System design document for ObPaint

Erik Anttila Ryderup Erik Bengtsson Axel Hertzberg
Jonas Nordin
Team JEEA
Version 1.003

23 oktober 2020

Innehåll

1 Introduktion

Detta dokument syftar till att beskriva designen av det grafiska ritprogrammet ObPaint. Den övergripande designariktekturen består av en Model View och Controller-arkitektur. Programmet utformas för att vara modulärt i den mån att modellen kan existera oberoende av de övriga modulerna. Ett tydligt mål med utvecklingen är att programmet ska vara utökningsbart.

1.1 Designmål

Under projektets utveckling har olika mål satts upp för kodbasen. Ett av dessa mål är att följa SOLID-principerna [1]. Ett annat designmål är att modellen ska vara så återanvändbar som möjligt vilket innebär att den har få externa beroenden. Den är bara beroende på standardbiblioteket i java.

1.2 Definitioner

Coupling - Coupling är ett mått på hur stark koppling det finns mellan olika moduler inom ett program. Inom objektorienterad programmering eftersträvas ofta låg Coupling.

MVC - syftar på designmönstret Model, View, and Controller.

Controller - syftar på ett package i projektet som vi skapat som står för Controller-delen i ett MVC mönster.

CanvasView - syftar på den vy mitt i applikationen där objekt ritas ut och som användaren kan interagera med.

ToolView - syftar på den vy ner till vänster i applikationen som håller i verktygen till att skapa nya eller modifiera objekt i CanvasView.

ObjectView - syftar på den vy ner till höger i applikationen som visar vilka objekt som finns i CanvasView.

SelectedObjectView - syftar på den vy upp till höger i koden som visar info om det valda objektet från CanvasView.

JavaFX - Ett bibliotek för att skapa grafiska program

FXML - Ett XML-baserat språk för att bygga upp gränssnitt

MenuBar - syftar på menyraden högst upp i applikationen.

Factory - En hjälpklass som är avsedd för att skapa objekt genom att kalla på deras konstruktorer.

Paket - En funktion i java för att dela upp kod.

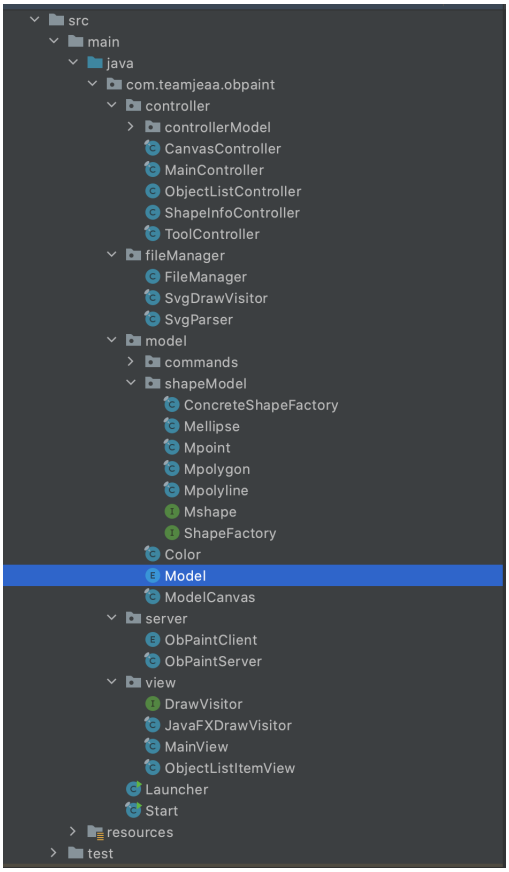
Commands - Del i designmönstret Command Pattern som syftar till att utföra kommandon utan att ge ut för mycket information om klasser/pakets konkreta implementation.

Pmd - ”Programming mistake detector” är ett analytiskt verktyg för att kontrollera kodbasen statiskt mot vanliga programmeringsfel.

Port - En port används för att skilja mellan olika trafik till och från en dator.

2 Systemarikitektur

Programmet skrivs i Java och använder Javafx som grafiskt bibliotek för att måla modellen på skärmen. Scenebuilder används för att skapa FXML-filer för att effektivt bygga upp grafiska scener. Programmet struktureras upp i flera paket och strukturen visas i sin helhet i figur 1.



