|  |
| --- |
| Högskolan i Gävle |
| Miniprojekt 1 |
| OODP2 |
|  |
| Datum  2024-02-11 |

|  |
| --- |
| Kurs: Objektorienterad design och programmering 2 |
|  |
| Lärare: Goran Milutinovic |

**Innehållsförteckning**

[1 Inledning 1](#_Toc158557351)

[2 Metod 2](#_Toc158557352)

[2.1 Factory 2](#_Toc158557353)

[2.2 Composite 2](#_Toc158557354)

[2.3 Iterator 3](#_Toc158557355)

[2.4 Facade 3](#_Toc158557356)

[3 Resultat 5](#_Toc158557357)

[4 Diskussion 6](#_Toc158557358)

1. Inledning

I denna rapport presenteras en detaljerad genomgång av utformningen och implementationen av ett dokumenthanteringssystem som använder sig av designmönstren Composite och Factory, samt tillägg av Iterator- och Facade-mönstren för att ytterligare förbättra kodens modularitet och flexibilitet. Systemets arkitektur är utformad för att hantera olika typer av dokumentstrukturer där varje dokument kan innehålla flera komponenter såsom text, tabeller, matriser och datum, vilket spelar komplexiteten i verkliga dokument.

Genom Composite-mönstret kan systemet hantera både enkla och sammansatta dokumentkomponenter på ett enhetligt sätt, vilket förenklarar strukturen. Factory-mönstret abstraherar skapandeprocessen för dessa komponenter, vilket bidrar till systemets flexibilitet och underlättar för framtida utvidgningar. Iterator-mönstret tillhandahåller en standardiserad metod för att navigera genom dokumentets innehåll, medans Facade-mönstret erbjuder ett förenklat gränssnitt för systeminteraktioner, detta minskar komplexiteten och förbättrar användarvänligheten.

1. Metod
   1. Factory

Factory-mönstret står för skapandet av de angivna dokumenttyperna, implementationen använder av 3 olika typer, *examDocument*, *AcademicCalendar* och *LetterDocument* vilket har designats så att flera kan läggas till vid behov. Klasserna består av interfacet DocumentFactory som skapar dokument ifrån en enum-klass DocumentType , se Figur 1.   
A computer screen shot of a computer code

Description automatically generated

Figur , Typer av dokument

Den överliggande klassen *ConcreteDocumentFactory*, som ärver ifrån *DocumentFactory*, används för skapandet av de olika typerna av dokumenten, se Figur 2.

A screen shot of a computer program

Description automatically generated

Figur ,ConcreteDocumentFactory

* 1. Composite

Implementationen av Composite-mönstret skapar en flexibel och hierarkisk struktur för att hantera dokument och deras komponenter. Genom att definiera *CompositeDocumentComponent* som en sammansättning av *DocumentComponent*, tillåter systemet att både enkla blad och sammansatta objekt behandlas enhetligt.  
denna struktur möjliggör rekursiv sammansättning där ett dokument kan innehålla enkla komponenter såsom textblock, tabeller och datum, vilket reflekterar den naturliga strukturen hos komplexa dokument.

Varje *CompositeDocumentComponent* kan innehålla en lista av *DocumentComponent*-objekt, vilket tillåter dynamisk tilläggning och borttagning av komponenter. Detta möjliggör enkel manipulation och iteration av dokumentets innehåll, vilket underlättar skapande av rika och strukturerade dokument, se figur 3.

A screen shot of a computer code

Description automatically generated

Figur , CompositeDocumentComponent

* 1. Iterator

Implementationen av Iterator-mönstret användes för att sekventiellt gå igenom och hantera de olika komponenterna i dokumenten. Genom *CompositeDocumentComponentIterator* tillhandahålls en standardiserad metod för att iterera över sammansatta och enkla komponenter inom *CompositeDocumentComponent*, vilket omfattar både dokument och deras inre element som tabeller, textblock och datumkomponenter.

Iteratorn möjliggör enkel navigering genom komponenterna utan att avslöja den underliggande representationen av sammansättningen, vilket håller åtkomstmetoden konsistent och abstraherad från de specifika detaljerna i dokumentstrukturen. Detta förenklar operationer såsom utskrift, modifiering och analys av dokumentinnehållet, samtidigt som de upprätthåller principen om inkapsling.

* 1. Facade

Genom att implementera Facade-mönstret i systemet har en förenklad interaktion mellan klientkod och systemets underliggande komplexitet gjorts. *DocumentFacade* agerar som ett högnivågränssnitt som erbjuder enklare och mer lättförståeliga metoder för att skapa dokument, lägga till och modifiera komponenter samt hantera specifika funktioner som datumändringar, se Figur 4.

A computer screen with colorful text

Description automatically generated

Figur , ConcreteDocumentFactory

1. Resultat

Slutprodukten blev ett fungerade dokumenthanteringssystem vilken kan användas för att skapa dokument med tillhörande komponenter, det går att skapade de olika typerna av dokument, *examDocument*, *letterDocument* och *academicCalendarDocument* samt lägga till eller ta bort de olika typerna av komponenter som finns tillgängliga, se Figur 5 och 6 som visar en testkörning med en dokumenttyp samt ändring av datum för detta dokument.

A screen shot of a computer code

Description automatically generated

Figur ,Testkörning

A computer screen shot of a black screen

Description automatically generated

Figur ,Testkörning resultat

Det går också att lägga till matriser och tabeller för dokument och går givetvis att antingen ta bort med hjälp av ett id för att identifiera valda tabellen, utöver detta går också dom andra komponenterna (*Title*, *Paragraph* och *Author*) också att ta bort med hjälp av att man skriver in den text som komponenten består av, se figur 7.

A computer screen shot of a black screen

Description automatically generated

Figur , borttagning av textkomponent

1. Diskussion

Det som har varit svårt under utvecklingen av programmet är till största del hur Factory- och Composite-mönstren används tillsammans, jag har försökt att få det så bra som möjligt men det har fortfarande varit en utmaning att försöka förstå om det är korrekt implementerat då det är många klasser som ärver ifrån andra och implementerar andra, dvs att det kan bli lite rörigt.

Utöver detta så är det bara designval som gjort som kan ifråga ställas från min egna sida, när valet att man måste skriva in texten för den komponenten man vill ta bort ( *Paragraph*, *Title* och *Author*) kändes logisk för mig, jag jämförda med word och tänkte enligt följande spår: Om jag vill ta bort något ifrån det jag skriver nu så markerar jag all den text jag vill ta bort och klickar på backspace, vilket jag då försökt härma i min implementation. På liknande sätt har jag gjort med klasserna Matrix och Table fast jag lade till en id istället, samma tankesätt gjordes här, markerar du en tabell eller matris i Word så har du själva id där och klickar därefter delete/backspace för att ta bort allt, vilket då jag åter igen försökt härma.

Gällande implementationen av Facade och Iteratorn så var det betydligt mycket enklare än dom andra 2 då iteratorn består av 1 klass och ett interface och Facaden bara ses som en ”bro” mellan min egen mer komplexa kod.

En lösning jag är nöjd med är hur jag hanterar matriser och tabeller, jag använde mig av varargs string och int, vilket möjliggör att tabellerna och matriserna kan ta flera inparametrar åt gången, inte riktigt något speciellt designmönster men en liten finurlig lösning som användes, se figur 8 och 9.

A screen shot of a computer code

Description automatically generated

Figur ,Table varargs

A screen shot of a computer program

Description automatically generated

Figur ,Matrix varargs