# Projektbeskrivning

”Illustrator Lite”

2015-02-16

## Projektmedlemmar:

Natanael Log <natlo809@student.liu.se>

# Innehållsförteckning

# 1. Introduktion till projektet

Detta projekt syftar till att ta fram ett program programmerat i språket Java. Programmet kommer heta Illustrator Lite och är inspirerat av det kända illustrationsprogrammet tillverkat av Adobe: Illustrator. Som namnet antyder är detta en lite-version av originalet, alltså en version med betydligt mindre funktioner och features som Adobe Illustrator har.

Illustrator Lite är alltså ett ritprogram där användaren ska kunna rita olika former och skriva text som sedan ska kunna konverteras till SVG-format (Scalable Vector Graphics) för att kunna användas i t.ex webapplikationer. Programmet ska också kunna ladda upp en redan existerande SVG-fil som användaren ska kunna redigera och spara.

# 2. Ytterligare bakgrundsinformation

SVG (Scalable Vector Graphics) är ett standardformat för att beskriva vektor-grafik. Det beskrivs med XML-syntax (eXtensible Markup Language) och är väldigt användbart, särskilt för webapplikationer då HTML5 direkt har stöd för SVG-kod. XML-syntax byggs upp med något som kallas DOM (Document Object Model) och för att kunna hantera en SVG-fil behöver Java en DOM-parser för att ta hand om varje element i SVG-filen.

**Referenser:**

*Wikipedia*, *Scalable Vector Graphics*. Uppdaterad 2014-11-06 [www]
< http://sv.wikipedia.org/wiki/Scalable_Vector_Graphics >
Hämtad 2015-02-16

*Wikipedia*, *Extensible Markup Language*. Uppdaterad 2013-08-08 [www]
< http://sv.wikipedia.org/wiki/XML >
Hämtad 2015-02-16

*Wikipedia*, *Document Object Model*. Uppdaterad 2013-03-10 [www]
< http://sv.wikipedia.org/wiki/Document_Object_Model >
Hämtad 2015-02-16

# 3. Milstolpar

Illustrator Lite kommer bestå av en hel del features och kärnan i koden ska byggd så att fler features ska kunna läggas till utan att behöva programmera om grundläggande delar. Programmet kommer behöva klasser som hanterar former, text, användargränssnitt, SVG-kompatibilitet, eget filformat och säkert mer som dyker upp under projektets gång.

Så för att bryta ner utvecklingen av Illustrator Lite skapas följande tabell av milstolpar:

*Tabell 1. Milstolpar för utveckling av Illustrator Lite.*

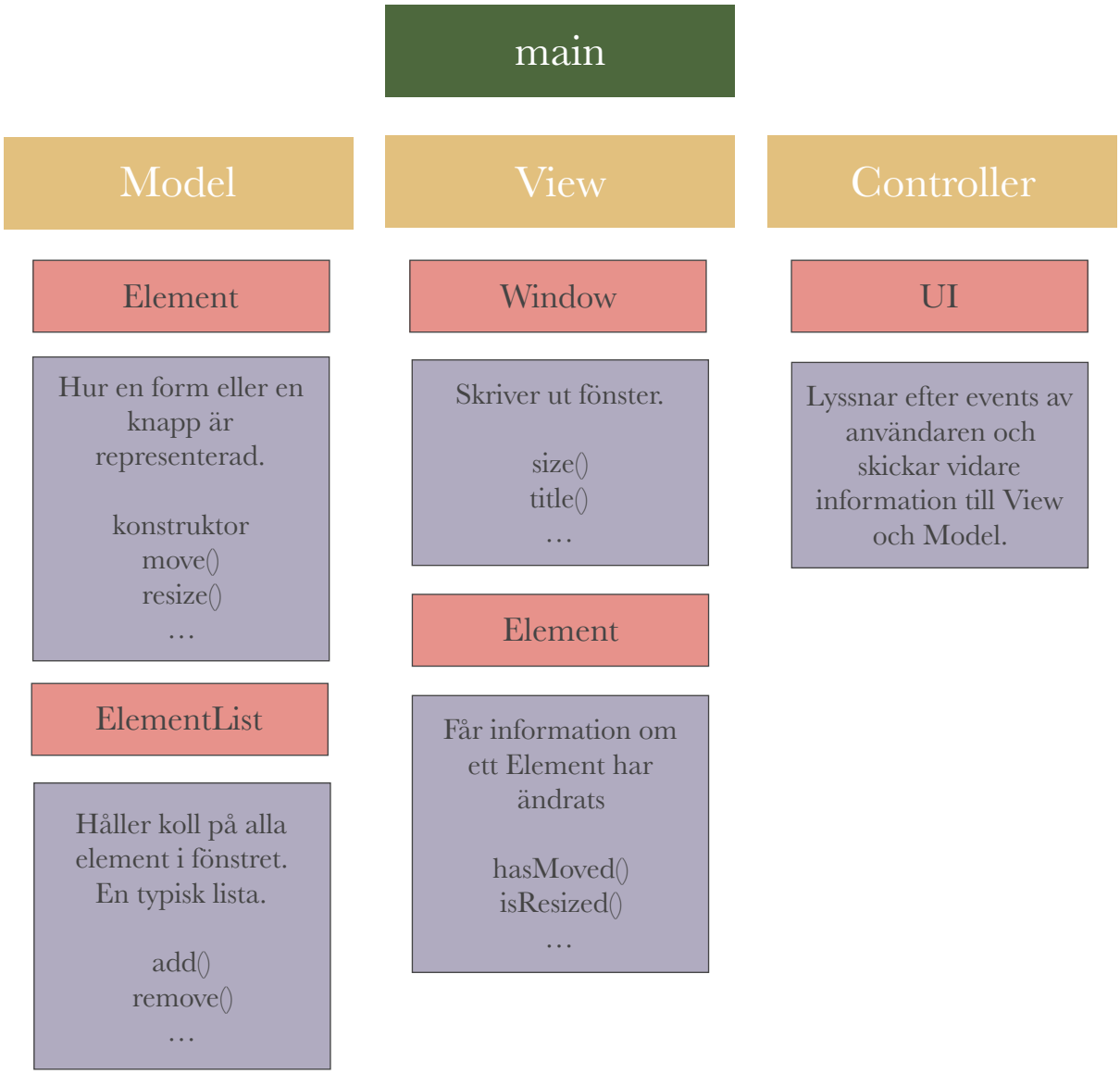
| # | Beskrivning |
|---|---|
| 1 | Ett programfönster ska dyka upp när man startar programmet. |
| 2 | Det ska finnas en knapp som skapar en cirkel med fix storlek. |
| 3 | Det ska finnas en knapp som skapar en rektangel med fix storlek. |
| 4 | Man ska kunna skapa cirklar och rektanglar där storleken bestäms av användaren vid skapandet. |
| 5 | Användaren ska kunna ändra färg på formerna. |
| 6 | Det ska finna en knapp som skapar en text-form som användaren får skriva in. |
| 7 | Det ska finnas en knapp som markerar en form. |
| 8 | Användaren ska kunna välja bland olika typsnitt hos en text-form. |
| 9 | Användaren ska kunna ändra storleken på en form med hjälp av siffror. |
| 10 | Användaren ska kunna flytta på en form. |
| 11 | Användaren ska kunna ta bort en form. |
| 12 | Användaren ska kunna ändra storlek på en form genom att markera en form och dra i dens hörn. |
| 13 | Användaren ska kunna kopiera en form och klistra in den igen. |
| 14 | Det ska finnas en knapp som sparar filen i ett eget bestämt filformat. |
| 15 | Det ska finnas en knapp som öppnar en fil i ett bestämt filformat. |
| 16 | Användaren ska kunna spara filen som .svg. |
| 17 | Användaren ska kunna öppna en SVG-fil. |
| 18 | Det ska finnas en knapp som skapar en n-hörning med n antal hörn. |
| 19 | Användaren ska kunna zooma. |

# 4. Övriga implementationsförberedelser

Programmet kommer till grund bygga på Model-View-Controller-mönstret. En Model ska t.ex kunna vara hur former är uppbyggda och representerade, likaså alla knappar. View kommer utnyttja något GUI som är inbyggt, förmodligen Swing, för att rita ut allting på skärmen. Controller kommer vara alla händelser som användaren gör på de olika formerna och knapparna.

