Your program should create the six triangles defining the footprint, and then add them to an arraylist.

3. Write a method `toString()` to create a string representation of the building footprint.

4. Within **Footprint**, write a method called `area()` that will compute the building area by walking along the arraylist and summing the triangle areas.

5. Finally, write methods getCentroidX() and getCentroidY() to compute the x- and y- coordinates of the building centroid.

`Note.` For parts 4 and 5, most of what you need is already defined in Triangle.java.

11.4 A polyline defines a set of one or more connected straight line segments. Polyline abstractions can be found in many areas of Civil Engineering (e.g., road trajectories in transportation, the orange line on the DC Metro, rebar trajectories in structural engineering). As illustrated by these examples, polyline elements typically define open shapes.

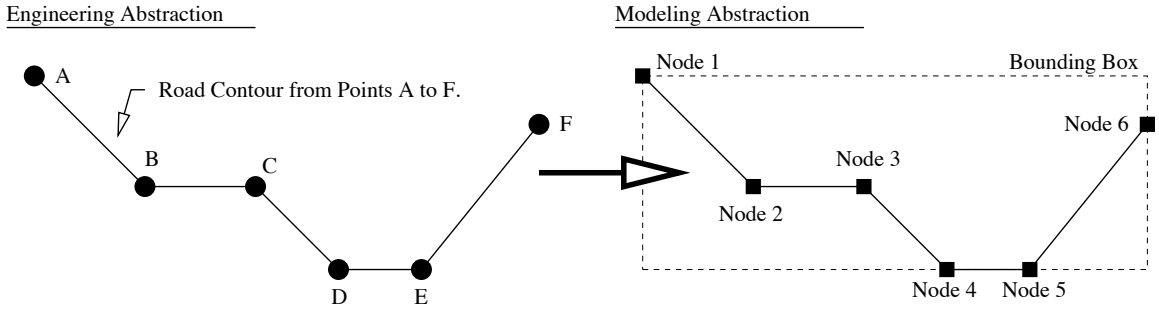

Figure 11.22. Real world and modeling abstractions for polylines.

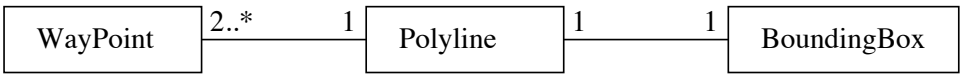

Figure 11.23. Polyline class diagram.

Modeling Polylines. Figure 11.23 shows the class diagram for a simplified implementation of polylines. A polyline can be modeled as an array list of (at least two) way points, and will store references to the source (first) and destination (last) way points. The polyline is said to be closed when the first and last way points have the same (x,y) coordinates.

Each way point will store the (x,y) coordinate of the point, plus the distance of the point from the source and the distance of the point from the destination. A bounding box covers the overall dimensions of the polyline and is a useful tool for simplify intersection computations.

Things to Do. Develop and test three classes: Polyline, BoundingBox and WayPoint. A way point will store the (x,y) coordinates plus additional information on disances to the source and destination nodes (see details below). A bounding box can be modeled with two pairs of coordinates, (xmin, ymin) and (xmax, ymax).

I suggest that you store the polyline as an arraylist of way points. You can save time by implementing WayPoint as an extension of Node2D (see the code in java-code-basics/src/geometry).

You will also need a test program to define an empty Polyline, systematically add way points to the model, and process the model to compute factors such as the total length and distance of individual way points from the source and destination nodes. The bounding box parameters should be updated when each new way point is added to the model.

The abbreviated output from my test program is:

```
geom01:
     [java] Run geometry.TestPolyline01() ...
     [java] ===================================  ...
     [java]
     [java] --- geometry.Polyline() ...
     [java] --- ==========================================
     [java] --- Name = Greenbelt Road
     [java] --- Is closed?   = false ...
     [java] --- Source         = Node A
     [java] --- Destination    = Node F
     [java] --- Pathway length = 694.974747
     [java] --- boundingbox[ x = (  0.00, 550.00), y = (  0.00, 200.00) ] ...
     [java] --- ==========================================
     [java]
     [java] WayPoint(Node A) ...
     [java] --- (x,y) = (0.000, 200.000) ...
     [java] --- distance from source      = 0.00 ...
     [java] --- distance from destination = 694.97 ...
     [java]
     [java] WayPoint(Node B) ...
     [java] --- (x,y) = (100.000, 100.000) ...
     [java] --- distance from source      = 141.42 ...
     [java] --- distance from destination = 553.55 ...
     [java]

     ... lines of output removed ...

     [java]
     [java] WayPoint(Node F) ...
     [java] --- (x,y) = (550.000, 150.000) ...
     [java] --- distance from source      = 694.97 ...
     [java] --- distance from destination = 0.00 ...
     [java]
     [java] ===================================  ...
     [java] Done! ...

BUILD SUCCESSFUL
Total time: 0 seconds
```

11.5 Now suppose that we wanted write a program where an object (e.g., a bug, a drone) traverses the pathway at a pre-defined velocity. We would need to know the position and orientation of the object as a function of time. Extend the capabilities of the previous question (see Figures 11.22 and 11.23) with two new methods: (1) compute the (x,y) coordinate of a point a distance d from the source, and (2) compute the slope of the pathway at a distance d from the source.