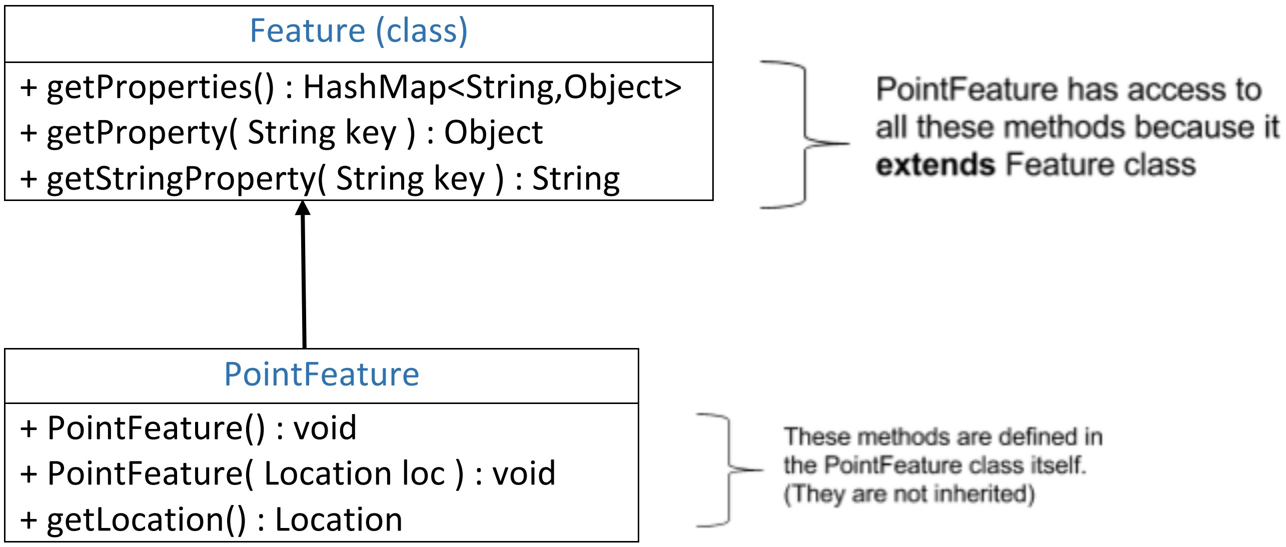
Support: Understanding Features and Markers

The programming assignments you will be doing will rely heavily on Features and Markers. This is a guide to help you understand how Features and Markers can be used in your programming assignments. You don't need to read through it now. It is here for you to refer to later on as you work through the programming assignments. This guide is intended as a help to learners who have difficulty differentiating between the different Feature and Marker classes.

NOTE: We have only included a few of the methods in these classes. You encourage you to be creative. Go to the API for each of these classes to find other methods that you may find helpful in completing the assignment.

Let's learn more about Feature and PointFeature!

