Informatics Large Practical: Introduction to Maven and Geo-JSON

Stephen Gilmore and Paul Jackson

School of Informatics

Friday, 25th September 2020

Using Maven as a build tool

Our project has Java as the application language and Maven as the build tool for the project. For us, Maven serves two purposes:

1. it records the project dependencies on other libraries (all significant software projects have these dependencies); and
2. it specifies how to compile our Java code and how to build a Java Archive file (JAR) for our project, with our code bundled together with the libraries which it depends on.

Here we consider only the first purpose and will come to the second later.

A new dependency: Generating Geo-JSON

- We wish to generate and export a Geo-JSON map from our application.
- Mapbox's Java Geo-JSON library* allows us to work with Geo-JSON concepts and generate a Geo-JSON string for export to a text file.
- This is a new dependency for our project, so we must add it to our Maven project-object model (POM) file, pom.xml.

*https://docs.mapbox.com/android/java/overview/

Maven concepts

- The pom.xml file describes libraries our code makes use of.
- Maven libraries/dependencies are stored in repositories in standard locations on the web.
- A dependency is specified using tags:
  - groupId,
  - artifactId,
  - version, and
  - scope.
- The scope tag says when a dependency needs to be used in the build process.

Interacting with Maven repositories

- If you start with just Java code and a pom.xml file, running Maven will connect to the Maven repositories and download the needed libraries to a local cache as a first step of the build process.
  - You will need to have a working internet connection at this point and, depending on your location, you may need to use a VPN to ensure that the Maven repositories can be seen.
- Each dependency is only downloaded once, not every time you compile your project.
- We will re-compile your code from source when testing your code, so you *must* include dependencies for all imported Java classes not in the Java SE library in your pom.xml file.

Maven: Adding to the existing dependencies in pom.xml

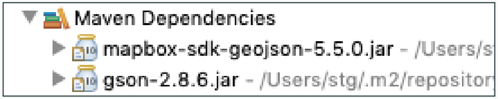
XML

```xml
 <dependencies>
  <dependency>
    <groupId>com.mapbox.mapboxsdk</groupId>
    <artifactId>mapbox-sdk-geojson</artifactId>
    <version>5.5.0</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.9</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Caution: UNICODE alert!

- If cutting-and-pasting from this PDF you may get UNICODE dash characters in `mapbox–sdk–geojson`.
- If so, delete these and put ASCII hyphen characters in their place to have `mapbox-sdk-geojson` instead.

Maven: Success! New dependencies added


Maven Dependencies
mapbox-sdk-geojson-5.5.0.jar - /Users/s
gson-2.8.6.jar - /Users/stg/.m2/repositor

- geojson 5.5.0 — 752Kb
- gson 2.8.6 — 880Kb
  - The Mapbox Geo-JSON library uses the Google Gson SDK (a JSON parser), so it added this dependency without us needing to write it explicitly.
  - The Gson library is included because it gets mentioned as a dependency in a pom.xml file included in the Geo-JSON jar file.

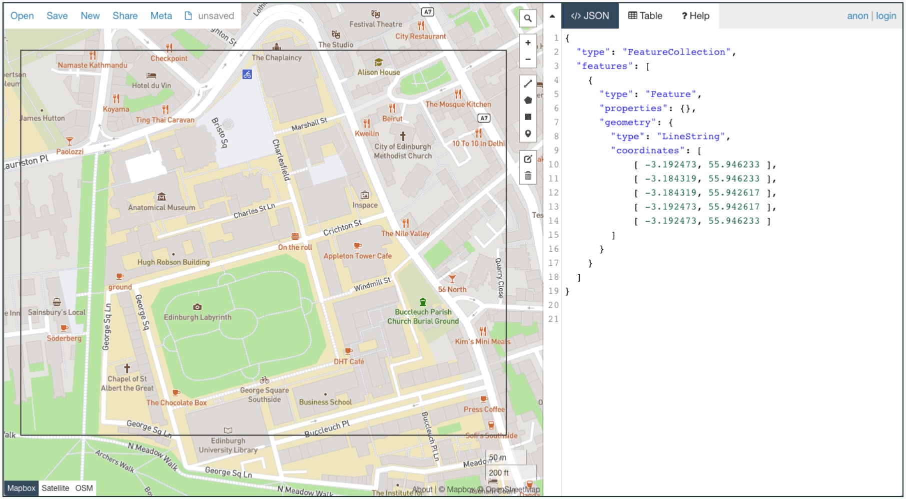
Geo-JSON concepts

- Every Geo-JSON map has a FeatureCollection which contains a list of Features.
- Every feature has a Geometry and every geometry has a property called coordinates.
- Examples of geometries are Point, LineString, and Polygon.
- For type LineString, the coordinates property is an array of two or more positions.
- In Geo-JSON, coordinates are presented in the order (longitude, latitude). For all the coordinates which we will see, latitude is always positive and longitude is always negative.

Example: a GeoJSON map with the drone confinement area

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "properties": { },
      "geometry": {
        "type": "LineString",
        "coordinates": [
            [ −3.192473, 55.946233 ],
            [ −3.184319, 55.946233 ],
            [ −3.184319, 55.942617 ],
            [ −3.192473, 55.942617 ],
            [ −3.192473, 55.946233 ]
         ]
      }
    }
  ]
}
```

Rendering the map on http://geojson.io



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "LineString",
        "coordinates": [
            [ -3.192473, 55.946233 ],
            [ -3.184319, 55.946233 ],
            [ -3.184319, 55.942617 ],
            [ -3.192473, 55.942617 ],
            [ -3.192473, 55.946233 ]
        ]
      }
    }
  ]
}
```

The Mapbox SDK

The Mapbox Java SDK gives us a collection of useful classes which we can use to parse or to create Geo-JSON documents:

- **class** com.mapbox.geojson.FeatureCollection
- **class** com.mapbox.geojson.Feature
  - can be used as java.util.List<Feature>
  - has properties and geometry
- **interface** com.mapbox.geojson.Geometry implemented by
  - **class** com.mapbox.geojson.Point
  - **class** com.mapbox.geojson.LineString
  - **class** com.mapbox.geojson.Polygon

Static methods in the Mapbox SDK

Instead of using constructors and making Geo-JSON objects with "**new** Point(...)" etc., the Mapbox SDK provides a collection of static methods for creating Geo-JSON objects:

- Point.fromLngLat
- LineString.fromLngLats
- Polygon.fromLngLats
- Points, line strings, and polygons can be cast to Geometry.
- Feature.fromGeometry
- FeatureCollection.fromFeatures

The Mapbox SDK; generating Geo-JSON maps (1/2)

- If lng and lat are doubles then Point.fromLngLat(lng, lat) is a Point.
- If pl is a List⟨Point⟩ then LineString.fromLngLats(pl) is a LineString.
- If pl is a List⟨Point⟩ then Polygon.fromLngLats(List.of(pl)) is a Polygon (without polygon holes).
- if x is a Point or LineString or Polygon then (Geometry)x is a Geometry.

The Mapbox SDK; generating Geo-JSON maps (2/2)

- If g is a Geometry then Feature.fromGeometry(g) is a Feature.
- If f is a Feature we can add a string property with f.addStringProperty("k", "v").
- If f is a Feature we can add a number property with f.addNumberProperty("x", 10).
- If fl is a List⟨Feature⟩ then FeatureCollection.fromFeatures(fl) is a FeatureCollection.
- If fc is a FeatureCollection then fc.toJson() is a JSON-formatted string of that feature collection.

Thank you for listening.