|  |
| --- |
| Högskolan i Gävle |
| Miniprojekt 2 |
| Dokument |
|  |
| *Hanna Medén, Niklas Nordgren* |
|  |
| 2020-02-02 |

|  |
| --- |
| Miniprojekt 2 i kursen Objektorienterad design och programmering II, DVG503 |
|  |

**Innehållsförteckning**

[1 Inledning 1](#_Toc31378888)

[2 Metod 2](#_Toc31378889)

[3 Resultat 3](#_Toc31378890)

[4 Diskussion 4](#_Toc31378891)

[5 Bilaga – kodlistningar 5](#_Toc31378892)

1. Inledning

I detta projekt skulle det implementeras ett back-end system för att generera och manipulera textdokument. Kriterierna för implementationen var att den skulle innehålla några olika lämpliga designmönster samt att programmeringen skulle ske på en abstrakt nivå för att öppna upp för återanvändning och utökning av koden.

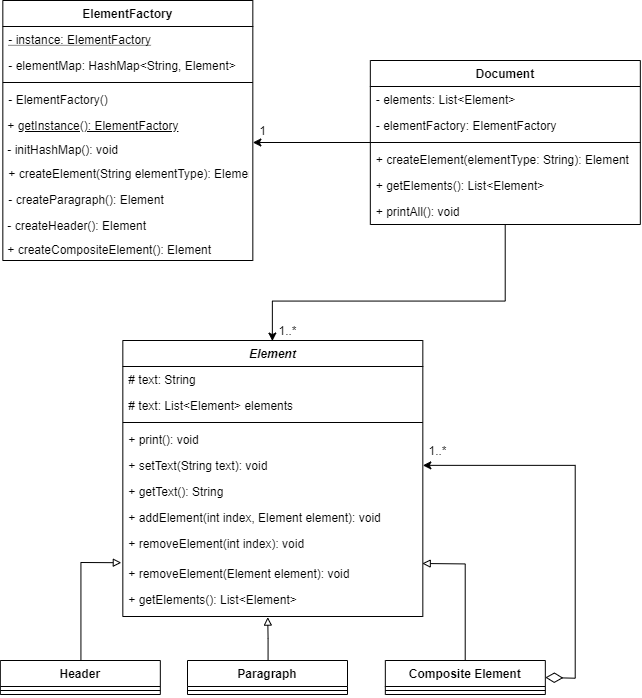
1. Metod

Inledningsvis identifierades vilka klasser som skulle ingå i systemet. Vi valde att bygga upp det på så sätt att det finns en klass Documentsom har en lista bestående av objekt av typen Element. De olika elementen är några olika delar som kan finnas i ett dokument, såsom Header*,* ParagraphochCompositeElement. Dessa är subklasser till superklassen Element*.* Därefter började arbetet med att identifiera de olika attribut och metoder varje klass behövde, och vad som var gemensamt för alla och därmed skulle finnas i den abstrakt klassen Element. Attribut som vi ansåg alla elementen krävde var ett – text. Till detta attribut skapades set/get-metoder. Det skapades även en lista av element samt metoden addElement(int index, Element element) och getElements()som CompositeElementinstansierar ochimplementerar*.* Klassen Element’s version av addElement(int index, Element element)-metoden utför ingenting utan existerar endast för att subklassen CompositeElement ska kunna implementera metoden genom överskuggning, samma sak gäller även för metoden getElements() som returnerar värdet null i klassen Element och listan av alla element som ett CompositeElement håller i. Även två olika varianter av publika borttagningsmetoder som verkar på listan Elements skapades där den ena tar emot ett index för objektet som skall tas bort och den andra det objekt som skall tas bort från listan. Klassen Element definierades som en abstrakt klass vilket motverkar att objekt av den klassen instansieras och implicit att metoderna som saknar beteende används på ett felaktigt sätt.

1. Resultat

Det slutgiltiga programmet är ett väldigt generiskt sådant där man genom hårdkodning kan skapa element, bestämma texter för dessa och även hämta samt skriva ut dem. Alla element hamnar i en lista som finns i *Document* där man kan iterera igenom alla element i listan och det finns även en metod för att skriva ut dem till konsolen. Composite element byggdes på så sätt att det i sig själv är en lista av element, där kan användaren lägga till flera olika element, exempelvis en rubrik följt av två paragrafer – ett avsnitt i en bok.

Det finns även en *ElementFactory*, den används av *Document* för att skapa olika *Element. ElementFactory* är som namnet berättar – av designmönstret Factory och även Singleton. *Document* fungerar som en Facade i detta projekt och innehåller implicit även en Iterator. Element och dess relation till subklasserna följer mönstret Composite.



Figur : UML klassdiagram av systemet.

1. Diskussion

Vi tyckte det var väldigt svårt att arbeta med så hög abstraktionsnivå. Skulle hjälpt mycket att ha någon form av konkretisering, hur det ska användas (utöver att vi sedan ska lägga till Visitor) och hur det ska användas rent praktiskt.

Såvida vi inte missförstått uppgiften väldigt mycket (tillsammans med några kurskamrater) så anser vi att det var en lagom stor uppgift för en vecka och det svåraste var som sagt att få ner faktisk kod trots så hög abstraktion. Något vi tydligen behöver jobba mer med.