Frequency Algo.
(Solution - Generation)

What all is present inside → (Similar of Carriers)

(minItems, maxItems) → (Similar for Items) (maxCarriers, minCarriers)

(maxDistributors, minDistributors) ⇒ Generate (Picks a Random Number b/w these two and generates that many distributors)

(maxDistributorsPerItem, minDistributorsPerItem)

(minCarriersPerItem, maxCarriersPerItem)

maxSolutionSize

Let's see what are these attributes.

(Step 1) : (carriers, distributors & Items Generated).

Now, time for generating the itemSolutions.

itemSolutions = (# carriers X # distributors X # Items)

Steps are getting added to Algo.

1st (Reader) →
2nd (Pruner) → No implementation is there
3rd (Generator) →
4th (LOGGER)

Enable Frequency type as (Node - CM)

Count Frequency of (Node - CM) combo (together)

Date :.....

Setting Algo as. ⇒ (FREQUENCY_BASED_Algo)

Types are → [LEGACY, FREQUENCY_BASED, (LOCAL_GUIDED_SEARCH)]

An Object called. Frequency Branching Context is created.

&

Frequency Branching Context{

FREQUENCY_BASED_ALGO;
Frequency type = NODE cm;

In Solution Generation Cm Context{

Frequency Branching Context → is set

}

(getSolutions)

Solution Generation Based On Frequency

Solution Generation Local Guided Search.

[Command Design Pattern.]

* Allows to convert actions in Objects.
* these objects contain all the info
  - which method() to call
  - method() arguments
  - who is the receiver of the response of this object. (Next Action)

[Components of Command Design Pattern]

* Command Interface → Interface for all commands
  - execute()
  - getNextStep()
* ConcreteCommand → Actual Implementation of the Command Interface
*

Solution Generation.

1. Command Interface.

```
public interface SolutionGenerationStep<ALGO_CONTEXT, CONTEXT, REQUEST, Response> {

    Response execute(ALGO_CONTEXT algoContext, CONTEXT context, REQUEST request);   // What to return after execute

    SolutionGenerationStep<ALGO_CONTEXT, CONTEXT, REQUEST, RESPONSE> getNextStep();   // Next Action

    void setNextStep(SolutionGenerationStep<ALGO_CONTEXT, CONTEXT, REQUEST, RESPONSE> nextStep);

    AlgoStep getStepName();
```

3

First Step is context Generation ⟹ Frequency Based Algorithm

Then 3 steps are there.

1). FirstStep = ReaderStep.

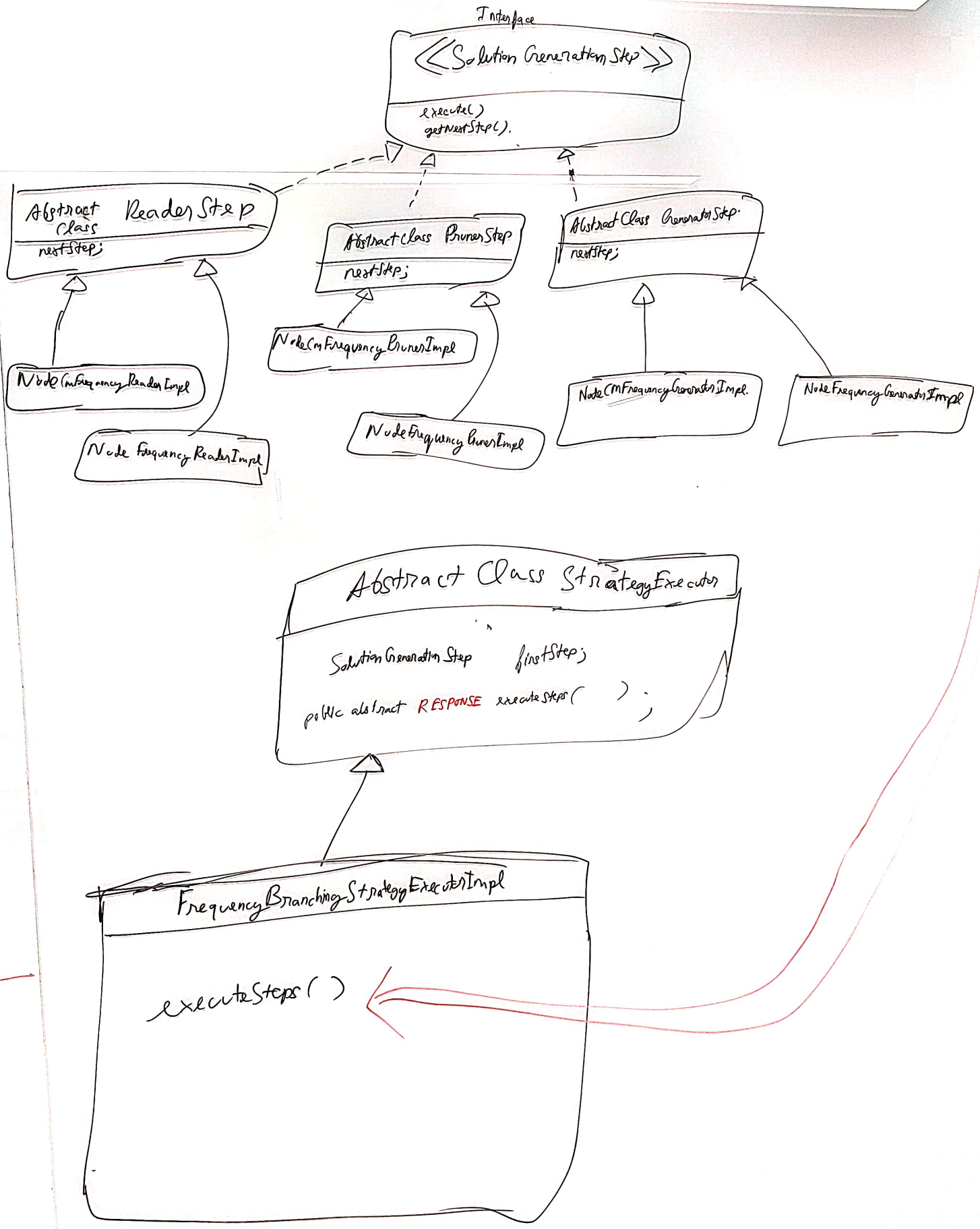
2). 2nd Step = PrunerStep

3). Generator Step.

Twist is there. Now Each of these steps can have their own implementations.

Hence Factory Pattern has been introduced here.

Each Step has an abstract class such that it is able to implement multiple variations.



Now comes the main method.

getSolutions(SolutionGenerationCmContext, SolutionGenerationRequestBucket){

~~if (Objects.nonNull(strategyExecutor.firstStep))~~ {

return strategyExecutor.executeSteps(...);

} else {

this will not be executed as during bean creation firstStep is always == null.

//Here we create the

Creating context → code

1st Step → ReaderStep = ReaderStep.
{ Call ReaderFactory to generate ReadStep
strategyExecutor.setFirstStep(readerStep);
}

2nd Step → Get PrunerStep and set it as next step for ReaderStep = PrunerStep.
{ Call PrunerFactory to generate PrunerStep
readerStep.setNextStep(prunerStep)
}

3rd Step → Get GeneratorStep and set it as next step for PrunerStep = GeneratorStep.
{
Call GeneratorFactory to generate GeneratorStep
prunerStep.setNextStep(generatorStep)
}

then call
strategyExecutor.executeSteps()

Now we have to study execution of each step.

}