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

Ans: It took me around approx. 2:00 hours completed the assignment.

Most challenging part was to preprocessing and structure of data on startup especially Facility and Covenant dataset, so they can be optimally used while processing each loan.

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?

Ans: For each type of know covenant I would implement concrete implementation using the chain of responsibility pattern or use the factory pattern, so we can easily introduce new covenants anytime in our process and un supported covenants could be either skipped or throw an exception as their default behavior.

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.

Ans: I would maintain facility information as an independent global cache outside the process JVM. Using any distributed cache system we can add/update facility globally while also process/allocating loans simultaneously.

So basic idea would be that facilities data will be updated by a batch job or by calling a REST service, the process will update a global distributed cache. As facility information is stored globally, all facility updates will be immediately available to the loan allocation process as well.

4. Your solution most likely simulates the streaming process by directly calling a method in your code to process the loans inside of the 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.

POST /api/requestLoan

**Input:** {

amount: nnnn

default\_rate: xx

state: XX

}

This api will allow requester to submit the loan request asynchronously, the request will be then sent to the backend service for processing. Api will only return a loan request id, which the submitter will use to check the status of the loan request via a separate REST API.

The reason we want to keep the call asynchronously as the loan allocation and covenant checking would take time as we might have to go through a lot of facilities and their respective covenants. The time complexity of this will increase as more facilities or covenants are introduced in the flow.

The backend service in this flow will allocate the loan and stored the result data into a persistence storage.

POST /api/loanStatus

**Input:** { loanRequestId: xxxxx }

Using the request id returned by the /api/requestLoan api this call will return loan status of the loan request (PENDING, REJECTED, APPROVED) and if approved will provide the facility id and any other necessary details as well.

POST /api/facilityCapacity

**Input:** { facilityId: xxxxx }

This api will return the facility information like capacity for a given facility id.

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.

Ans: If we are processing loan in batches then we can optimize the algorithm to reduce fragmentation, still keep in mind the better yield, in the facility capacity so the facility can give out maximum number of loans.

With batch process we can aggregate the requested loan amount by covenants, and try to match them with facilities for which we can utilities the max capacity of the facility (ideally we would use the full capacity amount for the facility). This way the facility won’t have any small unused amount left which most likely no one would request.

6. Discuss your solution’s runtime complexity.

Loading of Banks = O(b)

Loading of Covenant = O(c)

Loading of Facility = O(f\*c) + f\*log(f) // sorting for the collection

Per Loan Assignment = O(f \* c)