Generic Class Challenge

Now it's your turn to create your own generic class.

In the Interface Challenge, we created a Mappable Interface, and I introduced you to different Map geometry types, POINT, LINE, and POLYGON.

The challenge then created a map marker or icon, and a map label, but didn't do anything with locations.

In this challenge, you'll use another Mapping example, but use location data in the output.

As you are probably aware, you can use Google Maps to determine the location of any point on a map.

The Generic Class challenge

You'll start with a **Mappable Interface**, that has one abstract method, render.

You'll create two classes **Point** and **Line**, that implement this interface.

You'll create **two specific classes** that extend each of these, for a mappable item of interest.

The Generic Class challenge

I'll be mapping US National Parks, and a couple of major rivers in the US, so the parks will be points, and the rivers will be lines.

The data I'll be using is shown here.

I'll be creating a **Park** class that extends Point, and a **River** class that extends Line, to support this data.

US National Parks & selected locations

Name	Type	Googled Locations
Yellowstone	National Park	44.4882, -110.5916
Grand Canyon	National Park	36.0636, -112.1079
Yosemite	National Park	37.8855, -119.5360

US Rivers & selected locations

Name	Type	Googled Locations
Mississippi	River	47.2160, -95.2348 35.1556, -90.0659 29.1566, -89.2495
Missouri	River	45.9239, -111.4983 38.8146, -90.1218
Colorado	River	40.4708, -105.8286 36.1015, -112.0892 34.2964, -114.1148 31.7811, -114.7724
Delaware	River	42.2026, -75.00836 39.4955, -75.5592

The Generic Class challenge

You should have constructors or methods, to support adding a couple of attributes, and some location data, to your two specific classes.

Name	Googled Location of a Point
Yellowstone National Park	44.4882, -110.5916

You can pass the location data of a point type, as a String, or a set of double values, representing latitude and longitude.

You can pass the multiple locations of a line, as a set of strings, or a two dimensional array of doubles, that represents the multiple points on your line.

Name	Googled Locations of Points in a River
Mississippi River	47.2160, -95.2348 35.1556, -90.0659 29.1566, -89.2495

The Generic Class challenge

In addition to these classes, you'll create a **generic class** called **Layer**.

Your Layer class should have **one type parameter**, and should only **allow Mappable elements** as that type.

This generic class should have a **single private field**, a **list of elements** to be mapped.

This class should have a method or constructor, or both, to add elements.

You should create a method, called renderLayer, that loops through all elements, and executes the method render, on each element.

The Generic Class challenge

Your main method should create some instances of your specific classes, which include some location data.

These should get added to a typed Layer, and the render Layer method called on that.

Sample output is shown here:

```
Render Grand Canyon National Park as POINT ([40.1021, -75.4231])
Render Mississippi River as LINE ([[47.216, -95.2348], [29.1566, -89.2495], [35.1556, -90.0659]])
```

The plan - the class diagram

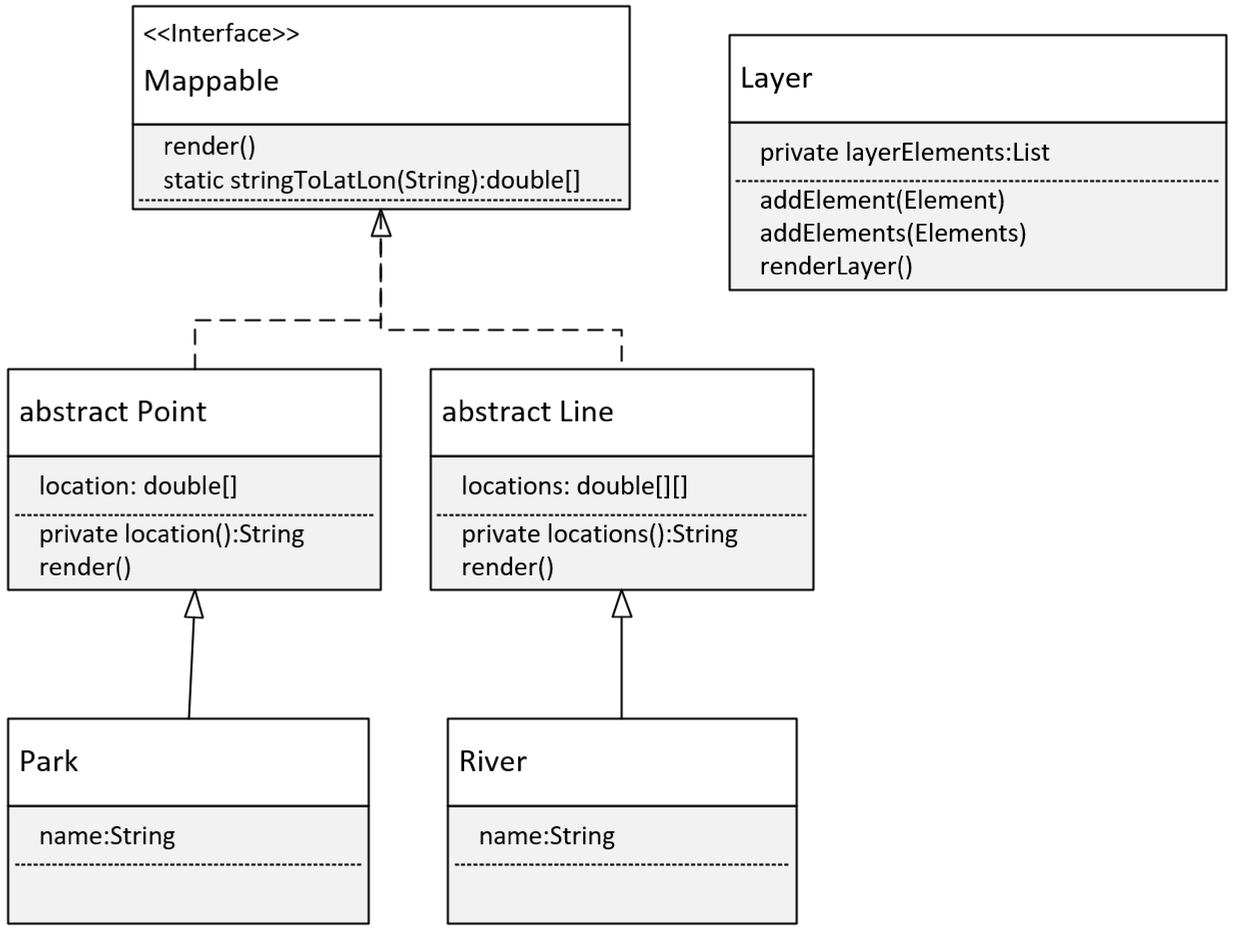
This diagram shows how I plan to build this.

