Navigation System v1

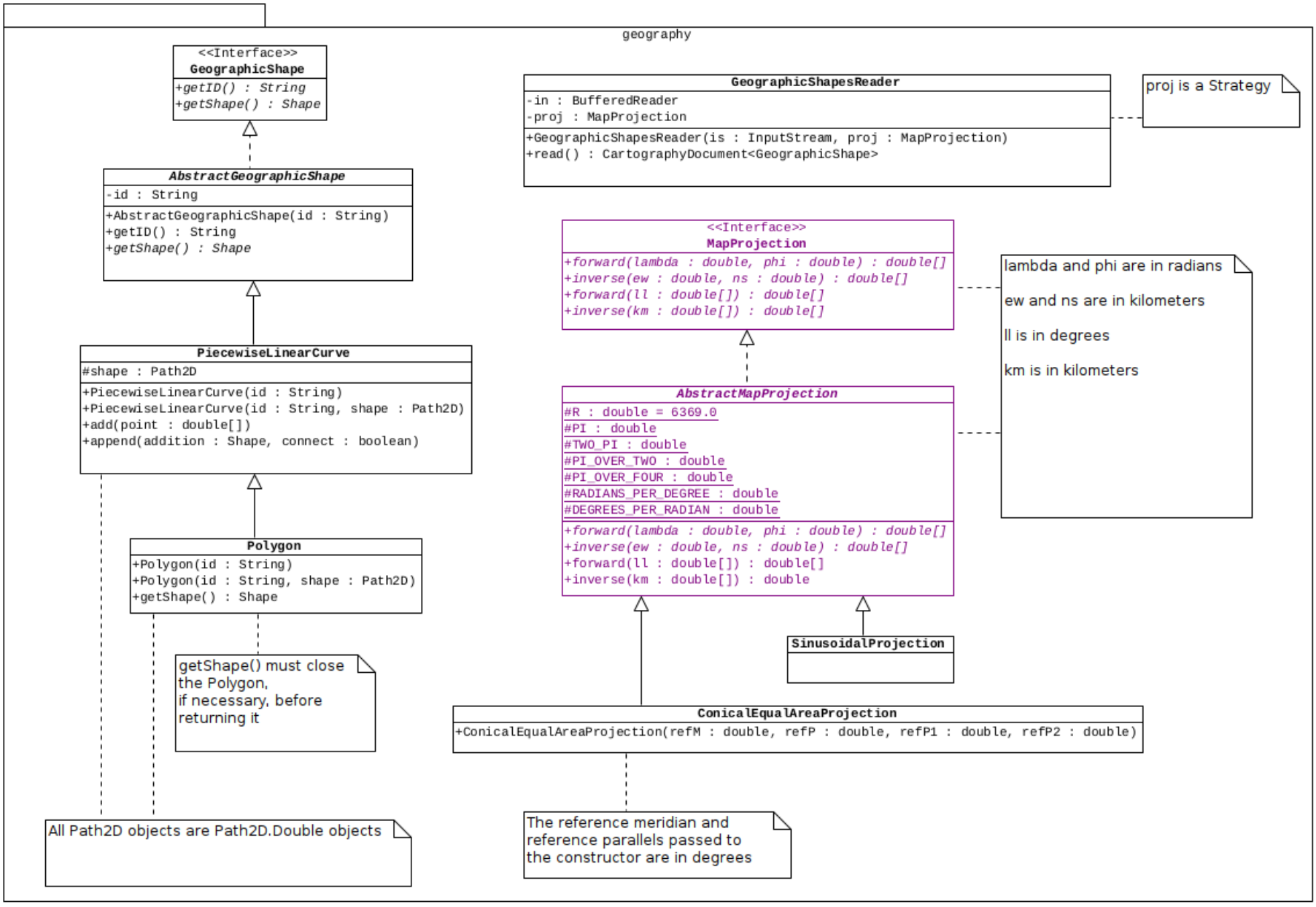
Purpose

Version 1 of the navigation system will provide the user with simple base maps.

