

DesignPatterns: Paint



Klas: HBO-ICT SE 2B

Project/Opdracht: Design Patterns - tekenprogramma

Auteurs: Damiaen Toussaint (4554752), Mail: damiaen.toussaint@student.nhlstenden.com)

Mathijs Grafhorst (4568540), Mail: mathijs.grafhorst@student.nhlstenden.com)

Datum laatste wijziging: 31-01-2020

Datum	Beschrijving van wijziging	Versie Document
27-05-2020	Afronding versie 1.0 document	1.0

Inhoudsopgave

1.	Inleiding	2
2.	Ontwikkelingsomgeving.....	3
2.1.	Gekozen taal en Frameworks	3
2.2.	Mockup van de user interface.....	3
3.	Stap 1: eenvoudig tekenprogramma.....	4
3.1.	UML diagrammen	4
3.2.	Eindproduct.....	5
4.	Stap 2: Command pattern	6
4.1.	UML diagrammen	6
4.2.	Eindproduct.....	7
5.	Stap 3: Composite pattern	8
5.1.	UML diagrammen	8
5.2.	Eindproduct.....	9
6.	Stap 4: visitor pattern	10
6.1.	UML diagrammen	10
6.2.	Eindproduct.....	10
7.	Stap 5: Strategy pattern & Composite pattern	10
7.1.	UML diagrammen	10
7.2.	Eindproduct.....	10
8.	Stap 6: Decorator pattern	11
8.1.	UML diagrammen	11
8.2.	Eindproduct.....	12

1. Inleiding

Dit document betreft de documentatie van de opdracht voor het vak “designPatterns”. Dit document is opgesteld door Damiaen Toussaint en Mathijs Grafhorst. In dit document zal er iteratief bijgehouden worden wat voor wijzigingen wij aan de code doorvoeren.

Om bij te kunnen houden welke wijzigingen er allemaal zijn doorgevoerd is het handig om deze in een document te noteren. Dit zal gaan volgens de 6 stappen die zijn aangegeven in de projectomschrijving.

Deze stappen zijn als volgt:

- Stap 1: eenvoudig tekenprogramma
- Stap 2: command pattern
- Stap 3: composite pattern
- Stap 4: visitor pattern
- Stap 5: strategy pattern en singleton pattern
- Stap 6: decorator pattern

In dit document zullen we bij elke uitgevoerde stap toelichten wat wij hebben aangepast aan het project en waarom wij bepaalde keuzes hebben gemaakt.

Tijdens de ontwikkeling van dit project zullen we gebruikmaken van git, en zal de code in een private repository op github komen te staan. Omdat er per stap broncode beschikbaar moet zijn hebben we besloten om aan het eind van een iteratie/stap de code naar een aparte branch te pushen. Zo kunnen we onze code history goed in 1 repository bewaren en hoeven we niet te prutsen met losse bestanden en zipjes.

2. Ontwikkelingsomgeving

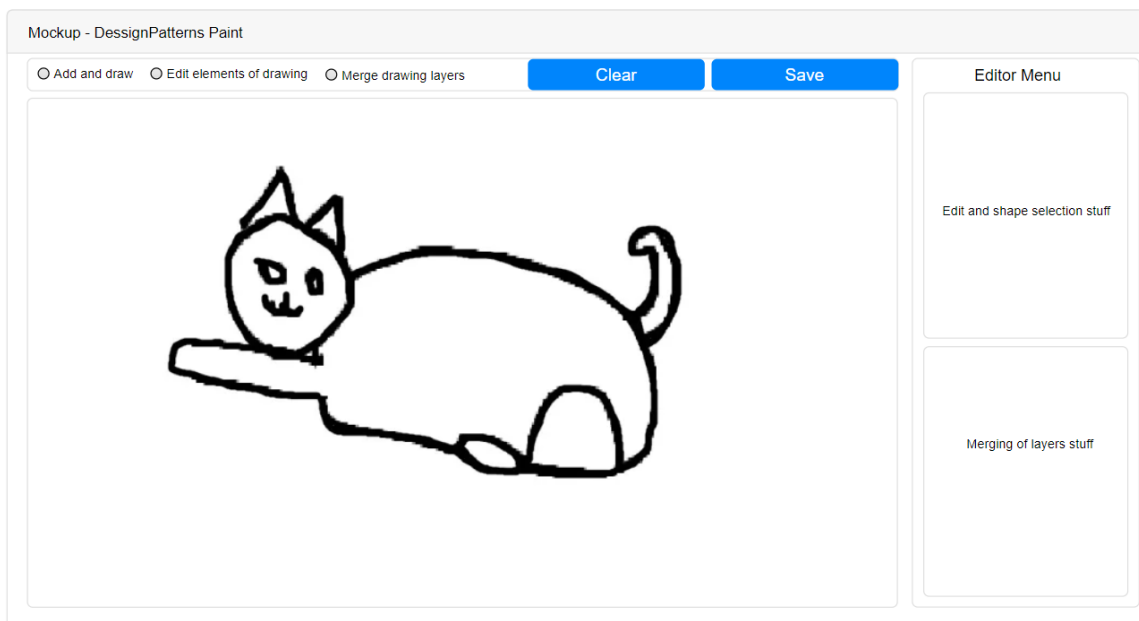
Hieronder is een kleine toelichting te vinden van de eerste keuzes die we als team hebben gemaakt, voordat we zijn begonnen aan de ontwikkeling van het DesignPatterns project. Het zal hier gaan over de gekozen programmeertaal, frameworks die we in het begin gebruiken en de hoe wij de userinterface voor ons zien.

