transformer
        Point    p1 = new Point ("A", 3, 4);
        Point    p2 = new Point ("B", 5, 6);
        out.println (p1 + "    " + p2);
```

```
        // test inspectors
        String    n = p1.getName ();
        int    x = p1.getX ();
        int    y = p1.getY ();
        out.println (n + " " + x + " " + y);

        // test a combiner and a comparator
        double    d = p1.distance (p2);
        out.println (d);
        boolean    b = p1.equals (p2);
        out.println (b);

        // test mutators
        p2.setX (1);
        p2.setY (2);
        out.println (p2);

        // test another constructor
        Point    p = new Point (p1);
        out.println (p);
    }
}
```

Exercises on planar points

1. Create a definition class called `Point`, representing a point in the plane. Run the test program `PointTest` on the created class.

2. Create an image that shows how the distance between two planar points is calculated. Relate the image to the code in the corresponding method; insert the name of the variables in the image.

B) 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 can be depicted like in figure 1.

A polyline is defined by its vertices, its colour and its width. An empty polyline has no edges.

One can create a character string that represents a polyline. This string can be on the form {[(A 3 4)(B 1 2)(C 2 3)(D 5 1)], black, 1}. 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.

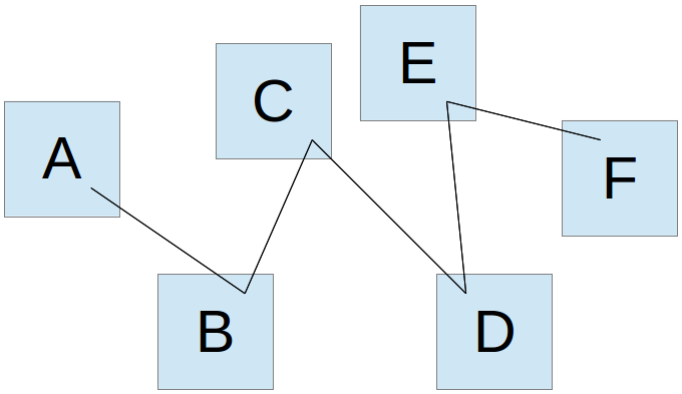

Figure 1: A planar polyline

A model of a polyline is to be created; a definition class called `Polyline`.

A model of a polyline — incomplete

```
public class Polyline
{
    private Point[]    vertices;
    private String     colour = "black";
    private int        width = 1;

    public Polyline ()
    {
        this.vertices = new Point[0];
    }

    public Polyline (Point[] vertices)
    {
        this.vertices = new Point[vertices.length];
        for (int i = 0; i < vertices.length; i++)
            this.vertices[i] = new Point (vertices[i]);
    }

    public String toString () {}

    public Point[] getVertices () {}

    public String getColour () {}

    public int getWidth () {}

    public void setColour (String colour) {}

    public void setWidth (int width) {}

    public double length () {}
```

```
    public void addLast (Point vertex)
    {
        Point[]    h = new Point[this.vertices.length + 1];
        int    i = 0;
        for (i = 0; i < this.vertices.length; i++)
            h[i] = this.vertices[i];
        h[i] = new Point (vertex);