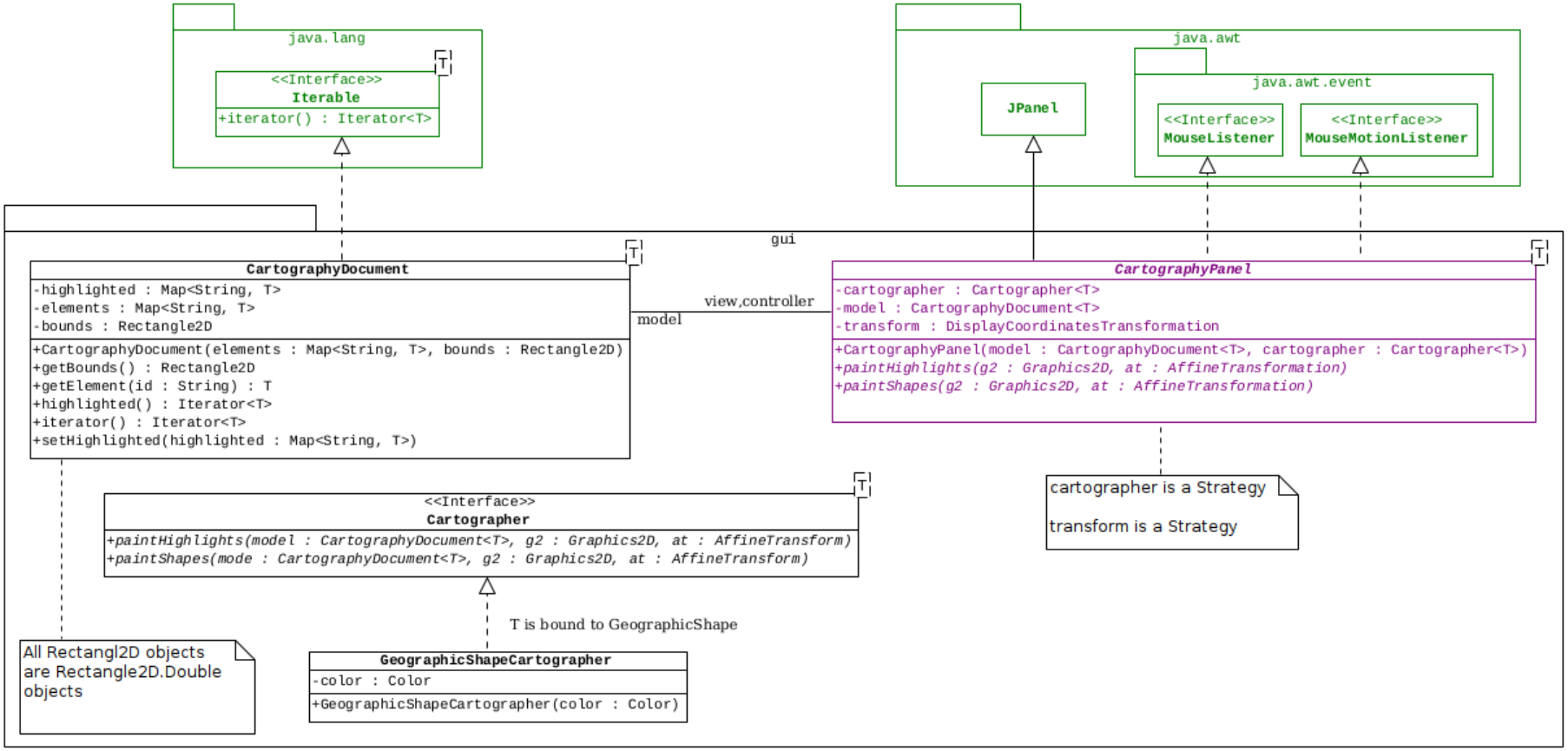
Design

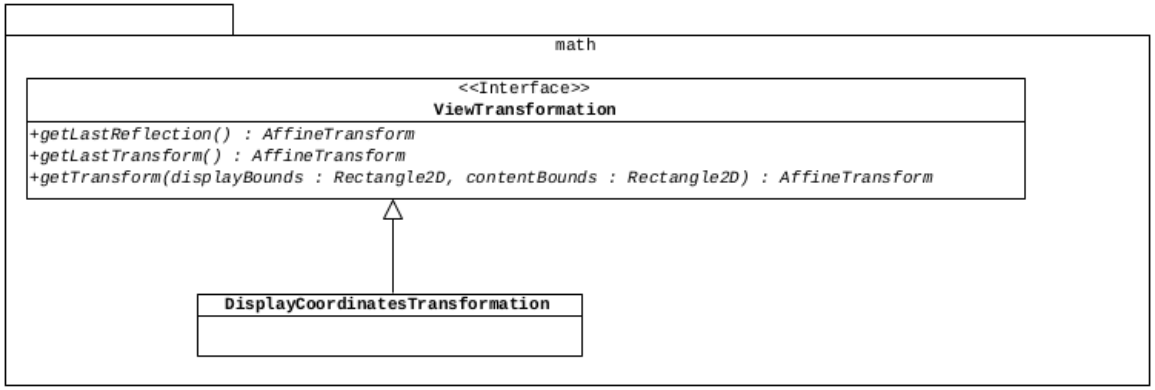
The design of the system is summarized in the following UML class diagrams.

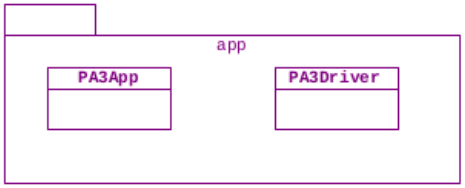
The Future of Navigation



The Future of Navigation







The Future of Navigation

Note that the components in jade green are part of the Java API and the components in purple have been provided to you.

Specifications

This section contains design specifications for some of the components above. For the others, the UML diagrams should provide all of the information that you need.

The `PiecewiseLinearCurve` Class

The `append()` method must append the given `addition` to the shape attribute of the `PiecewiseLinearCurve`. The `connect` attribute determines whether the `addition` should be connected to the `shape` attribute or not.

The `Polygon` Class

The `getShape()` method must determine if the `shape` attribute is closed or not. (It may be necessary to add an attribute to the class for this purpose.) If it is not, it must close the `shape` before returning it.

The `GeographicShapeReader` Class

The `read()` method must read a `.geo` file and return a `CartographyDocument<GeographicShape>`. The points that define the `elements` in the `CartographyDocument` must have been projected using the `MapProjection` that is passed to the constructor. (This `MapProjection` is a Strategy in the sense of the Strategy Pattern, and the `GeographicShapeReader` is the Context.) This method will also need to calculate the `bounds` in the `CartographyDocument`.

The `CartographyDocument` Class

