Alternative Designs

by Callum France

Alternative 1

Instead of using a composite pattern to store routes (that may contain other routes and so on), an alternative may be to simply enforce that a Route object can only contain Description objects. If a Route does contain a sub-route, then that sub-route is expanded into a list of Description objects and this list is inserted in its place.

The biggest advantage of this alternative is that it streamlines the use of an iterator pattern in Routes. Because every segment in Routes is a Description, you will never have to recurse down into a sub-route to find the next Description object.

One downside that is normally expected from this implementation is that if the data inside one Route (that was also a sub-route in another Route) were to change, because this Route is not referenced in the other Route, the data will not be changed in the other Route, despite no longer being correct. However, because GeoUtils will only ever update all data at once, this scenario will never occur.

One real consequence of this alternative is memory. If a Route is a sub-route, then the Descriptions inside this sub-route are stored multiple times despite being identical.

Alternative 2

An alternative to Way-to-Go's current UI class would be to refactor it using the State pattern. There are currently three separate calls to the View UI class within Controller - the UI methods stemming from these calls could all become their own separate View classes in different python files. Each of these View classes could aggregate a different 'State' interface, and the inheriting classes of these states would each represent a single state of the current UI class.

The UI class currently contains complicated control flow statements to write specific messages to the console. Using states and separate classes will separate the control flow clutter out of the UI class and into concrete state classes - i.e. code hiding. This will make the resultant UI classes highly readable.

However the biggest drawback to this alternative is that it requires a lot more classes to be written and tested, taking more time to implement in actual code.

Furthermore, it may increase inter-class coupling through the Directory and Tracker observers. The concrete observers would be added and removed from the respective concrete UI class. As such, the notify observers method will need to travel from their origin in the Model, through the observer interface, into the View UI class, then through the state interface and finally into the state class.