Team Umari

CS 410 Project Part 5

18 April 2020

Design Patterns

1. **Singleton**

The singleton design pattern will be utilized for each of the classes: CensicalInterface, Request, and Result. This will restrict the application to a single instance of each class at any given time. This is necessary because we will never want multiple instances of any of these classes. Instead, a single class object should be created and then modified.

For the Interface class, there should only be one instance because multiple interfaces are never needed for a given client. The Request class should be singleton because we never want to have multiple variations of requests being submitted by the same client. The request should contain all relevant information about the query the client submitted, and users are restricted to one request at a time. Even for requests that contain multiple data points, this will still be contained in a single request object. The Result class should also be limited to one instance for reasons similar to the Request class. All of the data for each result should be contained in a single object. Since there will only be one request at a time, there only needs to be one result to match the request.

1. **Iterator**

The iterator pattern will be used to iterate through returned results and display in the Censical interface. This will help the application to get through all returned visualization. This is necessary as our users will be requesting different graph types as desired results, which are based on varied data topics, timeframe and geographic regions. The iterator pattern allows us to access each object of returned results without worrying about implementation and representation differences.

The iterator pattern will focus on the interaction among interface, results and iterators. The aggregate object is visualized results, which comes with two parts -- an interface of Result and a concrete class to implement the interface. Similarly, the interator has an interface and a concrete class to implement the iterator interface.

1. **Adapter Design Pattern** - Converts request into API query and receives API result, converting it to work with CensicalInterface.

The adapter design pattern is used to solve the problem of incompatible interfaces working with each other with an adapter class interfacing between them. For our Censical website, we could use the adapter pattern to adopt our website to the API the government provides on their site. The government API has a set way of servicing requests that we cannot modify, so to use it, we must adapt our program to receive their result. We would create a separate adapter class that would receive the result from the API and then in turn send it to our CensicalInterface.py class in a form that can work with the CensicalInterface.py class.

Another way we could use the adapter design pattern is when adapting the CensicalInterface.py class to send the request to the API. The API has a set way of receiving requests.We are not able to ask the government to change the way their API services requests just to work better with our site. So what we do is create an adapter class to send our request in such a way that the API can service our request.

In a way, we are already using the adapter design pattern in our project because the Request.py and the Result.py serve the functions of acting as an interface between the CensicalInterface.py the government API.