EE461L Phase IV Report
Our Green Routine

Team Puma
Alan Penichet-Paul \\ penichet@utexas.edu \\ https://github.com/Penichet
Kan Vanthanasuksan \\ kanvan@utexas.edu \\https://github.com/kanvan
Donovan Mccray \\ donovan.mccray@utexas.edu \\ https://github.com/DoMarsAccount/
Lauren Jones \\ laurenEjones@utexas.edu \\ https://github.com/lej656
Indy Vermeerscsh \\ indy.vermeersch@utexas.edu \\ https://github.com/IndyVermeersch
Janvi Pandya \\ janvipandya@utexas.edu \\ https://github.com/jpandya1

Team repository \\ https://github.com/jpandya1/TeamPuma

Phase 4 project leader: Donovan Mccray
Website URL: greenroutine.appspot.com

Backup URL: https://tutorial5-255204.appspot.com/

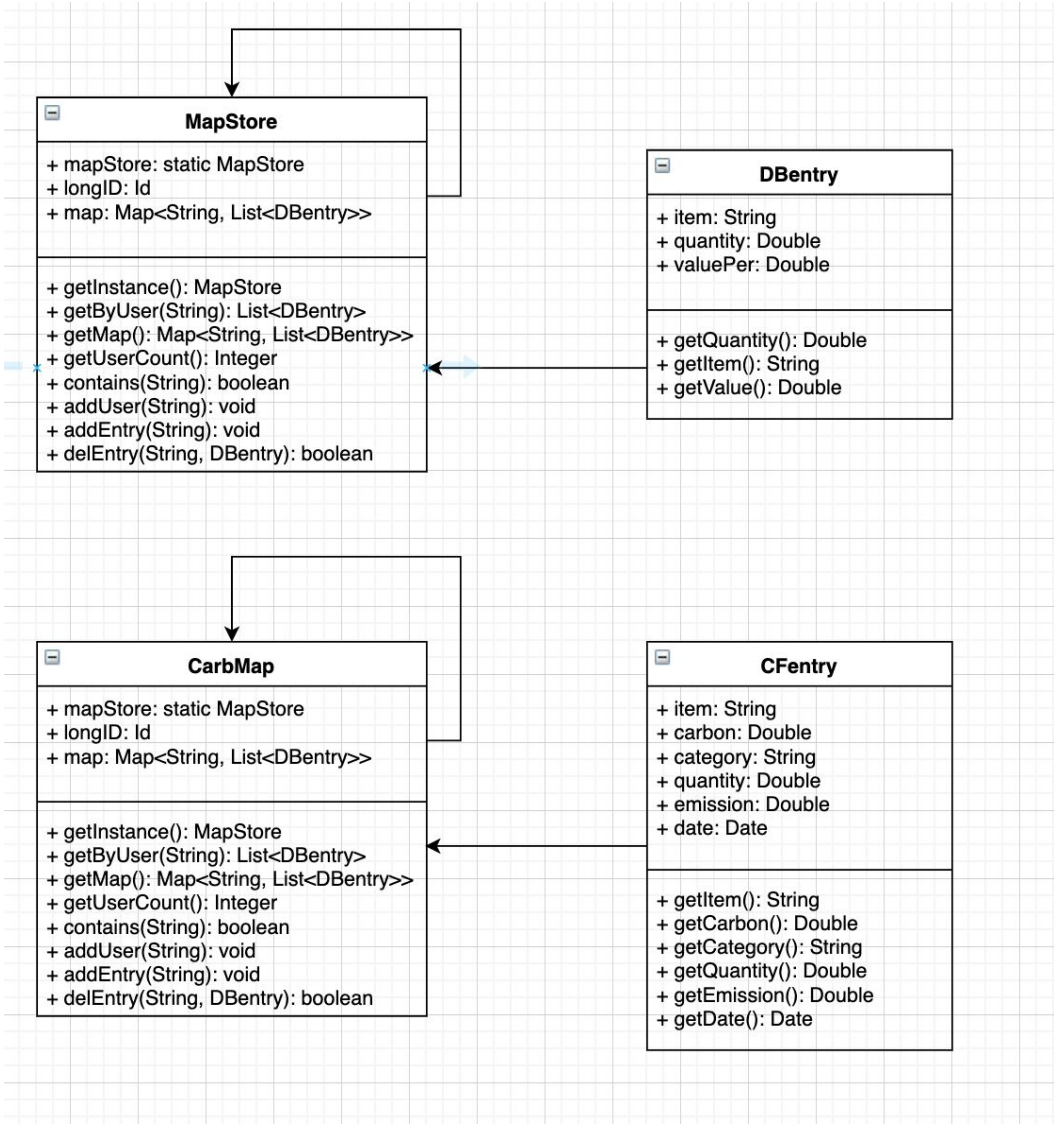
Information Hiding
The queryFetch and RDQueryFetch methods in jsonparser.js handle making calls to the Earth911 API. The only item passed to Fetch methods is a url, based on Earth911's API call structure. The methods providing these urls (get and search methods) have no involvement with the JSON results handled by the Fetch methods. The Fetch methods pass the JSON data that results from their API calls to methods that generate the content displayed on our website. These methods, generateList and generateGalleryOfLinkBoxes, have no knowledge of the API calls that were made to generate the JSON they receive.