2.1. Gekozen taal en Frameworks

Voordat wij een start maken aan de ontwikkeling van het programma hebben we als groepje even overleg gehad over onze keuze in programmeertaal. Uit dit overleg kwam naar boven dat de voorkeur van beide teamleden naar **Java** uitgaat.

Omdat het project van Periode 3 Data Science ook in Java is geschreven, kunnen we bepaalde onderdelen overnemen en zitten we al een beetje in de Java sfeer. Het Data Science project maakt gebruik van **Java Swing** in combinatie met de UIBuilder van IntelliJ. Deze manier van opbouwen van de user interface zullen wij ook in het DesignPatterns project gebruiken.

2.2. Mockup van de user interface

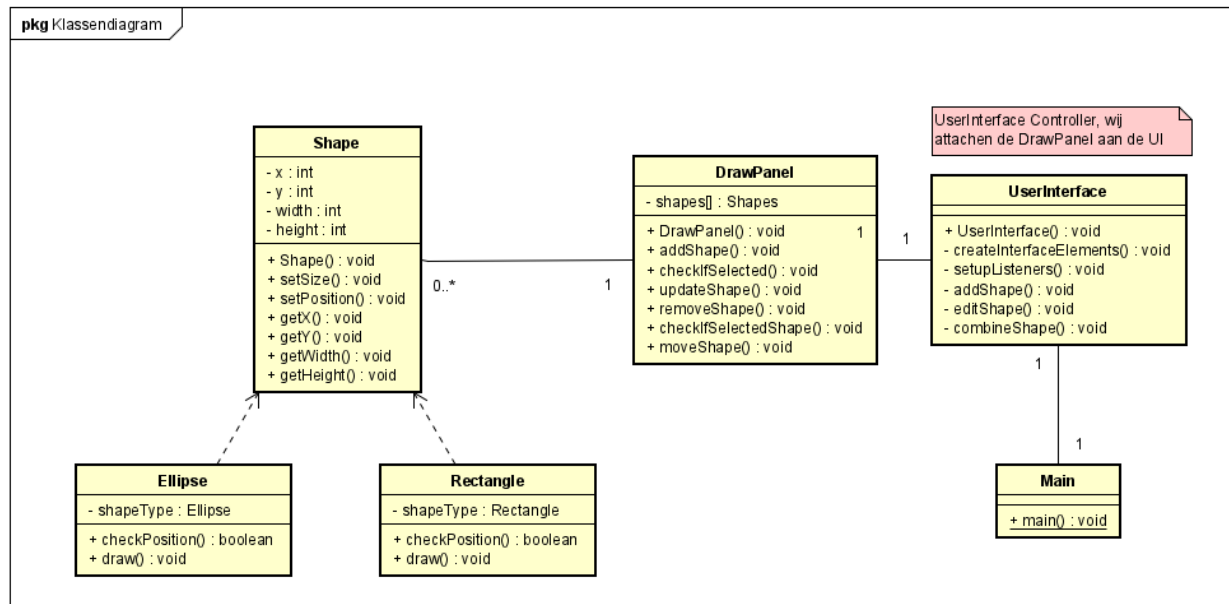


Figuur 1 - userinterface eerste versie, deze is zo gebouwd dat we aan elke kant nog wat kunnen aanpassen

3. Stap 1: eenvoudig tekenprogramma

Bij stap 1 zullen we de eerste versie van het tekenprogramma realiseren. We zullen in de eerste versie van de applicatie al wel gebruik maken van overerving en verdere OOP technieken, om zo het totaalplaatje overzichtelijk te houden.

