Högskolan i Gävle

Miniprojekt 3

Dokument

Hanna Medén, Niklas Nordgren

2020-02-09

Miniprojekt 3 i kursen Objektorienterad design och programmering II, DVG503

Innehållsförteckning

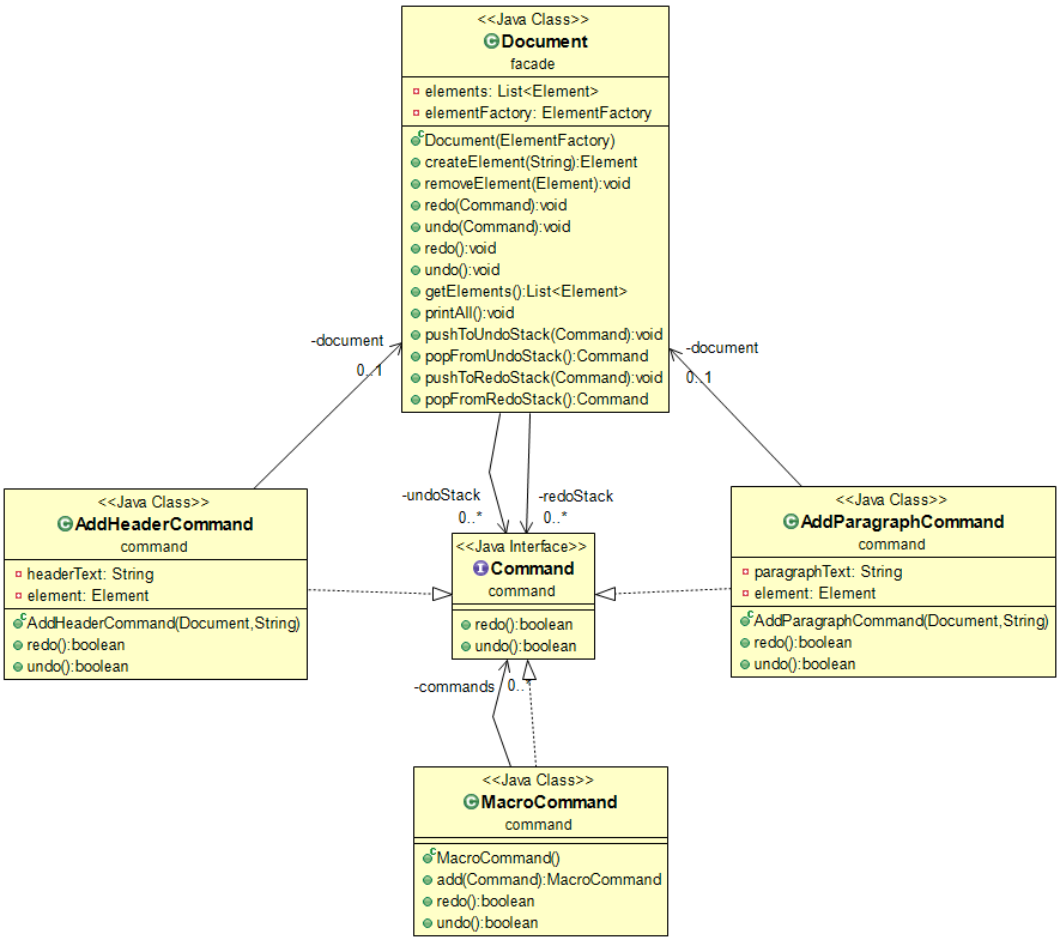
1 Inledning 1
2 Metod 2
3 Resultat 5
4 Diskussion 6
5 Bilaga – UML för systemet 7

1 Inledning

I detta projekt skulle det byggas på funktionalitet på det tidigare projektet där vi skulle skapa ett backend system för en dokumenthanterare som kan skapa olika beståndsdelar av ett dokument. Nu skulle detta förbättras på så sätt att det även skulle finnas konverterare, som kan konvertera dokumentet och dess innehåll till exempelvis HTML, LATEX eller MarkDown. Det skulle även vara möjligt att göra ändringar på dokumentet och sedan även ångra dessa genom att lagra vilka ändringar som gjorts i dokumentet.