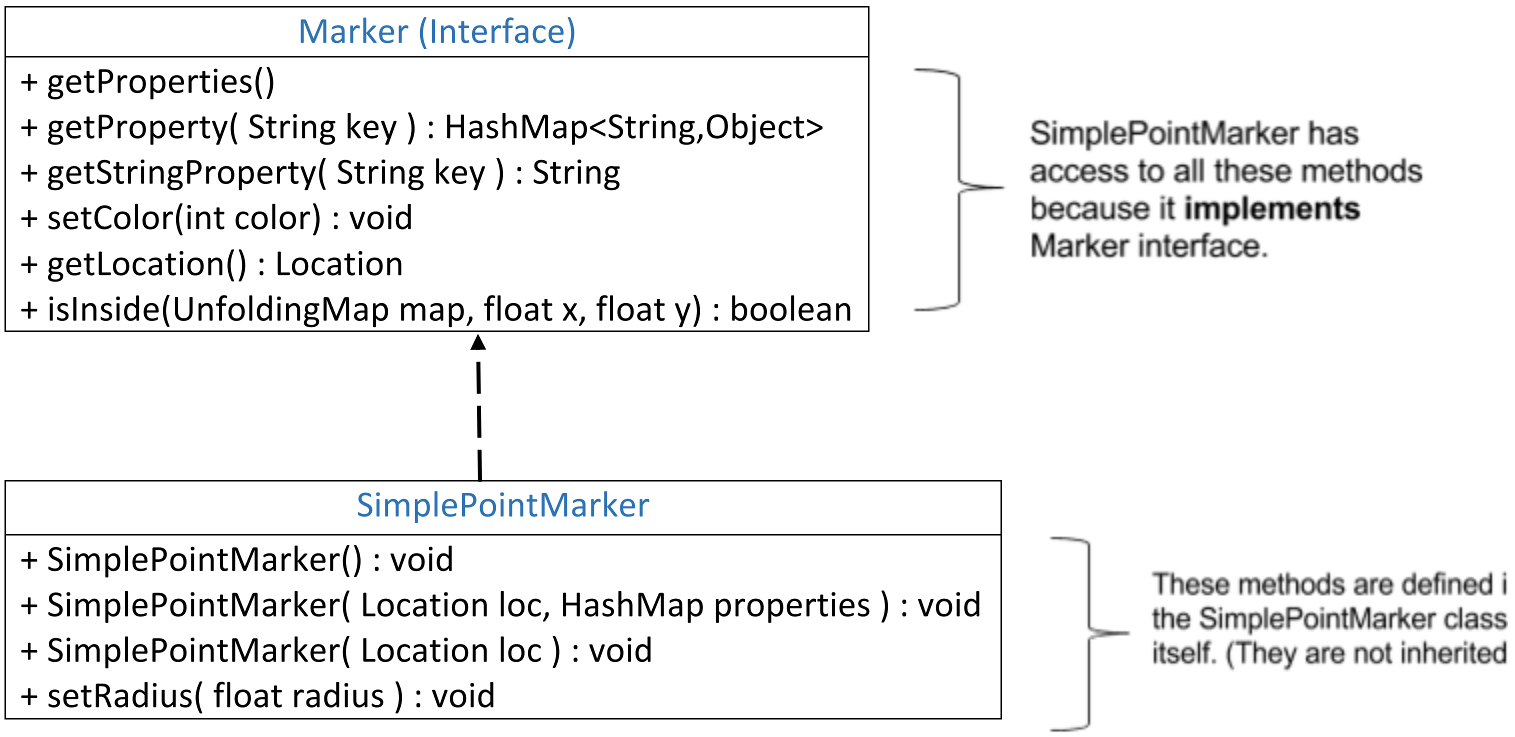
Here is a UML Diagram of the relationship between Feature and PointFeature. It is a visual representation of how Feature and PointFeature relate to each other. PointFeature is a subclass of (inherits from) Feature.


Feature (class)
+ getProperties() : HashMap<String,Object>
+ getProperty(String key) : Object
+ getStringProperty(String key) : String
PointFeature has access to all these methods because it **extends** Feature class
PointFeature
+ PointFeature() : void
+ PointFeature(Location loc) : void
+ getLocation() : Location
These methods are defined in the PointFeature class itself. (They are not inherited)

Be aware that even though PointFeature inherits all of the methods of Feature, the API lists the inherited methods separate. In the API, after the section titled Method Summary there is a section listing all of the methods that are inherited from Feature.

Let’s learn more about Marker and SimplePointMarker!

Here is a UML Diagram of the relationship between Marker and SimplePointMarker.

