# Project 3 - Web APIs

Course 342 - Fall 2017

Due Date: Midnight, Nov 17, 2017

*Any changes/updates to the project description will be in red -* ***Last Updated 10/25 12:30 p.m.***

## Driver Code and Test Input Files

- N/A

## Grading Rubric

Total: 42 points

- **Program:**
  - API Proxy
    - Uses a separate proxy class to access each API (9 points)
    - Has an API base class or module for shared functionality (3 points)
    - Encapsulates connection details in the API base class ( 3 points)
    - Has fallback for each API in case of call failure (3 points)
    - Uses the 'open_uri' or Net::HTTP module from the Ruby library (1 points)
  - Log Class
    - Uses the 'JSON' module from the Ruby library (2 points)
    - Log class is a Singleton, stores in real time as a json file (6 points)
  - Includes a readme in plain text (txt) explaining the design and how each pattern is used (10 points)
  - *Extra Credit assigned for increased complexity and functionality (5 points max to be determined by TA)*
- **Submission:**
  - (2 pts) Follows requested project structure and submission format
    - each class in a separate file, main() in a separate file, no globals
  - (2 pts) Follows formatting guidelines

## Guidelines

This is an individual lab assignment. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically *to* solve the specific problem you have been given, and you may not have anyone else help you write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

# Web APIs

## Description

One of the primary programming skills you will need in today's software developer landscape is working with web APIs. There are a lot of fun, free API's out there for you to use. Here is a quick tutorial on how accessing API's works in Ruby, and I will also cover the topic in class.

For this project, you are going to build a program that uses at least 3 web API's to generate and validate some data. You are then going to store the data locally in a JSON file.
The following are lists of different API's that you can use:

- https://github.com/toddmotto/public-apis
- https://any-api.com/

Feel free to use any publicly available API, even if it does not appear on this list.

You will need to create several API proxy classes (one for each API you use). From there, your design is entirely dependant on what you want to do with the API calls. Below is an example of the kind of project that would be acceptable:

**Example Program**
Your program generates random information for a user by making a call to the randomuser API (https://randomuser.me/) on initialization, and stores whatever information you deem as relevant. Using a User Proxy class, you encapsulate the connection and attributes of the API call, in addition to a fallback in case of no connection. You could then verify the user's email address with a Proxy class for the mailboxlayer API (https://mailboxlayer.com/). If the email address validates, you then use a weather api (https://www.metaweather.com/api/) to display the current weather for the user's city.

Repeat this until you have at least 10 records (users) a local JSON database. Then prompt the user on the command line to search the database. You may search by whatever attribute you wish, first name, last name, etc. On a successful search, the user can change any one of the records attributes (except the search attribute), then save the record back to the database. This should not create a duplicate record, but instead update the existing record.

Whenever you display a result on the console, you should also send it to a Log class that stores it in a JSON file.

# Requirements

## API Base Class

You will quickly notice that API calls all work similarly. You should create an API superclass or module (your choice) that can be inherited or included in your API Proxy classes. Within your super class you should use the 'open_uri' or Net::HTTP module from the Ruby library.

## Fallback Protection

You must create a fallback in your Proxy class in case the api call fails. You may return a generic value or a message that the api call failed; however, your program as a whole should not fail or crash. You should have some fallback data to use in its place.

## Log Class

You may approach designing the class however you wish, however, you must have a Log class that is a singleton and uses the json module to write out data to a file in real time, i.e. during execution not just when your programs exits.

## 3 Design Patterns

You must use at least 3 design patterns we discussed in class after Midterm I. I have already given you 2: Proxy and Singleton. Other options are decorator, command, component, etc.

In your submission, you should include a readme that explains your program, and how you used each pattern.

# Submission

- Required code organization:
    - Project3
        - Readme

            - Create a readme file containing an explanation of your project structure and any additional information you may need to add, and add it to your project folder
        - Required Classes
            - APIProxy.rb
                - Abstract Class
            - Additional Proxy Classes
                - One for each API
            - Log.rb
            - main.rb
            - *Include any additional files*
                - Any additional classes must go into their own file
- Create a zip archive of your project3 and upload the archive to Blackboard under Project 3.
    - Do not zip the folder itself, only the files required for the project
- You may demo your project by signing up for a demo time, then downloading your archive from Blackboard. Extract your archive, run your code, show your source, and answer any questions the TA may have.