Geotab Software Challenge

Martin Ritchie

# Table of Contents

[Table of Contents 1](#_Toc40653594)

[Introduction 2](#_Toc40653595)

[Smoke Test 2](#_Toc40653596)

[Execution 1 2](#_Toc40653597)

[Execution 2 2](#_Toc40653598)

[Execution 3 (this one was more successful) 3](#_Toc40653599)

[Conclusion 3](#_Toc40653600)

[Code Review 3](#_Toc40653601)

[ConsolePrinter.cs 4](#_Toc40653602)

[JsonFeed.cs 4](#_Toc40653603)

[public JsonFeed(string endpoint, int results) 4](#_Toc40653604)

[public static string[] GetRandomJokes(string firstname, string lastname, string category) 4](#_Toc40653605)

[public static string[] GetCategories() 4](#_Toc40653606)

[public static dynamic Getnames() 4](#_Toc40653607)

[Program.cs 4](#_Toc40653608)

[static void Main(string[] args) 4](#_Toc40653609)

[private static void PrintResults() 4](#_Toc40653610)

[private static void GetEnteredKey(ConsoleKeyInfo consoleKeyInfo) 4](#_Toc40653611)

[private static void GetRandomJokes(string category, int number) 5](#_Toc40653612)

[private static void getCategories() 5](#_Toc40653613)

[private static void GetNames() 5](#_Toc40653614)

[Conclusion 5](#_Toc40653615)

[Refactor Step #1 Add test cases and refactor for JsonFeed 5](#_Toc40653616)

[PrivServJsonFeed.cs 5](#_Toc40653617)

[NorrisJsonFeed.cs 5](#_Toc40653618)

[Refactor Step #2 Move logic from Program.Main to its own object 6](#_Toc40653619)

[Refactor Step #3 Write test cases for the program flow 6](#_Toc40653620)

[Refactor Step #4 Polish 6](#_Toc40653621)

[Final Thoughts 6](#_Toc40653622)

# Introduction

I was given an existing piece of code by Geotab and asked to refactor it to production standards.  The following areas are to be addressed:

Improve this software considering the following aspects;

1. Usability & UX

2. Future Maintenance & Extension

3. Reliability & Quality

There are also likely bugs in the solution that need to be fixed.

Note: This document will read more as a stream of activity instead of a direct analysis of the code. This is to give you more of an idea how I approached the problem then an actual analysis of the program.

# Smoke Test

After loading the project up in Visual Studio 19 I proceeded to run the program and see what the results were. The following are three examples of the output I received:

## Execution 1

Press ? to get instructions.

?

Press c to get categories

Press r to get random jokes

c

Unhandled Exception: System.AggregateException: One or more errors occurred. (Response status code does not indicate success: 404 ().) ---> System.Net.Http.HttpRequestException: Response status code does not indicate success: 404 ().

   at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()

   at System.Net.Http.HttpClient.GetStringAsyncCore(Task`1 getTask)

   --- End of inner exception stack trace ---

   at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)

   at ConsoleApp1.JsonFeed.GetCategories() in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\JsonFeed.cs:line 66

   at ConsoleApp1.Program.getCategories() in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\Program.cs:line 119

   at ConsoleApp1.Program.Main(String[] args) in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\Program.cs:line 30

## Execution 2

Press ? to get instructions.

?

Press c to get categories

Press r to get random jokes

rWant to use a random name? y/n

yWant to specify a category? y/n

How many jokes do you want? (1-9)

How many jokes do you want? (1-9)

Unhandled Exception: System.FormatException: Input string was not in a correct format.

   at System.Number.StringToNumber(ReadOnlySpan`1 str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)

   at System.Number.ParseInt32(ReadOnlySpan`1 s, NumberStyles style, NumberFormatInfo info)

   at System.Int32.Parse(String s)

   at ConsoleApp1.Program.Main(String[] args) in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\Program.cs:line 43

C:\Program Files\dotnet\dotnet.exe (process 15424) exited with code 0.

## Execution 3 (this one was more successful)

Press ? to get instructions.

?

Press c to get categories

Press r to get random jokes

rWant to use a random name? y/n

Want to specify a category? y/n

How many jokes do you want? (1-9)

4

[Count from one to ten. That's how long it would take Chuck Norris to kill you...Fourty seven times.]

Press c to get categories

Press r to get random jokes

rWant to use a random name? y/n

nWant to specify a category? y/n

How many jokes do you want? (1-9)

1

[There is no chin under Chuck Norris' beard, only another fist!]

Press c to get categories

Press r to get random jokes

c

Unhandled Exception: System.AggregateException: One or more errors occurred. (Response status code does not indicate success: 404 ().) ---> System.Net.Http.HttpRequestException: Response status code does not indicate success: 404 ().

   at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()

   at System.Net.Http.HttpClient.GetStringAsyncCore(Task`1 getTask)

   --- End of inner exception stack trace ---

   at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)

   at ConsoleApp1.JsonFeed.GetCategories() in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\JsonFeed.cs:line 66

   at ConsoleApp1.Program.getCategories() in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\Program.cs:line 119

   at ConsoleApp1.Program.Main(String[] args) in C:\Users\mdrit\OneDrive\Documents\GitHub\Geotab\CS-Challenge\ConsoleApp1\Program.cs:line 30

C:\Program Files\dotnet\dotnet.exe (process 4472) exited with code 0.

## Conclusion

There are some major obvious issues with the application. This includes but it not limited to:

* Unhandled Exceptions
* Confusing interface design
* Failure to successfully execute the category path
* Failure to respond correctly to the number of jokes

**Clearly there is a problem with this application and it will have to be looked at more closely**

# Code Review

The JokeGenerator solution has three components.