Programmet kommer utnyttja någon inbyggd DOM-parser för att hantera SVG-filerna och för att kunna konvertera till SVG när man sparar en fil. Jag tror detta kommer ta längst tid att implementera.

Hela programmet kommer vara uppbyggt (väldigt simplifierat) på det här sättet:



# 5. Utveckling

Jag kommer jobba ensam med detta projekt vilket gör att jag inte behöver ta hänsyn till hur andra gruppmedlemmar skulle vilja jobba. Jag kommer jobba med detta utanför schemat eftersom detta är en kurs jag läser utöver mitt ordinära schema. Dock kommer jag ändå använda GitHub för att kunna se historik och statistik över hur jag jobbat.

# 6. Implementationsbeskrivning

## 6.1 Milstolpar

| # | Beskrivning | Utförd | Kommentar |
|---|---|---|---|
| 1 | Ett programfönster ska dyka upp när man startar programmet. | Ja | |
| 2 | Det ska finnas en knapp som skapar en cirkel med fix storlek. | Ja | |
| 3 | Det ska finnas en knapp som skapar en rektangel med fix storlek. | Ja | |
| 4 | Man ska kunna skapa cirklar och rektanglar där storleken bestäms av användaren vid skapandet. | Nej | En form skapas alltid med en fix storlek. Användaren kan dock ändra storleken efter skapandet genom att dra i formens hörn eller ändra värden manuellt. |
| 5 | Användaren ska kunna ändra färg på formerna. | Ja | |
| 6 | Det ska finna en knapp som skapar en text-form som användaren får skriva in. | Ja | |
| 7 | Det ska finnas en knapp som markerar en form. | Ja | När knappen har tryckts kan användaren markera en form genom att klicka på formen. |
| 8 | Användaren ska kunna välja bland olika typsnitt hos en text-form. | Nej | Inte hunnit med. |
| 9 | Användaren ska kunna ändra storleken på en form med hjälp av siffror. | Ja | |
| 10 | Användaren ska kunna flytta på en form. | Ja | Antingen genom att dra formen med musen eller ändra X- och Y-koordinater. |
| 11 | Användaren ska kunna ta bort en form. | Ja | |
| 12 | Användaren ska kunna ändra storlek på en form genom att markera en form och dra i dens hörn. | Ja | |
| 13 | Användaren ska kunna kopiera en form och klistra in den igen. | ~Ja | Det finns en knapp som duplicerar en markerad form. |
| 14 | Det ska finnas en knapp som sparar filen i ett eget bestämt filformat. | Ja | Det egna bestämda filformatet är .svg. |
| 15 | Det ska finnas en knapp som öppnar en fil i ett bestämt filformat. | Ja | |
| 16 | Användaren ska kunna spara filen som .svg. | Ja | |
| 17 | Användaren ska kunna öppna en SVG-fil. | Ja | |
| 18 | Det ska finnas en knapp som skapar en n-hörning med n antal hörn. | Nej | Inte hunnit med. |
| 19 | Användaren ska kunna zooma. | Nej | Inte hunnit med. |

