The interface Challenge

In this challenge, we'll be working on creating some mappable output.

In the past decade or so, maps have become part of so many applications.

Most things, when drawn on a map, fall into three categories, a point, a line, or a polygon or geometric shape.

The result of your code, will be text that could be printed out to a file, for exchanging data with a mapping application.

One such file is a specially formatted file, called geojson, which is a JSON file extended for geographical elements.

You don't have to know JSON or geojson to be successful at this challenge.

The interface Challenge

For this challenge, you'll simply create a String for every feature that will be mapped.

An example of such a String is shown on this slide.

```
"properties": { "name": "Sydney Opera House", "usage": "Entertainment"}
```

The interface Challenge

First, Create a **Mappable** Interface.

The interface should **force classes to implement three methods**.

- One method should return a **label** (how the item will be described on the map).
- One should return a **geometry type** (POINT or LINE) which is what the type will look like on the map.
- The last should return an **icon type** (sometimes called a map marker).

The interface Challenge

In addition to the three methods described, the interface should also include:

- A constant String value called JSON_PROPERTY, which is equal to: "properties":{%s}. A hint here, using a text block will help maintain quotation marks in your output.
- Include a default method called toJSON() that prints out the type, label, and marker. I'll show examples shortly.
- A **static method**, that takes a Mappable instance as an argument. This method should print out the properties for each mappable type, including those mentioned above, but also any other fields on the business classes.

The interface Challenge

You'll also want to create two classes that implement this interface, a **Building** and **UtilityLine**.

- One class, in my case the Building, should have a geometry type of POINT, and One class should have a geometry type of Line. The UtilityLine class will be my example for a class that will be a LINE on a map.
- The Building will be shown on a city map, as a point with the icon and label specified, and the Utility Line will be a line on the map.

The interface Challenge

Your final output should look something like I show on this slide.

You should output the geometry type, the icon information, and the label.

Here is an example for a building, including type, label, and marker, but also the building name and usage, which are fields on building.

```
"properties": {"type": "POINT", "label": "Sydney Town Hall (GOVERNMENT)", "marker": "RED STAR", "name": "Sydney Town Hall", "usage": "GOVERNMENT"}
```

And here is an example for a fiber optic Utility line, so a LINE, a green dotted line, would get drawn for a fiber optic cable on College Street.

```
"properties": {"type": "LINE", "label": "College St (FIBER_OPTIC)", "marker": "GREEN DOTTED", "name": "College St", "utility": "FIBER_OPTIC"}
```

You can see that the properties are a comma delimited list, in curly braces, with the property or field name in quotes, then a colon, followed by the property value or field value, and that's also in double quotes.

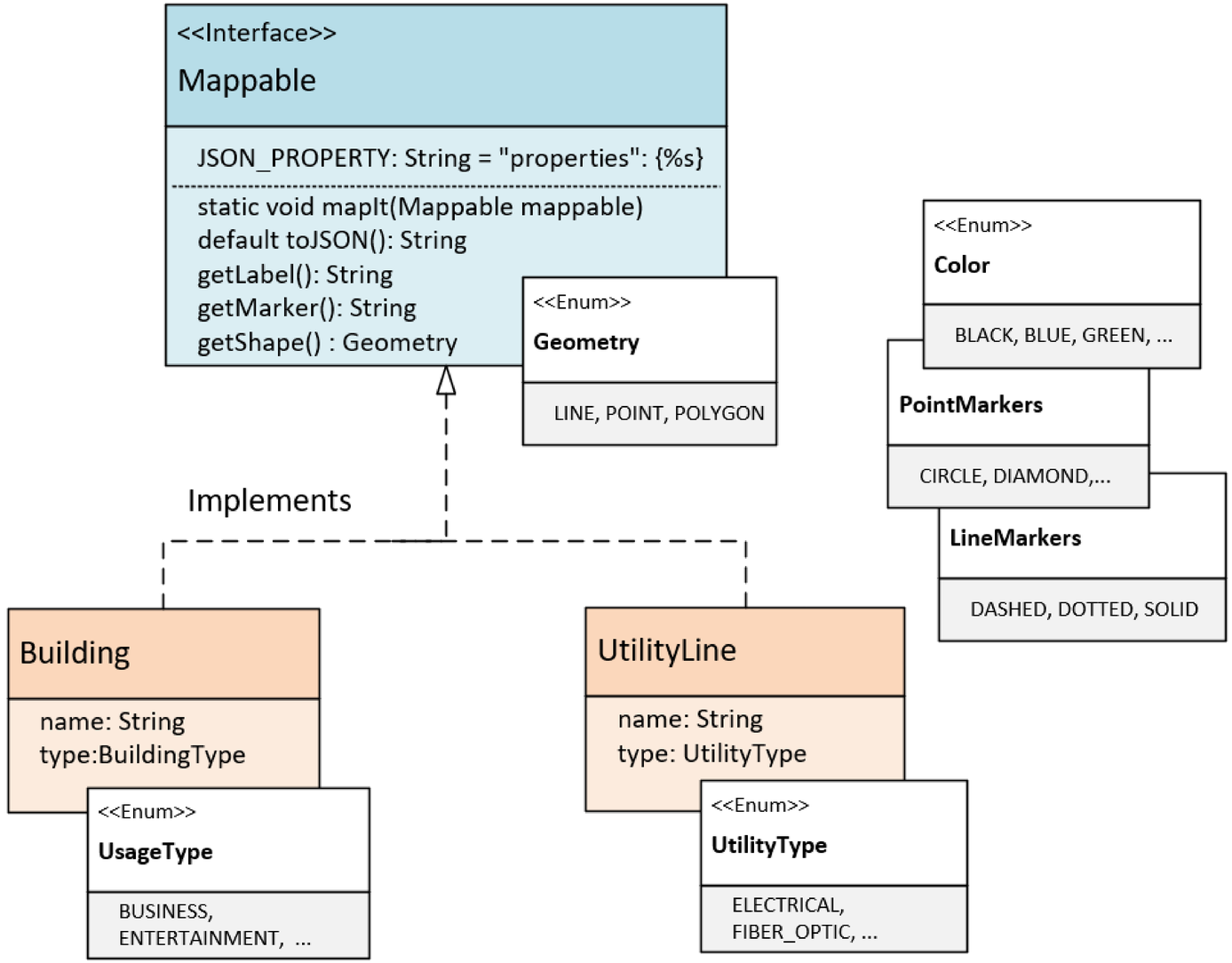
The Class Diagram

On the Mappable interface, I have one constant, JSON_PROPERTY, and 3 abstract methods, `getLabel`, `getMarker`, and `getShape`.

I'm also including a default method, toJSON, which is going to return a String.

The `getShape` method returns an enum type, Geometry, and the valid types on this enum are LINE, POINT, and POLYGON.

Use enums for Color, and the PointMarker and LineMarker types.



The Class Diagram

Two business classes to be mapped, Building and UtilityLine.

For building, use an enum to describe the building type.

For the UtilityLine, Use enum to describe the type of utility it is.