Figur 1: Programmets paketstruktur

2.1 Övergripande om modellen

Applikationen skrivs med utgångspunkt i ett MVC-mönster för att alla delar av programmet ska kunna existera så självständigt som möjligt. Själva kärnan i programmet är modellen som består av olika former och commands som används för att hantera anrop till modellen. Till modellen hör även en del hjälpklasser som används för att olika delar i modellen ska fungera smidigt.

Programmet syftar i huvudsak till att måla ut olika typer av figurer på skärmen. Figurerna representeras av klasser i modellen och figurerna ska kunna existera i modellen helt fristående från Javafx som används för att måla ut dem. Dessa figurer kallas ”Mshapes” och tillhör paketet ”shapeModel” som i sin tur är en del av Model-paketet. Då programmets vy har för avsikt att använda sig av Javafx för att måla på skärmen ska Visitor Pattern användas som en slags omvandlare. Detta leder till att det inte kommer finnas någon direkt koppling mellan modellen och Javafx. De designmönster som nämns

här kommer att diskuteras mer genomgående under 3.3.

Till modellen tillhör även paketet commands där de kommandon som kommer ifrån controllern hanteras och utförs på modellen. Verktyg som ska illustreras grafiskt ska ha en tillhörande kommando-klass i paketet commands.

En sak att tillägga är att Servern som finns i ObPaint tillhör också modellen. Mer om denna finns i punkten 2.2 om nätverk.

ModelCanvas är den klass ansvarar för att hålla alla de objekt som ritas ut på skärmen genom en lista av typen Mshape. Klassen ansvar ska i huvudsak ansvara för att lägga till och ta bort figurer från denna lista.

Controllern i MVC-mönstret kommer i själva verket bestå utav flera olika controllers som tar emot användarinput från sina respektive ansvarsområden. Därifrån kommer Controllern att fördela och skicka vidare kommandon i form av commands till modellen.

2.2 Systemets "flow" (high level)

Programmets flöde innebär att när man startar appen så skapar klassen "Start" en tråd där servern för programmet körs. Efter det skapas objektet "Model" som startar modellen. Klassen "Start" startar vidare upp Javafx som målar fönstret genom att använda en FXML-fil, sedan lämnar klassen över kontrollen till Javafx. Programmet styrs sedan av att användaren genererar "MouseEvent" som hanteras av Javafx. Det här fungerar genom att användaren klickar på ställen i programmet så binds klicken ihop med olika metoder. Detta sker med hjälp av "Controllers". Exempelvis är knapparna för programmets verktyg bundna till metoder som bygger ihop de olika kommanden som skickas till modellen. Om användaren stoppar programmet så anropas en metod som stänger ner Javafx och sedan stänger av programmet.

2.3 Nätverk

När programmet startas så skapar den en tråd av programmet genom ObPaintServer. I den tråden körs sedan servern. Servern startar en ServerSocket som hela tiden lyssnar efter "strings" som skickas till den från en annan instans av programmet på en vald port. "Stringsen" innehåller instruktioner om vad som sker på den andra instansen. Sedan utförs en enkel loop som kontrollerar om det kommit nya "strings" med instruktioner. Har det kommit nya instruktioner hanteras detta och utförs på den lokala instansen.

Då ändringar sker på den lokala instansen skickas dessa till den sammankopp-

lade instansen vid utförandet av ett command, samma command som skickar instruktioner till den lokala modellen skickar även dessa instruktioner till den andra instansen genom ObPaintClient som formaterar meddelandet skickar det till servern. På så sätt vet de två sammankopplade instanserna vad som sker hos varann hela tiden och kan uppdatera sig själva efter det.

Programmets meddelanden är en sträng som delas på ",". Det första ordet berättar vad som ska göras och resten av strängen är information om hur detta ska göras.