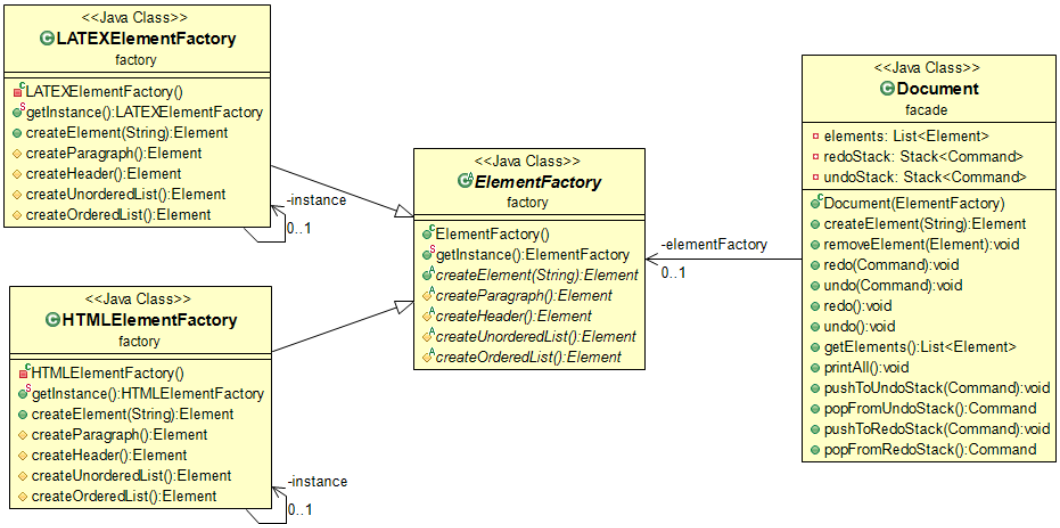
2 Metod

Inledningsvis implementerades designmönstret Command genom att skapa ett `Command` interface innehållande två metoder `redo()` och `undo()` som används för att göra ändringar i dokumentet och sedan även ångra ändringarna. Detta görs genom att systemet lagrar ändringar av dokumentet i två olika stackar, med namnen `redoStack` och `undoStack,` den ena sparar de ändringarna som utförs och den andra sparar de ändringar som ångrats. Nedan i Figur 1 visas de entiteter som tillsammans bygger upp designmönstret Command i systemet.



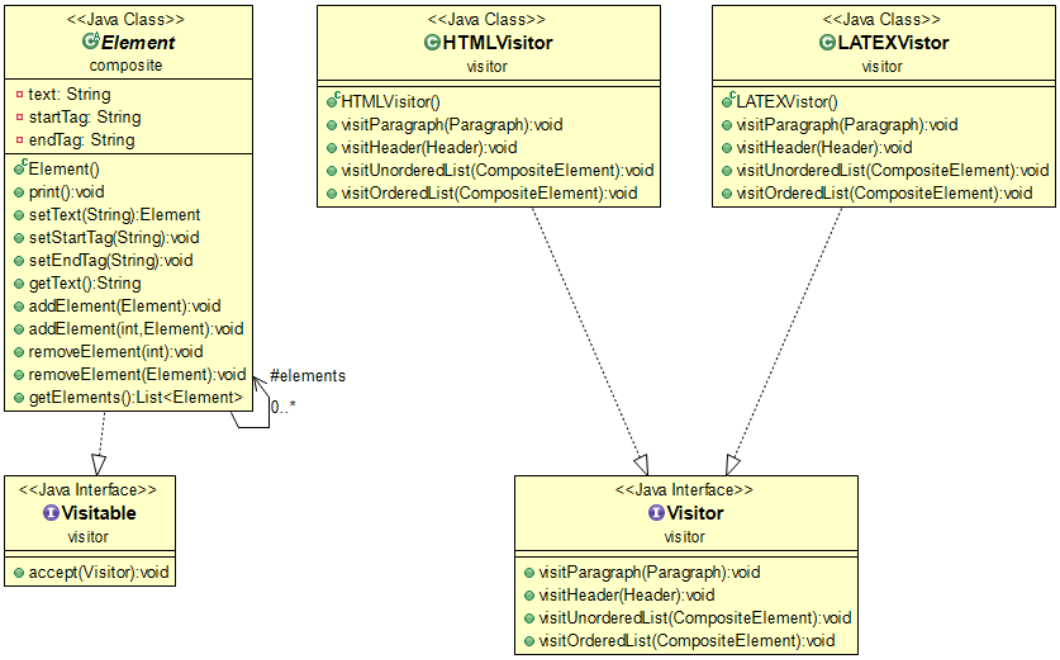
Figur 1: De ingående entiteter som utgör designmönstret Command i systemet.

Ett annat designmönster som introducerades i systemet var AbstractFactory, vilket gjordes genom att skapa två nya klasser som ärver från en abstrakt klass `AbstractFactory`. Namnet på dessa klasser är `HTMLElementFactory` och `LATEXElementFactory` som har i uppgift att skapa nya instanser av element. Se Figur 2.



Figur 2: De ingående entiteter som utgör designmönstret AbstractFactory i systemet.