This modularization scheme makes the website robust to changes. By separating the various tasks, utilizing the API in different ways became much easier than if one method were to handle everything. With this structure, new content can be created using a method whose only parameter is a JSON object array. And because the API has a shared result structure, each object array is known to contain the same information.

One potential problem stems from the specific design of Earth911's API. If for some reason they were to change the URL structure for API calls, or the content of the JSON array their API returns, our methods would need to be updated. Thankfully, these updates would only need to occur within their respective methods, so targeting the location for these changes would be quick.

Design Patterns:
We utilized a Map structure to store user data for the carbon footprint tracker and the financial calculator. Each user entry will be added to the respective map. Because of this, we have to ensure that only one map is initialized for each the carbon footprint and financial calculator. Thus, we implemented a **Singleton pattern** for the MapStore and CarbMap. There is a method called getInstance() that is called when both the maps are initialized and acts as a global point of access to the instance of the map. The method returns the map instance if it is already created or creates and returns the map if it hasn't. Although this pattern makes the code less efficient, it decreases the likelihood of bugs in the code or important user information being lost if duplicate maps are called and created.

Singleton Pattern UML diagram:

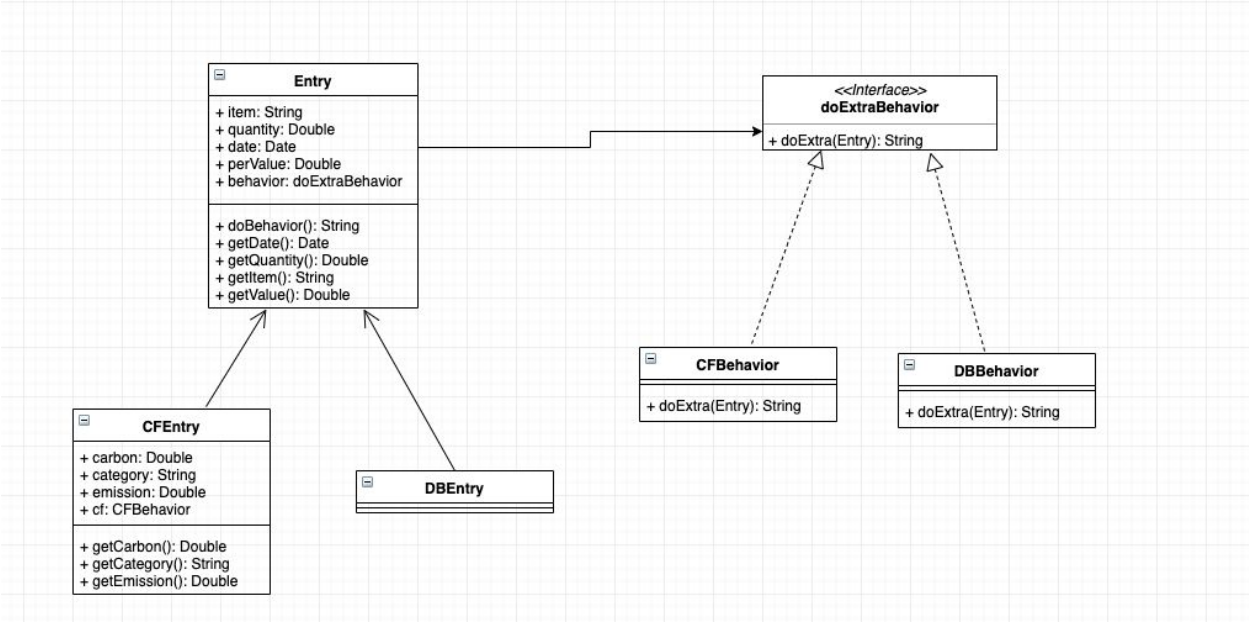


Our second design pattern is the **Strategy pattern**. The changes made were based on trying to hide information from the jsp files calling the financial calculator and the carbon footprint tracker entries. In addition, the financial calculator and carbon footprint classes that take in entries from their respective webpages shared a lot of the same code, so we implemented inheritance to create an Entry class.

The strategy pattern implements a doExtraBehavior interface (specifically the *doExtraStuff()* method) that is exercised by 2 behaviors: the behavior for the carbon footprint calculator (CFBehavior) and the behavior for the financial calculator (DBBehavior). The Entry class contains an instance variable of type doExtraBehavior named *behavior* that is instantiated in the constructor of the specific entry (either carbon footprint entry (CFentry) or financial calculator entry (DBentry)). The Entry class contains a *doBehavior()* method that calls *behavior.doExtraStuff()*. The entry class then executes the behavior of the entry based on what