        this.vertices = h;
    }

    public void addBefore (Point vertex, String vertexName) {}

    public void remove (String vertexName) {}
}
```

Exercises on polylines

1. Make complete the definition class `Polyline`. Implement the methods and comment its members. In the method `addBefore`, the vertex provided by the parameter is to be *copied*. In the method `getVertices`, a vector containing *copies* of the polyline's vertices is to be created, and a reference to the new vector returned.

2. Create another definition class, `Polyline1`. In this class it is the *references* to parameter objects that are to be copied, instead of the resources referred to. In the method `getVertices` it is the references to the own vertices that are returned.

Which strategy is the better one? To copy resources (as in `Polyline`) or to copy references (as in `Polyline1`)?

3. Create a simple test program, `PolylineTest`, that uses the constructors and methods in class `Polyline`.

4. Create an image that represents an object of type `Polyline`. Among other things, the image shall include the object and its references, the referred vector and *its* references, and the vertices that the vector refers to. The image and its components shall be labelled correctly.

5. Visualize the algorithm that is used in method `addBefore`. Draw a series of pictures that shows how the supplied vertex eventually is inserted in the correct place in the polyline. The images shall be labelled correctly.

6. An object of type `Polyline` has its own vector where the vertices are stored. Is this a memory-efficient strategy? What happens if vertices are added or removed frequently?

C) Create and use polylines

In a program called `SelectPolyline`, you create and use polylines of type `Polyline`.

A number of random polylines are created and presented. Each polyline has a random number of vertices and the colour is either blue, red or yellow. The vertices are named by single, uppercase letters from the english alphabet (A–Z). The names are chosen randomly but two vertices can not have the same name. The program determines and shows the shortest of the yellow polylines.

The program `SelectPolyline` has the following structure:

```
class SelectPolyline
{
    public static final Random    rand = new Random ();
    public static final int       NOF_POLYLINES = 10;

    public static void main (String[] args)
    {
        // Create a random number of polylines
        Polyline[]    polylines = new Polyline[NOF_POLYLINES];
        for (int i = 0; i < NOF_POLYLINES; i++)
            polylines[i] = randomPolyline ();

        // Show the polylines

        // Determine the shortest yellow polyline

        // Show the selected polyline
    }

    // The randomPoint method returns a new Point with a name
    // randomly chosen from the single letters A--Z. Coordinates
    // are random.
    public static Point randomPoint ()
    {
        String    n = "" + (char) (65 + rand.nextInt (26));
        int    x = rand.nextInt (11);
        int    y = rand.nextInt (11);

        return new Point (n, x, y);
    }

    // The method randomPolyline returns a random polyline,
    // with a colour either blue, red, or yellow. The names
    // of the vertices are single letters from the set A--Z.
    // Two vertices can not have the same name.
    public static Polyline randomPolyline ()
    {
        // Create an empty polyline and add vertices
        Polyline    polyline = new Polyline ();
        int    nofVertices = 2 + rand.nextInt (7);
```

```
        int    nofSelectedVertices = 0;
        boolean[]    selectedNames = new boolean[26];

        // Two vertices can not have the same name
        Point    chosenPoint = null;
        char     chosenChar = 0;
        while (nofSelectedVertices < nofVertices)
        {

        }

        // Assign a colour
    }
}
```

Exercise on SelectPolyline

Make complete and test the program `SelectPolyline`. What happens if there is no yellow line?

D) An iterator for a polyline

An iterator for a polyline can be defined and implemented. With such an iterator the vertices in the polyline can be visited in sequence.

An iterator for a polyline can be defined and implemented as an inner class in the class Polyline. This class can be named `PolylineIterator`, and look like this:

```
public class PolylineIterator
{
    private int    current = -1;

    public PolylineIterator ()
    {
        if (Polyline.this.vertices.length > 0)
            current = 0;
    }

    public boolean hasVertex ()
    {
        return current != -1;
    }

    public Point vertex ()
      throws java.util.NoSuchElementException
    {
        if (!this.hasVertex ())
            throw new java.util.NoSuchElementException (
```

```
                                        "end of iteration");

        Point    vertex = Polyline.this.vertices[current];

        return vertex;
    }

    public void advance ()
    {
        if (current >= 0  &&
            current < Polyline.this.vertices.length - 1)
            current++;
        else
            current = -1;
    }
}
```

Exercises on iterators

1. Create an iterator (an instance of class `PolylineIterator`) in the test program for class `Polyline`. With this iterator, visit each vertex in sequence and print them.

2. Visualize an iteration through a polyline. Follow the changes in the iterator object.