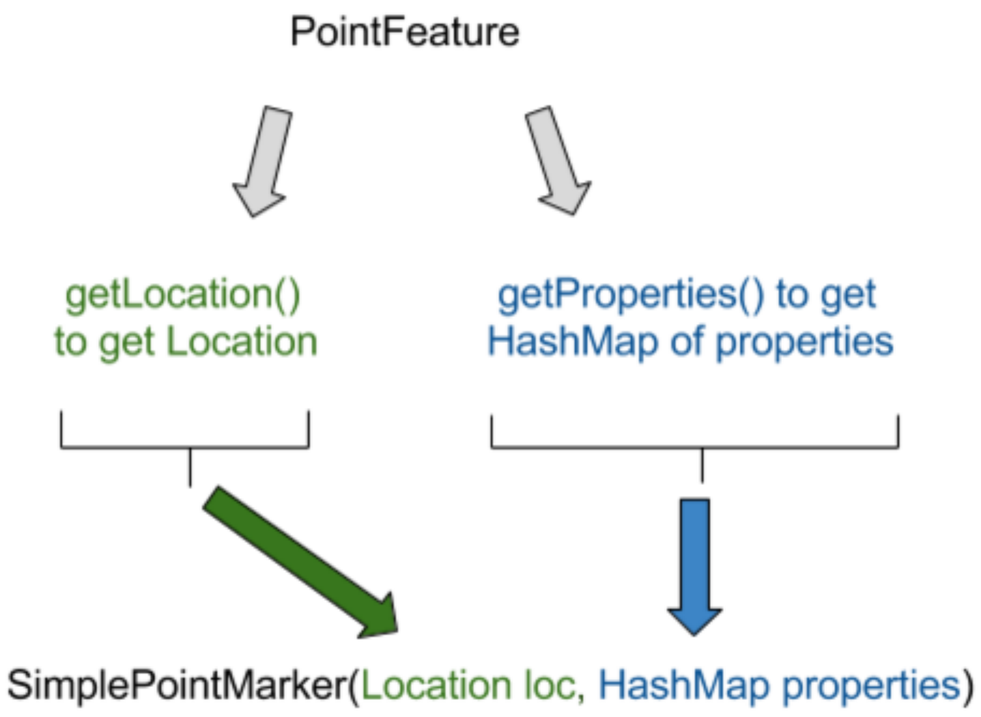

Marker (Interface)
+ getProperties()
+ getProperty(String key) : HashMap<String,Object>
+ getStringProperty(String key) : String
+ setColor(int color) : void
+ getLocation() : Location
+ isInside(UnfoldingMap map, float x, float y) : boolean
SimplePointMarker has access to all these methods because it **implements** Marker interface.
SimplePointMarker
+ SimplePointMarker() : void
+ SimplePointMarker(Location loc, HashMap properties) : void
+ SimplePointMarker(Location loc) : void
+ setRadius(float radius) : void
These methods are defined i
the SimplePointMarker class
itself. (They are not inherited

Since Marker is an interface that SimplePointMarker implements, the API for SimplePointMarker will not list methods inherited from Marker in the way we saw with the PointFeature API. There won’t be a section titled “Methods inherited from Marker”. Instead, we have to do some investigating to see what methods SimplePointMarker inherits from Marker. At the top of the API for any class there will be a section titled “All Implementing Interfaces”. For SimplePointMarker we will see Marker listed here. Since this is at the top of the page the developers of UnfoldingMaps assume we understand that all of Marker’s methods are inherited by SimplePointMarker. If we want to know what those inherited methods are, then we simply click on the link for Marker.

How do we associate PointFeature and SimplePointMarker?

1) Creating a single SimplePointMarker

If you already have a PointFeature object, then you can use that existing PointFeature to create a SimplePointMarker. The constructor for SimplePointMarker accepts a Location and a HashMap. The Location is where this PointFeature will appear on the map. The HashMap contains the properties of the PointFeature. Those two pieces of information we get from the PointFeature object using the getLocation() and getProperties() methods.


PointFeature
getLocation() to get Location
getProperties() to get HashMap of properties
SimplePointMarker(Location loc, HashMap properties)

This approach creates a single SimplePointMarker object with its location and properties set when you create it. Another approach is to create a SimplePointMarker using its default (no-argument) constructor, then use a PointFeature object to set the SimplePointMarker's location and properties after it's been created. There are many other ways to reach this same result. Be creative!