3.1. UML diagrammen



Figuur 2 – Klassendiagram eerste iteratie

De eerste versie van het klassendiagram is redelijk simpel, na wat uitzoekwerk kwam naar boven dat we een deel van onze eerder gemaakte userinterface konden overnemen. Om te kunnen tekenen moeten we aan een custom JPanel maken en deze koppelen aan een bestaande JPanel in de UserInterface klasse.

In deze DrawPanel zit dan weer de logica waarin al het tekenen gebeurt, vanuit de UserInterface krijgen wij met behulp van listeners binnen welke actie wij moeten uitvoeren. In de DrawPanel zelf zit een ArrayList met de verschillende Shapes. Op deze shape zelf kunnen wij weer verschillende functies uitvoeren, denk hierbij aan het aanpassen van de size of het instellen van de coördinaten.

3.2. Eindproduct

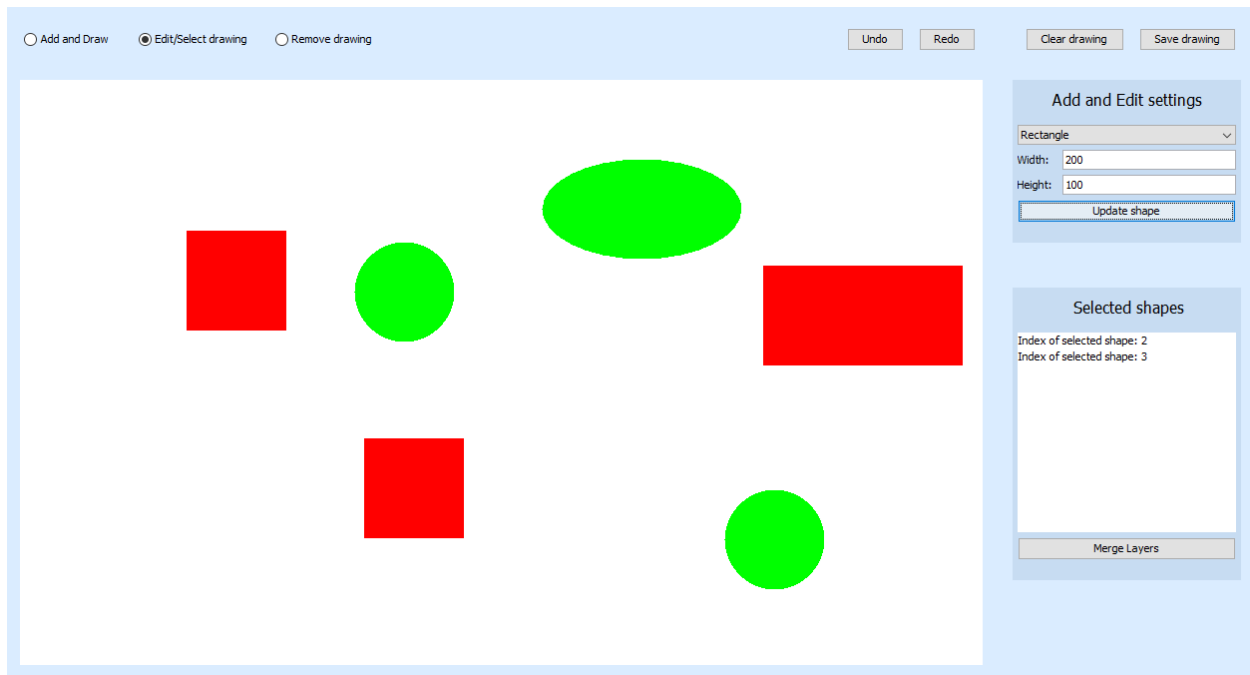


Figure 3 - Stap 1

Code:

Er staat nu een basis tekenprogramma, je kunt shapes toevoegen, resizen en verplaatsen. Het is ook al mogelijk om op een shape te klikken en terug te krijgen welke shape je nou hebt geselecteerd, als voorbereiding op het mergen van layers.

