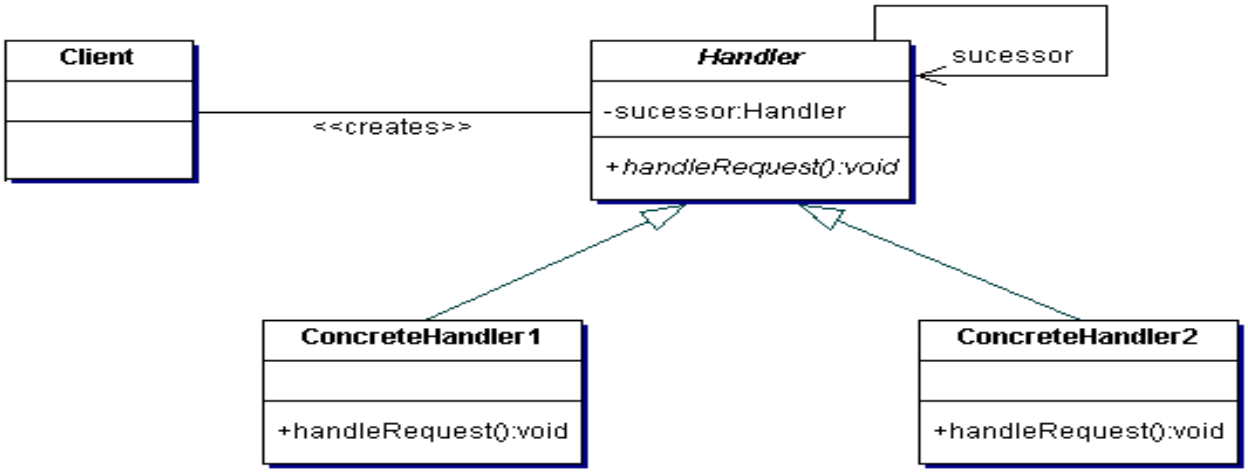
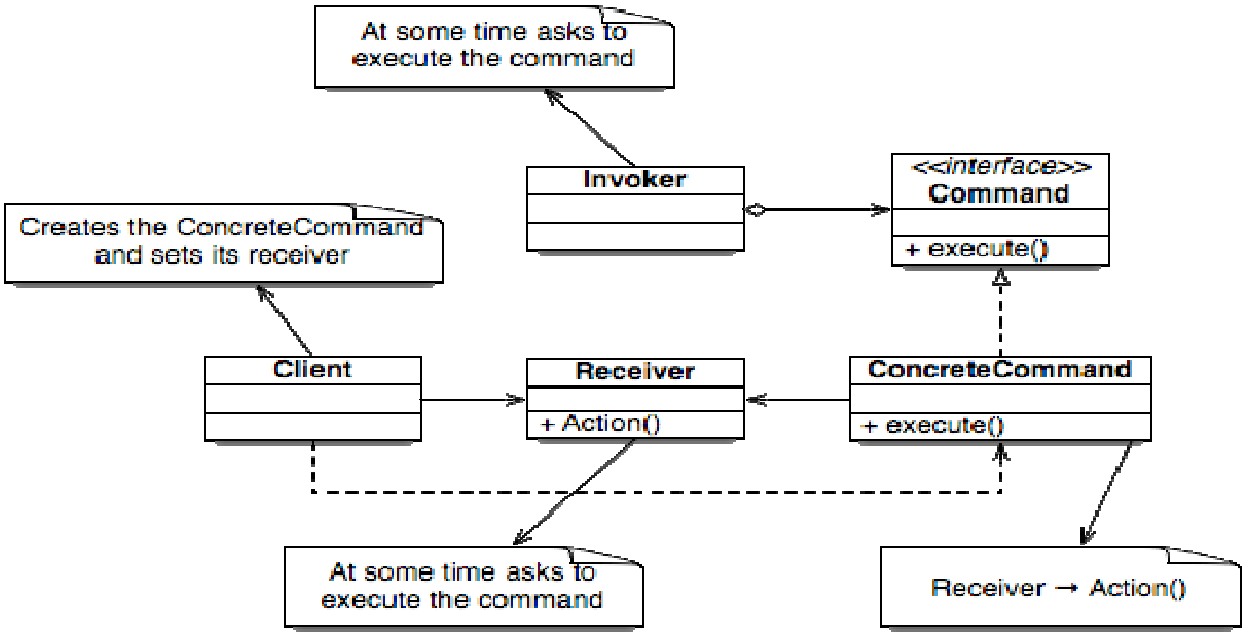
Software design patterns: 2010 – 2011
(exercise 5 – Chain of Responsibility, Command & Interpretor)

Поведенческите шаблони са насочени към алгоритми и разпределяне на взаимоотношения между обектите. В това отношение:

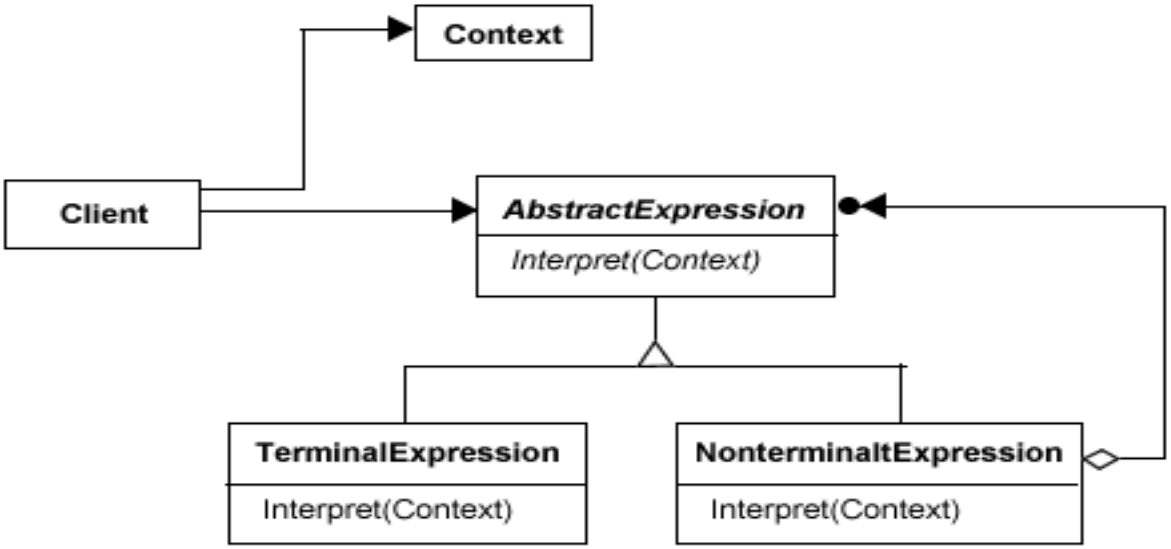
The Chain of Responsibility: позволява чрез неявно изпращане на заявки към даден обект чрез верига от команди



Command pattern: позволява ясно разграничаване на заявките (командите) капсулирани в обект, който може да се предава като параметър, който да се обработва и репродуцира (в run-time)

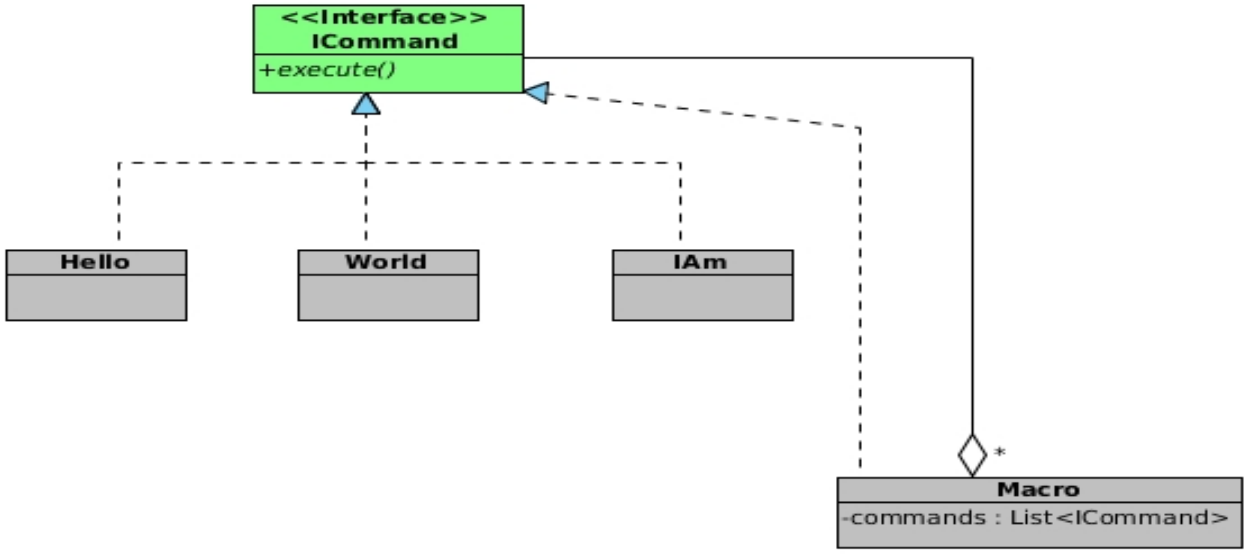


Interpreter: предсатвя граматика като йерархия от класове и имплементира интерпретатор като операции върху инстанции на тези класове



Задача 1:

В ***simpleCommandPattern*** има един единствен клас с коментирам сорс код.
Той отразява и тесства една семпла имплеметация на конадния шаблон, използвайки и т.н. макро-част от шаблона, изпълняваща поредица от команди.
Нека имплементирме така задачата, че като пуснем сорс кода в тестовия клас, да тръгне,използвайки следната UML нотация:



Като за конкретните команди, нека просто изписват на конзолата текст. Примерно :
Hello → “Hello”
World → “World”
Iam → “I am command pattern”

Macro класа има и add(ICommand command) method добавяща в списъка нова конада, която ще се изпълни в execute() функцията

Задача 2:

В предоставения сорс код в папка: ***designPatternsExplainator***
може да се види експертна система за шаблони за дезайн, позволяваща добавяне и принтиране на шаблоните, който тя съдържа, чрез въвеждане на команди от командния ред, когато се стартира тестовият клас: ConsoleTest
който за примерно начало добавя Singelton и Bridge шаблоните и принтирайки ги като ги сортира съответно по подадени параметри.

Командите са: print / add
Синтаксисът на командите е:

print var1, [var2], [var3] [sortby varN] (N = 1, 2, 3)
add designPatternName category explanation

където: varN е: designpatternname category explanation

Примери:

// добавяне на нов шаблон
: add Singelton creational Define a single instance of a class

: add Bridge structural Separate concrete commands in an instance of a class and allows executing of it in run-time

// Принтиране на шаблоните (категорията, името и значението) сортирани по име
: print category designPatternName explanation sortby designPatternName

Също така имаме и команда help, с която може потребителя да си помогне със синтаксиса и значението на коя да е команда. Примерно използване:

help add

help print

Използваните шаблони в системата са:

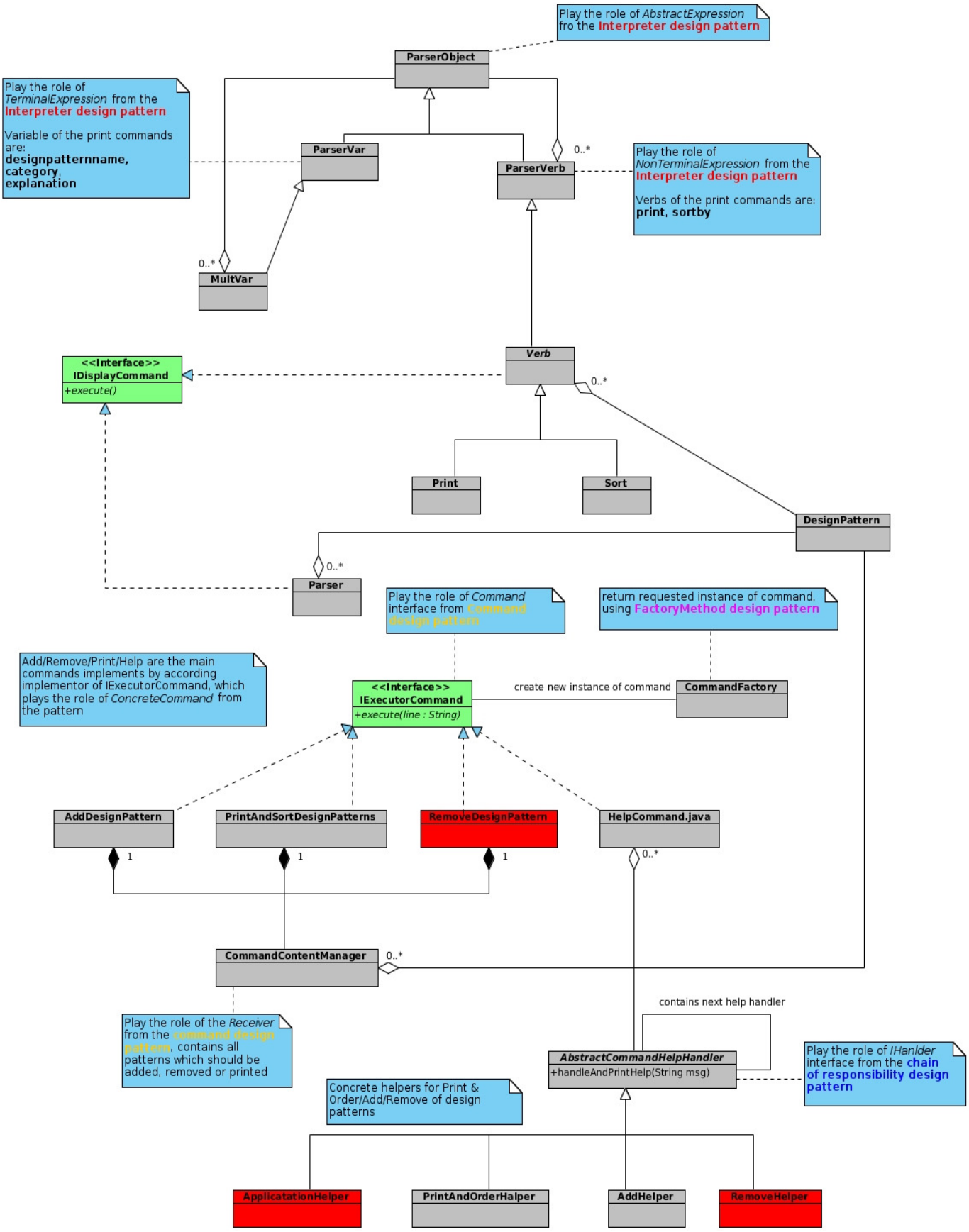
→ ***Chain of Responsibility***
→ ***Command***
→ ***Interpreter***
→ ***Factory Method***