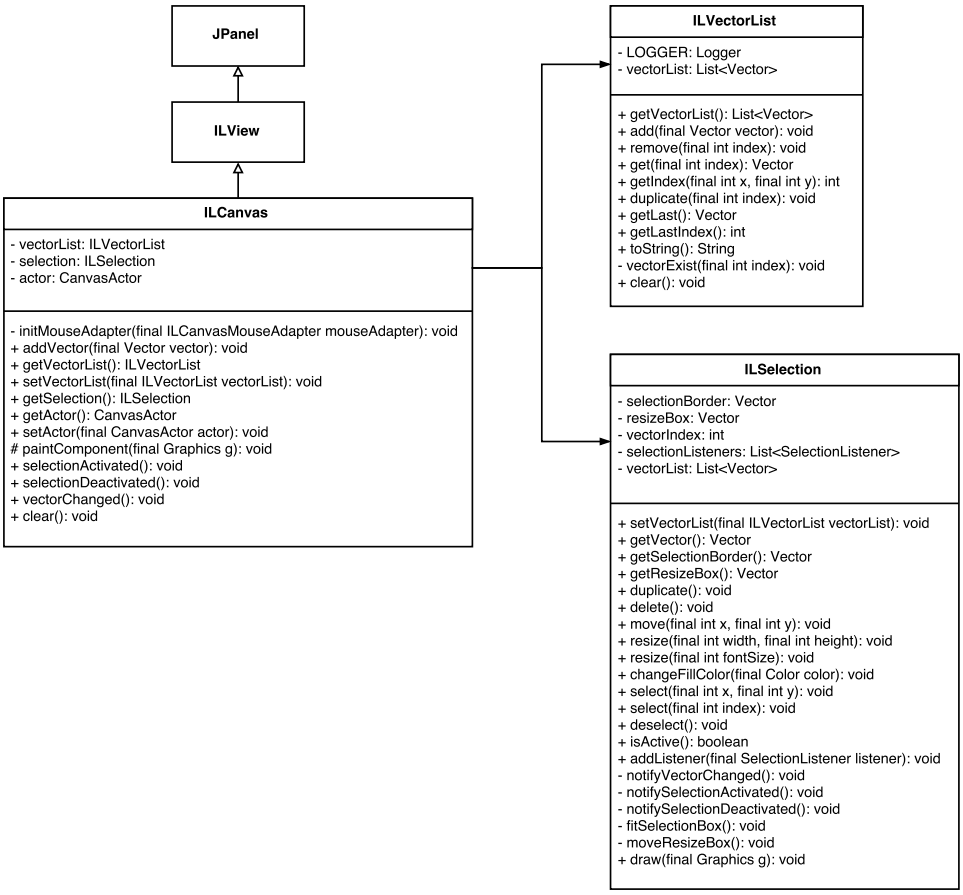
## 6.2 Programmets struktur

Programmets kärna är ”IllustratorLite”-klassen som bygger upp en fönster (ILFrame), en kanvas (ILCanvas), en filhanterare (ILFileHandler), en meny (ILCanvasMenu, ILFileMenu, ILSelectionMenu) och ett verktygsfält (ILModeBar, ILSelectionPropertyBar, ILSelectionBar).

### 6.2.1 ILCanvas

ILCanvas är en klass som håller reda på alla vektorer, markeringen och vilket läge programmet är i. Vektorerna hålls i klassen ILVectorList som utökar klassen ArrayList<Vector> med några metoder, bland annat `getIndex(int x, int y)` som returnerar index hos den första vektorn som innehåller koordinaterna x och y. ILSelection hanterar markeringen av en vektor. Den ritar t.ex. ut en blå ram med en blå låda i det nedre högra hörnet av den markerade vektorn för att tydligt indikera att vektorn är markerad. Den innehåller också en lista på lyssnare (SelectionListener) som behöver få reda på om någonting har hänt med markeringen (t.ex. behöver ILCanvas ritas om varje gång markeringen flyttats eller omformats).

