Command

- 將物件的操作(Operation)參數化，讓Client可以針對操作進行進一步的處理(如Job Queue, Logging或者是Undo)

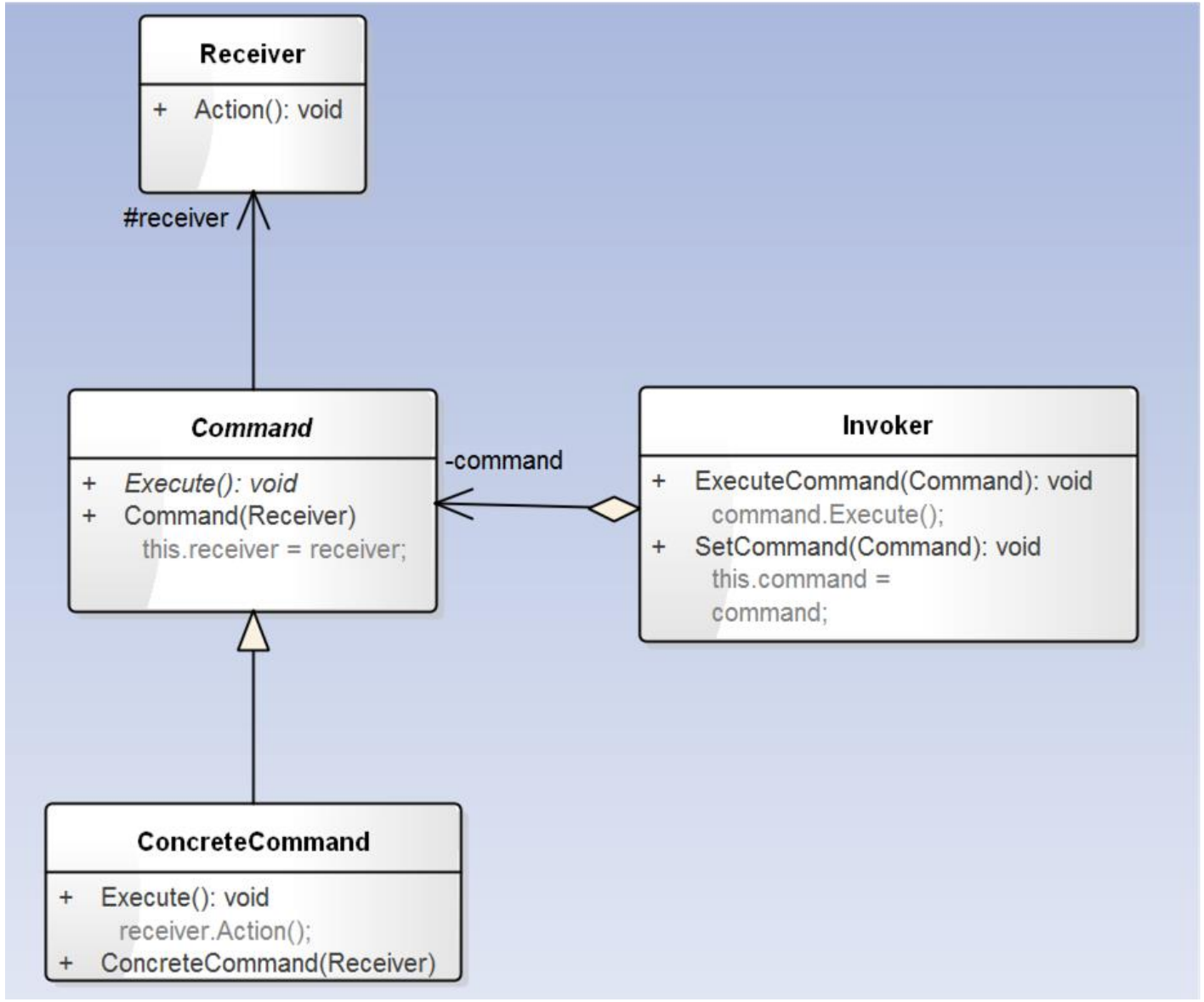
問題描述

將物件的複雜運算邏輯包裝起來，透過Invoker物件去執行運算，並且把運算當作是物件進行傳遞

解決方案

一般來說，Command Pattern會運用在當物件的Operation可以讓使用者自行去擴充時，此時，可以定義一個Command的Abstract Class(或是Interface)
把未來需要擴充的類別當作是Receiver傳遞給Command物件
把Command物件丟給Invoker來決定何時執行

結構


Receiver
+ Action(): void
#receiver
Command
+ Execute(): void
+ Command(Receiver)
this.receiver = receiver;
-command
Invoker
+ ExecuteCommand(Command): void
command.Execute();
+ SetCommand(Command): void
this.command =
command;
ConcreteCommand
+ Execute(): void
receiver.Action();
+ ConcreteCommand(Receiver)