Bovenin heb je een drietal radiobuttons, waarmee je kunt selecteren welke functionaliteit je wilt hebben. We dachten er eerst ook aan om mogelijk toetsencombinaties te gebruiken voor het uitvoeren van acties, maar hebben er toch voor gekozen om dat niet te doen. Een van de grootste redenen hiervoor, is dat je tijdens het switchen van “mode” wat code kunt uitvoeren om dingen in de achtergrond te resetten, waardoor je onverwachte side-effects kunt voorkomen.

Visueel:

Aan de visuele kant is bijna alles wat in de eerste mockup stond (figuur - 1), overgenomen naar de applicatie. Zoals al eerder aangegeven maken we gebruik van Java Swing voor de interface, hierdoor kunnen we ook eventlisteners toevoegen aan het witte tekenvlak.

4. Stap 2: Command pattern

Bij stap 2 zullen de versie van de vorige stap verder ontwikkelen en nieuwe functionaliteit toevoegen. In deze stap zal deze functionaliteit vooral bestaan uit het toevoegen van de File IO (load/save) en het implementeren van het command pattern.

4.1. UML diagrammen

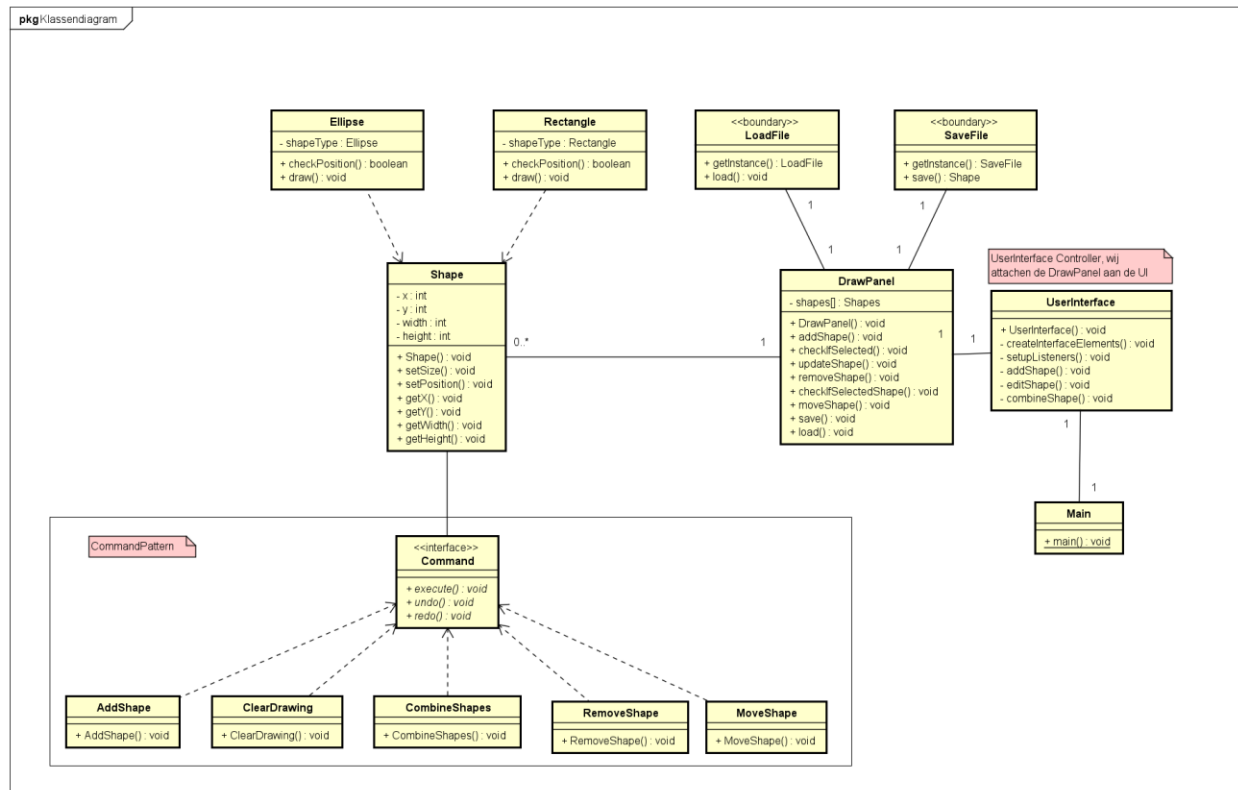
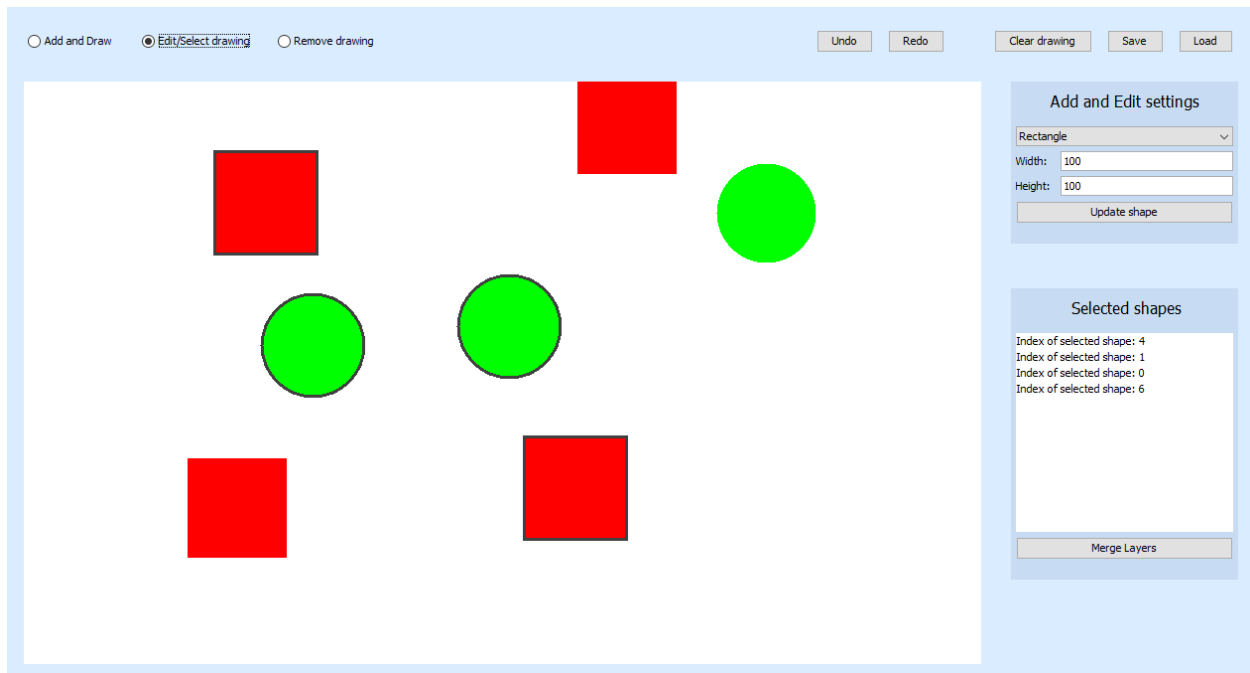


