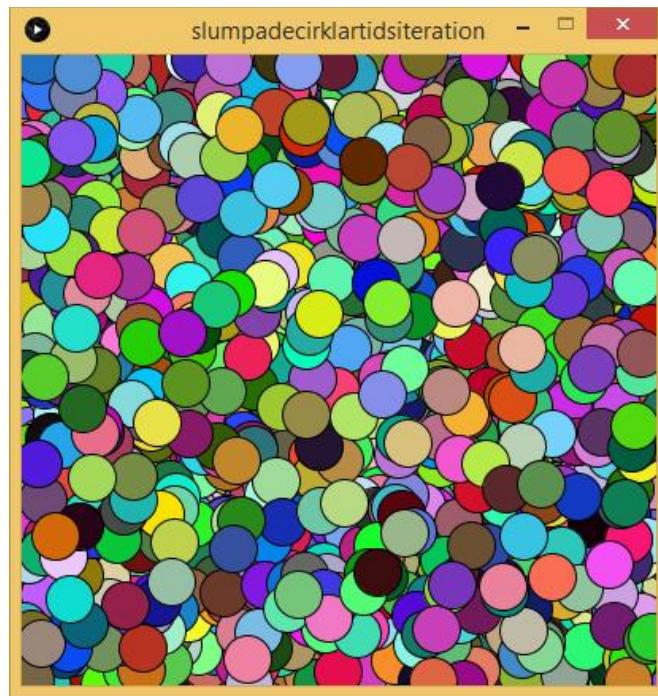


Lär dig programmera med Processing och Java



Jonathan Dahlberg

Copyright Jonathan Dahlberg 2020

Innehållsförteckning

1 Installera processing	11
1.1 Aktivera code completion.....	14
1.2 Installera Extended Code Completion.....	14
1.3 Processing i webbläsaren (<i>processing.js</i>)	14
2 Introduktion	15
2.1 Koordinatsystem.....	16
2.2 Grundläggande funktioner i processing	18
2.4 Färger.....	19
2.5 Color Selector.....	19
3 Variabler	22
3.1 Variabeltyper i processing	22
3.3 Skapa en variabel	23
3.4 Rita ut en bild från en fil.....	24
4 Inmatning från tangentbordet och omvandling mellan text och tal.	25
4.1 <i>Extended Code Completion</i> förenklar inmatning	27
4.2 Inmatning av tal och omvandling mellan text och tal	27
4.3 Skriva ut text.....	28
4.4 Flera sätt att skriva ut text.....	29
4.5 Tabell: Inmatning av textsträngar, omvandling av text till tal, samt utskrift av text.	31
4.6 Tabell: Aritmetiska operatorer i Processing (urval).....	31
5 Loopar.....	35
5.1 For-loopar - Iterera (Upprepa) ett visst antal gånger	35
5.2 Exempel for-loopar	36
5.3 While-loopar	42
5.4 Slumptal i Processing	43
5.5 While-loopar och slumptal	44

7 Mer om variabler	51
7.1 Ett exempel med int, for och heltalsdivision	51
7.2 Sammanfattning av skillnaden mellan heltal (int) och decimaltal (float).....	53
7.2.1 När ska vi använda heltal (int)?	53
7.2.2 När ska vi använda decimaltal (<i>float</i>).....	53
7.2.3 Vad händer när vi omvandlar decimaltal till heltal?	54
7.2.4 Vad händer när man omvandlar från heltal till decimaltal.	54
8 Fler uppgifter på variabler och iteration	55
9 Arrayer	58
9.1 Vi skapar en mask med hjälp av en array med fördefinierade värden	58
9.2 Vi skapar en tom array och låter användaren skriva in värdena.....	60
11 Tidsstyrda iteration.....	62
11.2 Tidstydrt iteration kombinerat med arrayer:.....	63
11.3 Tidsstyrda iteration kombinerat med slumptal.....	64
13 Selektion (if-satser)	66
13.1.2 Konvertera for-loop till animering.....	67
13.2 Skapa en boll som rör sig	70
13.3 Skapa en studsande boll	70
13.3.1 Övning Studsande boll	71
13.3.2 Övning Labyrint.....	72
13.4 Tabell: Jämförelseoperatorer	73
13.5 if-else-satser och if-satser med strängar	73
13.6 else.....	74
13.8 En kedja med flera if-satser	76
13.9 Logiska operatorer.....	77
13.10 Exempel logiska operatorer. Vem får åka Balder?	78
13.11 Inbyggda variabler som kan användas med if-satser	79
13.12 Exempel: ritprogram	79
13.13 Studsande boll med gravitation.....	82
13.14 * Hur gör vi för att bollen inte ska studsa högre och högre?.....	85

13.15 * Hur hanterar man flera knapptryckningar samtidigt?	86
13.16 Enkelt racerspel med Switch Case	89
15 Funktioner	93
15.1 Funktioner med returvärde.....	94
Teori om funktioner.....	96
15.2 Exempel på boolean funktion. Vem får åka Balder?	98
16 Tillståndsmaskiner	105
17 Aktivitetsdiagram och Pseudokod	109
17.1 Selektion.....	110
17.2 Iteration	112
17.3 Aktivitetsdiagram för ritprogrammet	113
17.4 Aktivitetsdiagram för tärningsspel	114
17.5 Pseudokod för tärningsprogram.....	115
17.6 Övning Aktivitetsdiagram och Pseudokod.....	117
18 Projektidéer.....	118
18.1 Sten, sax påse.....	118
18.2 Plattformsspel.....	118
18.3 Flappy bird.....	119
18.4 Enkelt agar.io.....	119
18.5 Snake	120
18.6 21 Spel/black jack.....	120
18.7 Hänga gubbe	120
19 Blandade repetitionsuppgifter.....	122
21 Klasser och Objekt.....	130
21.1 Skapa en mask som rör sig med klasser, objekt och ArrayList	137
22 Rekursiva funktioner	140
22.1 Kvadrater i kvadrat	141
23 Animerad rekursion	142
24 Att gå till Java från Processing - Skillnader mellan Java och Processing.....	147

24.1 Mainklass och main-metod	147
24.2 Skriva ut någonting	147
24.3 Datatyper	147
24.3.1 double istället för float	147
24.3.2 Omvandlingar	148
24.4 Funktioner och Metoder	148
25 Dina första Javaprogram	149
25.1 Att skapa ett enkelt program som enbart skriver ut några ord	149
25.2 Program med enkel inmatning med JOptionPane	151
25.3 Enkel inmatning i konsolen	153
25.3.1 Sammanfattning om Scanner:	154
25.4 Trevliga förkortningar i Netbeans	155
26 Skapa program med grafiska användargränssnitt i Java och netbeans	155
26.1 Sten Sax Påse-spel	159
26.2 Miniräknarexempel	169
26.3 Kvadrat och Kubikexempel	175
26.4 Rita i Java på Processingvis	181
26.5 Förteckning över metoderna i JDCanvas	190
26.6 Fler saker att tänka på när man använder JDCanvas	194
26.6.1 Färgobjekt och get()-metoden	194
26.6.2 Bilder i JDCanvas	195
26.6.3 Ändra storleken på en bild	195
26.6.4 Använda Javas standardritmetoder	196
27 Klasser i Java - Steg för steg	197
27.1 Exempel Klass som representerar en Boll	197
27.1.1 Steg 1 Skapa klassfilen	197
27.1.2 Steg 2 Skapa dina variabler	197
27.1.3 Steg 3 Infoga konstruktör	198
27.1.4 Steg 4 Infoga Setters and Getters	199
27.1.5 Steg 5 Skapa övriga metoder	200

27.1.6 Metod för att flytta bollen.....	200
27.1.7 Metod för att rita ut bollen.....	200
28.1 Skapa en JDCanvasklass som använder din Bollklass	202
28.2.1 Lägga till JDCanvas libbet.....	204
28.2.2 En boll som rör på sig.....	205
29 Teori om objektorientering.....	210
29.1 Tillgänglighet	212
30 ArrayList igen	213
31 Klasser i Java steg för steg - MP3.....	214
31.1 Steg 1 Skapa klassfilen	214
31.2 Steg 2 Skapa dina variabler.....	214
31.3 Steg 3 Infoga konstruktör.....	215
31.4 Steg 4 Infoga Setters and Getters.....	215
31.5 Steg 5 Lägg till övriga metoder	217
31.5.1 <code>toString</code>	217
31.5.2 <code>play</code>	218
31.6 Användargränssnitt till MP3Player	219
31.7 Steg 6 Skapa användargränssnittsfilen.....	219
31.8 Steg 7 Rita ut användargränssnittet.....	219
31.9 Steg 8 skapa en ArrayList.....	219
31.10 Steg 9 dubbelklicka på knappen.....	220
31.11 Steg 10 Skapa kod i händelsehanteraren	220
32 Sökning.....	221
33 Kundlistaexempel.....	223
34 Undantagshantering	231
34.1 Sammanfattning om undantag (exceptions).....	233
35 Blandade övningar	234

Förteckning över övningar

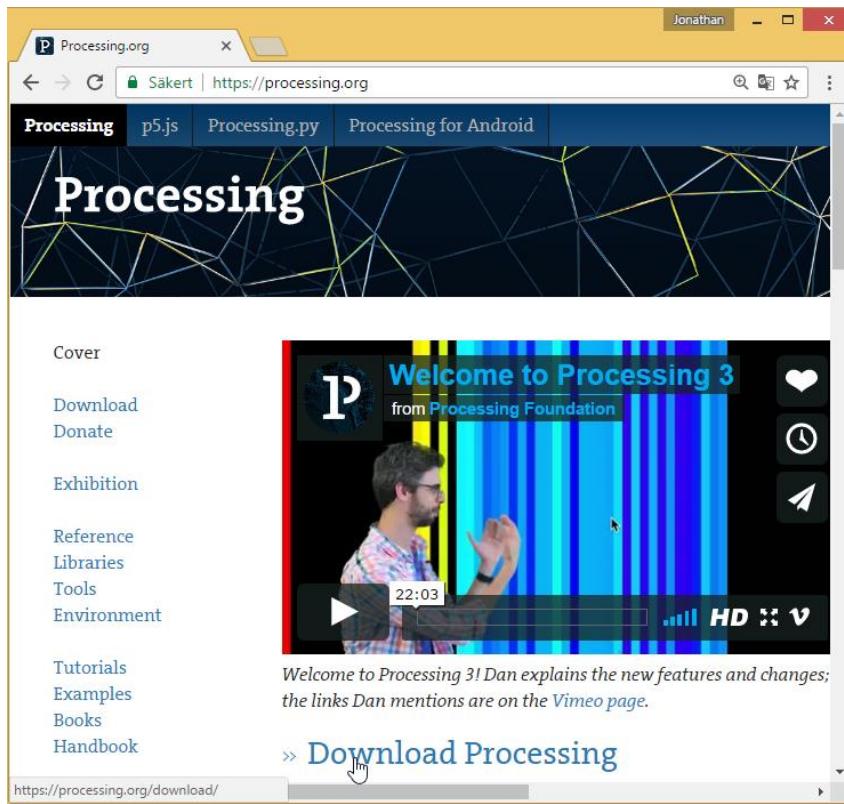
Övning 2-1 Rektangelövning 1	16
Övning 2-2 kvadrat i kvadrat	17
Övning 2-3 Rita orange rektangel	20
Övning 2-4 Rita en blomma.....	20
Övning 2-5 Rita sveriges flagga	21
Övning 2-6 Självporträtt	21
Övning 3-1 Övning kvadrat i kvadrat med variabler.....	22
Övning 3-2 Kollage	25
Övning 4-1 namnskylt	32
Övning 4-2 Kvadrat i kvadrat med inmatning.....	32
Övning 4-3 Fahrenheit till celsius	33
Övning 4-4 Valutaomvandlare	33
Övning 4-5 Rita cirklar med input	33
Övning 5-1 Övning större och större rektanglar.....	37
Övning 5-2 Övning gradient	37
Övning 5-3 Exempel linjer	38
Övning 5-4 Linjemönster.....	39
Övning 5-5 Rita kataloniens flagga.....	39
Övning 5-6 Rita staplar	40
Övning 5-7 Rutigt papper	41
Övning 5-8 Övning kvadrat i kvadrat med inmatning och storlekskontroll	45
Övning 5-9 Slumpa fram cirklar så länge mitten är fri	46
Övning 5-10 Numerisk lösning av ekvation	48
Övning 5-11 Övning på while och for-loop och slumptal.....	50
Övning 9-1 Övning stapeldiagram.....	61
Övning 9-2 Övning olika nyanser av blått.....	61
Övning 9-3 Övning bilder på rad	61
Övning 11-1 Slumpade animerade cirklar	65
Övning 13-1 Övning slumpa ett visst antal animerade cirklar	68
Övning 13-2 Antal frimärken 1	68
Övning 13-3 Övning stoppa bollen	69

