Angles and Directions: An Introduction to JTS Warped

By Landon Blake

Introduction

This article provides an introduction to the JTS Warped software library. It begins with a brief discussion of the JTS Topology Suite (JTS) and explains what functionality JTS Warped adds to the JTS Topology Suite. It then provides an overview of the code in JTS Warped that allows programmers to easily work with angles and bearings in JTS. It concludes with some code examples.

What Is JTS?

JTS is a software library and set of tools that support geometry calculations on the 2D Cartesian plane. JTS conforms to the Simple Features Specification for SQL published by the OGC. JTS strives to be (1) fast, (2) robust, and (3) implemented in pure Java. JTS was originally written by Martin Davis while at Vivid Solutions. He still maintains the library although he is no longer working for the company. JTS is used by popular geospatial software written in Java, and is the geometry library used by the open source desktop GIS OpenJUMP.

What is JTS Warped?

JTS Warped is a library of utility code related to JTS. It adds to the functionality of JTS but is not included in the main JTS distribution. JTS Warped is written by Landon Blake. The code for JTS is released under the GPL and is managed by the SurveyOS Project, a member of the Free Software Conservancy. JTS Warped is currently in Alpha status. You can download the source code, via SVN, for JTS Warped from here:

https://surveyos.svn.sourceforge.net/svnroot/surveyos

In this article we are going to focus our discussion on the code in JTS Warped that allows programmers to work with angles and bearings in JTS.

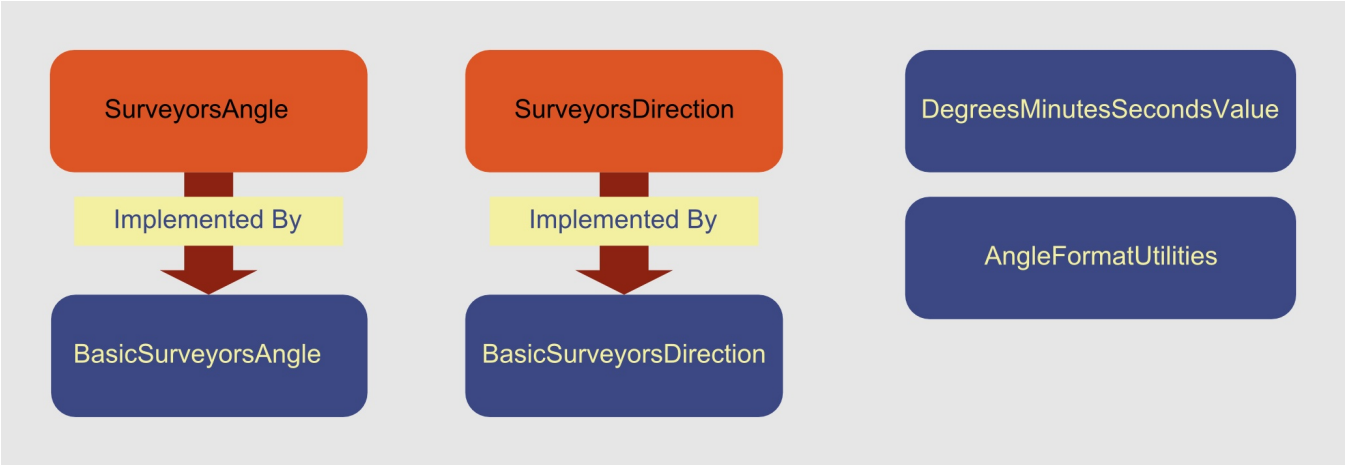
Overview of Angles and Directions Code in JTS Warped

Land surveyors frequently work with

angles and directions when moving data between 2D grid (also known as 2D Cartesian or rectangular coordinate systems) and polar coordinate systems.

Here is one example: Optical instruments used in terrestrial surveying employ polar coordinate systems to collect their measurements. In a polar coordinate system points are located in relation to the instrument point. This involves two measurements. The first is the measurement of the angle between the instrument point, a backsight point and the point being located (foresight point). The second is the distance between the instrument point and the point being located. Software is then used to transform the collected measurements in to point coordinates on the project grid coordinate system.