Figure 4 - Klassendiagram stap 2

Het laden en sparen van bestanden gaat via de LoadFile en SaveFile Klassen. Deze twee classes zijn singleton gemaakt, omdat er eigenlijk altijd maar 1 instantie van deze klasse hoeft te zijn. We zijn van plan om het opslaan van de data via json te laten gaan, omdat JSON wel mooi overzichtelijk is in vergelijking met plain opslaan naar tekst.

Het command pattern gedeelte spreekt redelijk voor zich. We hebben een command interface die weer verschillende acties kan uitvoeren. Dit zijn de acties: add, clear, combine, remove en move. Deze acties behoren tot een shape. Het mergen van een meerdere shapes naar een groep hoeft hier nog niet, maar we maken deze klasse wel alvast aan.

4.2. Eindproduct



Code:

Aan het einde van deze stap hebben we het command pattern geïmplementeerd in ons project. Het Adden, clearen, remove en moven van de shapes gaat nu via het command pattern. Verder is er ook een start gemaakt aan het toevoegen van de File IO, zo is de structuur van de shapes wat aangepast en kun je nu ook opslaan en laden.

Het opslaan en laden gaat via JSON, omdat een van de teamleden al wat ervaring had met het gebruik van JSON is er besloten om dit te implementeren. Voordat we dit konden doen moesten we eerst van het plain Java project, een Maven project maken. Om JSON functionaliteit in Java te krijgen, moesten we de plugin “org.json.simple” gebruiken.

Ook kun je nu alvast shapes selecteren vanuit de view zelf, aan de zijkant in het menu krijg je terug welke shapes dat zijn. Zo hebben we alvast een stukje klaar voor het mergen van shapes.