3 Systemdesign

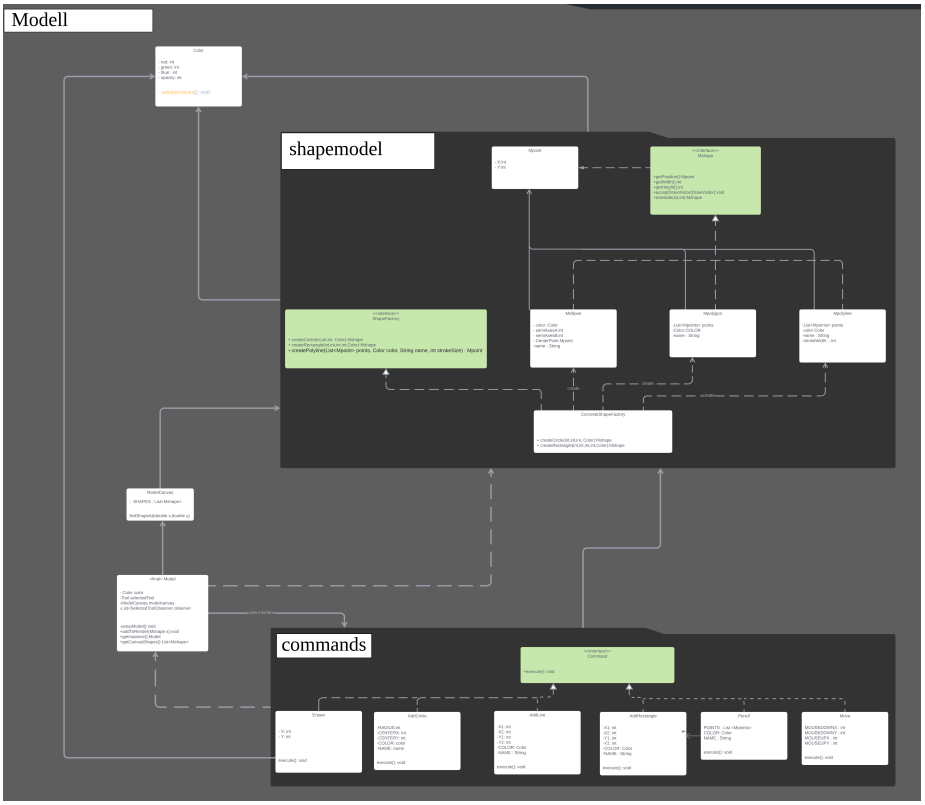
För programmet eftersträvas en god objekt-orienterad design. För att få till en bra design så används UML-diagram, en domänmodell och flera designmönster.

En viktig separation är mellan modellen och vad som visas upp. För att åstadkomma detta används designmönstret MVC. Programmet skrivs för att hålla modellen fri från beroenden till andra bibliotek och undvika stark coupling mellan olika paket.

Programmets vy är väldigt specialiserad till just det här programmet vilket kommer att göra den svår att återanvända.