Here is another example: Descriptions for land parcel boundaries often use bearings and distances to describe the parcel geometry. It is often useful to convert this bearing and distance information into 2D vector geometry in a 2D grid coordinate system.
The angles and directions code in JTS Warped was designed to make it easier to perform the transformations between polar coordinate systems and 2D grid coordinate systems. The angles and directions code has only two (2) key classes used to implement this design. The BasicSurveyorsAngle class represents angular measurements in a polar coordinate system, and the BasicSurveyorsDirection class represents direction measurements in



Class Diagram

a polar coordinate system. Both classes provide basic implementations of corresponding Java interfaces, which allow the library to support alternate implementations of the interfaces if desired.

The Basic Surveyors Angle Class

The BasicSurveyorsAngle is essentially a value class. It represents an angle measurement by storing the value of the angle in degrees, minutes, and seconds. The degrees value, minutes value, and seconds value are passed in and out of the class as ints. The fractional seconds value is passed in and out of the class as a double. As an alternative, you can wrap these values in a DegreesMinutesSecondsValue object for passage in and out of the BasicSurveyorsAngle methods.

There are four (4) ways to create a BasicSurveyorsAngle class. The default, no-argument constructor will create an BasicSurveyorsAngle class with a degree value of 0, a minutes value of 0, a whole seconds value of zero, and a fractional seconds value of 0. A second constructor allows you to pass these values as three ints and a single double. A third constructor sets the value during construction using an instance of the DegreesMinutesSecondsValue passed as an argument. The final constructor is a “copy constructor” which accepts another instance of SurveyorsAngle as an argument. The value of this argument is used to set the value of the new angle. (This constructor is a tool for cloning angle values.)

Code Listing #1 shows examples of how to create instances of the BasicSurveyorsAngle class.

The BasicSurveyorsAngle class is immutable. You can obtain its degree, minute, second, and fractional second components using traditional getter methods of the class. The decrease and increase methods allow you to increase or decrease the value of an angle by applying other angles as a rotation. These methods do not modify the subject angle, but create a clone, modify and return it. The value of the angle object can also be obtained in radians or decimal degrees through convenience methods. Convenience methods are also included to return the trigonometric ratios for the sins, cosins, and tangent of the angle. These methods handle the internal conversions of the angle values to and from radians so the trigonometric ratios can be calculated. Three (3) additional

methods provide information about the type of angle. These are the isAcute method, isObtuse method and isRightAngle method.

A few standard utility methods are also included in the class. There are three methods to compare angle values for equality and a toString method. The toString method returns the angle value in the following format: degrees-minutes-seconds. For example: 282-12-11.12

The last method of the BasicSurveyorsAngle class allows you to rotate a JTS geometry. You provide the geometry, center of rotation coordinate, and a boolean flag that indicates the direction of the method. Note that this method modifies the JTS geometry it is passed in-place. It does not create and modify a copy of the geometry.

The Basic Surveyors Direction Class

The BasicSurveyorsDirection class is also essentially a value class. It represents the direction of a line segment or line in reference to a system for angle measurement. In JTS Warped, the direction of a line is measured as an azimuth in degrees, minutes, seconds, and fractional seconds. A direction of 0-0-0.0 corresponds to cardinal north or the Y axis of the 2D coordinate grid. The degrees value, minutes value, and seconds value are passed in and out of the class as ints. The fractional seconds value is passed in and out of the class as a double.

There are five (5) ways to create a BasicSurveyorsDirection object. The default constructor creates a direction with a value that corresponds to 0, or true north. Two additional constructors allow you to create a BasicSurveyorsDirection object from an angle object. You can also create a BasicSurveyorsDirection by passing two JTS coordinate objects or a string representing the azimuth. The format for the string argument passed to this last constructor is the same as output by the toString method of the BasicSurveyorsAngle class.

You can obtain the value of the BasicSurveyorsDirection object as an Angle object, as a string formatted as a bearing, or as an angle value in degrees, minutes, seconds, and fractional seconds stored in a string. The format for the returned string is the same as output by the toString method of the BasicSurveyorsAngle class. Another method returns the quadrant of

the direction.

The BasicSurveyorsDirection class is not immutable. (It will be made immutable in the next release.) It can be modified by four (4) methods. Three (3) of these are convenience methods that allow the BasicSurveyorsDirection to be flopped 180 degrees or rotated forward and backward 90 degrees. The last method allows you to rotate the BasicSurveyorsDirection by passing in a rotation angle.