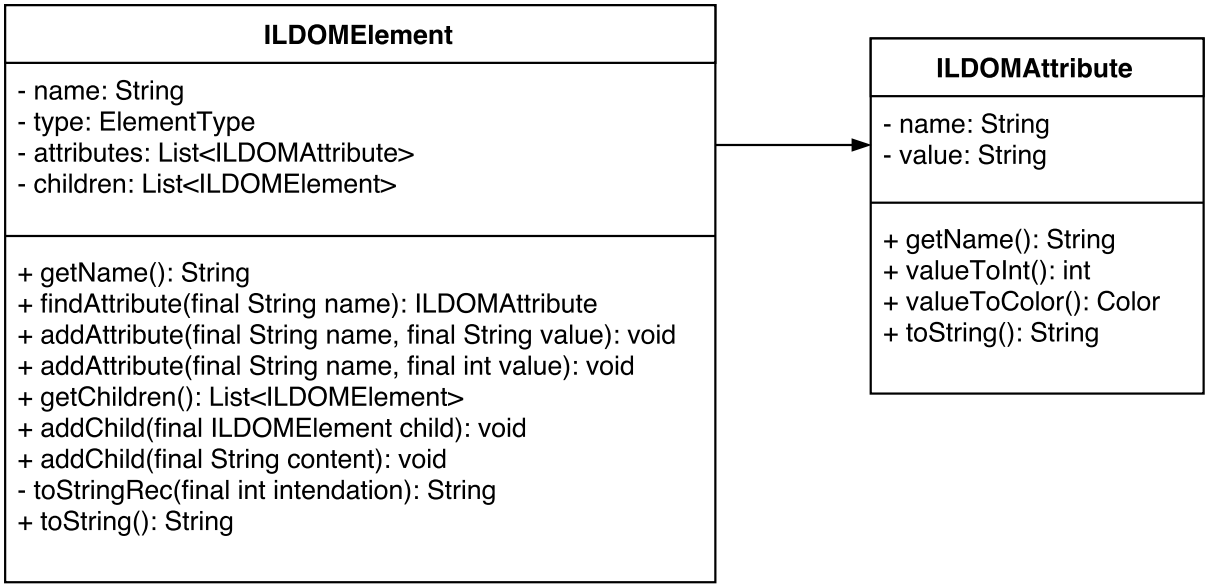
ILCanvas består också av ILCanvasMouseAdapter som utökar klassen MouseInputAdapter. Denna klass använder metoderna `MouseClicked()`, `MouseDragged()` och `MousePressed()` för att låta användaren skapa och ändra på vektorer. Beroende på programmets läge (CanvasActor) händer olika saker när man klickar på ILCanvas.




*Figur 1. UML-diagram över ILCanvas, ILVectorList och ILSelection.*

## 6.2.2 ILFileHandler

Vektorerna som är utritade på ILCanvas kan konverteras till SVG-format och sparas till en fil. På samma sätt kan en SVG-fil konverteras till vektorer och ritas ut på ILCanvas. Detta hanterar ILFileHandler. Den har en ILSVGAdapter som kan konvertera antingen från SVG till vektorer eller tvärtom. ILSVGAdapter utnyttjar klassen ILDOMParser som hämtar in hela SVG-filen som läses in och går igenom den, element för element och skapar så kallade ILDOMElement av de element den hittar. Varje element har ett namn, en lista på attributer och en lista på barn.




Figur 2. *UML-diagram över ILDOMElement och ILDOMAttribute.*

## 6.3 Användning av fritt material

Klassen MigLayout har använts som layouthanterare för de olika GUI-komponenterna.