## ConsolePrinter.cs

This is a simple class that provides the ability to write a string to the console. I am not sure what value it provides beyond a simple Console.WriteLine(string) command. I can understand wrapping up code that deals with external interfaces, but this execution does not really do that.

## JsonFeed.cs

This retrieves data from a JSON API is passed in by the program when the object is created.

These are the following important routines:

### public JsonFeed(string endpoint, int results)

A constructor that is passed in a string containing the url of the api to be used.

### public static string[] GetRandomJokes(string firstname, string lastname, string category)

A routine that returns an array of random jokes (as implemented it only returns a single joke). It also replaces the name of (Chuck Norris) with a new name (firstName + “ “ + lastName). It appears when this function is called by program.cs, the endpoint is set to <https://api.chucknorris.io>.

### public static string[] GetCategories()

A routine that returns an array of categories from a JSON API. It appears when this function is called by program.cs, the endpoint is set to <https://api.chucknorris.io>.

### public static dynamic Getnames()

A routine that returns a random name from a JSON API. It appears when this function is called by program.cs, the endpoint is set to <https://names.privserv.com/api/>. There is a comment attached to this routine that appears to have incorrect parameters and return values.

### Program.cs

Note: This is the startup object for the solution.

This component handles the main flow for the application. It uses JsonFeed to retrieve data from API endpoints, and ConsolePrinter to output results to the console.

### static void Main(string[] args)

The function called by default in .net at the start of program execution. This routine handles the question and answering using the console. All of the questions and preset answers are inline with the code.

### private static void PrintResults()

Uses a variable string[] results as effectively a global variable. This outputs the strings in the array separated by commas, surrounded by square brackets.

### private static void GetEnteredKey(ConsoleKeyInfo consoleKeyInfo)

Parses out a value from console.ReadKey() to get the character. It maps the consoleInfo.Key to the character that it represents. Seems very redundant as KeyChar could be used.

### private static void GetRandomJokes(string category, int number)

This retrieves random jokes from the chucknorris API using the JsonFeed object. It does not seem to be successfully returning the correct number of jokes.

### private static void getCategories()

This retrieves a list of categories from the chucknorris API using the JsonFeed object.

### private static void GetNames()

Retrieves a random name from the privserv.com API using the JsonFeed object.

## Conclusion

There are a lot of problems and needlessly convoluted logic with all of these routines. It is almost as though they have been deliberately written to make things difficult to understand. Having said that, in some ways, it has the right idea on how to do things, just the implementation is not working as intended.

The program is small enough that I am going to work from the outside (the modules) in (main). I also feel that test cases and some basic and correct inline documentation would be a good thing.

# Refactor Step #1 Add test cases and refactor for JsonFeed

The first thing I did was I added xUnit test cases for the JsonFeed file. I put in the test cases how I thought the routine should work.

Here is a test I wrote for the GetName() function

        /// <summary>

        /// test to make sure JsonFeed.GetName() returns a name and a surname.

        /// </summary>

        [Fact]

        public void GetName()

        {

            var result = JsonFeed.GetName();

            Assert.True(result.ContainsKey("name"));

            Assert.True(result.ContainsKey("surname"));

        }

At the same time I made several changes to clean up some details to make the code easier to use/maintain.

This includes

* Using a resource file rather than passing the url
* Changed the types returned to list and dictionaries as appropriate
* Made the whole class static
* Breaking up the privServ json feed from the Chuck Norris JSON feed.
* Passed a struct for the first and last names on the privServ.

The the old JsonFeed has now been replaced with the following:

## PrivServJsonFeed.cs

Static class with single method returning a Person struct with first and last name.

## NorrisJsonFeed.cs

Static class with 3 public methods:

#### public static string GetRandomJoke(string category)

Returns a random joke

#### public static List<string> GetRandomJokes(int numberOfJokes, string category)

Returns multiple jokes

#### public static List<string> GetCategories()

Returns a list of categories

# Refactor Step #2 Move logic from Program.Main to its own object

For this I replaced the logic in the main function to use a loop and switch statement to replicate the functionality. While doing this I fixed some of the major process flow bugs (for example it was not displaying multiple jokes). I also moved this logic to its own module called flow.cs. For writing to and from the console, I decided to wrap up that logic into a Print and Input functions. These functions were then made virtual. The reason for this was several. Firstly it would make writing test cases against this much easier. Secondly it would allow adding different interfaces such as a web interface or a voice interface in the future without affecting this module.

# Refactor Step #3 Write test cases for the program flow

I created another test case module and had it inherit from the flow module. As an example of dependency injection the Print and Input functions were then overridden so that I could easily replicate keystrokes and output from the module for testing purposes. Once this was done I started to write test cases for the simple, and then more complicated flow processes. As part of this I moved the strings used for questions and other outputs into a resource file that could be shared with the test module.

After some work I also cleaned up some of the interface, providing a command prompt, added some white-space, cleaned up some language, added some input validation and generally improved the management of the application. Finally some exception handling was provided to deal with errors that might occur more elegantly.

# Refactor Step #4 Polish

For this I worked on cleaning up some of the interface. I changed some of the text further. Then I added some logic to handle input validation and dealing with errors. There was an infinite loop issue that I also needed to fix.

# Final Thoughts

I hope you enjoyed my approach to this problem. I know I did, and my son enjoyed the Chuck Norris jokes. If there is one thing I hope you take away from my approach it is a focus on test cases and complexity encapsulation. I believe that test cases actually speed up development by clearly thinking about what a piece of code is supposed to do and then implementing it in a stable way. An example of my complexity encapsulation is the way I broke the JSON API feeds up to be simple static function calls. Calling a single function like PrivServJsonFeed.GetName() and getting a struct with a first and last name is easier than setting the url, calling a function, and then receiving a complicated type in return.