There are two (2) methods of the BasicSurveyorsDirection that create JTS geometry objects. The getLineStringAlongDirection method accepts a JTS Coordinate object and a double as arguments. It then creates a JTS LineString along the direction, using the Coordinate object as a start point and the double as the length of the LineString. The getCoordinateAtEndOfVector object accepts a JTS Coordinate object and a double as arguments. It returns a Coordinate object at the end of the vector represented by the direction stored internally in the class and the length passed in to the method as a double.

Code Listing #2 shows how to use the BasicSurveyorsDirection class to create a new line segment offset 50 units in a perpendicular direction from an existing line segment.

Converting Between Angular Unit Systems

A number of different units systems are used to measure angles. Surveyors typically measure angles in values recorded as degrees, minutes, and seconds. Mathematicians record angles as radians. Military surveyors record angles as Grads. You may measure angles in revolutions. Latitude and longitude values are often stored in decimal degrees. JTS Warped provides a utility class that allows for the easy conversion between these different angle formats. This utility class is named AngleFormatUtilties.

The class also contains three (3) convenience methods. Two of these provide BasicSurveyorsAngle objects for a given value in radians or revolutions. The third returns a DegreesMinutesSecondsValue from a string in the appropriate format.

Conclusion

JTS Warped contains code that enables programmers to integrate angles and directions into their creation and

manipulation of JTS geometries. In this article we looked at the two (2) most important classes of this code, the BasicSurveyorsAngle and the BasicSurveyorsDirection. JTS Warped includes other code to enhance JTS. This includes classes to support common coordinate geometry operations, additional coordinate filter implementations, and utility methods for manipulation of LineStrings and Coordinate objects.

JTS Warped is an open source library released under the GPL. It is maintained by the SurveyOS Project, a member of the Free Software Conservancy. Contributions of code or documentation for the library are welcome.

Code Listing #1

```
1)   // Use the default, no argument constructor.
2)   BasicSurveyorsAngle angle1 = new BasicSurveyorsAngle();

3)   // Create a BasicSurveyorsAngle with a specific value.
4)   BasicSurveyorsAngle angle2 = new BasicSurveyorsAngle(35, 22, 11,
0.2116);

5)   // Create a BasicSurveyorsAngle with using a
DegreesMinutesSecondsValue.
6)   DegreesMinutesSecondsValue dmsvalue = new
DegreesMinutesSeconsValue()
7)   BasicSurveyorsAngle angle3 = new BasicSurveyorsAngle(dmsValue);

8)   // Clone a BasicSurveyorsAngle using the copy constructor.
9)   // The value of clone will be the same as the value of angle2.
10) BasicSurveyorsAngle clone = new BasicSurveyorsAngle(angle2);
```

Code Listing #2

```
1) // Rotate the target LineString 90 degrees.
2) // Create the angle needed to perform the rotation.
3) BasicSurveyorsAngle rotationAngle = new BasicSurveyorsAngle(90, 0, 0, 0.0);

4) // Apply the rotation. “target” holds the JTS LineString to be
rotated. “rotationBase” holds
5) // a JTS Coordinate object that is the basis of the rotation.
6) rotationAngle.rotateGeometry(target, baseCoordinate, true);
```

Code Listing #3

```
1) // Get the end coordinates of the existing LineString.
2) // “targetLine” holds copy of existing LineString to offset 50
units.
```

```
3) Coordinate coord1 = targetLine.getCoordinateN(0);
4) Coordinate coord2 = targetLine.getCoordinateN(1);

5) // Create a BasicSurveyorsDirection from the two (2) coordinates.
6) BasicSurveyorsDirection dir = new BasicSurveyorsDirection(coord1, coord2);

7)  // Rotate the direction 90 degrees so we can create a point on a
line perpendicular to the existing
8)  // LineString.
9)  dir.rotateForward90Degrees();

10) // Create a point 50 units away on the perpendicular line.
11) Coordinate newStartPoint  = dir.getCoordinateAtEndOfVector(coord1, 50.0);

12) // Create a parallel LineString that is 200 units long and offset
50 unites from the existing LineString.
13) LineString parallelLine = dir.getLineStringAlongDirection(newStartPoint, 200.0);
```