## 6.4 Användning av objektorientering

1. **Användning av gränssnitt.**
   Klassen *SelectionListener* är ett gränssnitt som används av t.ex. ILSelectionBar. ILSelection innehåller en lista på dessa lyssnare och så fort markeringen har markerats, avmarkerats eller ändrat form anropas respektive metod hos alla lyssnare som implementerar SelectionListener. ILSelectionBar utnyttjar detta för att veta om knapparna ”Delete”, ”Duplicate” och ”Color” ska synas eller inte (de ska bara synas när en vektor är markerad). Att lösa detta på ett icke-objektorienterat sätt vore att t.ex. direkt i klassen ILSelection ha en referens till knapparna och direkt där göra dem osynliga eller inte. Definitivt så är det mycket smidigare att utnyttja ”lyssning” med gränssnitt. Då räcker det med att någonstans ha en referens till ILSelection och vardera lyssnare så att man kan lägga till lyssnarna i ILSelections lista. Sen kommer ILSelection loopa igenom alla lyssnare och kalla på respektive relevant metod och lyssnarna kommer köra sina egna implementationer av funktionerna.
2. **Sen/dynamisk bindning**
   Den abstrakta klassen ILVector har en metod som heter resize(), den tar in en bredd och en höjd och sätter respektive instansvariabel till det. Klassen Text överskuggar superfunktionen resize(), anledningen är att en text-vektor inte kan ändra storlek hur som helst - ändringen måste vara proportionerlig ursprungsstorleken hos vektorn. Att göra detta på ett icke-objektorienterat vis vore att varje gång man ska ändra storlek hos en

vektor kollar man manuellt om vektorn är av text-typ, och isåfall ändrar man storlek utefter det. Nu räcker det med att bara kalla på resize() oavsett vilken sorts vektor det är. I klassen ILCanvasMouseAdapter och metoden mouseDragged() utnyttjas dynamisk bindning då metoden resize() kallas oavsett vilken sorts vektor som getSelection() pekar på. Att utnyttja detta kan spara mycket kod eftersom om man skulle behöva ändra storlek på vektorer på flera ställen i programmet skulle man behöva skriva in det speciella fallet ”if textvektor”. Och skulle man lägga till ytterligare en vektor som har en speciell resize skulle det blir ännu mer if-satser.

3. **Konstruktorer**
   Klassen ILParseException (filehandler.svg.dom) används för att hantera fel när man läser in en DOM-fil. Den kan visa var i DOM-filen felet låg och skriva ut det meddelande som skickades med. I klassen ILDOMParser och metoden validateTree() kastas ILParseException två gånger, en gång med tre parametrar och en gång med bara en. Det är för att man inte alltid behöver visa var felet låg om det är uppenbart. T.ex. *”Did not find a ’<’-character at the end of the file.”* behöver inte också skriva ut slutet av filen eftersom meddelandet redan säger det. Men i det andra fallet, *”Should not find a ’<’-character here.”* skickas radnummer och input-strängen med i konstruktorn till ILParseException därför att man vill enkelt kunna se i konsolen exakt var i strängen felet låg. Här utnyttjas konstruktor-konceptet med flera olika konstruktorer till samma klass för att initiera olika fält i olika sammanhang. På ett icke-objektorienterat sätt skulle detta kunna lösas genom att man t.ex. hade en metod i ILParseException som antingen satte klassen som ”bara-meddelande” och en metod som satte klassen som ”meddelande-och-felsök”. Så varje gång man ska kasta en ILParseException ställer man först in den och sen kastar man den. Det blir väldigt mycket smidigare att bara kunna ställa in klassen direkt i konstruktorn.

## 6.5 Motiverade designbeslut

Programmet har strukturerats om ett antal gånger, främst på grund av dålig planering. Här följer några designbeslut som tagits.

1. ILCanvas är bara en ”behållare” som utöver utritning av alla vektorer kan lägga till en vektor, ge tillgång till markeringen (ILSelection) eller ändra läget (Actor). Tidigare kunde ILCanvas göra fler saker, men det mesta har flyttats över till ILSelection. Den största anledningen till det är att det bara är en markerad vektor man kan göra saker med, vilket gjorde det logiskt att det är ILSelection som ska ha metoder som t.ex. flyttar och omformar.
2. När en användare klickar på en knapp som skapar en ny form skapas inte formen förrän användaren klickar på själva ILCanvas. Tanken var att det skulle dyka upp en dialogruta som frågade efter höjd och bredd hos vektorn som skulle skapas. Istället ritas det direkt ut en vektor som har en förinställd storlek (50 pixlar) och användaren får istället markera vektorn och ändra dess storlek i verktygsfältet eller dra i dess hörn. Detta var en mycket

smidigare lösning då det var krångligt att få till en dialog-klass som skulle innehålla några textfält vars värden skulle skickas ut för att ställa in vektorns storlek.