Visueel:

Aan de visuele kant zijn er ook enkele wijzigingen doorgevoerd, zo krijg je nu visuele feedback bij het selecteren van shapes en is het plaatsen/editen van de shapes wat gebruiksvriendelijker gemaakt. Voorheen als je een shape plaatste, kwam deze niet op het punt van de cursor terecht. Met gebruik van wat berekingen komt deze nu wel op de correcte xy positie en gaat het slepen ook wat soepeler.

5. Stap 3: Composite pattern

Bij stap 3 zullen de versie van de vorige stap verder ontwikkelen en nieuwe functionaliteit toevoegen. In deze stap zal deze functionaliteit vooral bestaan uit het toevoegen van het groepsysteem, het implementeren van het composite pattern en het refactoren van de bestaande code.

5.1. UML diagrammen

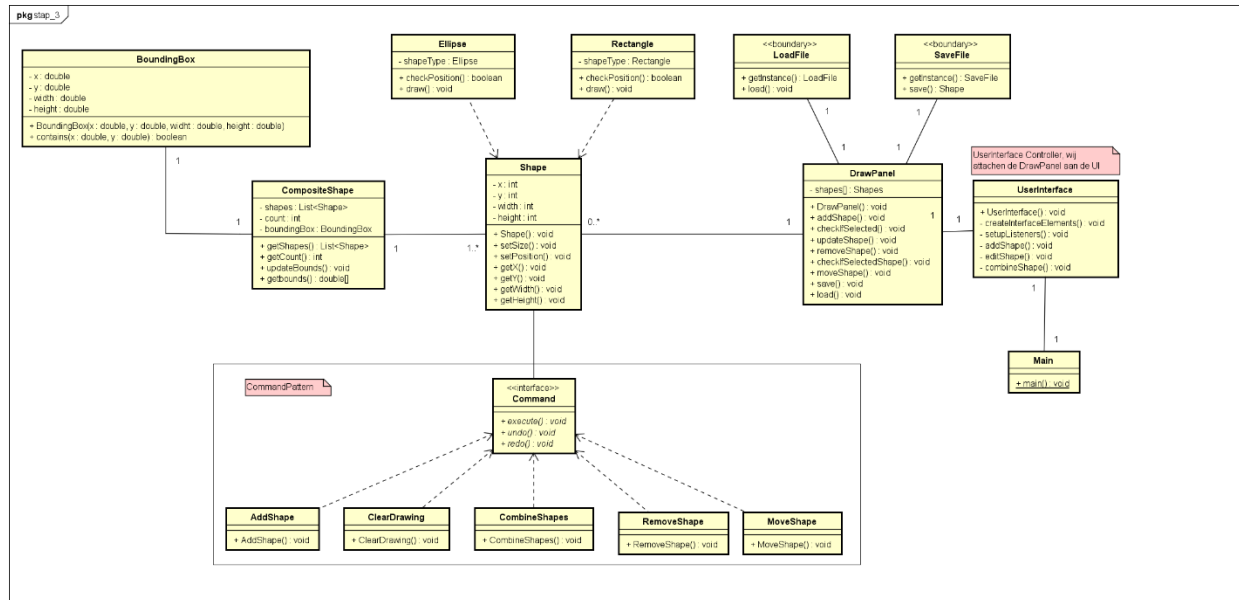


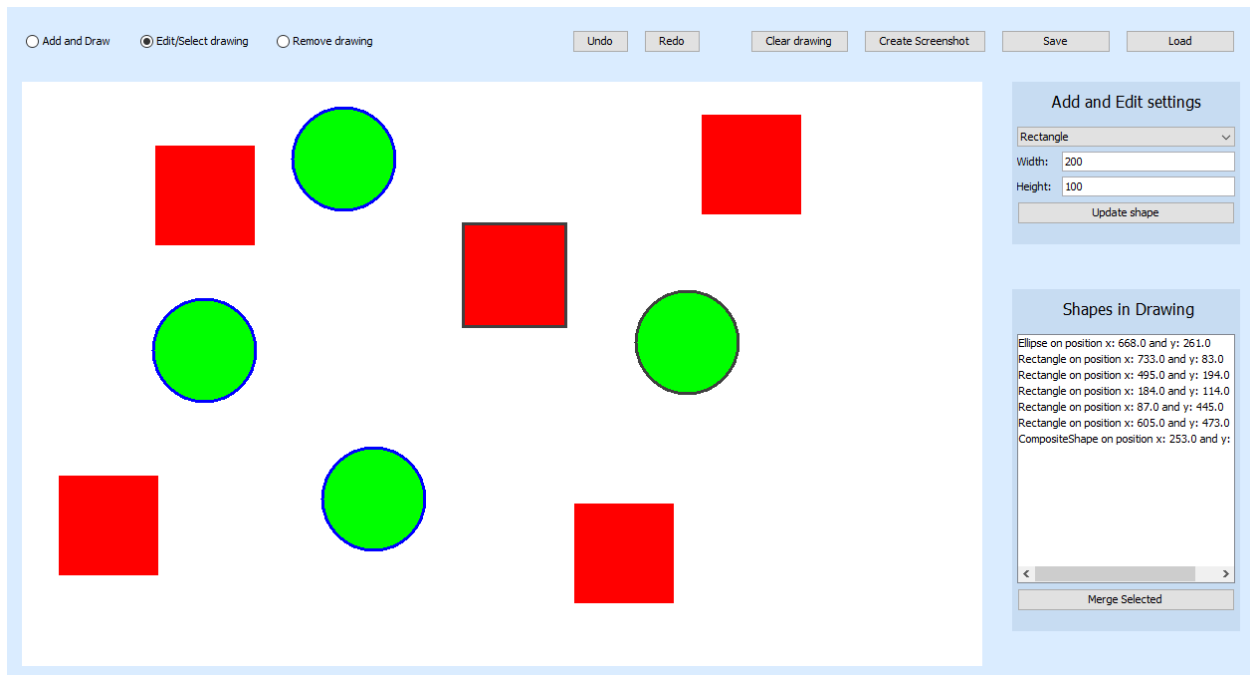
Figure 5 - Diagram stap 3

Bij deze stap gaan we het composite pattern voor het maken van groepen toevoegen. Het idee is dat je vanuit het zijmenu of via de view shapes kan selecteren en deze dan kan samenvoegen tot 1 nieuwe groep. Omdat **CompositeShape** ook een shape is, kunnen we dus meerdere groups in elkaar plaatsen.

De uitdaging van deze group is het selecteren en resizen, want je moet als gebruiker kunnen zien welke shapes er allemaal in de group zitten. Ook voor het verplaatsen van een group zal je rekening moeten houden met alle shapes die in de group zelf zitten.

Het idee is om een **BoundingBox** om de group te zetten, dit kun je een beetje voor je zien als een vierkant om alle shapes heen. Zo kun je bij het selecteren van een group meteen zien welke shapes er in deze group zitten en bij het slepen van een group hoeven we alleen rekening te houden met de afmetingen van de **BoundingBox** zelf. Dus dan zit je niet te prutsen met het steeds berekenen van de afmetingen van elke shape in de group zelf, want dit kan best wel in een spaghetti veranderen als je met meerdere nested groups zit.

