**Chain of responsibility pattern**

**Overview**

Chain of responsibility design pattern is very simple pattern to understand. So, a bunch of objects in a collection of objects talk to each other. Every object would have different responsibility, so client’s request is transferred to chain of object to process. In turn, whatever object is qualified to solve the problem in chain of objects would process that request.

A simple example would be simple mathematical problem such as addition , deletion, divide and multiplication processes run by 4 objects. But, that scenario is too simple to understand.

**Benefits**

* Each object would not need to know about logic of other object
* Duties can be reassigned, deleted, modified, added and it doesn’t impact in huge for whole codebase.
* It increases number of requests that can be processed as each object is added in chain.
* Number of concurrent processes can be run at same time so it increases speed of application if used properly, and decoupled process supports microservice architecture.

On flip side, many objects would interact with object that builds this chain and it increases lot of objects creation; otherwise it's a fantastic approach to decouple processes in codebase.

**Problem/Scenario**

A problem that can be solved using this pattern would be 4 database operations that can be run through backend code such as SELECT, INSERT, DELETE and UPDATE operations and update student database table where these 4 operations are repetitive and we don’t want one operation to stagnate other request so these 4 operations could run concurrently behind the scene.

With refactoring step, this will be well defined with lot of objects, but it is worth investing time to learn this as it decouples concerns of SQL queries.

Refactor would this problem by introducing multithreading in codebase where I would use 4 objects to run these operations by what is asked from command if user is looking for 'SELECT' query, only SelectQuery object would be assigned to do that task. This approach will separate concerns of objects by their assignment of task which increases speed/performance. Also, if we don't need to run particular query either we don’t send that command or delete objects related to run that command.

**Before Refactor**

With normal flow, one object would run 4 possible methods to update, delete, select, insert with database queries attached inside. This would increase time to do so specially when there are multiple operations in table.

**After Refactor**

With refactoring to chain of responsibility pattern 4 database operations could be run by 4 objects which will be DELETE, SELECT, UPDATE, INSERT objects which take request through SQL query processor object and redirect request to respective object through chain of responsibility. I have used ENUMS so that user’s request can be captured and request processor would process it through relevant object through chain of objects jumbled together.

* If request equals to DELETE, process it through DeleteRequestProcessor
* If request equals to UPDATE, process it through UpdateRequestProcessor
* If request equals to SELECT, process it through SelectRequestProcessor
* If request equals to INSERT process it through InsertRequestProcessor
* Codebase will use StudentRequestChain object to build requestChain then pass it onto objects in chain,and objects do the process if one of them is responsible to fulfil the request

Before/after link

<https://github.com/sppanday/S120-PRT583-Group-A/tree/feat/design-pattern/s260598-PandaySurendra/Sprint-3-Deliverables/Task024_Chain_of_responsibility/ChainOfResponsibility/ChainOfResponsibility>

**UML Diagram**

Please find it inside relevant project UML in pattern folder

/chainofresponsibility.png

**Reference**

Freeman Eric., Freeman Elisa. 2004. 1st ed. O’Reily Media Inc. 1995, Sebasttopo, CA