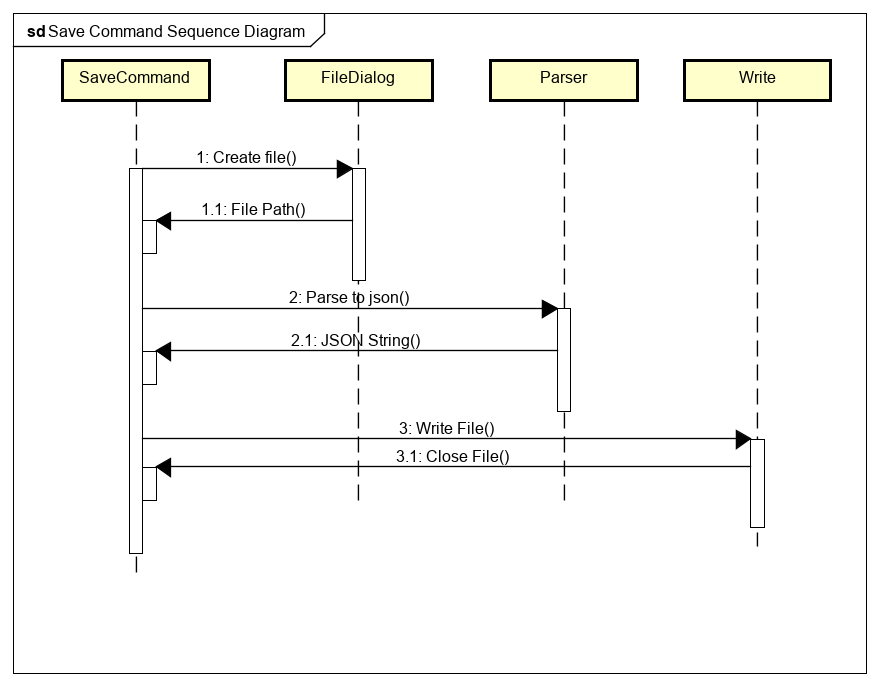
**Command Pattern Klassen Diagram**

Het command pattern maak gebruik van een Command Manager die Commands uit kan voeren. Een command bestaat uit 3 functies, namelijk: Execute() het uitvoeren van het command, Undo() het ongedaan maken van de command en Redo() het opnieuw uitvoeren van de command.

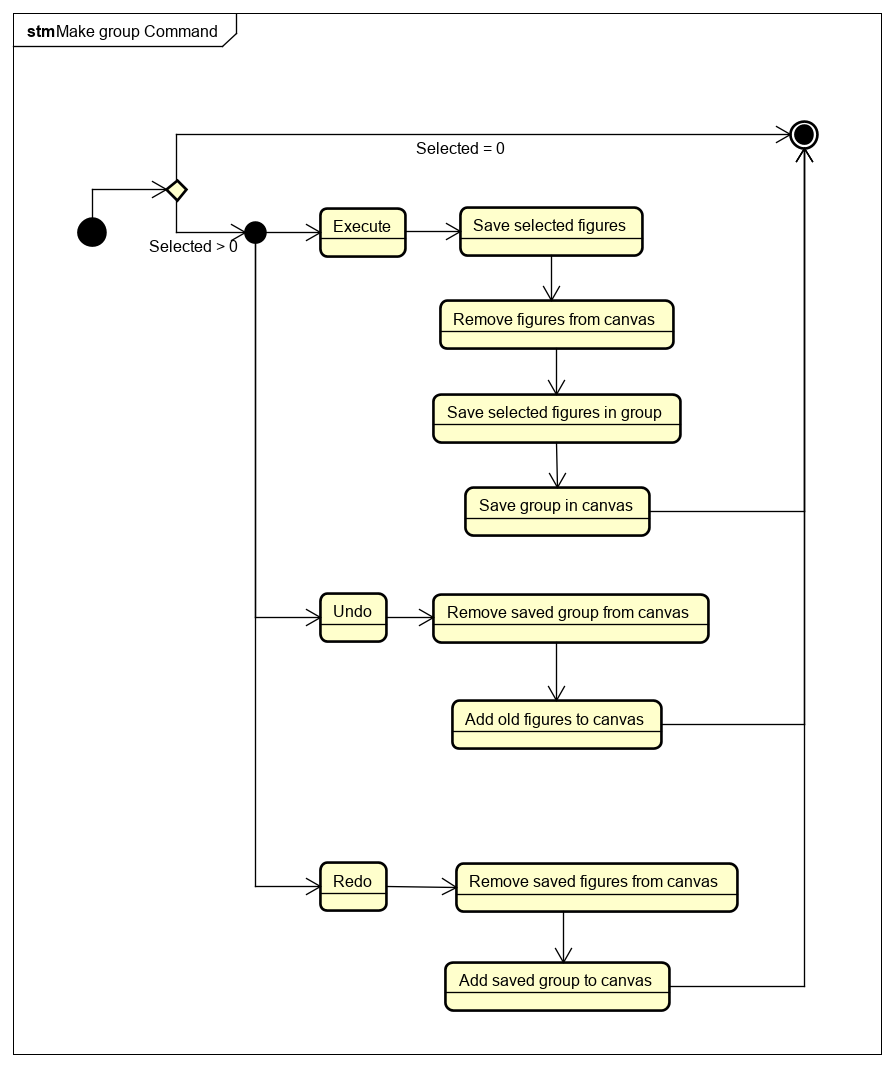
Een specifiek command implementeert een command interface, dit komt omdat een command steeds uit dezelfde functies bestaat alleen zijn deze anders geimplementeerd.

****

**Command Pattern Sequence Diagram**

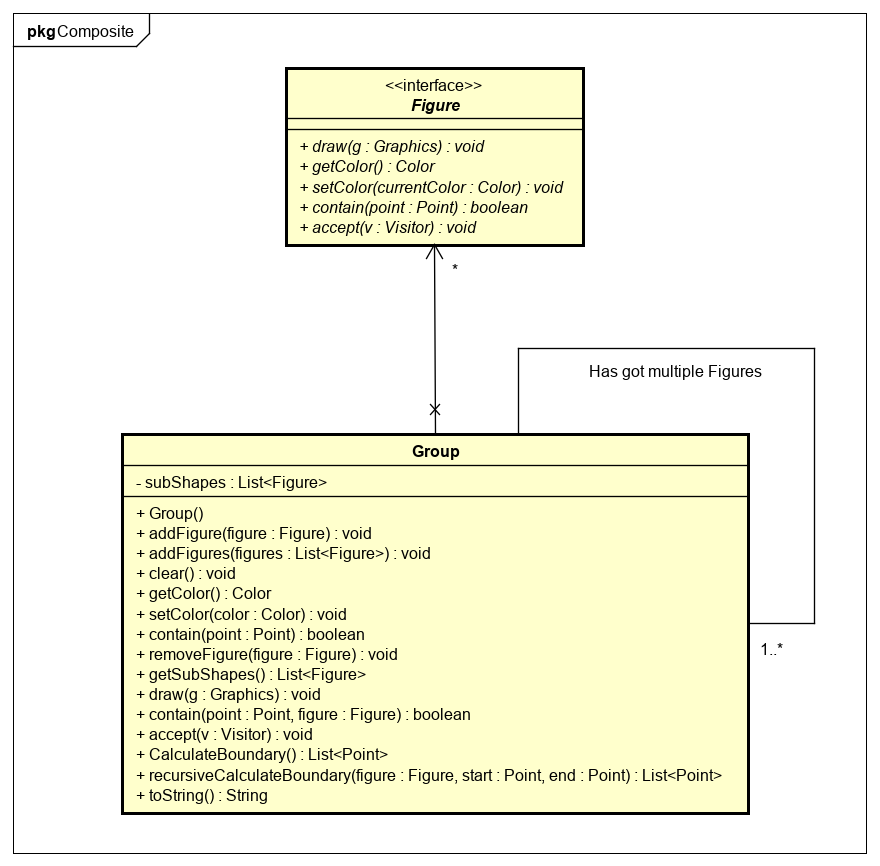
Hierboven wordt sequence diagram getoond van een save command.

Het Save command opent een File Dialog waarin je een file kan selecteren en geeft het pad terug van de file. Vervolgens parse je de figuren die in de canvas staan naar een json. Daarna schrijf je deze content naar de file met het pad wat je hebt terug gekregen van de file dialog.



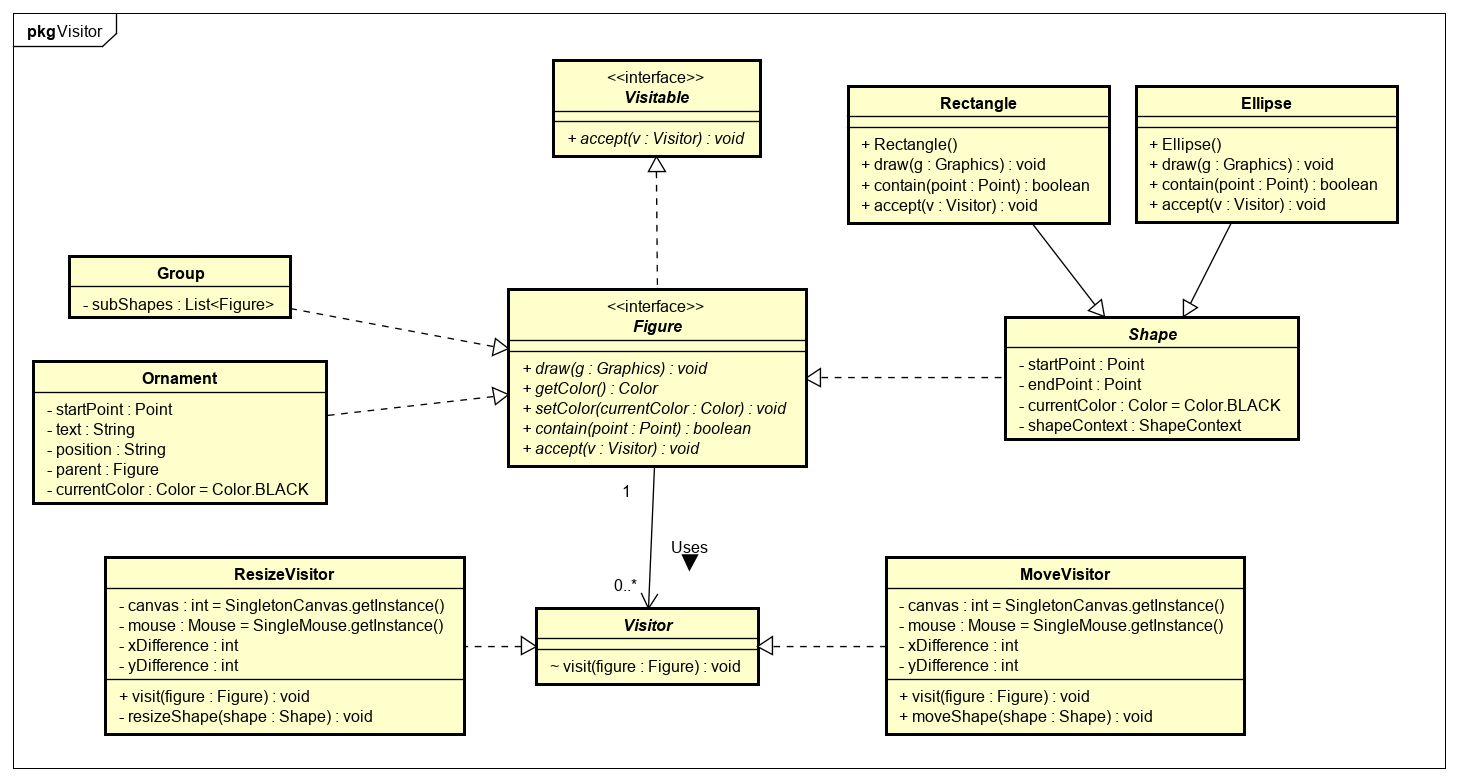
**Commmand Pattern State Diagram Group Command**

Bij het group command onthoud hij de figuren waarvan je een groep wou maken, Vervolgens zet hij deze in een nieuwe group. Daarna verwijderd hij de oude figuren van het canvas en voegt tenslotte de nieuwe groep toe aan het canvas.

****

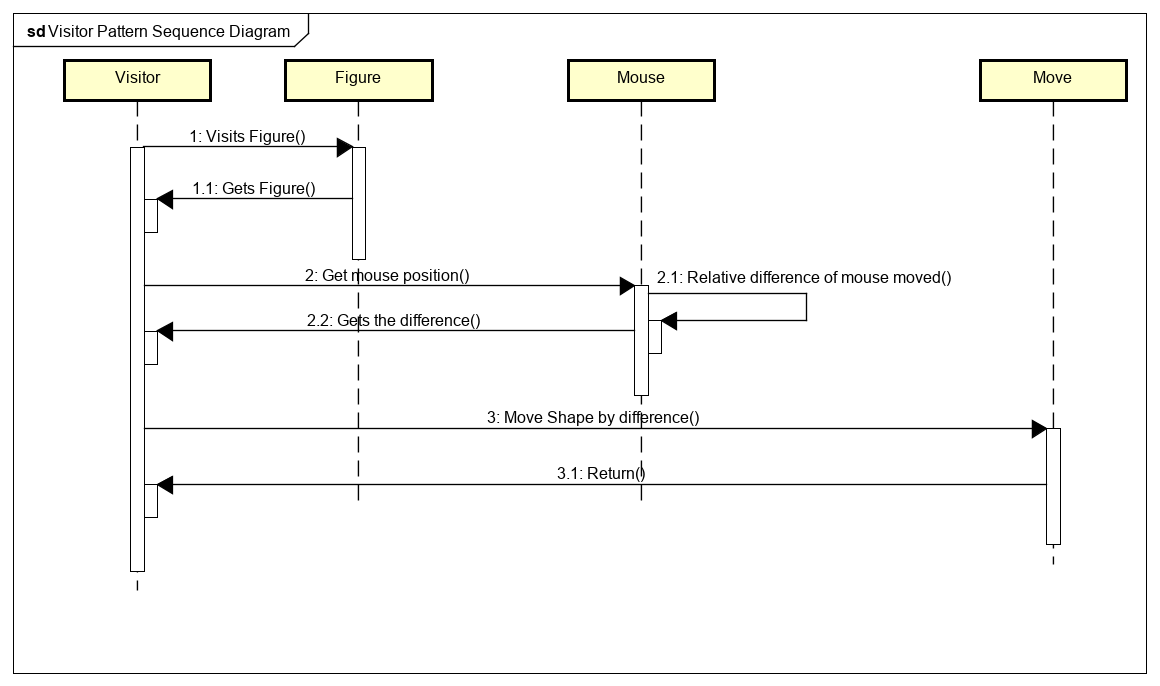
**Composite Pattern Klassen Diagram**

Het Composite pattern zorgt er voor dat je een groep kan maken van figuren waar ook weer groepen en figuren in kunnen zitten. Je kunt bij alle figuren door recursief door de lijst met sub figuren heen te lopen.



**Visitor Pattern Klassen Diagram**

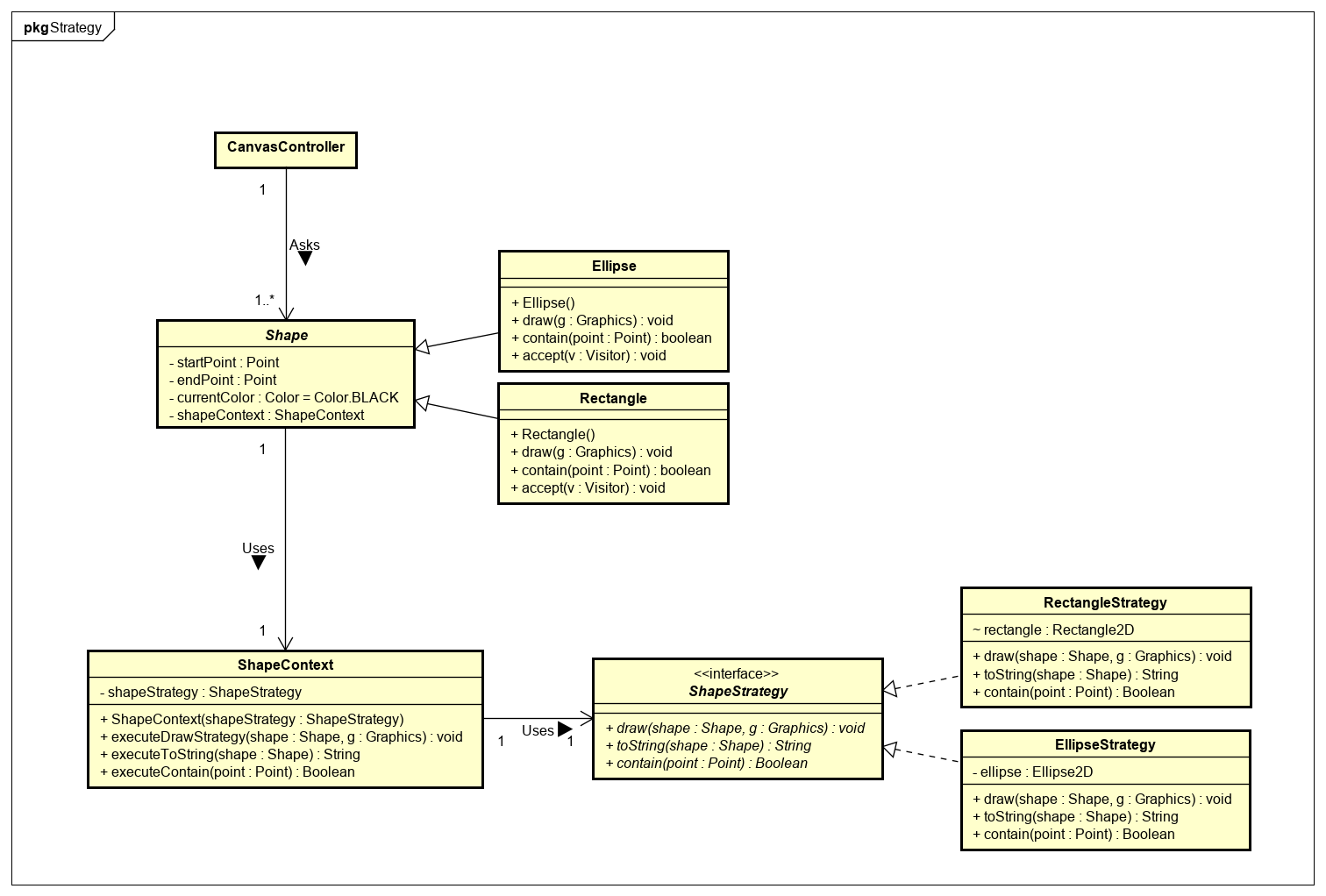
Een figuur implementeert het interface Visitable, hierin zit een functie die een visitor accepteert. Een figuur geeft zichzelf mee aan de visitor zodat de visitor dingen aan het figuur kan veranderen. Hierdoor kun je functionaliteit toevoegen aan de figuur zonder de klasse figuur aan te roepen.