The `iterator()` method must return an Iterator containing the elements in the `CartographyDocument`. Similarly, the `highlighted()` method must return an Iterator containing the elements in the `CartographyDocument` that are to be highlighted (which may be empty).

The `getBounds()` method must return the smallest rectangle that bounds all of the elements in the `CartographyDocument`.

The `GeographicShapeCartographer` Class

A `Cartographer` is a Strategy (in the sense of the Strategy Pattern) that a `CartographyPanel` uses to render the elements and the highlighted elements in a `CartographyDocument`. A `GeographicCartographer` is a Concrete Strategy.

The `paintShapes()` method must transform each of the elements (using the `AffineTransform` it is passed) and then render them in the `Color` that was passed to the constructor. The `AffineTransform` is used to translate, reflect, and scale the elements (which, as explained in the `GeographicShapeReader`, are in kilometers).

The `paintHighlights()` method, similarly, must transform each of the elements and then render them (in an appropriate highlight color that is partially transparent).

The `DisplayCoordinatesTransformation` Class

The DisplayCoordinatesTransformation class converts from coordinates that are in kilometers to display coordinates that are in pixels with (0,0) at the upper left.

The `getTransformation()` method must return an `AffineTransform` that translates, rotates, and scales the coordinates that are in kilometers (and bounded by the parameter `contentBounds`) to display coordinates that are in pixels (and bounded by the parameter `displayBounds`). Note that `AffineTransformation` objects can be combined using their `concatenate()` and/or `preConcatenate()` methods

The `getLastReflection()` method must return the reflection that was used.

