Class Diagram Explanation

Graphical user interface, application, Word

Description automatically generated

**Command:**

Java POJO that represents every single step of processing a Signal.

Name – name of the step which will be used to match method of Algo class(Third party class)

Type- a category that represents whether this command is part of Algo class or some other similar class which contains the necessary behavior to execute this command. In future with the rate of growth of new signals, there is a possibility that Algo class can be over loaded with different behaviors. Hence the behaviors can be split among multiple Algo classes. In such case, this type property of the command tells which algo class to be referred to execute the command.

Params – A Map of parameters that can be passed to behaviours that execute the command

**Signal:**

Java Pojo that represents a list of commands for a signal

Signal – id of the signal

List<Commands> - list of commands

**Handler Interface:**

Contains method that handles the execution of a command against a particular algo class that deals with the behavior of the command

**TradingApp:**

SpringBoot Application main class.

**SignalController:**

HTTP Post Rest API that take signalId(Integer) as input path parameter.

Sample **POST** URI: <http://localhost:8081/trading-app/signal/1>

The Rest API that takes an integer as Input in case of successful execution of Signal. Else it returns an Internal Server Error with related exception.

**SignalProcessor( Implementation of ISignalHandler):**

SignalController calls SignalProcessor to process the signal.

Input: signalId (Integer) and an optional body of parameters **(For Future use: to handle overriding input or similar use cases in future)**.

This class has an association to **SignalConfig** and **CommandExecutor** described below.

Once the SignalProcessor receives right signal spec for the given sigId, it passes the list of commands to the commandExecutor to execute those commands.

Once the set of commands for the given signal are executed, this class executes doAlgo command as the last step of execution for the input signal.

**SignalConfig:**

This class contain method(getCommands) that takes signalId as input and returns the list of matching commands for the given signalId.

SignalConfig refers to signalConfig\*.json that contains the json representation of list of Signal objects.

These json files will be available under signal-config folder under resources folder.

signalConfig-default.json contains the command configuration for default handling of a signal.

Since the addition of new signals keep growing at a rate of 50 per month, once the number reaches 100, the specification of signals can be spread against multiple config files with a suffix of 1-100 or 1-500 etc.

This way once a signal arrives, the suffix of a file name can be used to compare against signalId to pick the right file to load the signal spec.

Lets say a signal with signalId 10 arrives, the config file signalConfig-1-100.json can be picked up as 10 being part of the range 1-100.

For now the logic to select a file is not implemented as represent as TODO in the comments of method(getConfigFileName)

of this class. For now this method simply returns signalConfig-1-100.json as that is the only file exists in the system.

Once the signalConfig loads all the specs in the json file, it matches each spec against signalId and returns the right spec that contains list of commands to execute this signal.

Maintaining the spec in json file is much scalable option rather than creating factory instances for each spec. Also the adding of new signal is easy with quick turn around and without touching the code.. it will be adding the spec in the appropriate sibnalConfig json file.

This way the code need not be updated until and unless these exists any change in Algo class with respect to addition of behaviors or changing the input or outpit of existing behaviors.

Since all the behaviors does not return anything, I have used void as returnType for all the methods processing the given signal. Hoeever in future, when a case where some of the behaviors return some data, then the return type of all these newly implemented methods can be changes to a generic Response class.

**CommandExecutor:**

This is a default executor that executes commands in given order. For every command, it looks for the corresponding Adapter class(represented by the type property of **Command** class) to execute the command and calls it.

For now since only one **Algo** class is available and **AlgoAdaptor** class that implements **IAdaptor** interface is created

**AlgoAdaptor:**

This the adaptor class that executes the command of a signal against the method of the actual third party algo class.

A post construct method called *loadHandlerMap* is implemented to load the map with key as command name(which will be used to match method name of Algo class) with value as i dynamic implementation of handler object handling the execution of actual method inside **Algo** class.

Separate handlers can be created for each of the method of Algo class. However due to the simplicity of methods (with in the scope of this assessment), they are implemented as dynamic handler being part of a lookup map.

Once the signal is executed successfully by this call chain, 200 response will be returned by the user with a **“Success”** message.

Any exception during the call chain will be captured by appropriate exception classes and returned with HTTP status code as 500 along with the exception message.

Flow of Request:

SignalController

SignalProcessor

SignalConfig

CommandExecutor

AlgoAdapter

Algo

Config jsons…….

**Major Benefit of this design:**

1. With this design, I have implemented a new signal **10** by just adding new config spec to signalConfig-1-100.json file without making any changes to java code.

POST: <http://localhost:8081/trading-app/signal/10>

1. Extracted and centralizing common functionality and delegated dynamically changing data to configuration files.
2. Applied Single Responsibility Principle so that each class deals with its own purpose of creation
3. Applied Open Close Principle by using IAdaptor Interface, ICommandExecutor Interface, ISignalHandler interface
4. Used interface base implementation so that objects talk to each other using behaviours exposed through interfaces rather than depending on the class implementation

JDK 11 is used for development of this solution

Spring Boot version 2.7.3 has been used.

Junit 5 is used to implement unit tests along with mocking

**Assumptions:**

1. Commands of a signal are assumed to be executed in the sequential order( the order that is given in the problem statement)
2. Every signal is assumed to be a series to commands that need to be executed against behaviors of Algo class.
3. Structural/Behavioral changes to Algo class are assumed to be minimal.
4. Algo class is created as part of a separate maven module trading-algo. All the active code of this solution is in the module trading-app which contains dependency to trading-algo
5. Security aspect of HTTP API is assumed to be not within the scope of this assessment as no relevant requirements are provided as part of problem statement.

Created this design document as a reference to understand solution design

Refer to Questions.docx in the root folder of the project which contain questions/assumptions after the initial brief analysis of the problem statement.