You can see my Mappable interface has the method render on it, and by default that's both public and abstract.

I've also added a static method, that will take a String, and split it into a double array, which will have the latitude and longitude values in them.

I've made two classes, Point and Line, abstract, because I don't really want anyone to instantiate these classes.

Point has a location field, which is a double array, and will just have two doubles, the latitude and longitude.



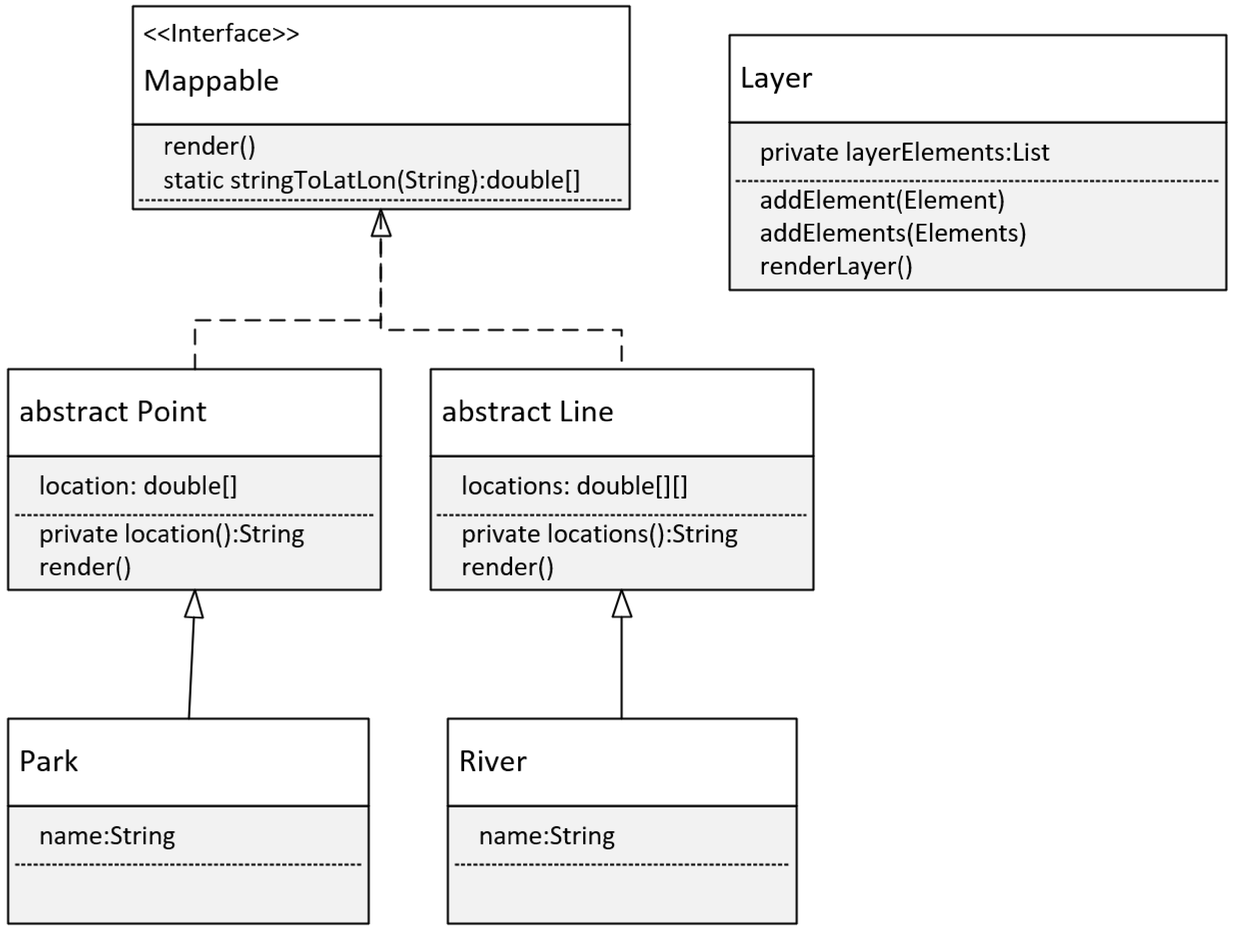
The plan - the class diagram

And I have a method that will print that array to a string, called location.

And then the render method is implemented.

And the same with abstract Line, except a line will have multiple latitude, longitude pairs, and these will be represented as a two-dimensional array.

And then I'm going to have a Park class extend Point, with just a name field.



The plan - the class diagram

River will extend Line, and that has just a name as well, for simplicity.

Lastly, there's the Layer class, this is the generic class, and it has a list of layerElements, and methods to add one or more of these.

It also has the method, renderLayer.