3. Varje vektor har en bredd och en höjd. Detta kan kännas konstigt eftersom en cirkel kan beskrivas med en radie, men cirkeln togs bort och ersattes utav en ellips som har en bredd och en höjd. Detta har gjort det enklare, dels för att den abstrakta klassen ILVector kan ha dessa fält hos sig (de behövs alltså inte direkt i Ellipse, Rectangle och Text) och dels för att markeringen kunde förenklas eftersom den bara tar bredden och höjden hos vektorn som ska markeras och ritar sen ut sin blåa markering med samma höjd och bredd som vektorn.
4. För att kunna ändra storlek på en vektor direkt genom att skriva in värden i ett textfält hårdkodades logiken tidigare. Nu används istället logiken i klassen FieldInputFilter (som ärver DocumentFilter). På samma sätt utnyttjas gränssnittet DocumentListener för att ändra storlek eller flytta på en form beroende på värdet som skrivs in i ett textfält. Helt enkelt så används ett inbyggt dokumentsystem i Java istället för ett egenbyggt. Ett problem uppstod dock, alla DocumentListener's ändrar storlek eller position hos en vektor när värdet i respektive textfält ändras. Men så fort vektorn ändrar form (ILSelectionPropertybar implementerar SelectionListener) kallas metoden vectorChanged() som i sin tur ändrar värdena i varje textfält som i sin tur ändrar formen på vektorn, och så vidare. För att hindra denna loop används variabeln allowUpdate som helt enkelt synkar denna läs- och skrivning för textfälten. Det kanske inte är den smidigaste lösningen, men den fungerar.
5. Till en början skulle programmet bestå av en stenhård MVC-struktur, alltså en Model-, View- och Controller-klass. ILController skulle ta hand om programmets läge (ClickMode) och göra ändringar på vektorer och så vidare. ILView skulle vara grunden till alla GUI-element (vilket den faktiskt är nu med) och ILModel skulle innehålla alla vektorer. Stor vikt lades vid ILController, den fick alldeles för mycket att göra och kopplades till många ologiska klasser. T.ex. skulle den innehålla dimensionsvärden på vektorer som skapades genom en dialogruta. Allting är nu mer uppdelat och relevanta metoder sitter hos relevanta klasser.

## 6.6 Bredd inom Java och objektorientering

1. **Egen enum-typ:** T.ex. MousePressState som används för att hålla koll på var användaren trycker ner muspekaren.
2. **Grafiskt gränssnitt:** Fönstret består av knappar och alla vektorer kan ändras på grafiskt.
3. **Egen GUI-komponent:** ILCanvas är en komponent som ritar ut alla vektorer.
4. **Designmönstret State:** När användaren klickar på ILCanvas händer olika saker beroende på vilken CanvasActor som är satt. Syns tydligt i klassen ILCanvasMouseAdapter.

5. **Filhantering:** Klassen ILFileHandler kan öppna en SVG-fil och konvertera den till vektorer. Den kan också konvertera vektorer till SVG-format och skriva det till en fil.
6. **Egen exception-klass:** T.ex. ILParseException.
7. **Loggning:** Finns i klassen ILVectorList.
8. **Tangentbordsstyrning:** Alla knappar i menyn kan styras med hjälp av tangentbordet. Syns t.ex. i klassen ILFileMenu.

# 7. Användarmanual

Nedan följer en manual som kan användas för att få ut det mesta av Illustrator Lite.

## 7.1 Definitioner

• Verktygsfältet: Området mellan menyn och den svarta linjen.
• Kanvas: Det stora området under verktygsfältet och dess gråa skiljelinje.
• Markeringsruta: Den blå rutan som ritas ut på en markerad form.
• Omformingslåda: Den blå fyrkanten som ritas ut i det nedre hörnet på markeringsrutan.

