**Balance the Loan Books**

**Build from the Source Code**

* Make sure you have installed .net framework
* Make sure you installed visual studio 2019 or higher

Then run the code directly from visual studio

**Input Files**

* Input Folder for small dataset is located at **/Affirm.Challenge.LoanBalance.Startup/Inputs/small/**
* Input folder for large dataset is located at **/Affirm.Challenge.LoanBalance.Startup/inputs/large/**

**Output Files**

* Input Folder for small dataset is located at **/Affirm.Challenge.LoanBalance.Startup/output/small/**
* Input folder for large dataset is located at **/Affirm.Challenge.LoanBalance.Startup/output/large/**

Write-Up Questions

**1- How long did you spend working on the problem? What did you find to be the most difficult part?**

**Ans:** 3hr for coding and 1 hour for testing and little bit refactoring.

Most difficult part was to understand the requirement, and then the testing against the provided data.

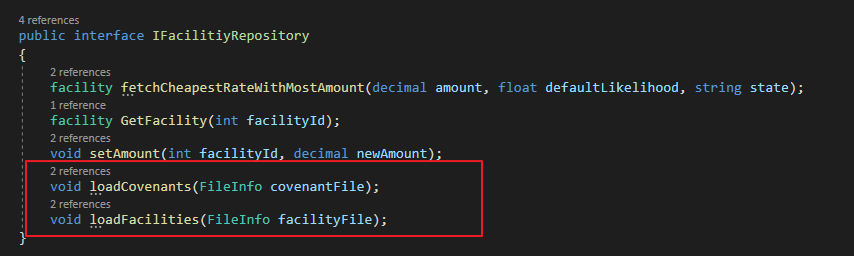
**2- How would you modify your data model or code to account for an eventual introduction of new, as-of-yet unknown types of covenants, beyond just maximum default likelihood and state restrictions?**

* More covenants can be added to the Covenant.cs
* To change the condition, we can add more in
  + **class** “FacilityRepository”
  + **Method:** fetchCheapestRateWithMostAmount”

**3- How would you architect your solution as a production service wherein new facilities can be introduced at arbitrary points in time. Assume these facilities become available by the finance team emailing your team and describing the addition with a new set of CSVs.**

In **FacilityRepository** has methods available to load both covenant and Facilities

Please see the screen shot below



**4- Your solution most likely simulates the streaming process by directly calling a method in your code to process the loans inside of a for loop. What would a REST API look like for this same service? Stakeholders using the API will need, at a minimum, to be able to request a loan be assigned to a facility, and read the funding status of a loan, as well as query the capacities remaining in facilities.**

* Add Api Controller
  + LoanServicesController.cs
    - assignLoan ( Method: PUT, parameter : <Json Loan object >, url: LoanService/assignLoan)
    - getFundingStatus ( Method: Get, parameter : <LoanId >, url: LoanService/assignLoan/1)
    - getFacilityCapicity ( Method: Get, parameter : <facilityID >, url: LoanService/FacilityCapicity /1)

**5- How might you improve your assignment algorithm if you were permitted to assign loans in batch rather than streaming? We are not looking for code here, but pseudo code or description of a revised algorithm appreciated.**

In case of the batch, We could load facilities as already sorted list by interest rate. We could save some time for larger dataset.

**6- Discuss your solution’s runtime complexity.**

Lets say

* Initialization or loading Facilities & Covenants
  + **Facilities** O(F) (F is no facilities)
  + **Covenants**
    - O(C) + O(F)
      * O(C) is for grouping the items by Facility
* Initializing the Loan
  + O(L) where L is number of loan Items
* Assignment :

O(L \* FlogF)