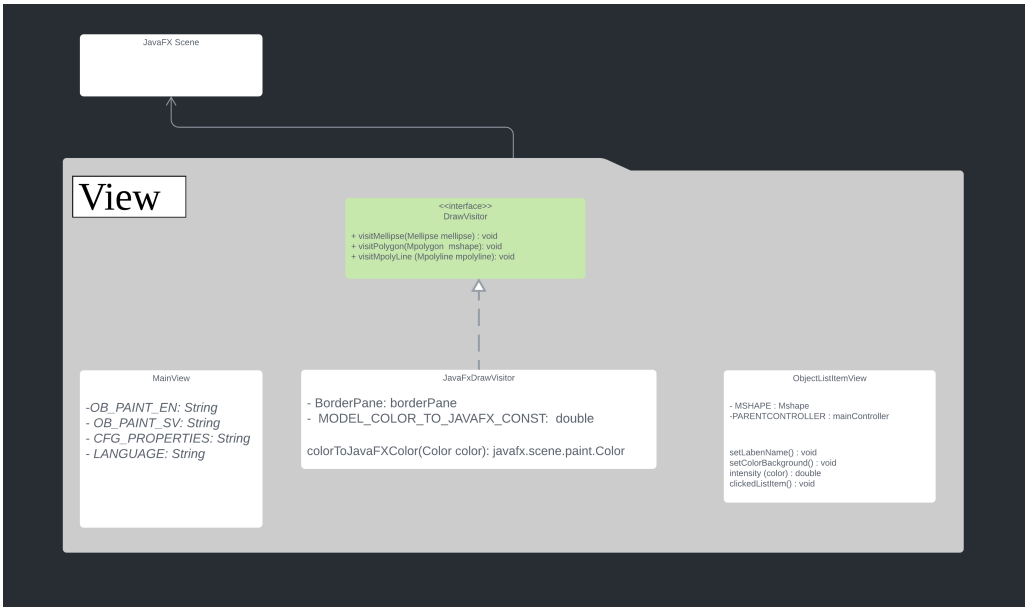
3.1 UML-klassdiagram

I denna sektion presenteras en övergripande vy av UML-diagram för de olika paketen. För en mer detaljerad vy går hela UML-diagrammet att hämta från [2].

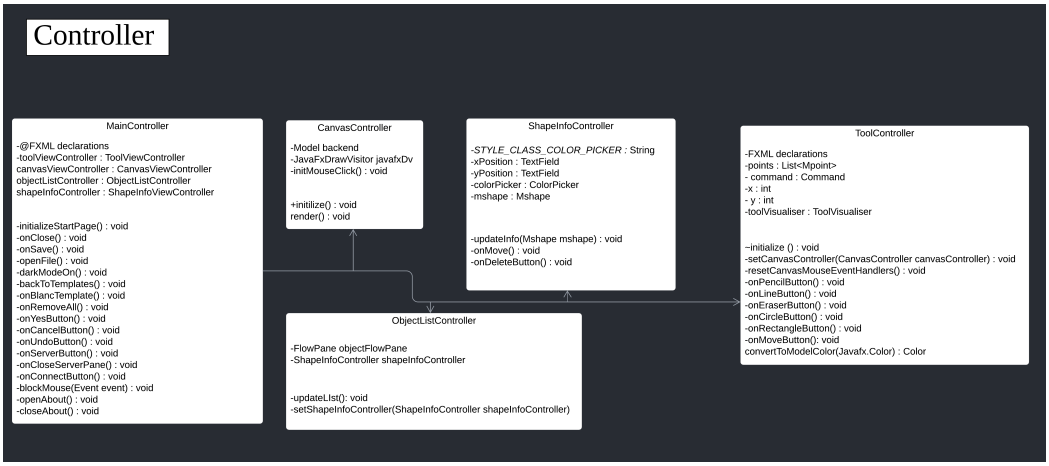


Figur 2: UML klassdiagram för modellen

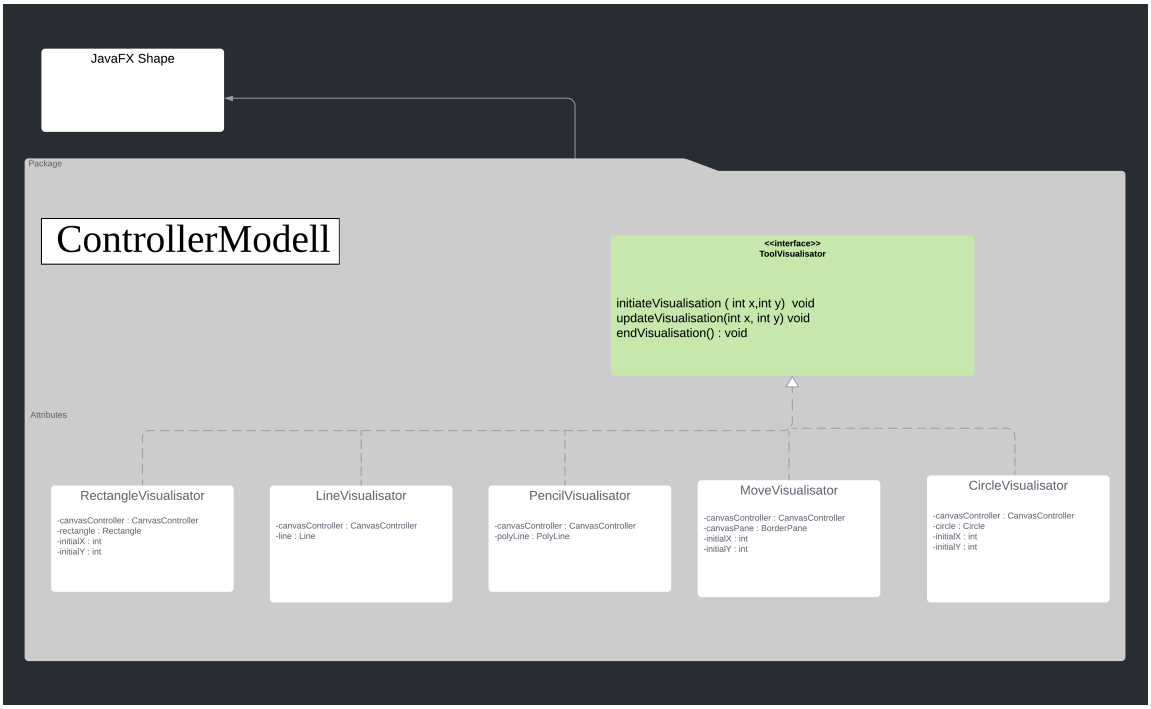
Figur 3: UML klassdiagram för FileManager och Server