Използваните Задачата ни ще е да добавим фунционалност, позовавайки се на изброените шаблони, с която ще можем и:

- изтриваме вече добавени шаблони за дезайн
- добавим помощни класове за командата изтриване: remove designPatternName и друг помощен клас: ApplicationHelper, който фактически ще е в дъното на веригата във веригата от отговорници и ще изписва информация на конзолата (както всички други помощтни класове) обща информация за всички команди. Нека се активира с въвеждането само на help командата на конзолата

За целта:

1.1. Разгледайте следната диаграма, отразяваща цялата система в UML анотация. Червените класове ще бъдат довбавени от вас, за да допълним липсващата функционалност описана по-горе. Ръководейки се от коментарите и йерархията от класове, открийте четирите използвани шаблона и направете връзка с конвенционалната нотация на съответните шаблони, предоставена в началото на упражнението


Play the role of AbstractExpression fro the Interpreter design pattern
ParserObject
Play the role of TerminalExpression from the Interpreter design pattern
Variable of the print commands are:
designpatternname,
category,
explanation
ParserVar
0..*
ParserVerb
Play the role of NonTerminalExpression from the Interpreter design pattern
Verbs of the print commands are:
print, sortby
0..*
MultVar
<<Interface>>
IDisplayCommand
+execute()
Verb
0..*
Print
Sort
DesignPattern
0..*
Parser
Play the role of Command interface from Command design pattern
return requested instance of command, using FactoryMethod design pattern
Add/Remove/Print/Help are the main commands implements by according implementor of IExecutorCommand, which plays the role of ConcreteCommand from the pattern
<<Interface>>
IExecutorCommand
+execute(line : String)
create new instance of command
CommandFactory
AddDesignPattern
PrintAndSortDesignPatterns
RemoveDesignPattern
HelpCommand.java
1
1
1
0..*
CommandContentManager
0..*
contains next help handler
Play the role of the Receiver from the command design pattern, contains all patterns which should be added, removed or printed
AbstractCommandHelpHandler
+handleAndPrintHelp(String msg)
Play the role of IHanlder interface from the chain of responsibility design pattern
Concrete helpers for Print & Order/Add/Remove of design patterns
ApplicatationHelper
PrintAndOrderHalper
AddHelper
RemoveHelper

1.2. Додавете клас **RemoveDesignPattern**, (конкретна команда) който използвайки инстанция на класа CommandContentManager изтрива шаблон, използвайки иемто му. Нека синтаксиса на командата е:

remove designPatternName

Незабравяйте да го добавете и като връщаша инстанция във факторния метод на класа: **CommandFactory** . И в commandNames променливата в CommandLineHandler класа, която прави проиверка за съществуването на командата

1.3. Нека сега добавим и helper клас (**RemoveHelper**) за изтриването на шаблон в системата, която да обясни синтаксиса и занчението на командата, позовавайки се на Chain of responsibility шаблона (ще се активира при въвеждане в конзолата на следдната команда : help remove). Добавете също така и **ApplicationHelper** клас, изписващ в конзолата обща информация за командите и значението им. Активира се при въвеждане в конзолата само help.

Незабравяйте в **HelpCommand** класа да додефинирате последователността на помошнтиците (Helper класовете).

1.4. Тествайте така имплентираната система. Примерно въведете следната последователност от команди:

→ *add Facade provides a simplified interface to a large number of classes*
→ *add Prototype make a copy of a state of a class and return new instance of the class with this copy*
→ *print category designPatternName explanation sortby designPatternName*
→ *help remove*
→ *remove Prototype*
→ *print designPatternName sortby designPatternName*

1.5. (Дискусии)

1.5.1. Възможно ли е изпозлването на защитно пълношощно (protection proxy design pattern) за всеки от командите, имащо за цел да 'скрие' в себе си конкретната имплементация на съответната команда и да предаде на нея бизнесс логиката, като първо провери синтаксиса на командата в конзолата?

1.5.2. Кои от класовете биха могли да бъдат Singleton ?