behavior was previously instantiated. The main advantages of this design is that the method call from the jsp files has no knowledge of the inner workings of the entries, it simply calls *entry.doBehavior()* and the results are executed, regardless of the entry type. A disadvantage of this design is that many more classes were created in order to implement the design. This can get confusing, and is prone to small bugs, as we experienced in our implementation.

Strategy Pattern UML diagram:



Refactorings

Duplicated Code: In jsonparser.js, the methods generateGalleryOfLinkBoxes and generateGalleryOfMatchingMaterials both created itemBox divs to contain recycled material information. The solution to the code duplication was to create a new method, “createLinkBox” that returns a div element. Now the itemBox variable is created by calling this method.

```
function createLinkBox() {
    var itemBox = document.createElement('div');
    itemBox.setAttribute("class", "linkBoxes");

    var title = document.createElement("h3");

    var brk = document.createElement("br"); // for spacing
    var descr = document.createElement("p");

    title.innerHTML = results[j].description;
    descr.innerHTML = results[j].long_description;

    itemBox.appendChild(title);
    itemBox.append(brk);
    itemBox.append(descr);

    return itemBox;
}
```

Comments: The queryFetch method in jsonparser.js had several comments that became redundant over the course of development. For example, the clearSearchList method is named in a way that clearly defines its purpose.

```
function queryFetch(url) {
//    fetch(CORS_PREFIX+url, {mode: 'cors'}).then( resp => {
    fetch(CORS_PREFIX+url, {mode: 'cors'}).then( function(resp){
        return resp.json();
//    }).then(data => {
    }).then(function(data) {

        var response = data;
        var results = response.result;

        if (document.getElementById('APIResponseList').hasChildNodes()) {
            // clear the list of former search results
            clearSearchList();
        }

        if (!Array.isArray(results) || !results.length) {
            // No results were returned for this request
            document.getElementById("APIResponse").innerHTML = "No items were found for your request.";
        } else {
            // Some number of results were returned
            document.getElementById("APIResponse").innerHTML = " ";

            // fill in the list with search results
            document.getElementById('APIResponseList').appendChild(generateList(results));
            console.log(results);
        }

    });
}
```