2) Creating a List of SimplePointMarkers

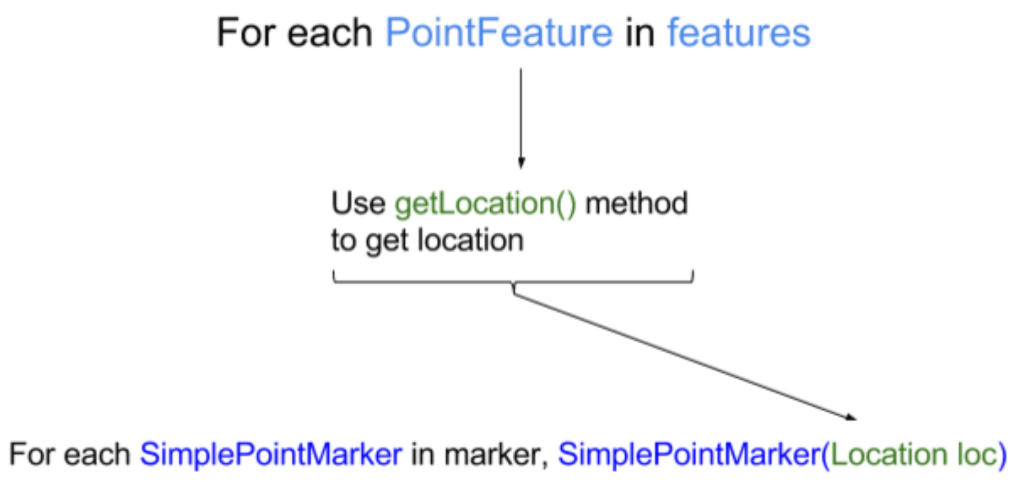
Let’s say you have a List of PointFeature objects.

```
List<PointFeature> features = new ArrayList<PointFeature>();
```

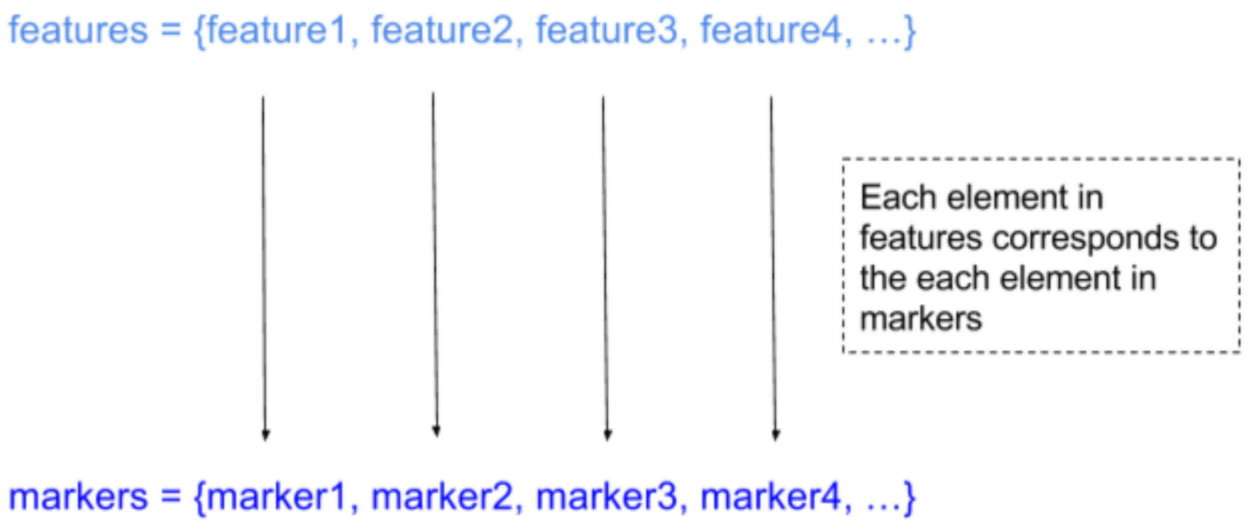
Assume that this List is already populated with PointFeatures, each with its own location. Now you want to visually represent those PointFeatures on a map. To do that, create an empty List of SimplePointMarkers.

```
List<SimplePointMarker> markers = new ArrayList<SimplePointMarker>();
```

We will now take the locations of each PointFeature in our features List and use it to create a new SimplePointMarker in our markers List.


For each PointFeature in features
Use getLocation() method to get location
For each SimplePointMarker in marker, SimplePointMarker(Location loc)

Creating our markers List using this method gives us is a correlation between the two lists. The first feature relates to the first marker, the second feature to the second marker, etc.


features = {feature1, feature2, feature3, feature4, …}
Each element in features corresponds to the each element in markers
markers = {marker1, marker2, marker3, marker4, …}

Any method is the right way as long as it does its job.
So just use the one that suits your need the best!