Assignment 4

Kaibo Ma (32400129) -> kiddo122
David Henderson (19475128) -> D-H
Matthew Piaseczny (32993131) -> mpiasek

Exercise 4.1
Features that are important to test:

- Searching for a destination
- directions from one destination to another
- zooming in and out on the map
- different map layers

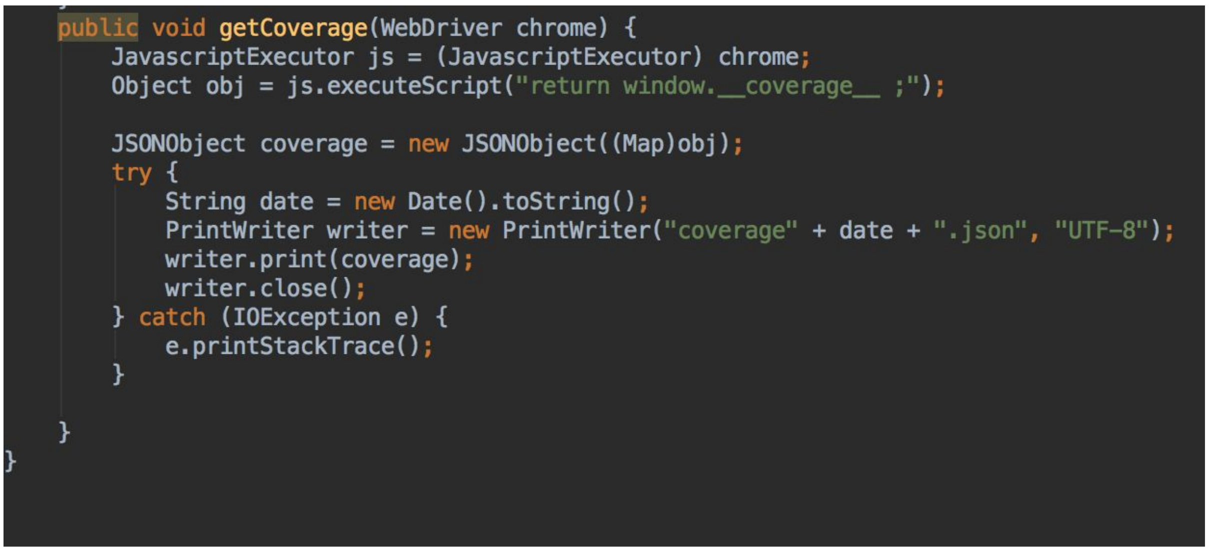
Which tests cover which feature:

- chromeTitleTest() simply tests to make sure the page loaded properly
- chromeSearchTest() tests searching for a destination
- chromeDirectionsTest() tests getting directions from one destination to another
- chromeCycleLayerTest() tests changing the current label to the cycle layer
- chromeTransportLayerTest() tests changing the current layer to the transport layer
- chromeHumanitarianLayerTest() tests changing the current layer to the humanitarian layer
- chromeTestZoomingIn() tests zooming in five times
- chromeTestZoomingOut() tests zooming out five times
- chromeTestZoomingInMaxed() tests zooming in to the maximum allowed value
- chromeTestZoomingOutMaxed() tests zooming out to the maximum allowed value

Exercise 4.2
Test quality has been assured through the use of assertions, checking whether variables that showed in the pages' URLs were equal to what we expect them to be if we did what the test cases did manually. Additionally, functions were created to take all the complex code that would not mean anything to a non-coder and turn it into something more simple to decode. Functions such as openPage() to open a specified URL, zoomIn() to zoom in on the map, and cycleLayer() to change the current layer to the specified layer.

Exercise 4.3
The way our group approached this assignment is to run a coverage test on the .js files for the client on the server. Our selenium test will trigger certain javascript functions to be executed. We first set up a localhost server of the open source project from the github (https://github.com/openstreetmap). Then we had to instrument all our js files through istanbuljs using the commands (~$ istanbul instrument --embed_sources <path/to/original/js/files> --output <path/to/location/of/instrumented/sources>. Then we set up the rails environment to run the server such as Vagrant and PostGreSQL. We ran our server through a standalone virtual machine dedicate for our server through Vagrant. Once the server is up and running we are able to run our selenium test along with a function called

getCoverage() shown below:

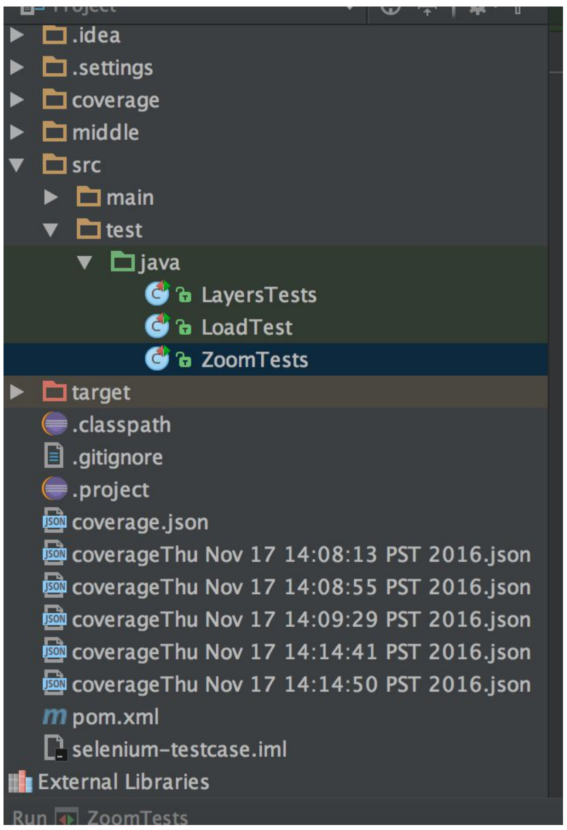


```
    public void getCoverage(WebDriver chrome) {
        JavascriptExecutor js = (JavascriptExecutor) chrome;
        Object obj = js.executeScript("return window.__coverage__ ;");

        JSONObject coverage = new JSONObject((Map)obj);
        try {
            String date = new Date().toString();
            PrintWriter writer = new PrintWriter("coverage" + date + ".json", "UTF-8");
            writer.print(coverage);
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

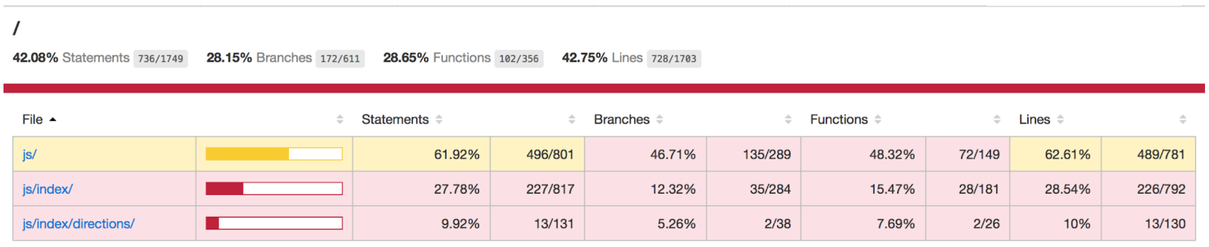
This function would read the instrumented javascript files via a javascript window.variable called __coverage__. This is a mapping of the branches, statements, and coverage of the executed when selenium runs each javascript function. It will then convert it to JSON format and export it to a .json file with timestamps. These files are shown below:



```
▶ .idea
▶ .settings
▶ coverage
▶ middle
▼ src
    ▶ main
    ▼ test
        ▼ java
            LayersTests
            LoadTest
            ZoomTests
▶ target
  .classpath
  .gitignore
  .project
  coverage.json
  coverageThu Nov 17 14:08:13 PST 2016.json
  coverageThu Nov 17 14:08:55 PST 2016.json
  coverageThu Nov 17 14:09:29 PST 2016.json
  coverageThu Nov 17 14:14:41 PST 2016.json
  coverageThu Nov 17 14:14:50 PST 2016.json
  pom.xml
  selenium-testcase.iml
External Libraries
```

Once the files are generated, istanbul can parse these json files and output a final report with the function (~$ istanbul report --dir <path/to/root/directory> --include <all .json files>).

The final report will be outputted to a directory called coverage where you can click the .html file in there to show the total coverage of the executed scripts.

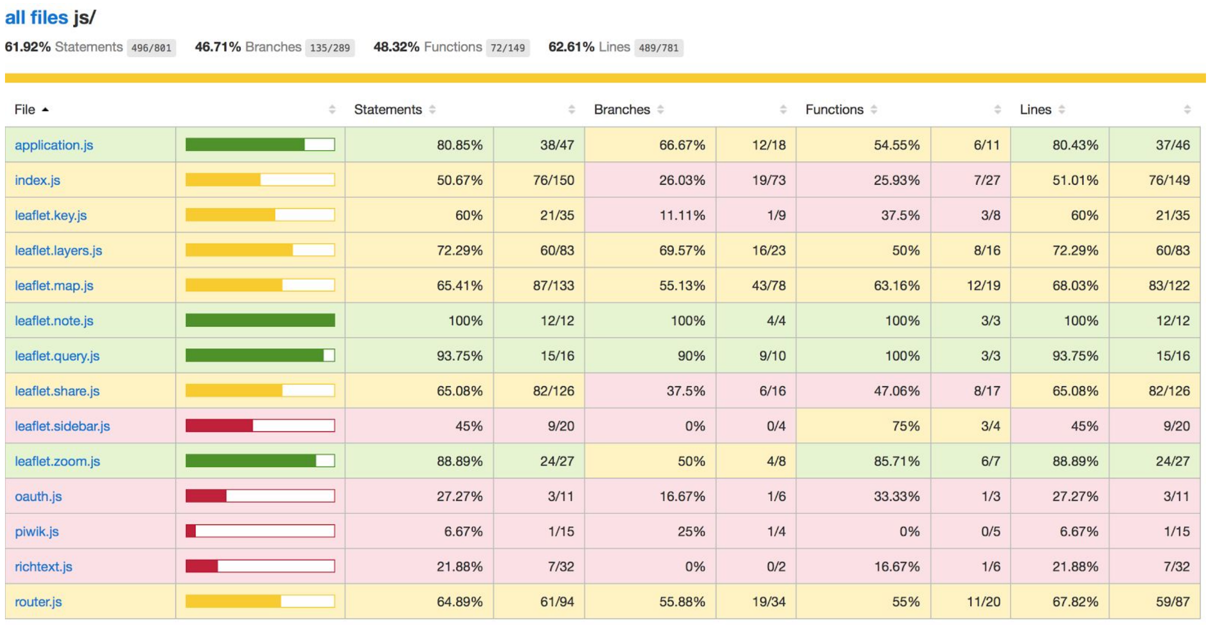


/

42.08% Statements 736/1749 **28.15%** Branches 172/611 **28.65%** Functions 102/356 **42.75%** Lines 728/1703

File ▲		Statements		Branches		Functions		Lines	
js/		61.92%	496/801	46.71%	135/289	48.32%	72/149	62.61%	489/781
js/index/		27.78%	227/817	12.32%	35/284	15.47%	28/181	28.54%	226/792
js/index/directions/		9.92%	13/131	5.26%	2/38	7.69%	2/26	10%	13/130

In this picture, we have about 61% coverage of from our selenium test cases. This covers almost 100% of our main functionality of our website such as zoom in, zoom out, directions, location, map layers. We did not test functions like Login, Signup, and History as they are miscellaneous features unimportant to our web app.



all files js/

61.92% Statements 496/801 **46.71%** Branches 135/289 **48.32%** Functions 72/149 **62.61%** Lines 489/781

File ▲		Statements		Branches		Functions		Lines	
application.js		80.85%	38/47	66.67%	12/18	54.55%	6/11	80.43%	37/46
index.js		50.67%	76/150	26.03%	19/73	25.93%	7/27	51.01%	76/149
leaflet.key.js		60%	21/35	11.11%	1/9	37.5%	3/8	60%	21/35
leaflet.layers.js		72.29%	60/83	69.57%	16/23	50%	8/16	72.29%	60/83
leaflet.map.js		65.41%	87/133	55.13%	43/78	63.16%	12/19	68.03%	83/122
leaflet.note.js		100%	12/12	100%	4/4	100%	3/3	100%	12/12
leaflet.query.js		93.75%	15/16	90%	9/10	100%	3/3	93.75%	15/16
leaflet.share.js		65.08%	82/126	37.5%	6/16	47.06%	8/17	65.08%	82/126
leaflet.sidebar.js		45%	9/20	0%	0/4	75%	3/4	45%	9/20
leaflet.zoom.js		88.89%	24/27	50%	4/8	85.71%	6/7	88.89%	24/27
oauth.js		27.27%	3/11	16.67%	1/6	33.33%	1/3	27.27%	3/11
piwik.js		6.67%	1/15	25%	1/4	0%	0/5	6.67%	1/15
richtext.js		21.88%	7/32	0%	0/2	16.67%	1/6	21.88%	7/32
router.js		64.89%	61/94	55.88%	19/34	55%	11/20	67.82%	59/87