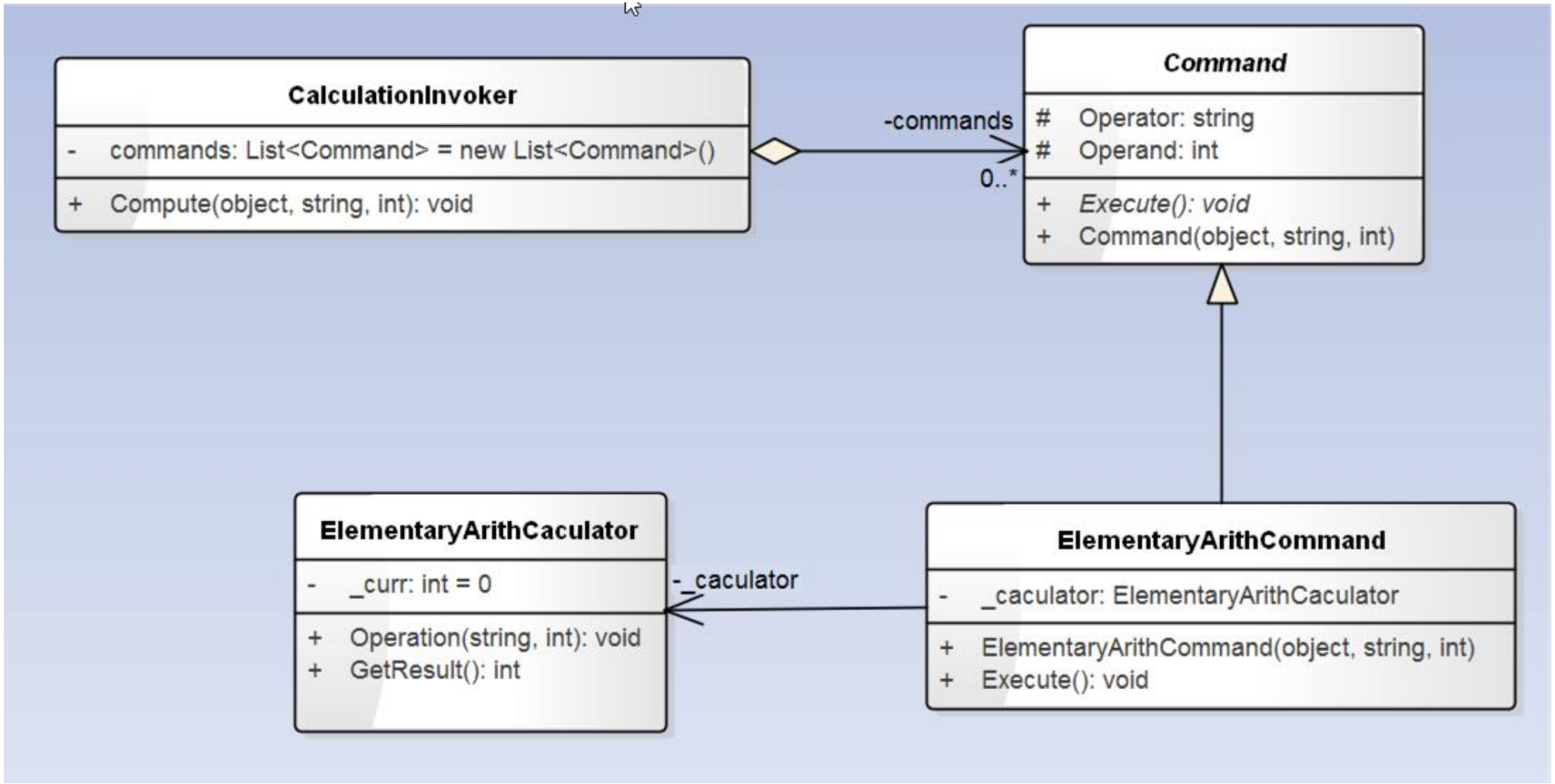
說明

Invoker物件會Hold住所有執行的Command，因此，可以透過Invoker決定是否要Undo或是Redo某些特定的Command
Command的執行(Execute)需要透過Receiver來進行
在.NET中的delegation也是某種程度的Command Pattern的實作

案例描述

建立一個簡單的四則運算的計算機，並且預留未來可以進行更複雜的運算(如三角函數的運算)。
四則運算的邏輯，透過ElementaryArithCalculator來實作，Invoker只要Hold住進行的Command即可(包括運算元(數值)與運算子(+, -, *, /))
Command的執行透過ElementaryArithCalculator的Operation來運作，並且把最終結果記錄在ElementaryArithCalculator中
ElementaryArithCalculator是一個Stateful的物件，在Web端，可以記錄在Session當中

簡易計算機


CalculationInvoker
- commands: List<Command> = new List<Command>()
+ Compute(object, string, int): void
-commands
0..*
Command
Operator: string
Operand: int
+ Execute(): void
+ Command(object, string, int)
ElementaryArithCaculator
- _curr: int = 0
+ Operation(string, int): void
+ GetResult(): int
-_caculator
ElementaryArithCommand
- _caculator: ElementaryArithCaculator
+ ElementaryArithCommand(object, string, int)
+ Execute(): void