Övning 13-4 Övning byt håll på bollen.....	69
Övning 13-5 Boll fram och tillbaka.....	69
Övning 13-6 Övning Studsande boll fortsättning	78
Övning 13-7 Övning ritprogram	81
Övning 13-8 Övning antal frimärken 2	81
Övning 13-9 Gravitationsspel	85
Övning 13-10 Övning Racerspel	92
Övning 15-1 Funktion som ritar ut en blomma.....	100
Övning 15-2 Övning funktion som ritar ut många cirklar	101
Övning 15-3 Övning slumpad cirkelrad-funktion	101
Övning 15-4 Övning upphöjt till-funktion.....	102
Övning 15-5 Funktion kontrollerar om muspekaren är ovanför en rektangel.....	103
Övning 15-6 Övning primtalsfunktion	104
Övning 15-7 Övning flera animerade slumpcirklar.....	104
Övning 19-1 Placera ut dina fina blommor med hjälp av arrayer	122
Övning 19-2 Olympiska ringarna.....	123
Övning 19-3 Animera din bokstav	124
Övning 19-4 Funktion som summerar fyra tal.....	125
Övning 19-5 Funktion omkrets	125
Övning 19-6 Medelvärde	126
Övning 19-7 Omvänd ordning	126
Övning 19-8 Slumpat linjemönster	126
Övning 19-9 Reaktionstest.....	128
Övning 19-10 Geometry dash	129
Övning 23-1 Övning gör din egen stack	146
Övning 26-1 Övning Fahrenheit till Celsius i java	159
Övning 26-2 Övning summera två tal.	159
Övning 26-3 Övning Sten Sax Påse.....	168
Övning 26-4 Miniräknarövning fortsättning	174
Övning 26-5 Omvandling till brittiska mått enheter	180
Övning 26-6 Övning Ritprogram i Java	189
Övning 26-7 Testa att själv konvertera en uppgift från Processing till Java och JDCanvas	196
Övning 35-1 Räntekalkylator	234

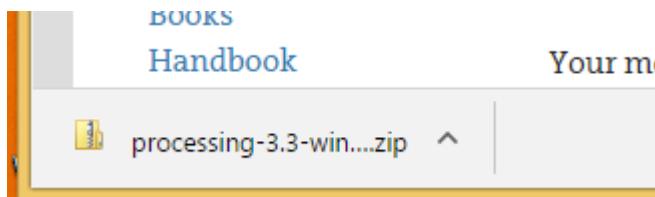
Övning 35-2 Färgvisare.....	235
Övning 35-3 Omvandlare från kilometer i timmen till meter per sekund.....	236

1 Installera processing

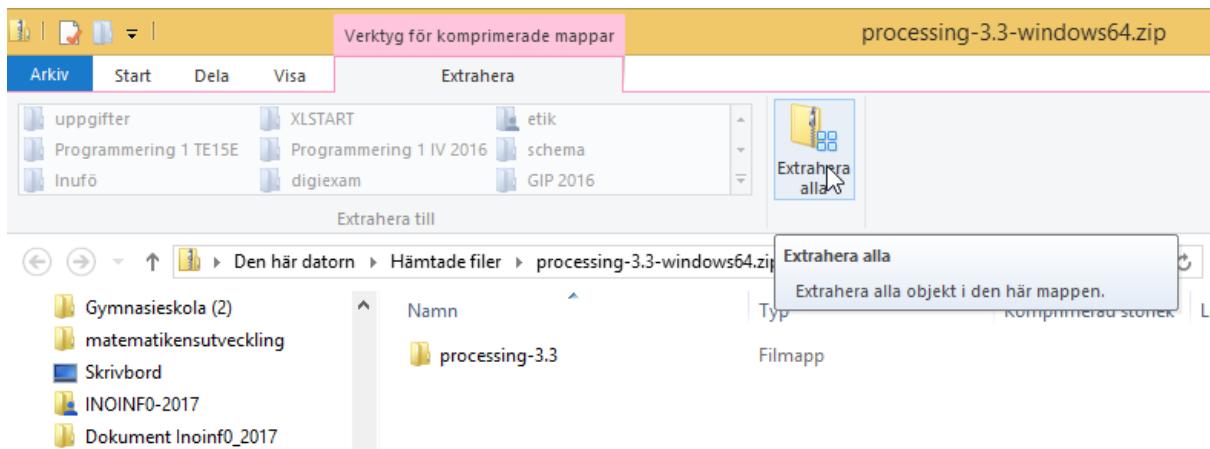
Gå till processing och processing.org och välj *Download processing*



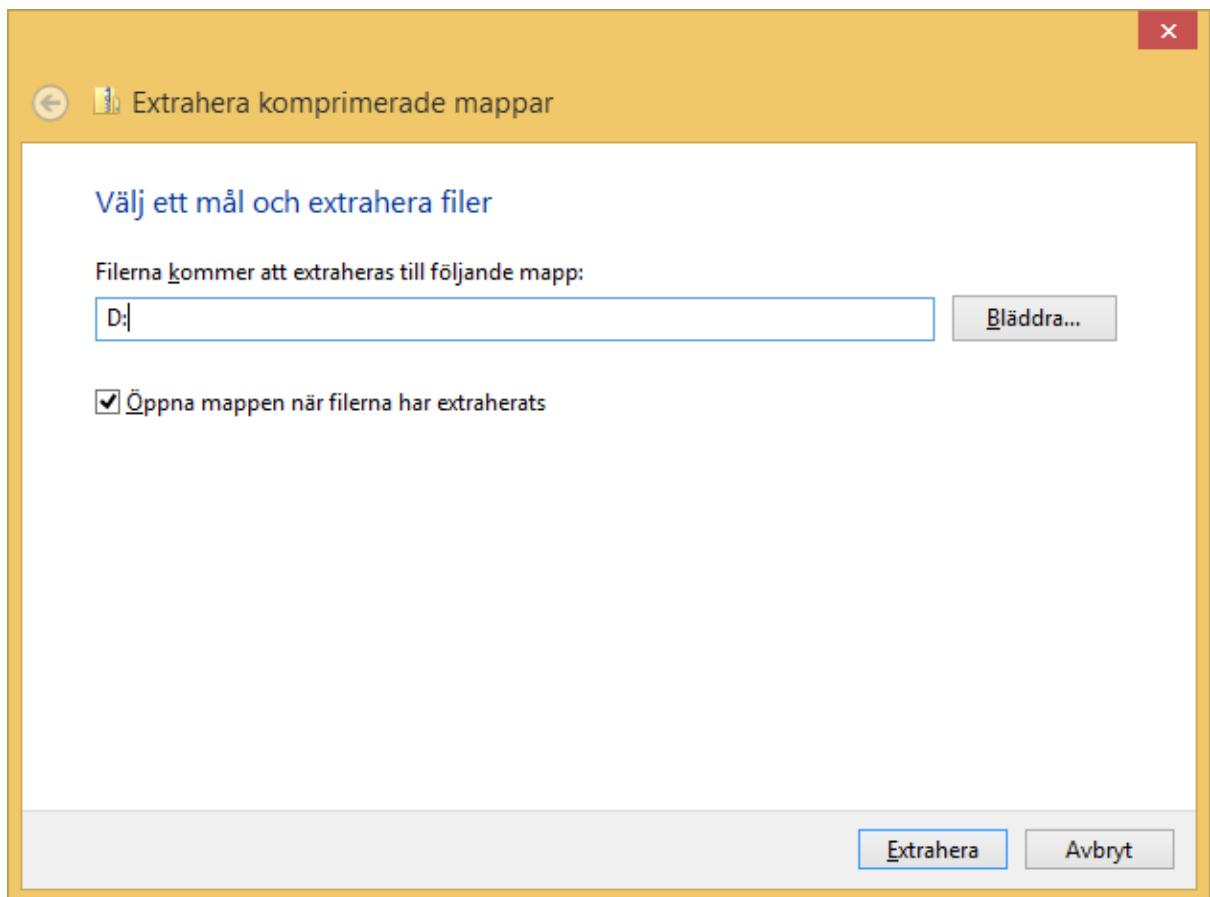
Välj ditt operativsystem (vilket oftast är Windows 64-bit). En zip-fil laddas ner.



Öppna filen. Om du använder chrome gör du det enklast genom att dubbelklicka på filen som kommer att finnas längst ner i vänstra hörnet. Verktyget för komprimerade filer öppnas då.



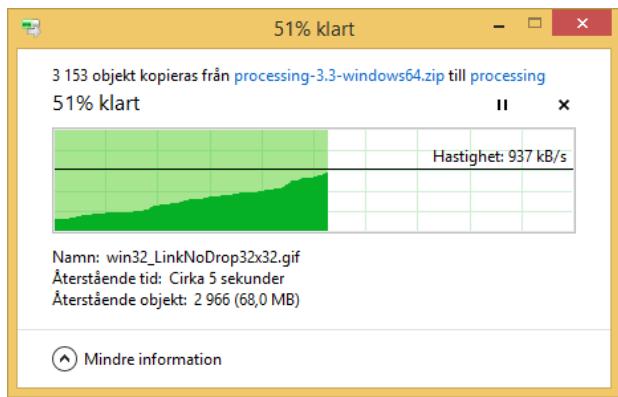
Tryck på knappen *Extractera alla*.