Deleted unused Code: Unused functions in carbonfootprint.jsp were deleted i.e. updateChart(), edit_row(), save_row(), delete_row(), add_row(), getDate(), totalEmission().

```
373
374         function add_row() {
375             var new_name = document.getElementById("new_name").value;
376             var new_country = document.getElementById("new_country").value;
377             var new_age = document.getElementById("new_age").value;
378             var new_quantity = document.getElementById("new_quantity").value;
379             var new_emiss = new_country * new_quantity;
380             var new_date = new Date().toJSON().slice(0,10).replace(/-/g,'/');
381
382             var table = document.getElementById("data_table");
383             var table_len = (table.rows.length) - 1;
384             var row = table.insertRow(table_len).outerHTML = "<tr id='row" + table_len + "'><td id='name_row" + table_len + "'>" + new_name
                    </td><td id='country_row" + table_len + "'>" + new_country + "</td><td id='age_row" + table_len + "'>" + new_age + "</td><td
                    id='quantity_row" + table_len + "'>" + new_quantity + "</td><td id='emiss_row" + table_len + "'>" + new_emiss +  "</td><td
                    id='date_row" + table_len + "'>" + new_date + "</td><td><input type='button' id='edit_button" + table_len + "' value='Edit'
                    class='edit' onclick='edit_row(" + table_len + ")'> <input type='button' id='save_button" + table_len + "' value='Save'
                    style='display: none' class='line save' onclick='save_row(" + table_len + ")'> <input type='button' value='Delete'
                    class='delete' onclick='delete_row(" + table_len + ")'></td></tr>";
385
386
387             data.labels.push(new_name);
388             data.population.push(new_emiss);
389             data.area.push(new_age);
390
391             document.getElementById("new_name").value = "";
392             document.getElementById("new_country").value = "";
393             document.getElementById("new_age").value = "";
394         }
395         function getDate(){
396             var date = new Date().toJSON().slice(0,10).replace(/-/g,'/');
397             document.getElementById("new_date").innerHTML = date;
398
399         }
400         function totalEmission(){
401             var new_country = document.getElementById("new_country").value;
402             var new_quantity = document.getElementById("new_quantity").value;
403             var new_emiss = new_country * new_quantity;
404             document.getElementById("new_emiss").innerHTML = new_emiss;
405         }
406
407         setInterval(getDate(), 5000);
408         setInterval(totalEmission(), 500);
409
```

Extract HTML Code: Previously we had repeated copies of code for the navigation banner, title banner, and footer for every page. We created a header.html and footer.html file and then used a load function that's available from a JavaScript library, jQuery. This assisted in refactoring so that the snippet of code could be imported and modified from a single source for every website page. The code below is used to import html code for the navigation banner, title banner, and bottom banner.

```
 9     <script
10         src="https://code.jquery.com/jquery-3.3.1.js"
11         integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
12         crossorigin="anonymous">
13     </script>
14     <script>
15         $(function(){
16             $("#header").load("header.html"); //imports navigation and title banner
17             $("#footer").load("footer.html"); //imports bottom banner
18         });
19     </script>
20
21     <body>
22         <div id="header"></div>
```

Rename variables:

We utilized Objectify from Google Cloud to implement our Carbon footprint and Financial calculator page. For this reason, we reused a lot of code from our previous project, specifically the Blog Post. Previously, we used the exact same class called OfySignGuestbookServlet.java to implement the calculator. However, this is an example of a bad and confusing naming

convention. Therefore, we renamed this class to FinancialCalcServlet along with other variables in the class.