Figur 4: UML klassdiagram View-paketet



Figur 5: UML klassdiagram Controller-paketet

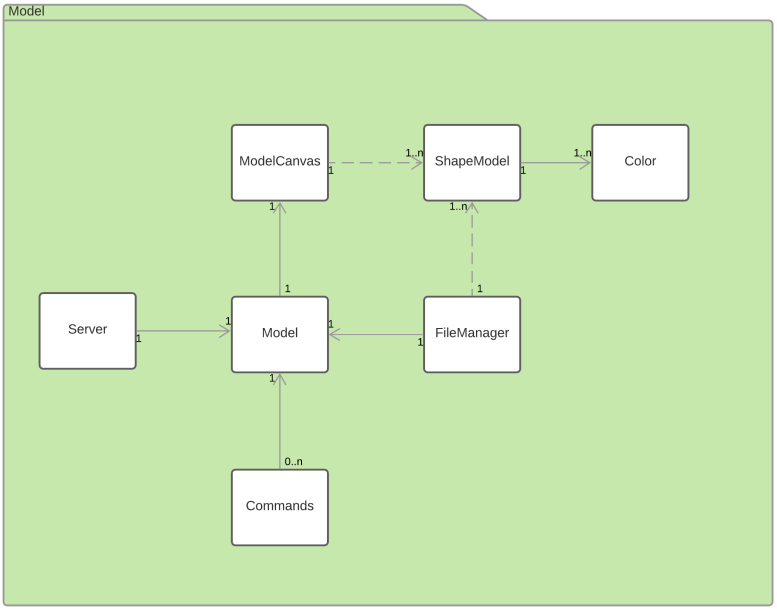


Figur 6: UML klassdiagram ControllerModel-paketet

3.2 Domänmodell

Här presenteras programmets design i en domänmodell.

3.2.1 Domänmodell



Figur 7: Domänmodell

3.2.2 Analys av beroenden

Det finns ett cirkulärt beroende i programmet. Vilket är dåligt men skapas på grund av att Visitor Pattern används Gränssnittet ”Mshape” har metoden acceptDrawVisitor som har en parameter med typ DrawVisitor. Gränssnittet DrawVisitor däremot har metoderna visitX där X är en klass som implementerar gränssnittet Mshape. Detta har risk att bli farligt men det är tänkt att de som implementerar gränssnittet DrawVisitor får hantera ändringar av modellen.

3.3 Designmönster

Här listas och förklaras de viktigaste designmönster som förekommer i systemdesignen.

3.3.1 Visitor Pattern

Visitor Pattern är ett designmönster som låter andra klasser definera nya operationer på en fast datastruktur [3].

ObPaint använder designmönstret för att översätta den interna modellen till ett anpassat gränssnitt som är kompatibelt med javafx. För utritning av modellen använder ObPaint visitor pattern genom ”JavaFxDrawVisitor.java” som implementerar gränssnittet ”Drawvisitor.java”. När programmet ska måla ut modellen i det visuella gränssnittet översätts modellen till Javafx-grafik. Ett annat fall är när modellen ska representeras i SVG-format [4], då implemterar klassen ”SvgDrawVistitor.java” gränssnittet. Klassen har metoder för att omvandla modellens representation av former till SVG.