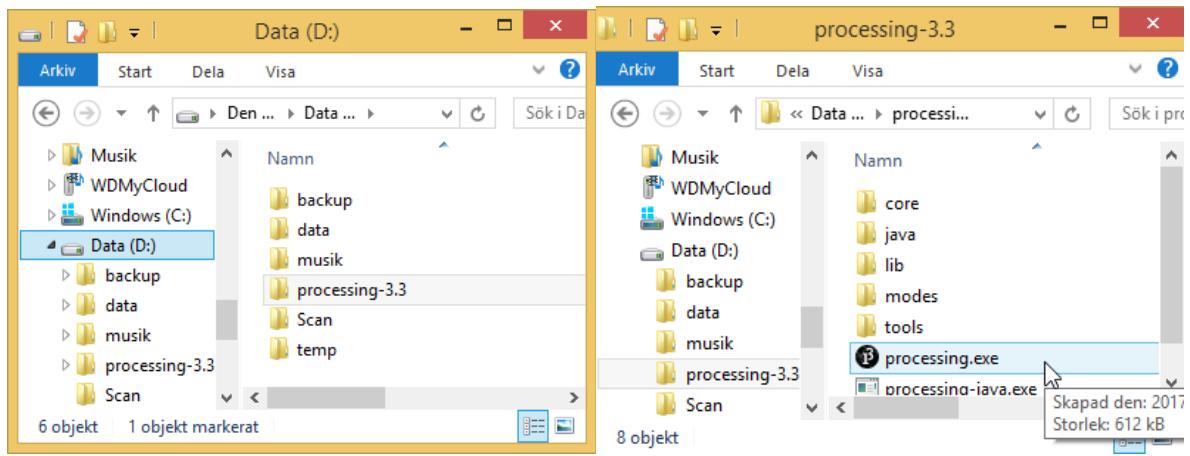
Du får nu möjlighet att välja var processing ska packas upp/installeras. Mitt förslag är att lägga det på en plats som är lätt att hitta. Till exempel d:\. Du behöver inte skapa någon mapp eftersom en mapp kommer att skapas när du packar upp filen.

Du väljer enklast d:, genom att helt enkelt skriva D: i textfältet.

Tryck sedan *Extractera* för att packa upp/installera Processing.



D: öppnas sedan automatiskt

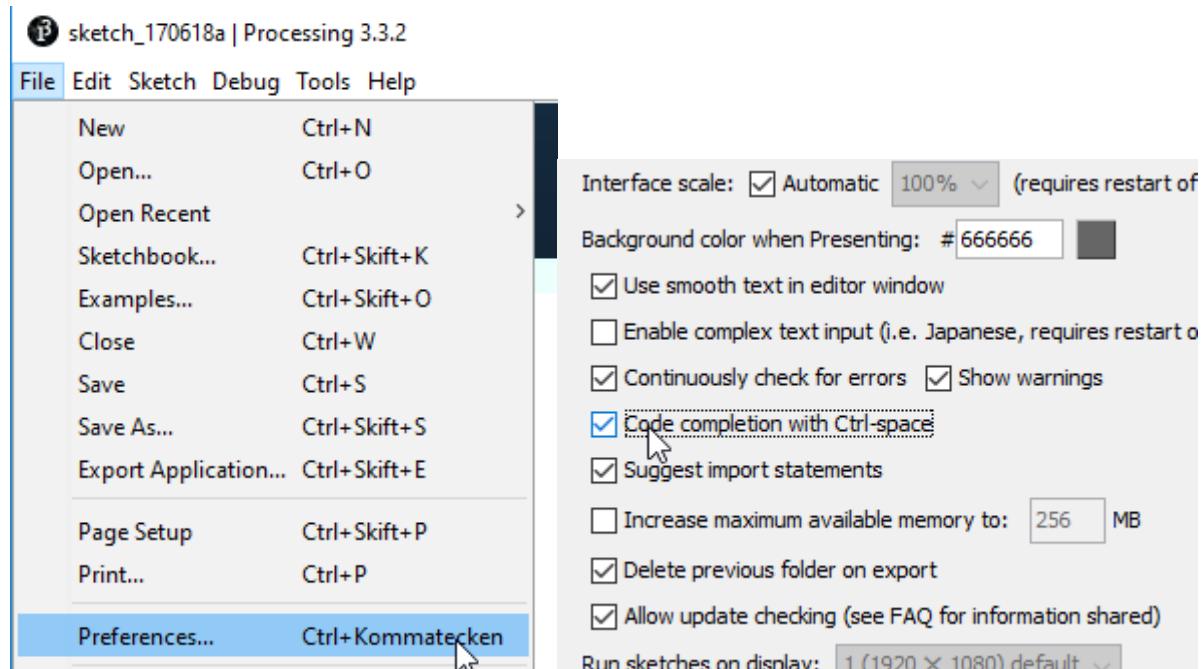


Dubbelklicka på *processing-3.3* och dubbelklicka sedan på *processing.exe* för att starta processing.

1.1 Aktivera code completion

För att enklare hitta alla kommandon i processing så är det bra att aktivera *code completion*. Med detta installerat kan man få förslag på kommandon genom att trycka *ctrl+mellanslag*.

Gå till *File/Preferences* och kryssa sedan i *Code Completion with Ctrl-space*



1.2 Installera Extended Code Completion

För att få ytterligare kod som automatiskt fylls på av Processing så kan man installera tillägget *Extended Code Completion*. Den kan hjälpa dig med if, for och while-satser med mera.

För att installera tillägget väljer du *Tools/Add tool*, och väljer sedan *Extended Code Completion* och trycker sedan på *Install*.

1.3 Processing i webbläsaren (*processing.js*)

Om du inte vill eller kan installera Processing på en Windows, Linux eller Mac-dator, så finns det möjlighet att köra en begränsad variant av Processing direkt i webbläsaren. Den heter *processing.js*. Det finns en smidig tjänst för att testa *processing.js* på nätet:

<http://sketchpad.cc/>

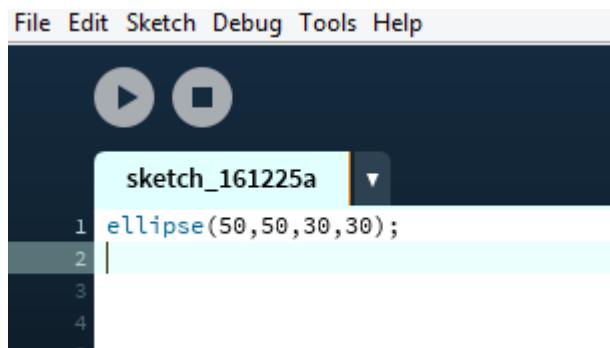
I *sketchpad.cc* kan man både programmera direkt i webbläsaren, eller kopiera sin kod dit så att man kan visa upp sina fina program för sina kompisar, familj och vänner.

Allt i *processing.js* fungerar inte som i Processing. Lek gärna med *sketchpad.cc*. **Men lägg ingen tid på att felsöka i den miljön. Det är kanske inte du som gjort fel. Det kan vara något som inte stöds i *processing.js* men som fungerar i Processing. Testa istället först i riktiga Processing om du behöver felsöka!**

2 Introduktion

Vår första utmaning i Processing är att rita en cirkel med diametern 30, på position 50,50.

Detta gör vi genom att använda funktionen `ellipse`, med parametrarna 50,50, 30,30 för positionen 50,50, och 30,30 för att ange att både bredden och höjden ska vara 30. Lägg märke till att den position vi anger är mitten på cirkeln.



The screenshot shows the Processing IDE interface. The menu bar includes File, Edit, Sketch, Debug, Tools, and Help. Below the menu is a toolbar with a play button and a square button. The sketch title is "sketch_161225a". The code area contains the following lines:

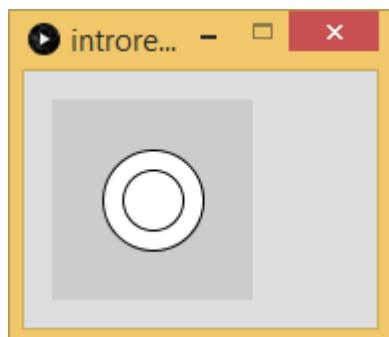
```
1 ellipse(50,50,30,30);  
2  
3  
4
```



Tryck sedan på play knappen:



Då borde du se detta:



Svar:

```
ellipse(50, 50, 50, 50);  
ellipse(50, 50, 30, 30);
```

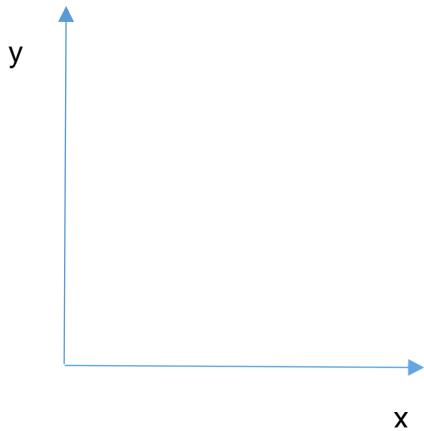
Övning 2-1 Rektangelövning 1

Rita en rektangel som har sidan bredden 60, höjden 40, och placera den på position 20,20.
Lägg märke till att positionen som anges är övre vänstra hörnet.

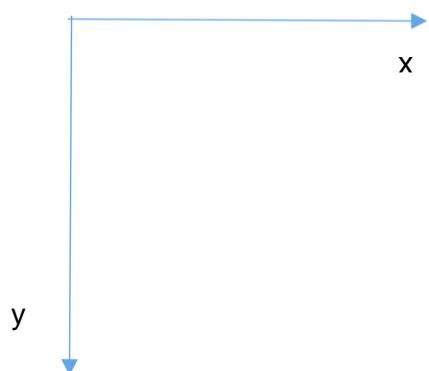
2.1 Koordinatsystem

Som ni kanske märkt i exemplen ovan så är origo i koordinatsystemet i Processing uppe i övre vänstra hörnet.

Från matten är ni vana med sådana här koordinatsystem:



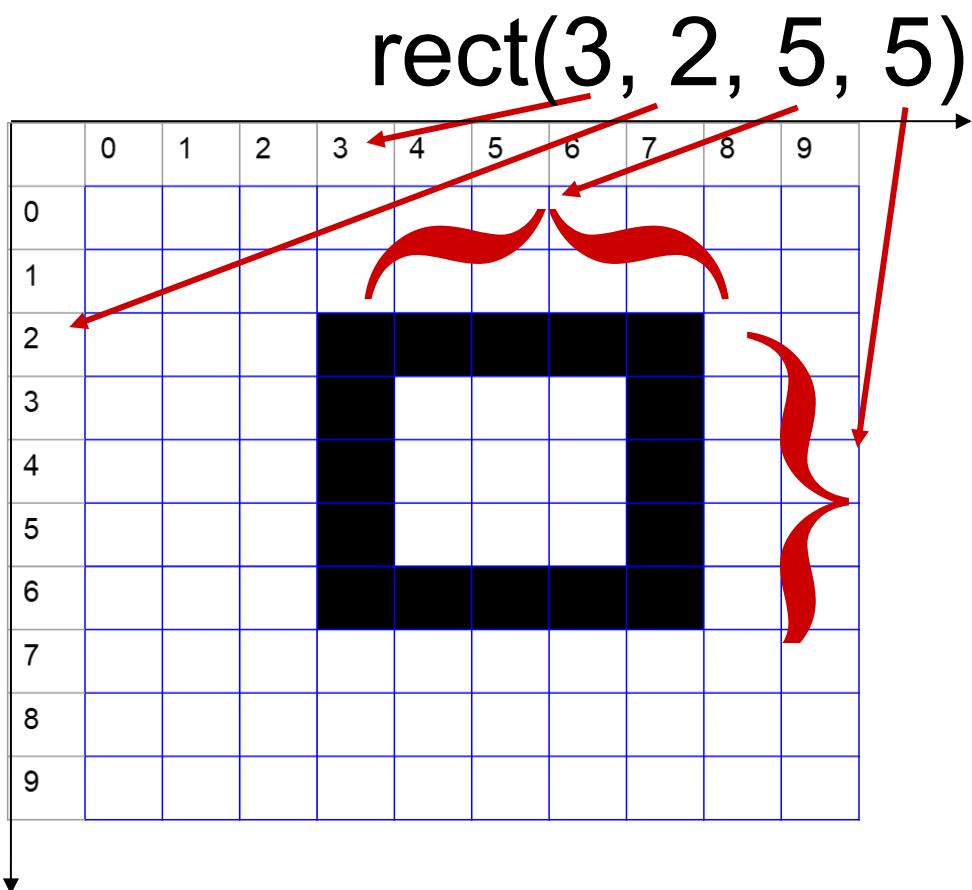
Men i Processing är koordinatsystemet upp och ner:



När vi kör kommandot:

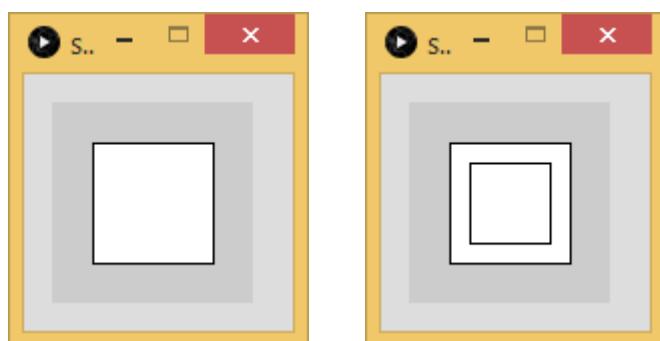
```
rect (3, 2, 5, 5);
```

så innebär det:



Övning 2-2 kvadrat i kvadrat

- Rita en kvadrat med sidan 60, på position 20,20.
- Rita en kvadrat inuti den andra kvadraten med avståndet 10 mellan de båda kvadraterna sida.



2.2 Grundläggande funktioner i processing

Beskrivning	Funktion	Exempel
Storlek på rityta	size(bredd, höjd)	size(800,600); // Sätter ritytan till bredden 800 och höjden 600
Rita ellips	ellipse(x,y,bredd, höjd)	ellipse(50,50,70,70); //ritar cirkeln med diametern 70 på position 50,50
Rita rektangel	rect(x,y,bredd, höjd)	rect(50,50,70,70); //ritar kvadrat med sidan 70 på position 50,50
Rita linje	line(x1, y1, x2, y2)	line (20,20,100,100); //Ritar en linje mellan punkten 20,20 till punkten 100,100
Rita triangel	triangle(x1, y1, x2, y2, x3, y3)	triangle(50,10,90,90,10,90);
Ladda in en bild från en Fil.	PImage img = loadImage(<filnamn>);	PImage img = loadImage("Blomma.jpg"); image(img,20,20);
Rita ut en bild på skärmen	image(img, x, y);	PImage img = loadImage("Blomma.jpg"); image(img,20,20);
Bestäm färgen på det grafiska element som ritas ut	fill(röd,grön,blå);	fill(0,0,128); // nästa grafiska objekt som ritas ut blir mörkblått
Måla bakgrundens i en bestämd färg	background(röd,grön,blå);	background(0,0,128); // Hela ritytan blir mörkblå
Bestäm färgen på kantlinjen	stroke(röd, grön, blå)	stroke(0,0,128); // Kantlinjen blir mörkblå
Ta bort kantlinjen	noStroke()	noStroke();
Skriva ut text på en rad.	text(meddelande, x,y)	text("Detta är ett meddelande", 10,10);

Fler funktioner finns på <https://processing.org/reference/>.

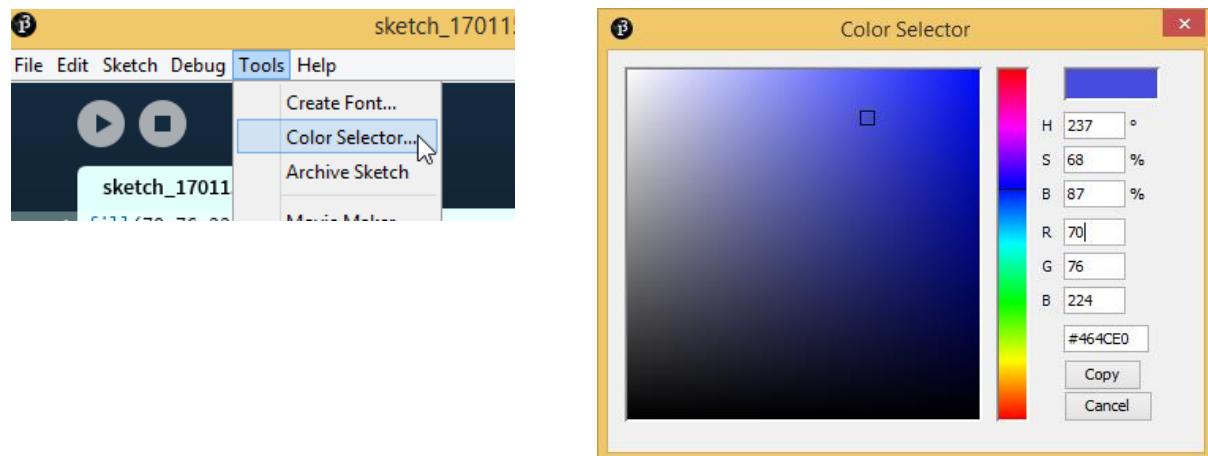
2.4 Färger

För att bestämma vilken färg ett grafiskt objekt ska fyllas med så skriver man `fill(röd, grön, blå)`, se exempel nedan:



I exemplet ser ni att den röda parametern är ifylld med 128, och de andra är 0, vilket ger en mörkröd färg. Det är inte så lätt att lista ut rätt färgkombination för hand, därför innehåller Processingmiljön en *Color Selector*.

2.5 Color Selector



Då är det bara att välja en färg med färgverktyget och sedan skriva av R, G och B värdena.

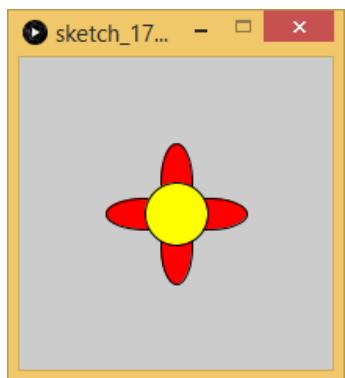


Övning 2-3 Rita orange rektangel

Gör den mittersta rektangeln orange i övningen ovan:

Övning 2-4 Rita en blomma

Rita en bild som liknar blomman nedan.



Övning 2-5 Rita sveriges flagga

Du ska rita Sveriges flagga med Processing.

- Ritytan ska vara 600x400.
- Bredden på ränderna 80 pixlar.
- Den horisontella linjen ska vara mitt på flaggan.
- Vänsterkanten på den vertikala linjen ska vara 190 pixlar från vänsterkanten på fönstret

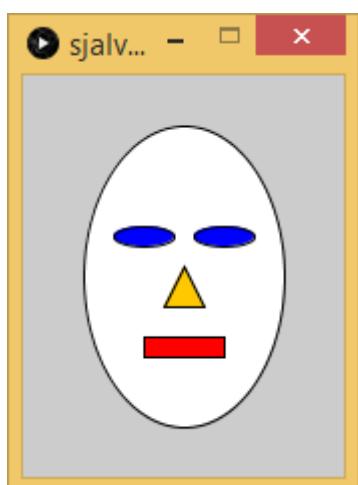
Svenska flaggan har inte vilka blå eller gula färger som helst. Nyanserna är reglerade i lag, och omräknat till RGB-värden så blir det:

- Den blå färgen ska ha rgb-värdena R: 0, G: 128, B: 230
- Den gula färgen ska ha rgb-värdena R: 255, G: 204, B: 0



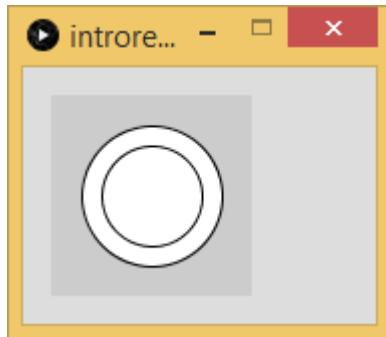
Övning 2-6 Självpörträtt

Rita ett självpörträtt av dig själv med hjälp av ellipser, rektanglar och trianglar.



3 Variabler

Nu ska du ändra i ellipsexemplet ovan så att du kan ändra ellipsernas storlek, men behålla avståndet 20 mellan den yttre och den inre ellipsens ytterkant. Den inre cirkelns diameter ska sparas i en variabel.



Figuren visar när den inre cirkeln är 30. Vi introducerar nu variabeln d1, och d2. dr2 kommer alltid att vara d1+20, vilket betyder att det alltid kommer att vara 10 pixlar mellan cirklarna. Så det räcker att sätta värdet på d1 för att ändra storleken på cirkeln.

```
int d1=50;  
int d2=d1+20;  
ellipse(50,50,d2,d2);  
ellipse(50,50,d1,d1);
```

Övning 3-1 Övning kvadrat i kvadrat med variabler

Modifiera övningen [kvadrat i kvadrat](#) ovan så att du enkelt kan ändra sidstorlek på den yttre rektangeln samt avstånd med hjälp av variabler.

3.1 Variabeltyper i processing

Namn	Kan lagra	Exempel	typ av typ
int	Heltal	int tal=23;	primitiv
float	Decimaltal	float vinkel=2.34;	primitiv
char	ett tecken	char c='a';	primitiv
String	Text (dvs flera tecken)	String s="Hej";	Objekt
PImage	En pixelbaserad bild, ofta inlästa från bildfiler (png, jpg etc)	PImage img = loadImage("Blomma.jpg");	Objekt
color	En färg	color farg=color(128,0,0);	Special

3.3 Skapa en variabel

Man skapar en variabel genom att ange en typ följt av ett variabelnamn. Om man vill kan man också initiera variabeln, det vill säga sätta ett startvärde på variablen på samma rad som man skapar den.

```
int points;
```

Typ Variabelnamn

Skapa en heltalsvariabel

```
float points;
```

Typ Variabelnamn

Skapa en decimaltalsvariabel.

```
String name;
```

Typ Variabelnamn

Skapa en variabel som kan lagra text, en strängvariabel (stringvariable på engelska).

```
int points = 0;
```

Typ Variabelnamn Startvärde

Skapa en heltalsvariabel och sätta ett startvärde (initiera) på den.

```
float points = 0.0;
```

Typ Variabelnamn Startvärde

Skapa en decimaltalstalsvariabel och sätta ett startvärde (initiera) på den.

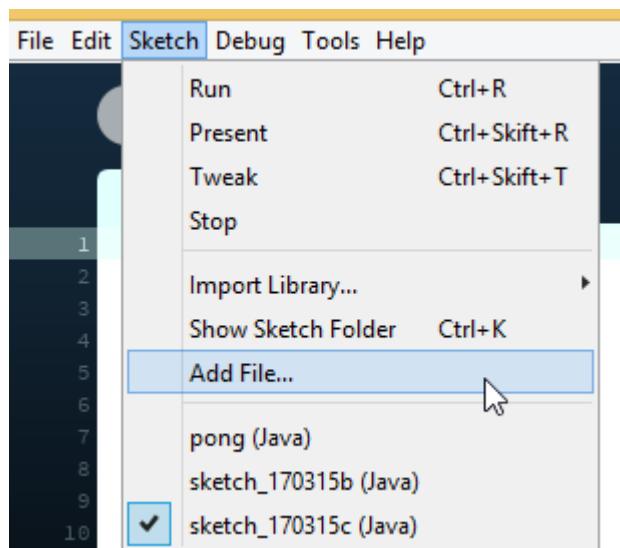
```
String name = "Jonathan";
```

Typ Variabelnamn Startvärde

Skapa en strängvariabel och sätta ett startvärde (initiera) på den.

3.4 Rita ut en bild från en fil.

Först ska man välja vilken fil som man ska rita ut och lägga till den till projektet/sketchen. Detta gör man enklast genom att välja *sketch->Add File...* och sedan välja bilden som du ska rita ut. Fyra typer av bilder kan hanteras av Processing. Det är de bilder som slutar på .gif, .jpg, .tga, eller .png. Jag kommer att välja en bild som heter "Blomma.jpg"



Ibland vill man rita ut en bildfil i sitt program, för detta krävs två steg. Först läser man in bildfilen i en variabel:

```
PImage img = loadImage(<filnamn>);
```

Exempelvis:

```
PImage img = loadImage("Blomma.jpg");
```

Nu när bilden finns i variabeln så är det bara att rita ut den ungefär på samma sätt som du ritar ut en rektangel. De första paramaterna till *image*-funktionen är variabeln som innehåller bilden, och de andra två är x och y-koordinaten som bilderna ska ritas ut på.

```
image(img, x, y);
```

I vårt exempel blir det:

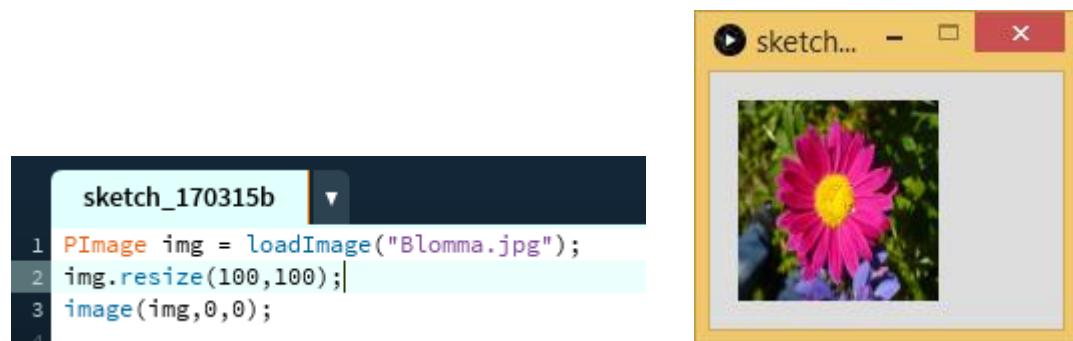
```
image(img, 20, 20);
```

Hela programmet blir:



Lägg märke till att jag lagt till `size(640,490)`. Storleken på fönstret behövs med andra ord ofta sättas till lämpligt värde annars så kan man enbart visa mycket små bilder.

Om man istället vill att bilden ska rymmas på 100*100 pixlar kan man använda funktionen/metoden `resize` för att ändra storleken på bilden.



Övning 3-2 Kollage

Skapa ett snyggt kollage av minst tre bilder. Skriv också en beskrivande text någonstans i bilden.

4 Inmatning från tangentbordet och omvandling mellan text och tal.

Om vi vill kunna mata in text som är mer än ett tecken långt så behöver vi använda oss av `JOptionPane`, som är en javaklass som kan användas för ändamålet.

Vi kanske till exempel vill lägga till en beskrivande text i programmet som visade en bild på en blomma ovan. Vi öppnar blomexemplet och lägger till `import javax.swing.*`

```
laddabild1
```

```
1 import javax.swing.*;
2
3 size(640,490);
4 PImage img = loadImage("Blomma.jpg");
```

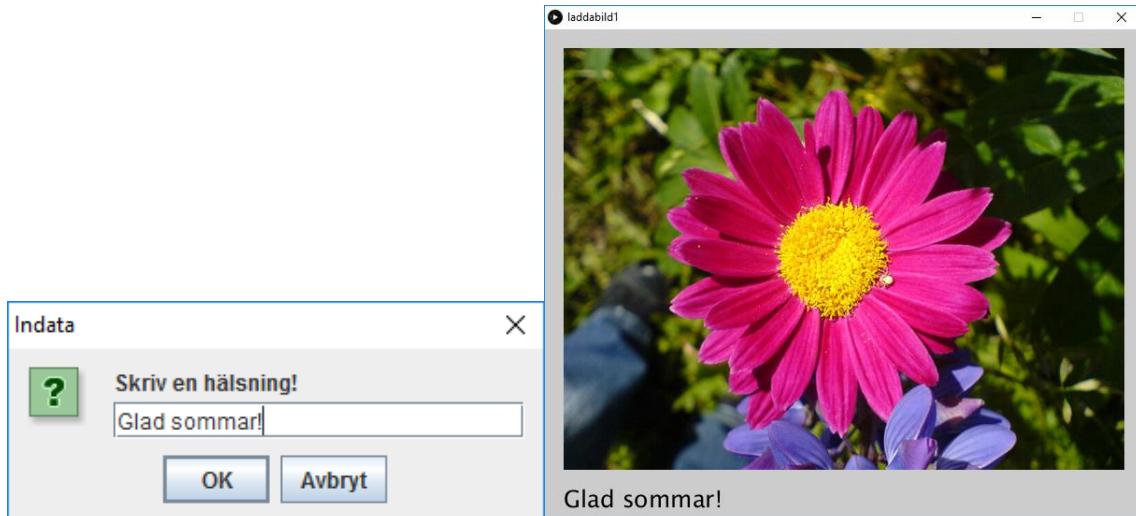
För att be användaren mata in något så anropar vi JOptionPane.showInputDialog("Fråga");
Ofta vill vi stoppa in svaret i en String så då blir det:

```
String str = JOptionPane.showInputDialog("Fråga");
```

Sedan använder vi variabeln str för att skriva ut hälsningen under bilden.

```
laddabildmedinput
```

```
1 import javax.swing.*;
2
3 size(640,530);
4 PImage img = loadImage("Blomma.jpg");
5 image(img,20,20);
6 fill(0);
7 textSize(25);
8 String str = JOptionPane.showInputDialog("Skriv en hälsning!");
9 text(str, 20,510);
10
```



4.1 Extended Code Completion förenklar inmatning

Om du har installerat och startat *Extended Code Completion* så räcker det med att skriva `input` följt av `ctrl + mellanslag` så ersätts `input` med

```
JOptionPane.showInputDialog("")
```

och dessutom importeras `JOptionPane` automatiskt, dvs raden

```
import javax.swing.JOptionPane;
```

infogas längst upp.

4.2 Inmatning av tal och omvandling mellan text och tal

Tidigare så gjorde vi ett litet program som skrev ut två cirklar:

```
int d1=50;
int d2=d1+20;
ellipse(50, 50, d2, d2);
ellipse(50, 50, d1, d1);
```

Vi vill nu låta användaren få sätta värdet på `d1` varje gång som programmet körs, så vi ändrar lite i programmet ovan. Vi börjar med att importera `JOptionPane`.

```
sketch_170615c
import javax.swing.JOptionPane;
int r1=50;
```

När vi gjort detta så kan vi byta ut den fasta tilldelningen av `d1`, med en inmatning från användaren. Det som krånglar till det hela lite grann är att vi enbart kan få ut en Sträng från `showInputDialog`. Nu skapar vi därför en extra variabel som heter `d1str`.

```
int r1str = JOptionPane.showInputDialog("mata in diameter");
```

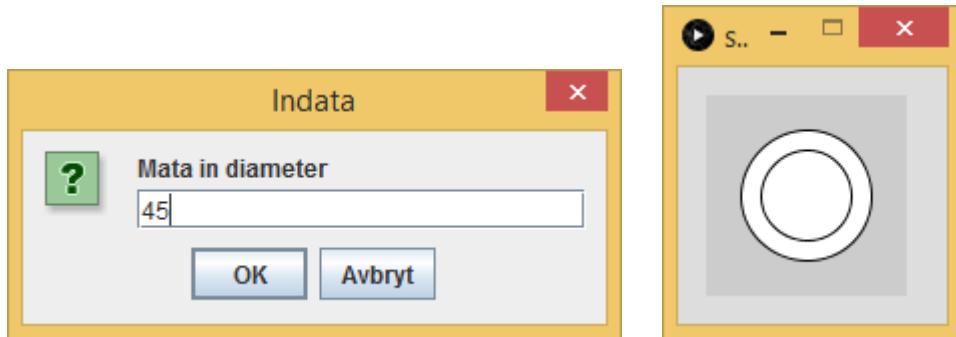
Den variabeln kan vi sedan omvandla till en `int` genom att använda funktionen `int(d1str)`.

Programmet blir då:

```

sketch_171103a | ▾
1 import javax.swing.JOptionPane;
2
3 String d1str = JOptionPane.showInputDialog("Mata in diameter");
4 int d1=int(d1str);
5 int d2=d1+20;
6 ellipse(50,50,d2,d2);
7 ellipse(50,50,d1,d1);
8

```



Nu när vi kör programmet dyker det upp två fönster på skärmen. En med ett inmatningsfält och ett som visar grafiken. När vi trycker på entertangenten i inmatningsfönstret så dyker cirkeln upp i grafikfönstret, och inmatningsfönstret försvinner.

4.3 Skriva ut text

På liknande sätt gör man också om man vill omvandla från en sträng till ett decimaltal. Vi tar ett exempel där vi ska omvandla mellan kilometer i timmen och meter per sekund. Inmatning och omvandling blir då:

```

String str = JOptionPane.showInputDialog("Ange hastighet i km/h");
float kmh = float(str);

```

Omvandlingen blir:

```

float ms = kmh/3.6;

```

För att skriva ut resultatet skriver vi:

```

text(kmh + " kilometer i timmen blir " + ms + " meter per sekund",
10,10,80,80);

```

```

kmh
1 import javax.swing.*;
2
3 String str = JOptionPane.showInputDialog("Ange hastighet");
4 float kmh = float(str);
5 float ms = kmh/3.6;
6 fill(0);
7 text(kmh + " kilometer i timmen blir " + ms + " meter per sekund", 10,10,80,80);

```

4.4 Flera sätt att skriva ut text

Förutom att skriva ut text direkt i Processings vanliga fönster så kan man skriva ut text på två andra sätt i Processing. Det enklaste är att använda `println()`. Då skrivs texten helt enkelt ut i den nedre delen av Processingutvecklingsmiljön.

```

kmhprintln
1 import javax.swing.*;
2
3 String str = JOptionPane.showInputDialog("Ange hastighet");
4 float kmh = float(str);
5 float ms = kmh/3.6;
6 fill(0);
7 println(kmh + " kilometer i timmen blir " + ms + " meter per sekund");

```

45.0 kilometer i timmen blir 12.5 meter per sekund 10 10 80 80

Det tredje sättet att skriva ut text är att använda `JOptionPane`. Det är alltså samma bibliotek som vi använder för att mata in text. `JOptionPane.showMessageDialog` tar två stycken parametrar. Den första är vilket fönster som popupfönstret ska vara kopplat till. Här anger man enklast `null`, vilket är ett värde som betyder *inget*. Den andra parametern är texten. Om man har installerat och startat *Extended Code Completion*, så räcker det att skriva `output` och sedan trycka på `ctrl+ mellanslag`, för att få fram uttrycket `JOptionPane.showMessageDialog`, samt den nödvändiga importen av `javax.swing.JOptionPane`;

The screenshot shows a Java code editor window with the title bar 'kmhjoptionpane'. The code is as follows:

```
1 import javax.swing.*;
2
3 String str = JOptionPane.showInputDialog("Ange hastighet");
4 float kmh = float(str);
5 float ms = kmh/3.6;
6 fill(0);
7 JOptionPane.showMessageDialog(null, kmh + " kilometer i timmen blir " + ms + " meter per sekund");
8
```

Below the code editor is a screenshot of a 'Meddelande' (Message) dialog box. The dialog has a yellow border and a white interior. It contains an information icon (a blue circle with a white 'i'), the text '45.0 kilometer i timmen blir 12.5 meter per sekund', and an 'OK' button at the bottom.

4.5 Tabell: Inmatning av textsträngar, omvandling av text till tal, samt utskrift av text.

Beskrivning	Kommando	Exempel
Mata in en sträng från tangentbordet	JOptionPane.showInputDialog();	String str = JOptionPane.showInputDialog("Fråga");
Omvandla från text till heltal	int()	r=int(str);
Omvandla från sträng till decimaltal	float()	float hastighet = float(str);
Skriva ut text på en rad i.	text(meddelande, x,y)	text("Detta är ett meddelande", 10,10);
Skriva ut text på flera rader.	text(meddelande, x,y, bredd, höjd)	text("Detta är ett meddelande", 10,10, 80,80);
Skriva ut en rad text i Konsolen.	println(meddelande)	println("Detta är ett meddelande");

4.6 Tabell: Aritmetiska operatorer i Processing (urval)

Vi har i exemplen ovan använt olika aritmetiska operatorer såsom + och -. Nedan finns en tabell av de vanligaste aritmetiska operatorerna i Processing.

Operator	Beskrivning	Exempel	prioritet
+	Plus (addition)	float summa = tal1 + tal2;	5
-	Minus (subtraktion)	float diff = tal1 - tal2;	5
*	Gånger (multiplikation)	float produkt = tal1 * tal2;	4
/	Delat (division)	float kvot = tal1 / tal2;	4
++	Öka med ett	int tal=0; tal++;	
--	Minska med ett	int tal=0; tal--;	

Mer om operatorer finns på <https://processing.org/reference/> under *Operators*.

Övning 4-1 Namnskylt

Be användaren mata in ett namn. Lagra namnet i variabeln namn och skriv sedan ut namnet på textrutan. Textstorleken ska vara 40. För att få texten på mitten behöver du använda:

```
textAlign(CENTER, CENTER);
```

Ritytan ska vara 400x150 stor.

Lägg till denna rad sist i programmet för att bilden ska sparas:

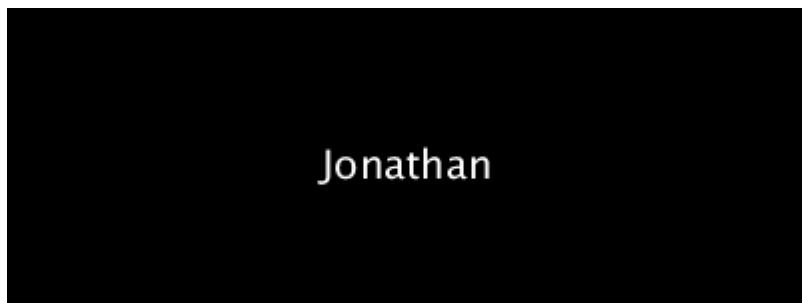
```
saveFrame("procass.png");
```

körexempel:

Om man skriver:

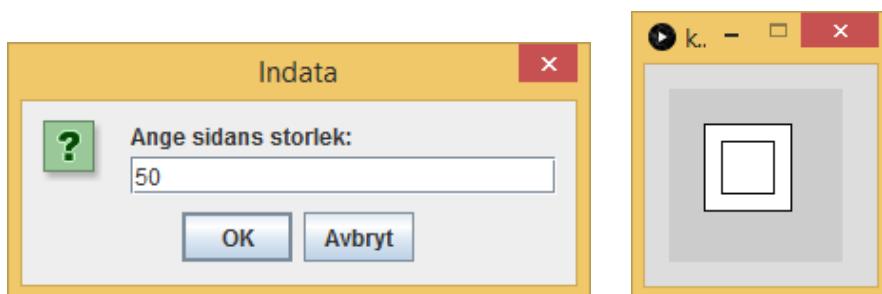


Så ska det bli så här:



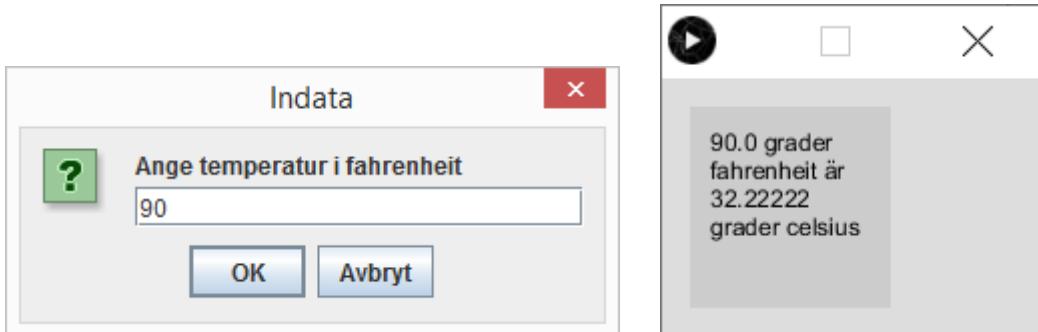
Övning 4-2 Kvadrat i kvadrat med inmatning

Ändra i övningen [kvadrat i kvadrat med variabler](#) ovan så att användaren får möjlighet att ange den yttre rektangelns värde.



Övning 4-3 Fahrenheit till celsius

Skriv ett program som omvandlar från fahrenheit till celsius. Programmet ska be användaren att mata in en temperatur i fahrenheit och sedan ska temperaturen skrivas ut i celsius. Formeln är ${}^{\circ}\text{C} = ({}^{\circ}\text{F} - 32) \cdot 5/9$.



Övning 4-4 Valutaomvandlare

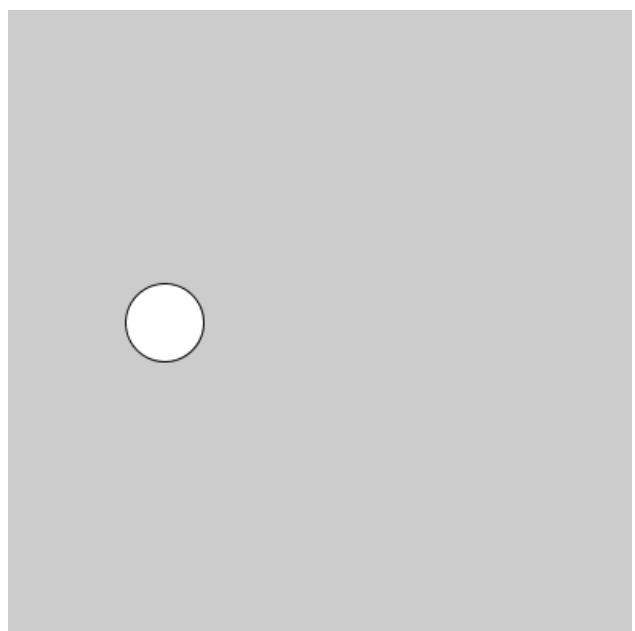
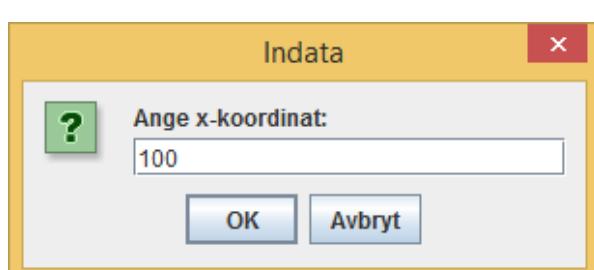
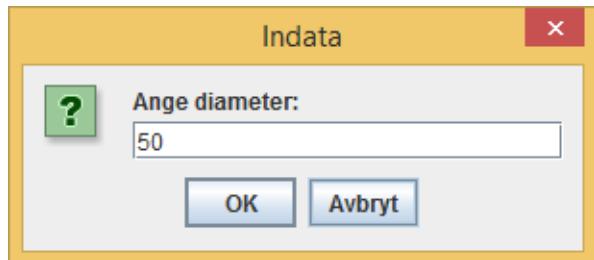
Skriv ett program som räknar ut hur många amerikanska dollar man får från ett antal kronor, och sedan ska programmet be användaren mata in växlingskurserna. Programmet ska be användaren att mata in ett belopp i kronor. Programmet ska räkna ut hur mycket det motsvarar i USD.



Övning 4-5 Rita cirklar med input

Skriv ett program som tar invärdarna diameter, samt x och y-koordinat, och ritar en cirkel

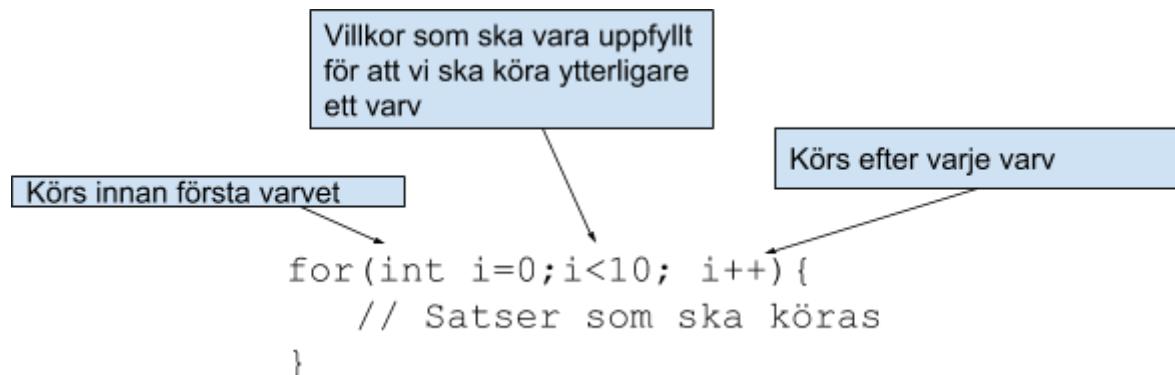
med motsvarande värden, se körexemplet nedan. Ritytan ska vara 400x400.



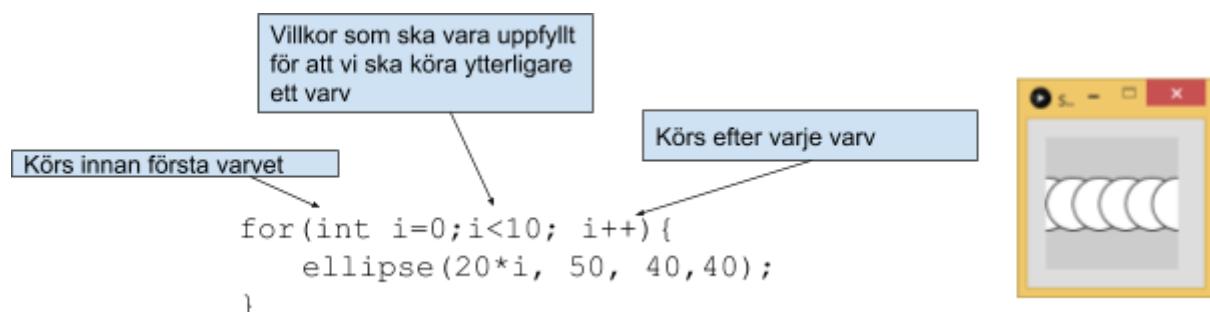
5 Loopar

5.1 For-loopar - Iterera (Upprepa) ett visst antal gånger

Ofta i programmering vill man att saker ska köras många gånger. Vi ska börja med att titta på for-loopar. For-loopar används när man vill köra en eller flera satser ett visst antal gånger direkt efter varandra.



Exempel på ett helt program:



Det ser ju fint ut. Men alla cirklar syns inte. Vi behöver skriva ett *size*-kommando först. Programmet blir då:

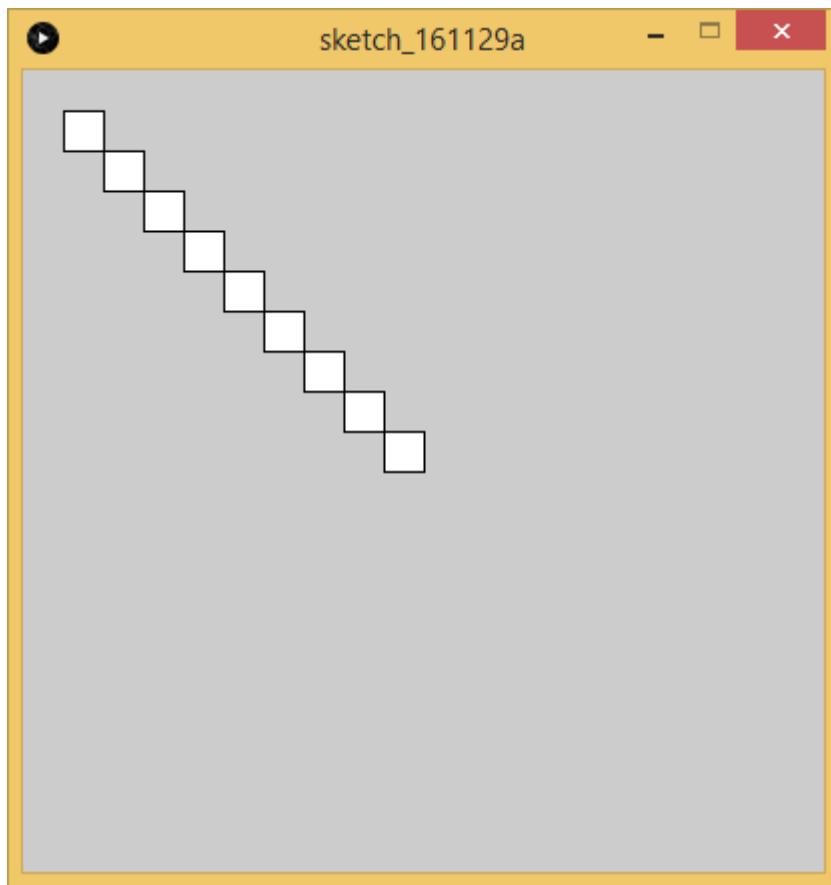
The screenshot shows the Processing IDE interface. On the left is the code editor with the following code:

```
1 size(300,100);
2 for(int i=0;i<10; i++){
3   ellipse(20*i, 50, 40,40);
4 }
```

On the right is the preview window titled "sketch_161225b" showing 10 circles drawn in a horizontal row.

5.2 Exempel for-loopar

Använd en for-loop för att skapa mönstret nedan. Storleken på sidan på kvadraterna är 20, och storleken på ritytan är 400*400.

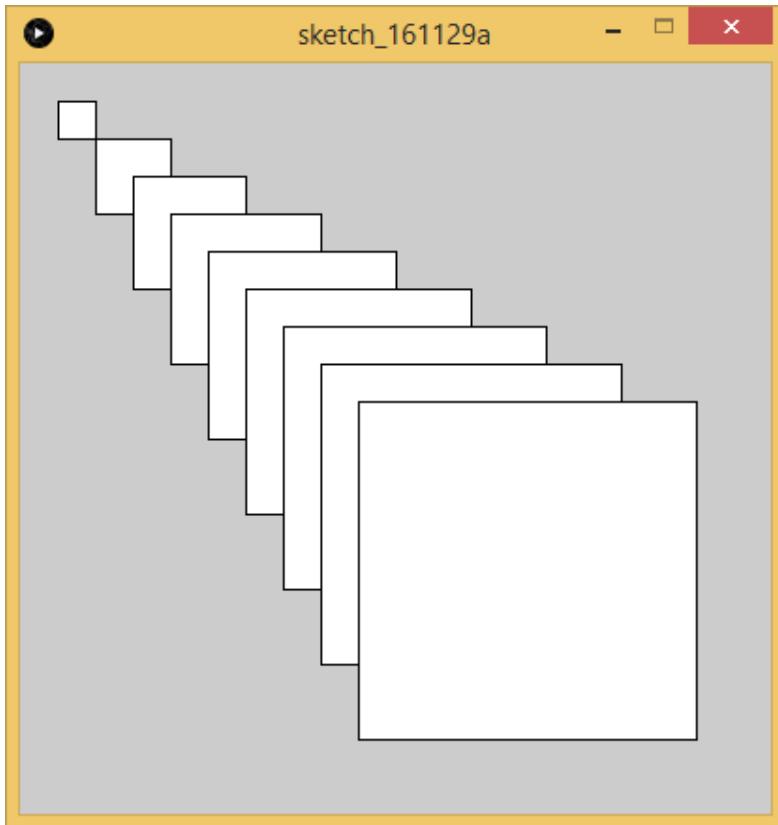


Svar:

```
size(400, 400);
for(int i=1; i < 10; i++){
    rect(20*i, 20*i, 20, 20);
}
```

Övning 5-1 Övning större och större rektanglar

Använd en for-loop för att skapa mönstret nedan. Storleken på sidan på den första kvadraten är 20, och storleken på ritytan är 400*400.



Övning 5-2 Övning gradient

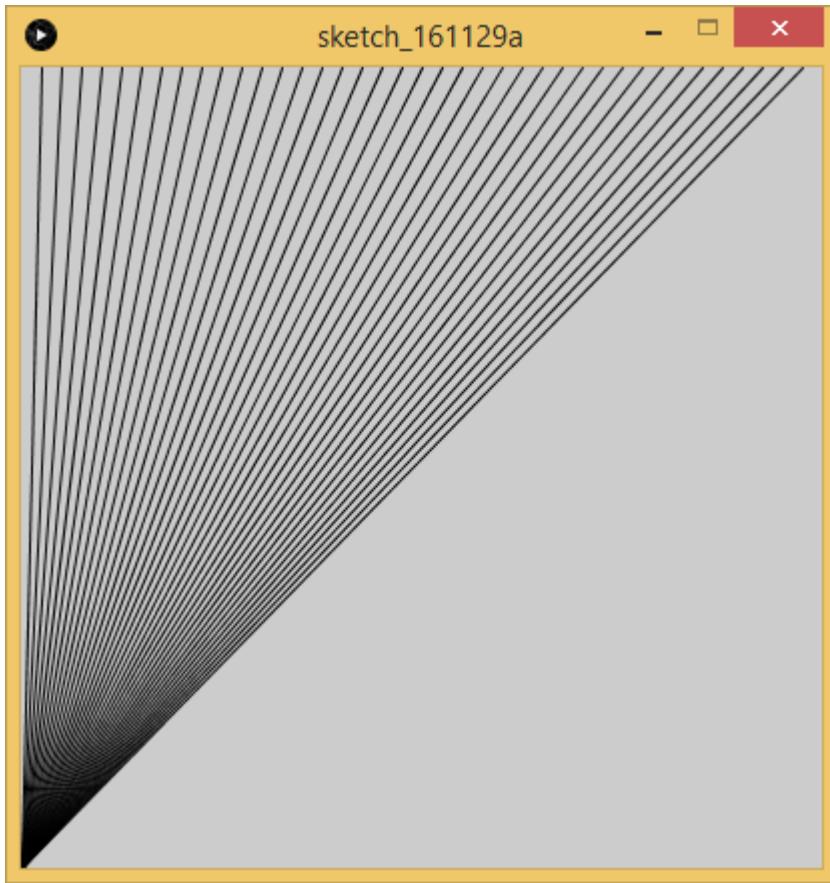
Skapa ett program med storleken 255*255 som ritar ut en gradient enligt bilden nedan.

Tips: Rita ut 255 linjer nedanför varandra, och innan du skriver ut linjen så ändrar du färg på linjen med stroke().



Övning 5-3 Exempel linjer

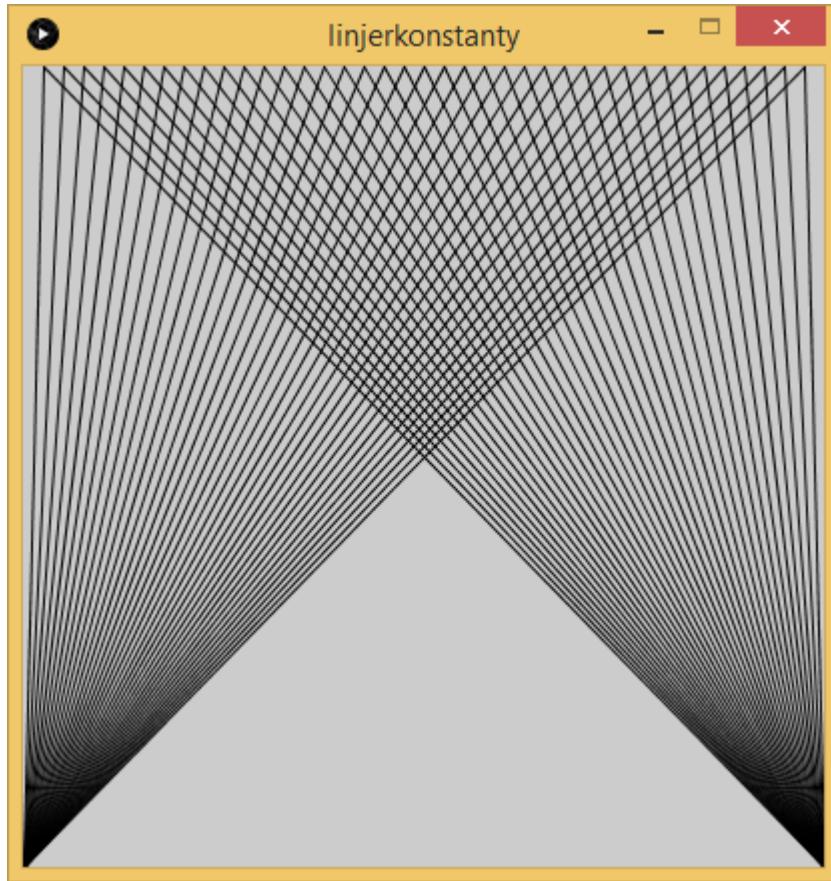
Skapa mönstret nedan med hjälp av en for-sats. Ritytan är 400*400, och avståndet är 10px på överkanten.



```
size(400,400);
for(int i=1; i < 40; i++){
    line(i*10, 0,0, 400);
}
```

Övning 5-4 Linjemönster

Fortsätt på ovanstående exempel för att skapa mönstret nedan.