5.2. Eindproduct



Code:

Aan het einde van deze opdracht zit het Composite pattern voor shapes groups in de applicatie verwerkt. Het is nu mogelijk om shapes met elkaar te mergen tot een group, dit kan oneindig door, want er zit geen limiet op het mergen.

Het opslaan van de tekening hebben we nu ook aangepast van JSON naar tekst, omdat we er toch achter kwamen dat in de opdrachtsomschrijving stond dat moesten opslaan via een bepaalde structuur. Het is mogelijk om nu .json en .txt te laden, want de applicatie checked zelf wat de file-extension is en roept de correcte functies aan. Maar waarschijnlijk gaan wij later in de ontwikkeling over naar alleen opslaan naar tekstbestanden.

Visueel:

Aan de visuele kant zijn ook wat dingen aangepast, zo kan je nu via het sidemenu shapes selectere, is er een screenshot save button en zijn er verschillende outlines die de shapes omringen gebaseerd op de geselecteerde shapes. Als een shape of group geselecteerd is via het zijmenu, zal deze blauw zijn. Als dit via de DrawPanel gaat, zal deze blauw zijn. Bij het selecteren van meerdere shapes vanuit de view en het zijmenu, zal de applicatie deze bij elkaar pakken en samenvoegen tot 1 group. Als een gebruiker een group selecteerd, zal elke shape een border krijgen en kan de gebruiker deze ook bewegen.

6. Stap 4: visitor pattern

Bij stap 4 zullen de versie van de vorige stap verder ontwikkelen en nieuwe functionaliteit toevoegen. In deze stap zullen we waarschijnlijk grotendeels bezig zijn met het refactoren van de applicatie en het implementeren van het visitor pattern.