The `getLastTransform()` method must return the last complete transformation that was used.

.geo File Format

`.geo` files are simple tab-delimited ASCII text files that contain one or more components.

Each component begins with a line that contains a description of the component and ends with a line containing the string `END`. Each description contains four fields: the keyword `Type:`, the type of the component, the keyword `ID:`, and the identifier (which is not necessarily unique) of the component.

For example, the following is a simple component:

```
Type:     Polygon   ID:  Washington01
     -0.122715386336304E+03    0.487485426692278E+02
     -0.122611554920771E+03    0.486508989342796E+02
     -0.122718833000000E+03    0.487168180000000E+02
     -0.122715386336304E+03    0.487485426692278E+02
END
```

Comments

Comments have a type of `Comment`. Comments have no line between the description and the `END`.

Piecewise Linear Curves

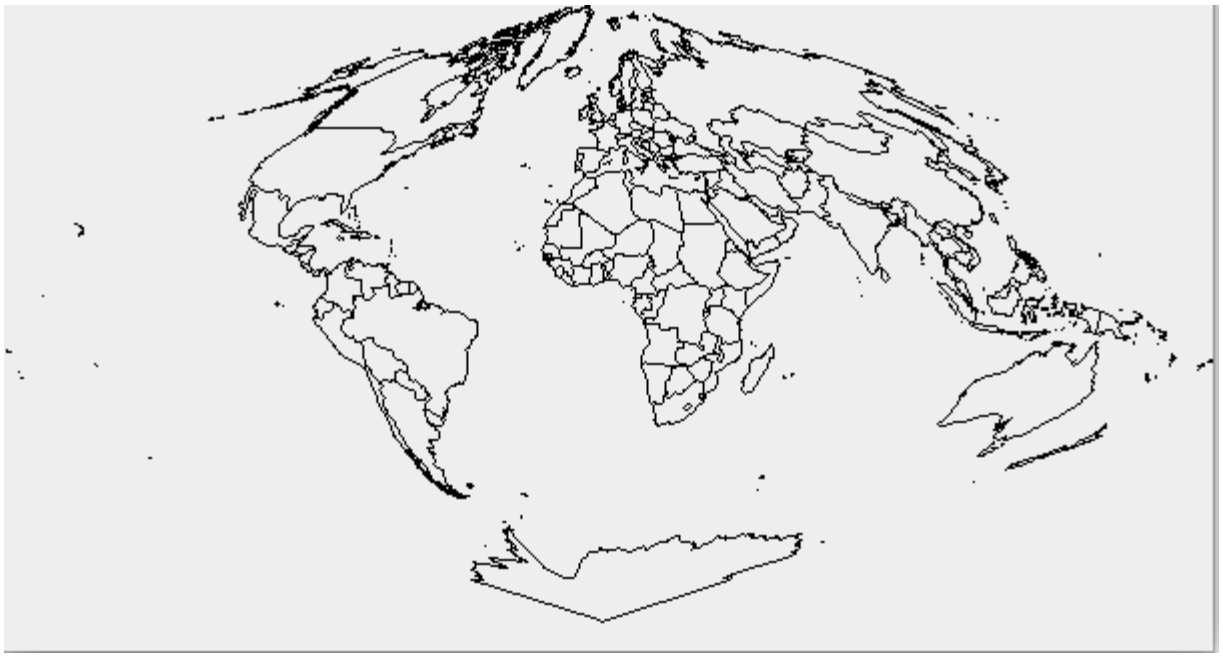
Piecewise linear curves have a type of `PiecewiseLinearCurve`. There will be 2 or more lines between the description and the `END`. Each of these lines contains the longitude and latitude (in floating-point degrees; not degrees, minutes and seconds) of the points that define the curve.

Polygons

Polygons have a type of `Polygon`. There will be 2 or more lines between the description and the `END`. Each of these lines contains the longitude and latitude of the vertices that define the polygon. The first and last of these vertices will always be the same.

Examples

This section contains examples of what we home the maps will look like. The first shows the borders of the countries of the world. The second shows the streets in Rockingham County, VA.



The Future of Navigation