3.3.2 Factory Pattern

Factory Pattern används för att abstrahera skapandet av olika former. För att exempelvis skapa en cirkel, så skapas en ”ConcreteShapeFactory” som implementerar gränssnittet ”ShapeFactory” som håller metoder för att skapa former, bland annat ”createCircle”. Metoden som anropas i kommandot ”AddCircle” är då ”createCircle”. Mer om programmets olika commands under 3.3.3

3.3.3 Command Pattern

För att på ett flexibelt sätt kunna sköta direktiven från Controller-klasserna har ett Command Pattern implementerats. Det innebär att ett Interface som heter Command har skapats och alla command-klasser tvingas implementera metoderna execute samt undo. Beroende på vilket Command som skapas och som kallas på i controller-klasserna så utför command-klasserna det som varje klass själv har ansvar för. I programmet ansvarar exempelvis kommandot “addRectangle” för att kalla på mPolygons konstruktor via en ShapeFactory och skapar rätt objekt. Alla objekt som skapas för att målas ut skapas via factory för att öka abstraktionen i programmet.

3.3.4 Model View Controller (MVC)

MVC är den generella arkitekturen som projektet följer. Där ingår paketet “Model” som håller i basprogrammet utan något grafiskt gränssnitt kopplat. Till modellen finns sedan paketen “Controller” och “View” som står för det grafiska gränssnittet samt möjligheten för användaren att interagera med modellen.

3.3.5 Singleton Pattern

Singleton - Används då man vill komma åt modellen för att endast en instans ska bli skapad och använd mellan alla paket. Singleton ses ofta som ett ”Anti-Pattern”. Användningen av Singleton Pattern försvårar testningen av programmet på grund av att samma instans stannar kvar. Användningen av en singleton skapar problem med god objekt-orienterad design. En singleton kan bli som en global referens [3]. Vilket inte överensstämmer med bra objekt-orienterad design. I ObPaint så är klasserna ”Model” och ”ObPaintClient” Singletons. Vilket är en brist med designen.

4 Datahantering

Programmet skapar en fil “cfg.properties” när programmet startas med 2 olika inställningar. Vilken port som programmet ska använda och vilket språk som programmet ska ha. Den här filen använder en enkel struktur, varje rad sparar en nyckel och ett värde. Med nyckeln kan inställningen till den raden hämtas.

I programmet så går det att spara och öppna bilder, då används formatet SVG. I SVG formatet används bara polyline, polygon och ellipse. Det går också att läsa in en SVG-fil i programmet men då läser den bara in polyline, polygon och ellipse. Här finns en begränsning att polygon och ellipse tolkas som en rektangel respektive cirkel. Dock har programmet bara verktyg för att skapa dessa former. Ingen bild öppnas automatiskt när programmet startas.

Utöver detta sparas ingen information av programmet.

Programmet använder vissa resurser som ikoner, FXML-filer, CSS-filer och Properties-filer. Dessa sparas i roten av programmet. Under mapparna “images”, “fxml” och “css”. Properties filer används för att spara strängar på språken svenska och engelska. I gränssnittet används nycklar för att läsa ut vilken sträng som ska användas.

5 Kvalitet

Testning av programmet kommer att vara svårt eftersom en stor del av programmet sker dynamiskt på ett grafiskt gränssnitt. Tester har utförts via test ramverket JUnit. Testfilerna består utav Java kod och är placerade i en speciell mapp placerad enligt Mavens standard för mappar.[5]
Testerna som utförs är på modellen. De olika formerna från modellen testas med avseende på deras olika egenskaper. Exempel på saker som kan testas

är storlek, position och färg. Ytterligare tester som utförs är tester om vad som ska renderas på skärmen samt vad som ska tas bort från renderingen. Testerna körs automatiskt varje gång koden laddas upp till github. Detta sker via Travis [6].