## 7.2 Skapa ny form

Det finns tre stycken olika former som kan skapas: rektangel, ellips och text. För att skapa en ny form klickar du på dess respektive knapp i verktygsfältet eller under menyn ”Canvas” och klickar sen på kanvasen. Det ska nu ha skapats en svart form med en standardstorlek där du klickade. Observera att när du skapar en ny text-form frågar programmet efter vad texten ska innehålla. Du kan inte ändra detta efter att texten är skapad.

## 7.3 Markera en form

Klicka på knappen ”Select” i verktygsfältet eller under menyn ”Canvas” och sen på en form på kanvasen. Det ska då ritas ut en markeringsruta och en omformningslåda runt formen som visar på att den är markerad.

## 7.4 Flytta en form

Se till att formen ska flyttas först är markerad. Håll sen in musen i markeringsrutan och dra musen runt för att flytta formen. Du kan också flytta den genom att ändra på dess X- och Y-koordinater i verktygsfältet.

## 7.5 Ändra storlek på en form

Se till att formen som ska omformas först är markerad. Håll sen in musen i omformningslådan och dra runt för att ändra dess storlek. Du kan också ändra storlek genom att ändra på dess W- (width, bredd) och H- (height, höjd) eller S- (size, textstorlek) värden i verktygsfältet.

## 7.6 Ta bort, duplicera och ändra färg

Du kan enkelt ta bort en form genom att först markera den och klicka på ”Delete” i verktygsfältet eller under menyn ”Selection”. Duplicering görs på samma sätt men genom att klicka på ”Duplicate”. Ändra färg genom att markera formen som ska ändras och klicka på ”Color”. Välj sen den färg som passar och tryck på ”OK”.

## 7.7 Öppna, spara och skapa en ny kanvas

Under menyn ”File” kan du enkelt skapa en ny kanvas, öppna eller spara en fil i SVG-format som sen ritas ut på kanvasen. För att spara, välj först en mapp och skriv sen in vad filen ska heta (utan filformat!). För att öppna, leta reda på din .svg-fil och välj den.

## 7.8 Snabbknappar

| **Knapp** | **Snabbknapp** |
|---|---|
| Select | V |
| Rectangle | R |
| Ellipse | E |
| Text | T |
| Color | C |
| Duplicate | D |
| Delete | Backspace |
| New | ctrl-N |
| Open | ctrl-O |
| Save | ctrl-S |

# 8. Slutgiltiga betygsambitioner

Jag satsar på betyg 4.

# 9. Utvärdering och erfarenhet

Jag tycker det sammanfattningsvis har varit ett väldigt roligt projekt. Att man fick göra allting från 0 och inte börja med ett kodskelett har varit väldigt givande. Jag har strukturerat om mitt projekt tre gånger vilket gjort att det tagit lite längre tid än väntat. En stor anledning till det var att jag utgick från mina milstolpar en efter en utan att se det slutgiltiga programmet. När jag bockade av milstolpe ett, två, tre och kom till en lite längre fram upptäckte jag att det blev väldigt krångligt att implementera den på grund av hur jag implementerat de tidigare milstolparna. Tillslut gjorde jag ett schema över hela programmet som har hjälpt jättemycket. Jag tror det jag främst lärde mig var att försöka få en helhetsbild över programmet innan man börjar koda. Försöka dela upp allt i moduler och veta vad varje modul förväntar sig få som indata och vad den har som utdata. På det viset kan man jobba med varje modul för sig och man kan bygga ganska komplext rätt så enkelt. Så jag önskar jag hade gjort min projektbeskrivning mer utförlig och vågat se hela programmet och inte bara några milstolpar.

Jag har tyckt det varit svårt att hitta tid och prioritera tid för projektet, mycket för att det var en kurs jag läste utöver mitt vanliga schema. Jag tror inte det spelar så stor roll att man kan jobba hemifrån, man behöver ändå avsätta tid.