**Bookshout Development Project**

**Prepared by**

**Shruti K. Rawool**

**June 5, 2016**

**GitHub link: https://github.com/skrawool/bookshout-project**

**Introduction**

This turned out to be a really interesting project for me. I learned a lot about Ruby programming and this was something that I was willing to learn from a very long time. The tasks in this project gave me an insight into Ruby programming.

The purpose of this report is to demonstrate the workflow that I implemented for successful execution of the tasks.

For writing the code, I did not use any IDE such as IntelliJ IDEA. I used a simple text editor-Notepad++ to write the Ruby and Java programs. I used NetBeans for Java though.

**Task implementation**

This project involves three tasks. The implementation is described separately for each task:

**Task 1: BookShout API Request / Parse**

This task involved parsing a JSON file in Ruby. Initially, I was unsure about the approach, but there were some really useful documentations available online such as ruby-doc.org and tutorialspoint.com that helped me a lot in finding out the methods/functions which were required if we’re parsing a JSON file in Ruby.

We need to use " require ‘json’ " before we parse JSON, so I added this statement. Before this, I ensured I did "gem install json" so that " require ‘json’ " does not return false (I went through the stack overflow link that was provided and I just wanted to be sure I installed the required gems).

Then, to access the JSON file with the help of the REST client library, I wrote:

*url = https://www.bookshout.com/api/books/#{book\_id}.json*

Here, the *#{book\_id}* value would change depending on the argument that we pass when executing the program. Then, for RestClient request, I used the following function:

*response = RestClient.get url, {:accept => :json}*

Now, since we’re passing the book\_ID and the tax\_rate as an argument, in the command line, we need to use "ARGV", which is an input method used to pass values in the script. This is done as follows:

*book\_id = ARGV[0]*

*tax\_rate = ARGV[1].to\_f*

Since the tax rate is a float value, I appended *".to\_f"* to ARGV.

The logic of the program is explained in the following two operations:

Subtotal = current price \* tax rate

Total price = subtotal + current price.

This operation is written in Ruby as follows:

current\_price = ((JSON.parse(response))["current\_price"][1..-1]).to\_f

total\_price = current\_price\*(1 + tax\_rate)

The current price of the book is retrieved using JSON.parse(response), which converts the string to JSON object.

The output which displays the Deception is printed using the puts statement as follows:

*puts "Deception - $#{'%.2f' % total\_price}"*

The output obtained is displayed in the following screenshot:



**Task 2: CSV parse**

In order to parse a CSV file in Ruby, we use Ruby’s built-in standard library by using " require ‘csv’ "

Now, similar to the first task, this program is executed by passing an argument in the command line, which in this task is a CSV file. So, we use the following:

csv\_path = ARGV[0]

I then declared the no\_of\_books variable and the sum\_price variable and initialized them to zero. I also declared the most\_exp and least\_exp variable which will be used to return the most expensive and the least expensive book.

no\_of\_books = 0

sum\_price = 0.0

most\_exp = "None"

least\_exp = "None"

max\_price = -1

min\_price = 99999999

I also used the max\_price and the min\_price variable which are initialized to the given values such that they are set to the price of first book in first iteration. This is because the price cannot go below zero, max\_price will always be less than the book price for first row, forcing it to set the max\_price to first book's price initially and similarly for the min\_price.

The CSV file is read as follows:

CSV.foreach(csv\_path) do |row|

In the program, I ensured I have not included headers, so the preceding command explains the same. Here, csv\_path specifies the path where the CSV file is stored on the system.

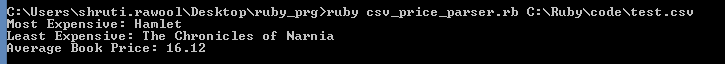
I included three functions that performed the operation of finding out the average book price, the most expensive, and the least expensive book. These values are printed using the puts statement as follows:

puts "Most Expensive: #{most\_exp}"

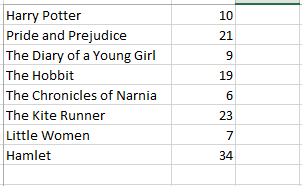
puts "Least Expensive: #{least\_exp}"

puts "Average Book Price: #{'%.2f' % (sum\_price/no\_of\_books)}"

The output obtained is shown in the following screenshot:



The CSV file that I've used (test.csv) consists of eight books and their costs:



**Task 3: Array implementation of Stack**

The *DynamicStack.java* program uses a dynamic-sized stack. In this Java program, I've used the push, pop, top, and isEmpty operations.

The implementation of these operations is explained in the form of comments of the Java program.

The obtained output is as follows:

