

# Projektbeskrivning

”Illustrator Lite”

2015-02-16

Projektmedlemmar:

Natanael Log <natlo809@student.liu.se>

## Innehållsförteckning

1. Introduktion till projektet	1
2. Ytterligare bakgrundsinformation	2
3. Milstolpar	3
4. Övriga implementationsförberedelser	5
5. Utveckling	6
6. Implementationsbeskrivning	7
6.1 Milstolpar	7
6.2 Programmens struktur	8
6.2.1 ILCanvas	8
6.2.2 ILToolbar	9
6.3 Användning av fritt material	10
6.4 Användning av objektorientering	10
6.5 Motiverade designbeslut	11
6.6 Bredd inom Java och objektorientering	12
7. Användarmanual	13
7.1 Definitioner	13
7.2 Skapa ny form	13
7.3 Markera en form	13
7.4 Flytta en form	13
7.5 Ändra storlek på en form	13
7.6 Ta bort, duplicera och ändra färg	13
7.7 Snabbknappar	14
8. Slutgiltiga betygsambitioner	15
9. Utvärdering och erfarenhet	16

## 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](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](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.*

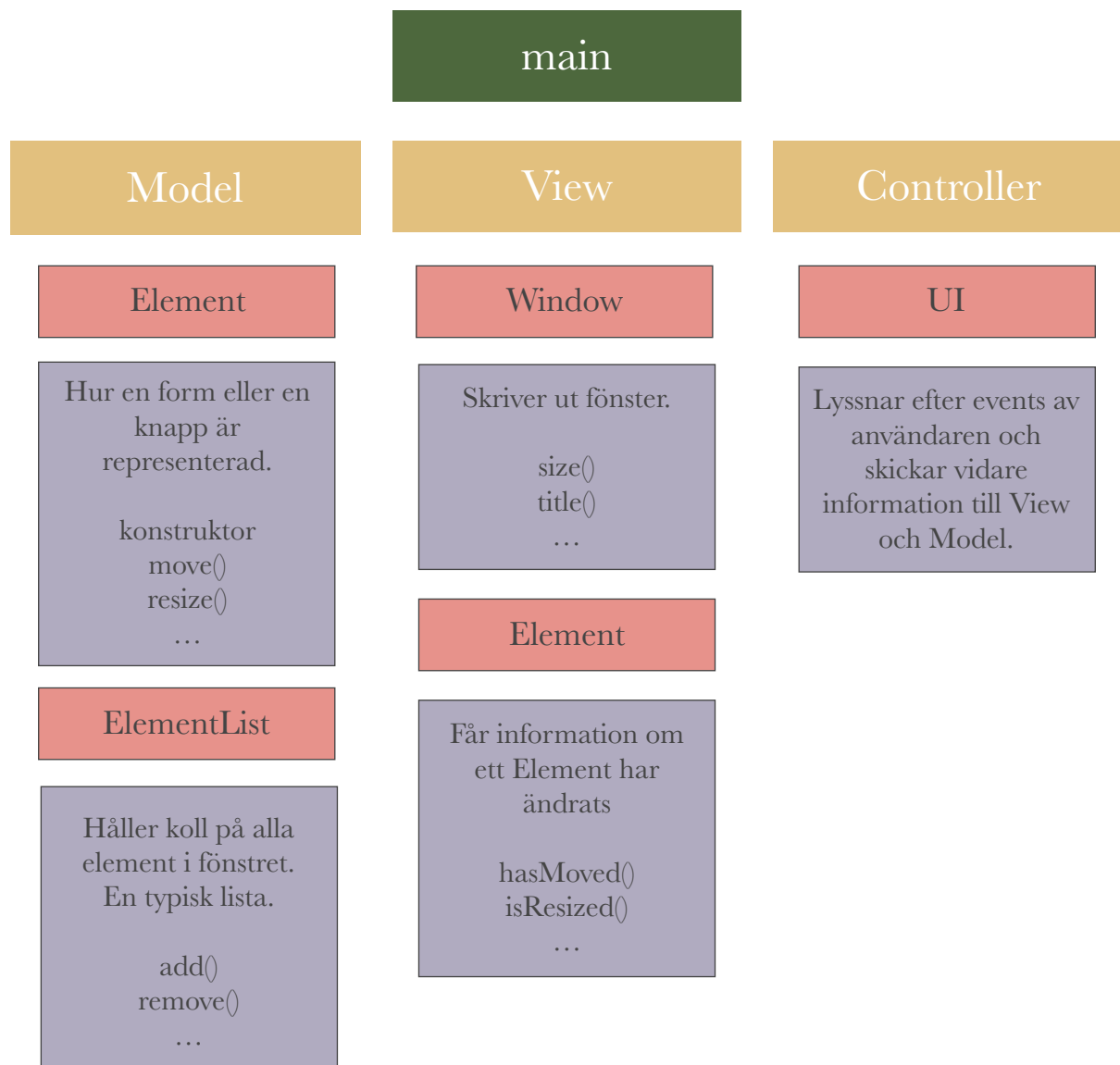
#	Beskrivning
1	Ett programfönster ska dyka upp när man startar programmet.
2	Det ska finnas en knapp som skapar en cirkel med fix storlek.
3	Det ska finnas en knapp som skapar en rektangel med fix storlek.
4	Man ska kunna skapa cirklar och rektanglar där storleken bestäms av användaren vid skapandet.
5	Användaren ska kunna ändra färg på formerna.
6	Det ska finnas 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 (våldigt simplificerat) 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.



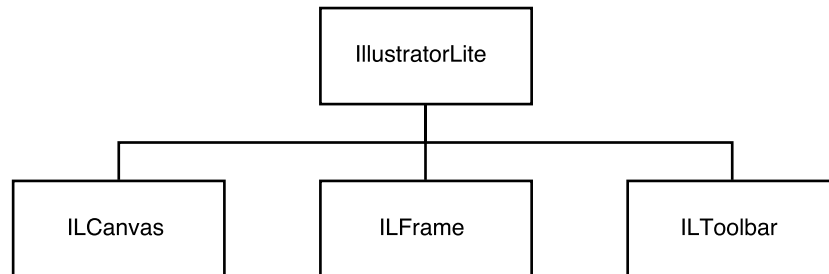
## 6. Implementationsbeskrivning

### 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 finnas 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.	Nej	Inte hunnit med.
15	Det ska finnas en knapp som öppnar en fil i ett bestämt filformat.	Nej	Inte hunnit med.
16	Användaren ska kunna spara filen som .svg.	Nej	Inte hunnit med.
17	Användaren ska kunna öppna en SVG-fil.	Nej	Inte hunnit med.
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

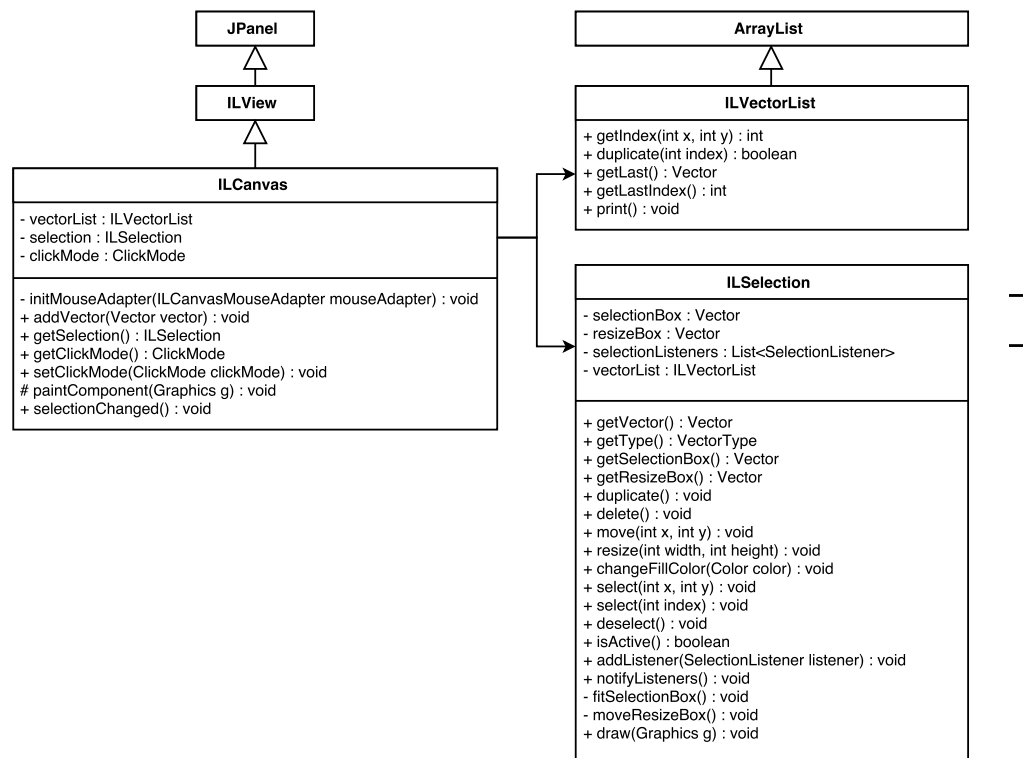
Programmets kärna är ”IllustratorLite”-klassen som består av tre delar: fönstret (ILFrame), kanvasen (ILCanvas) och verktygspanelen (ILToolbar).



### 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 ritat 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).

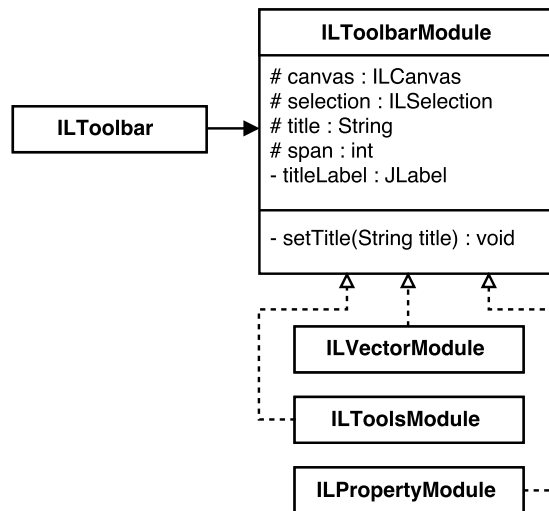
ILCanvas består också av `ILCanvasMouseAdapter` som utökar klassen `MouseInputAdapter`. Denna klass använder metoderna `MouseClicked()`, `MouseDragged()` och `MousePressed()` för att låta användaren skapa och ändra på vektorer. Beroende på programmets läge (`ClickMode`) händer olika saker när man klickar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



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

### 6.2.2 ILToolbar

ILToolbar är en klass som fungerar som en verktygspanel användaren kan använda för att skapa nya vektorer, ta bort vektorer, ändra färg, ändra storlek med mera. ILToolbar består i sin tur av tre stycken moduler (ILVectorModule, ILToolsModule och ILPropertyModule) som utökar den abstrakta klassen ILToolbarModule. Varje modul har olika uppgifter; bland annat att visa knappar som ändrar ClickMode hos ILCanvas och att visa den markerade vektorns position och storlek.



Figur x. UML-diagram över *ILToolbar* och dess moduler.

## 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 *ILCanvas*, *ILPropertyModule* och *ILVectorModule*. *ILSelection* innehåller en lista på dessa lyssnare och så fort markeringen har ändrats på något sätt anropas metoden *selectionChanged()* som används av lyssnarna. Detta är ett väldigt smidigt sätt att låta flera andra klasser få koll på ändringar hos en annan klass.

### 2. Abstrakta klasser.

Abstrakta klasser används på flera ställen. Bland annat klassen *ILVector* är en abstrakt klass som realiserar gränssnittet *Vector*. Den har gjort det enkelt att slippa skriva samma kod hos varje vektor (*Ellipse*, *Rectangle*, *Text*) som ändå har vissa gemensamma fält och metoder.

### 3. Konstruktörer.

Konstruktörer har används i nästan alla klasser för att initiera alla fält. Många klasser (t.ex. *ILCanvasMouseAdapter*) har behövt tillgång till markeringen (*ILSelection*) för att veta om en vektor redan är markerad. Då har *ILCanvas* skickats med i konstruktorn hos *ILCanvasMouseAdapter*.

## 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 (`ClickMode`). 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 under "Property"-modulen 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. `ILPropertyModule` består av ett gäng textfält. Dessa textfält ska bara kunna innehålla siffror vilket krävdes en `KeyAdapter` som kunde hålla koll på det. Så klassen `ILInputField` skapades som utökar klassen `JTextField` och håller koll på tangentbordet genom klassen `ILToolbarKeyAdapter`. Denna klass innehåller en lyssnare som får reda på om något har hänt, denna lyssnare är alltså `ILPropertyModule` som uppdaterar vektorn varje gång en ändring skett i `ILInputField`. Dock skulle inte `ILInputField` behöva finnas eftersom `ILPropertyModule` direkt skulle kunna bestå av några `JTextField` som direkt kunde kopplas till en ny `ILToolbarKeyAdapter`. Men `ILInputField` skapades för att automatisera skapandet av `ILToolbarKeyAdapter` och slippa en hel del onödig kod.
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. ClickMode som styr vad som ska hända när användaren klickar på ILCanvas.
2. **Grafiskt gränssnitt:** Fönstret består av knappar och alla vektorer kan ändras på grafiskt.
3. **Egen GUI-komponent:** ILCanvas är en komponent som ritat ut alla vektorer.
4. **Designmönstret State:** När användaren klickar på ILCanvas händer olika saker beroende på vilket ClickMode som är aktivt. Syns tydligt i klassen ILCanvasMouseAdapter.
5. **Designmönstret Strategy:** När en vektor ska ritas ut körs metoden draw() som är olika beroende på typen av ILVector som ska ritas ut.
6. **Egen exception-klass:** IndexOutOfBoundsException.
7. **Loggning:** Finns i klasserna ILCanvas, ILSelection och ILVectorList.
8. **Tangentbordsstyrning:** Klassen ILButton ger stöd för tangentbordsstyrning.

## 7. Användarmanual

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

### 7.1 Definitioner

- Verktysfältet: Den översta panelen i fönstret som innehåller knappar och textfält.
- Tools-menyn: De fyra knapparna längst till vänster i verktysfältet under rubriken "Tools".
- Properties-menyn: De fem textfälten i verktysfältet under rubriken "Properties".
- Vector-menyn: De tre knapparna längst till höger i verktysfältet under rubriken "Vector".
- Kanvas: Det stora området under verktysfältet och dess gråa skiljelinje.
- Markeringsruta: Den blå rutan som ritas ut på en markerad form.
- Omformningslåda: 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 under Tools-menyn 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 Tools-menyn 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 Properties-menyn.

### 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) värden i Properties-Menyn. Observera att detta inte fungerar hos en text-form. En text-form kan endast ändra text-storlek vilket görs i Propertius-Menyn i Size-fältet.

### 7.6 Ta bort, duplicera och ändra färg

Du kan enkelt ta bort en form genom att först markera den och klicka på "Delete" under Vector-menyn. 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 Snabbknappar

<b>Knapp</b>	<b>Snabbknapp</b>
Select	V
Rectangle	R
Ellipse	E
Text	T
Color	C
Duplicate	D
Delete	Backspace



## 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.