5.1 Kända problem

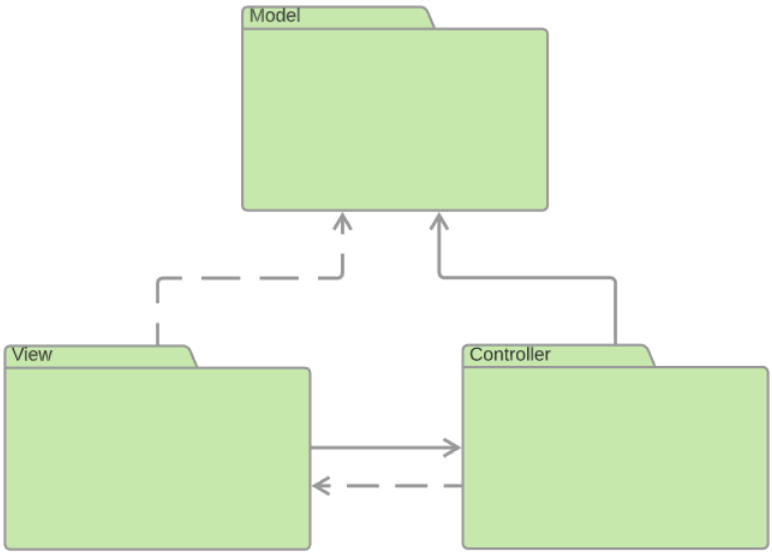
Som nämnts ovan har det varit en utmaning att testa viss funktionalitet med hjälp av JUint-tester. Speciellt har det varit en utmaning att testa de dynamiska ändringar som sker vid knapptryck. Vissa tester har även varit svåra att se resultat på då testmiljön har saknat ett grafiskt gränssnitt. En av de stora bristerna i designen har varit att en Controller måste definieras i vyn eftersom Javafx används. Detta skapar ett ”onödigt” beroende mellan kontrollen och vyn. Det beror på hur javafx är uppbyggt och något som inte har gått att undvika i designen.
Programmet använder sig av mycket minne, speciellt vid stora bilder. Detta beror på att det skapas 2 olika objekt för varje form skapas. En som represenation i modellen och en som målas ut med JavaFX.

I ObPaint finns möjligheten att samarbeta synkront. För att detta ska vara möjligt måste man portvidarebefodra samt ha tillgång till varandras IP-adresser. Detta ses som ett problem eftersom majoriteten av applikationer där möjligheten att arbeta synkront finns har en annan lösning. Exempelvis en inloggning eller en länk.

5.2 Resultat av analytiska verktyg

Resultat från Maven site samt hela “Sitemapp” med pmd finner du här. Anledningen till att programmets PMD har en ”UnusedLocalVariable ry” är för att programmet just nu bara kan skapa cirklar men inte ellipser, därför finns denna variabel som inte används. Den skulle däremot användas vid senare implementation av ett commando för att skapa andra olika typer av ellipser. Ytterligare en sak man finner är en ”UnusedLocalVariable port”. Anledningen till det är att programmet vid uppstart skapar ”cfg.properties” som är en konfigureringsfil där port och språk ställs in. Om programmet inte startats innan så finns inte denna fil. I analysen påträffas även alla metoder som anropas i Javafx. De metoder som listat är alla metoder som reagerar på knapptryck och vars funktionalitet hittas i Controllern. Anledning till reaktionen är att metoderna används först vid körning vilket resulterar i att analysen indikerar på att metoderna inte används.

5.3 MVC



Figur 8: MVC

5.3.1 Beskrivning av programmets MVC-arkitektur

Figur 8 illustrerar programmets MVC-arkitektur. ”Controller”:n i MVC-mönstret har en ”Model” att tillgå samtidigt som ”View” använder modellen för att läsa data. ”View” håller i sin tur en controller där ”View”:en skickar tillbaka en referens till controllern. Det hade varit önskvärt att inte ha någon koppling mellan View och Model. Som nämnt tidigare i kapitlet har det på grund av hur Javafx är uppbygg inte varit möjligt att komma ifrån beroendet mellan Controller och View. I och med att delar av modellen översätts till Javafx för utritning finns ett beroende mellan View och Model genom klassen DrawVisitor 3.3.1. Detta beroende är inte heller önskvärt och hade kunnat undvikas med hjälp av en annan struktur.

6 Referenser

Lucidchart, verktyg som användes för att skapa alla UML-diagram. [7]

JavaFX, bibliotek som används för att rita upp på skärmen. [8]

JUnit, plattform som används för att utföra enhetstester. [9]

Travis CI, Continuous Integration för att utföra tester när ändringar av programmet görs. [6]

Maven, plattform för att sköta byggandet av programmet. [10]

Github, versionshanteringsplattform på internet. [11]

[1] Wikipedia. (26 sept. 2020). SOLID, URL: https://en.wikipedia.org/wiki/SOLID. (Hämtad 22/10 2020).

[2] Team JEEA. (23 okt. 2020). ObPaint UML, URL: https://app.lucidchart.com/publicSegments/view/3dc8a3a9-9576-4d1b-9cd8-d82c3bc9aeb3/image.png. (Hämtad 23/10 2020).

[3] D. Skrien, *Object-Oriented Design Using Java*. New York: McGraw-Hill, 2008, (Hämtad 23/10 2020).

[4] W3. (4 okt. 2018). Scalable Vector Graphics (SVG) 2, URL: https://www.w3.org/TR/SVG2/. (Hämtad 23/10 2020).

[5] Apache. (19 okt. 2020). Introduction to the Standard Directory Layout, URL: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html. (Hämtad 23/10 2020).

[6] Travis CI. (2020). Travis CI, URL: https://travis-ci.com/. (Hämtad 23/10 2020).

[7] Lucid Software Inc. (2020). Lucidchart, URL: https://lucid.app/. (Hämtad 23/10 2020).

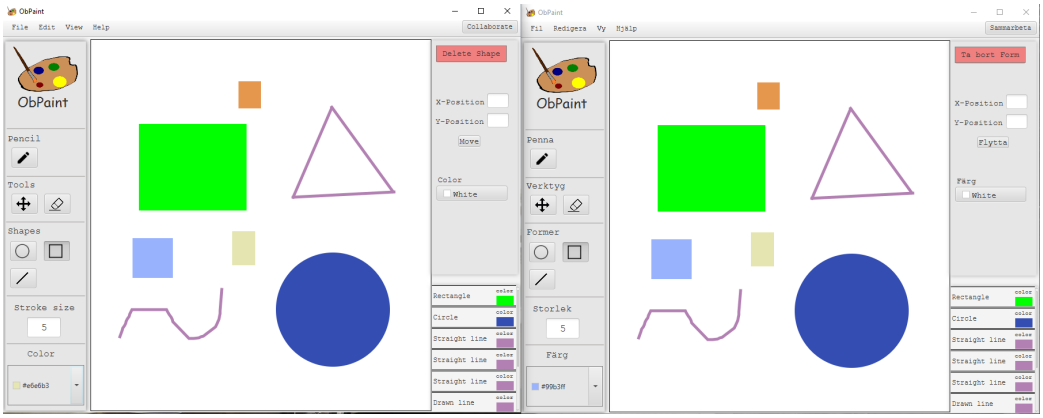
[8] Oracle. (2018). JavaFX Javadocs, URL: https://openjfx.io/javadoc/11/. (Hämtad 23/10 2020).

[9] The JUnit Team. (2020). Junit, URL: https://junit.org/. (Hämtad 23/10 2020).

[10] Apache. (19 okt. 2020). Maven, URL: https://maven.apache.org/. (Hämtad 23/10 2020).

[11] Team JEEA. (23 okt. 2020). ObPaint Github, URL: https://github.com/0x3D/TDA367OOP. (Hämtad 23/10 2020).

7 Bilagor


ObPaint
File Edit View Help
Collaborate
ObPaint
Pencil
Tools
Shapes
Stroke size
5
Color
Delete Shape
X-Position
Y-Position
Move
Color
White
Rectangle
Circle
Straight line
Straight line
Straight line
Drawn line
ObPaint
Fil Redigera Vy Hjälp
Samarbeta
ObPaint
Penna
Verktyg
Former
Storlek
5
Färg
Ta bort Form
X-Position
Y-Position
Flytta
Färg
White
Rectangle
Circle
Straight line
Straight line
Straight line
Drawn line