6.1. UML diagrammen

6.2. Eindproduct

Code:

Visueel:

Tijdens het uitvoeren van deze stap zijn er geen visuele wijzigingen doorgevoerd aan de applicatie.

7. Stap 5: Strategy pattern & Singleton pattern

Bij stap 5 zullen de versie van de vorige stap verder ontwikkelen en nieuwe functionaliteit toevoegen. In deze stap zullen we waarschijnlijk grotendeels bezig zijn met het refactoren van de applicatie en het implementeren van het strategy pattern, singletons zitten al in de applicatie, maar zullen ook toegevoegd worden waar het kan.

7.1. UML diagrammen

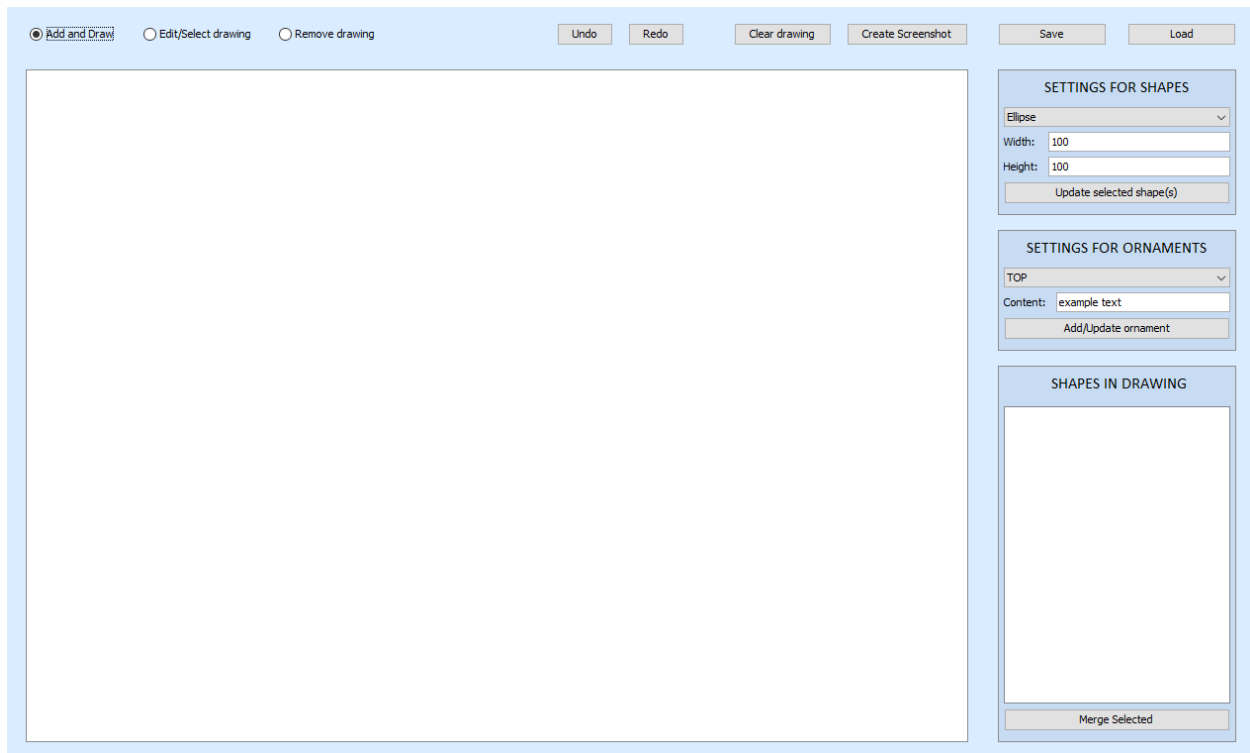
7.2. Eindproduct

Code:

Visueel:

Tijdens het uitvoeren van deze stap zijn er geen visuele wijzigingen doorgevoerd aan de applicatie.

8.2. Eindproduct



Code:

Visueel:

Aan de visuele kant is de grootste wijziging in deze stap, dat je nu ornaments kunt zien bij een shape of group. Verder zijn er aan de visuele kant enkele kleine aanpassingen doorgevoerd. Zo hebben bepaalde elementen nu een border en de achterliggende forms structuur is aangepast, zodat deze nu beter scaled met verschillende monitor resoluties.