**Visitor Pattern Sequence Diagram**

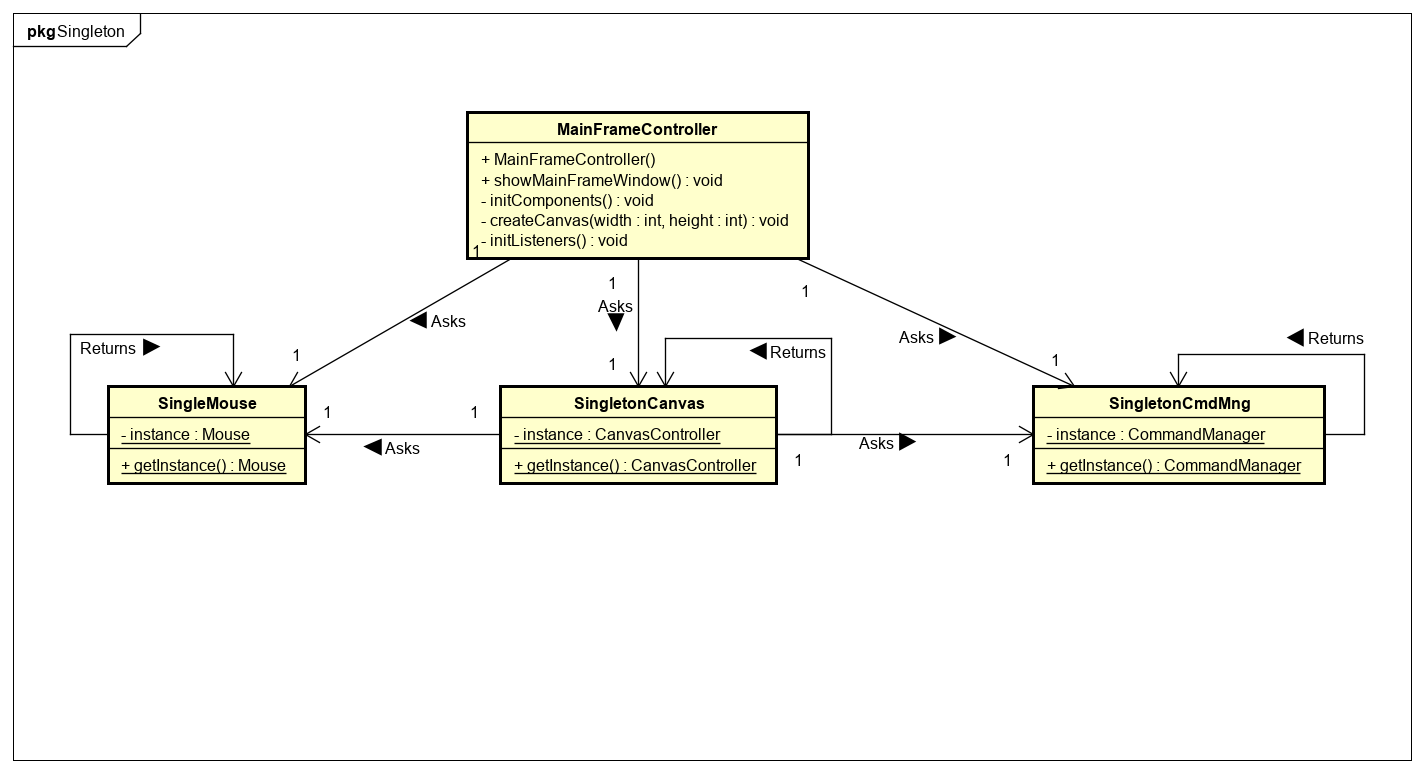
Hierboven wordt de Move Visitor getoond.

De visitor visits de figuur en de figuur geeft zichzelf mee. Vervolgens pakt de visitor steeds het verschil van de positie van de mouse. Tenslotte veranderd hij het start en eind punt van het figuur relatief aan het verschil van de muis.



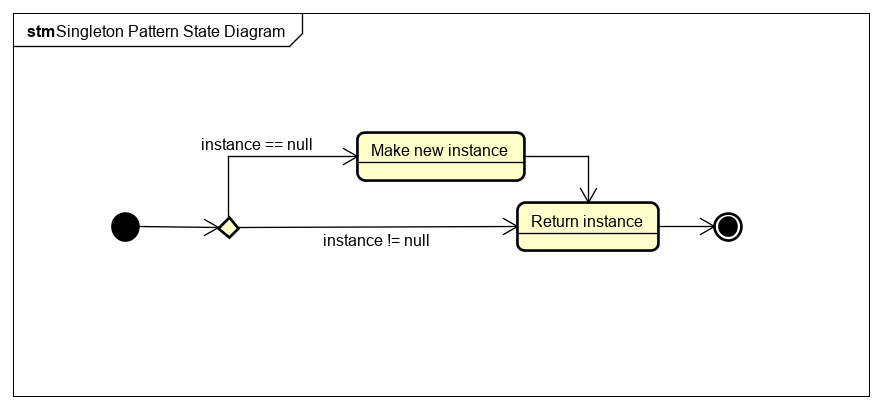
**Strategy Pattern Klassen Diagram**

Bij een shape is zowel het tekenen van de shape als het kijken of de muis in de figuur zit anders. Daarom wordt een strategy pattern gebruikt die deze functionaliteiten apart houd. De shape heeft een shapecontext die weer een shapestrategy heeft.



**Singleton Pattern Klassen Diagram**

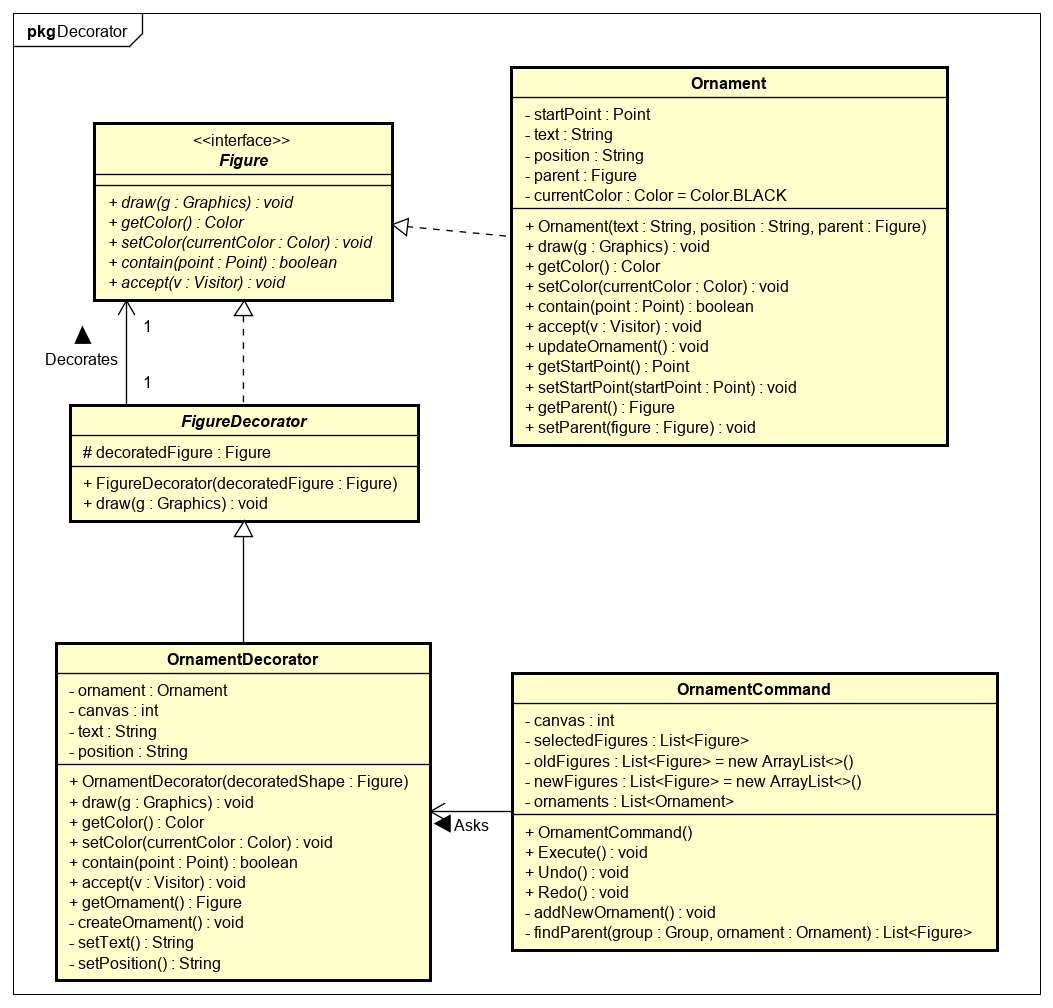
Het Singleton pattern wordt gebruikt om steeds 1 instantie aan te houden van een klasse. Bij ons kwam dit goed van toepassing aangezien we maar 1 Muis adapter , 1 Canvas en 1 CommandManager willen hebben. Door bijvoorbeeld 1 command manager te hebben zullen alle commands hierin terecht komen waardoor de Undo en Redo functies goed kunnen werken.



**Singleton Pattern State Diagram**

Hierboven wordt een state diagram getoond van het singelton pattern

In de singelton class zit een functie getInstance(), deze functie geeft een instance van de klasse terug. Wanneert deze instance nog niet gezet is zal en nieuwe instance maken en deze terug geven.



**Decorator Pattern Klassen Diagram**

Het decorator pattern zorgt er voor dat je nieuwe toe kan voegen aan een figuur qua uiterlijk door een soort wrapper te maken om de een figuur. Hierdoor zal de structuur van de figuur niet zelf aangepast worden.