För att konvertera texten i dokumentet mellan olika textrepresentationer användes designmönstret Visitor. Visitor delen av systemet består av två stycken interface, `Visitor` och `Visitable` där klasserna `HTMLVisitor` och `LATEXVisitor` implementerar Visitor interface:et och klassen `Element` implementerar interface:et `Visitable`. Anledningen till dessa implementationer var att varje Element skall kunna besökas av ett objekt av typen `Visitor` som i sin tur kan verka på objekt av typen `Element` utan att behöva göra runtime kontroller. De ingående entiteter för designmönstret Visitor åskådliggörs nedan i Figur 3.



Figur 3: De ingående entiteter som utgör designmönstret Visitor i systemet.

Vi gjorde även om klassen `Element` under detta projekt, två attribut av typen `String` adderades, `startTag` och `endTag`. Syftet med dessa var att förenkla konvertering mellan texttyper då man inte behöver söka igenom ett elements innehåll för att förändra dess typ. Detta förutsätter dock att metainformationen av ett elements finns innan och efter själva innehållet, som till exempel HTML.

3 Resultat

Slutresultatet innehåller det system från Miniprojekt 2 där man skapar ett `Document` som agerar som en Facade och håller i en lista av alla element och kan även skapa de olika typer av element, header, paragraf etc. Det har sedan utvecklats så man kan skapa element i format av HTML eller LATEX direkt, man kan även skapa ett dokument med dess element som vanligt – utan format - och sedan, med hjälp av en Visitor kan man konvertera detta till det ena formatet och sedan det andra om man så önskar.

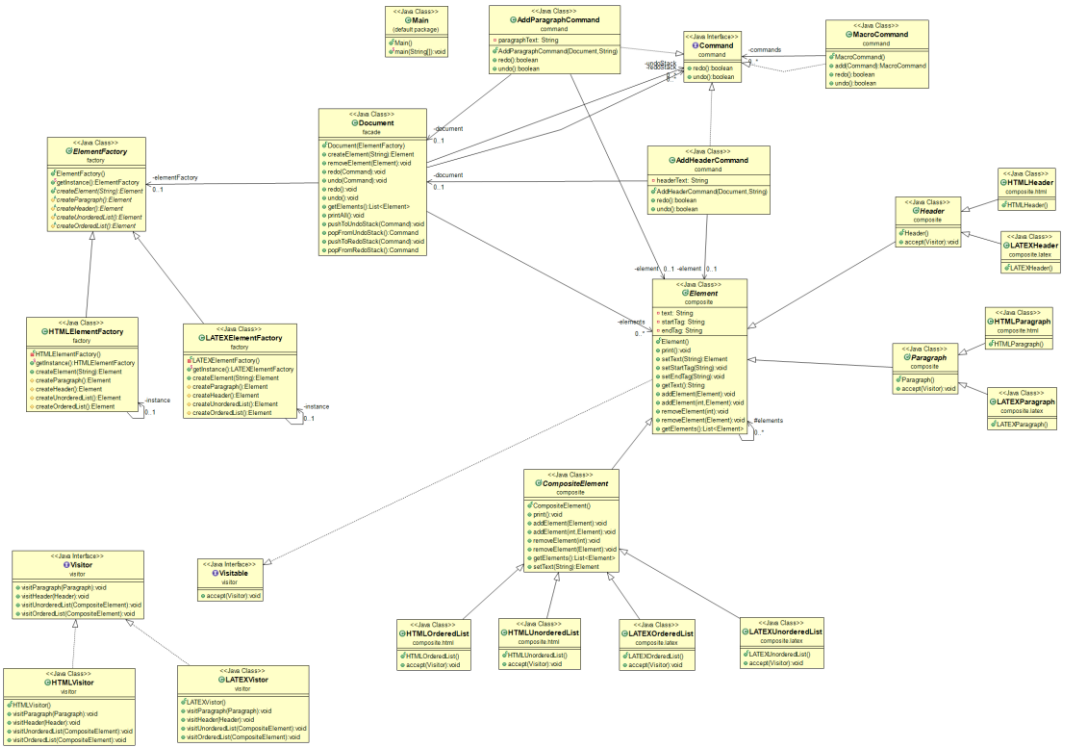
Det implementerades även ett MacroCommand vilket håller i en lista av Commands och möjliggör att göra flera command på ett och samma objekt och sedan kalla på `redo()` metoden endast en gång för att utföra alla i listan. Detta fungerar på samma sätt med `undo()`.

4 Diskussion

Anledningen till att vi utökade klassen `Element` med två attribut av typen `String` var att ha dessa två som kommer runt brödtexten hellre än att sätta taggarna i brödtexten, då detta gör det lättare att hantera om man vill konvertera om ett redan konverterat dokument.

Eftersom ingen av oss kunde syntax för LATEX element så valde vi att använda av samma markeringar som för HTML, där endast taggarnas namn byttes ut. Vi gjorde ett antagande att detta var okej då uppgiftens primära syfte var att ge kunskap om designmönster inom objekt orienterad programmering.

5 Bilaga – UML för systemet



Figur 4: UML för systemet.