Övning 5-5 Rita kataloniens flagga

Skapa kataloniens flagga genom att använda två for-loopar i Processing.

- Ritytan ska vara 600x396.
- Bredden på ränderna 44 pixlar.

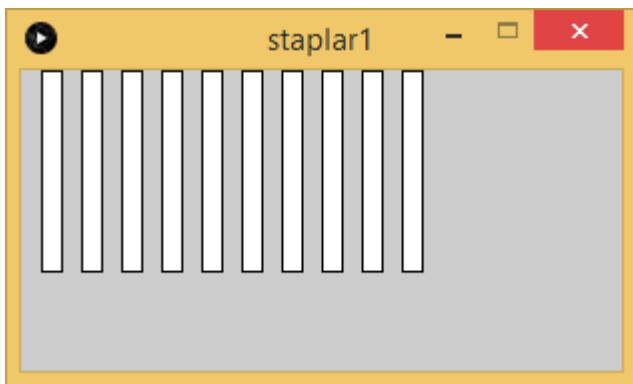
Färgnyanser

- Den röda färgen ska ha rgb-värdena R: 218, G: 18, B: 26
- Den gula färgen ska ha rgb-värdena R: 252, G: 221, B: 9



Övning 5-6 Rita staplar

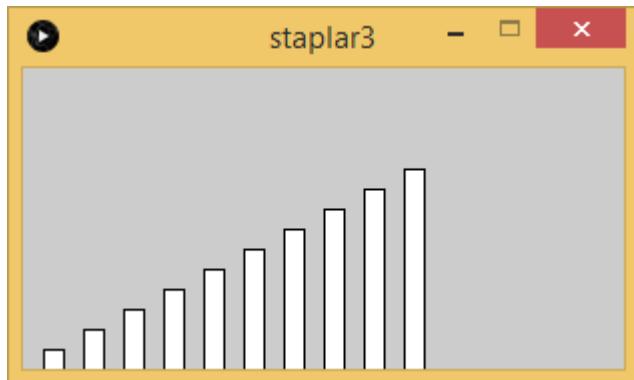
Använd en for-loop för att skapa staplar som ser ut som bilden nedan. Staplarna är 100 pixlar höga och 10 pixlar breda.



Ändra i programmet så att den första stapeln är 10 pixlar hög och sedan ökar varje stapel med 10 pixlar.



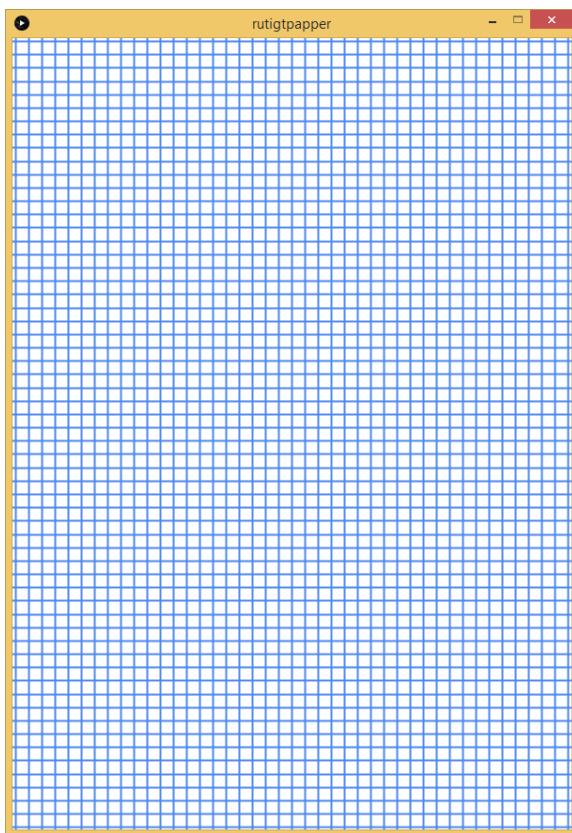
Nu vill vi istället att staplarna ska utgå från nederkanten, så att det ser ut som ett stapeldiagram.



Övning 5-7 Rutigt papper

Skapa ett rutigt papper enligt nedan som ska kunna användas för att skriva ut för att få ett rutigt papper.

Skapa ett fönster med upplösningen 595x842. Låt avståndet mellan varje ruta vara fem millimeter. För att räkna ut hur många pixlar det motsvarar så behöver du veta att ett A4 är 210x297 mm stort. Spara bilden genom att avsluta med saveFrame("rutigtpapper.png");



5.3 While-loopar

While-loopar är den typ av loop som vi använder när vi inte vet hur många gånger vi ska köra en loop.

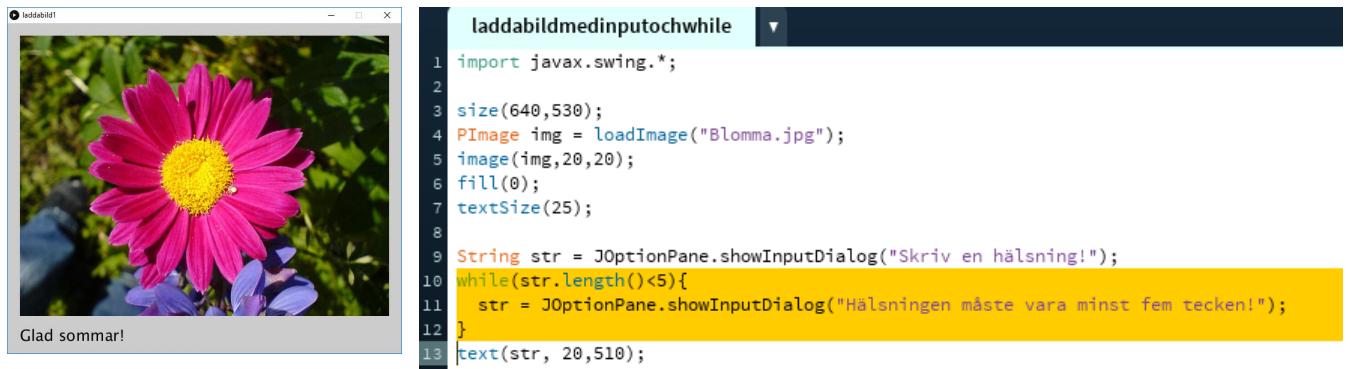
```
while(villkor){  
    saker som ska utföras  
}
```

Om ni kommer ihåg exemplet med blomman med en hälsning ovan, så kanske vi vill lägga till att man minst måste skriva fem tecken i hälsningen, annars blir användaren tvungen att försöka en gång till.

Vi lägger till en while-sats som körs så länge som strängen är mindre än fem tecken.

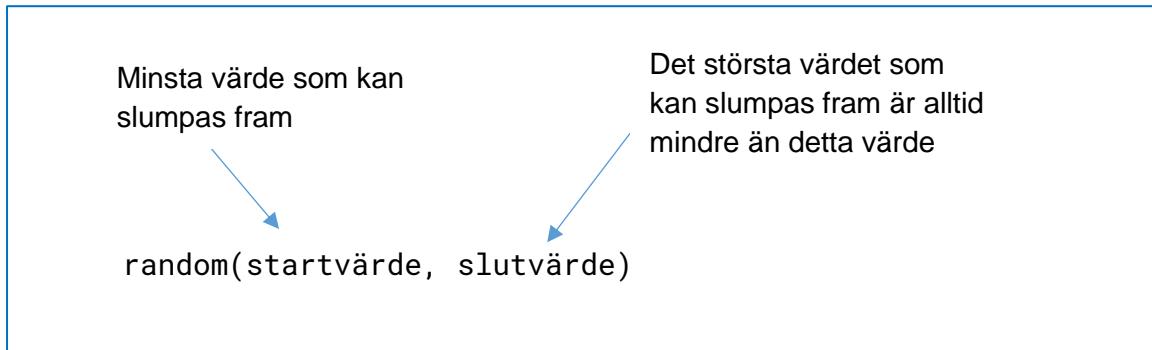
```
9 String str = JOptionPane.showInputDialog("Skriv en hälsning!");  
10 while(str.length()<5){  
11     str = JOptionPane.showInputDialog("Hälsningen måste vara minst fem tecken!");  
12 }  
13 text(str, 20,510);  
14
```

Hela programmet blir då:



5.4 Slumptal i Processing

Ganska ofta när vi programmerar så önskar vi att programmet beter sig på ett slumptat sätt. Det kan vara för att skapa fina mönster med ett slumptat inslag, det kan vara hinder i ett spel som ska placeras ut på ett slumptat sätt, eller så vill vi kanske skapa matteövningar med slumpade tal. Som tur är så har skaparna av Processing förstått detta och skapat en funktion random som kan hjälpa oss med detta.



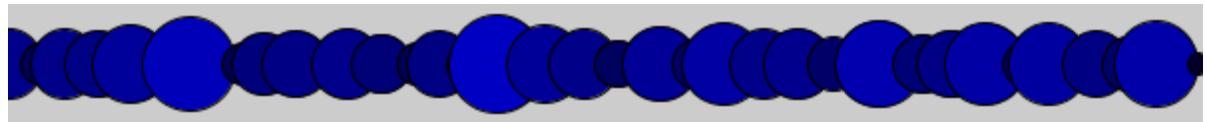
För att visa hur det fungerar i praktiken så har jag exempel nedan som slumpar 20 stycken tal och skriver ut dem, som ni ser så

The screenshot shows a Processing sketch window titled "visaslumpal". The window displays 20 randomly generated floating-point numbers between 2 and 5. The code for the sketch is as follows:

```
1 size(150,650);
2 background(0);
3 fill(255);
4 textSize(25);

6 for(int i=0; i < 20; i++){
7   float r = random(2,5);
8   text(r,20,40+30*i);
9 }
```

5.5 While-loopar och slumptal



While-loopar använder man alltså om man inte vet hur många gånger en loop ska köras. Det kan till exempel vara när man har med slump att göra. Om till exempel vill skapa en bild som den ovan med ett antal olika stora cirklar, som vi placerar bredvid varandra, med en halv diameters avstånd. Vi vet att vi ska sluta när x-värdet blir mer än 600, men vi vet inte exakt hur många gånger det tar.

För att slumpa ett tal så används funktionen `random(min, max)`, som vi såg ovan. Om vi till exempel vill slumpa fram en diameter mellan 5 och 50 och lagra i variabeln d, så skriver man:

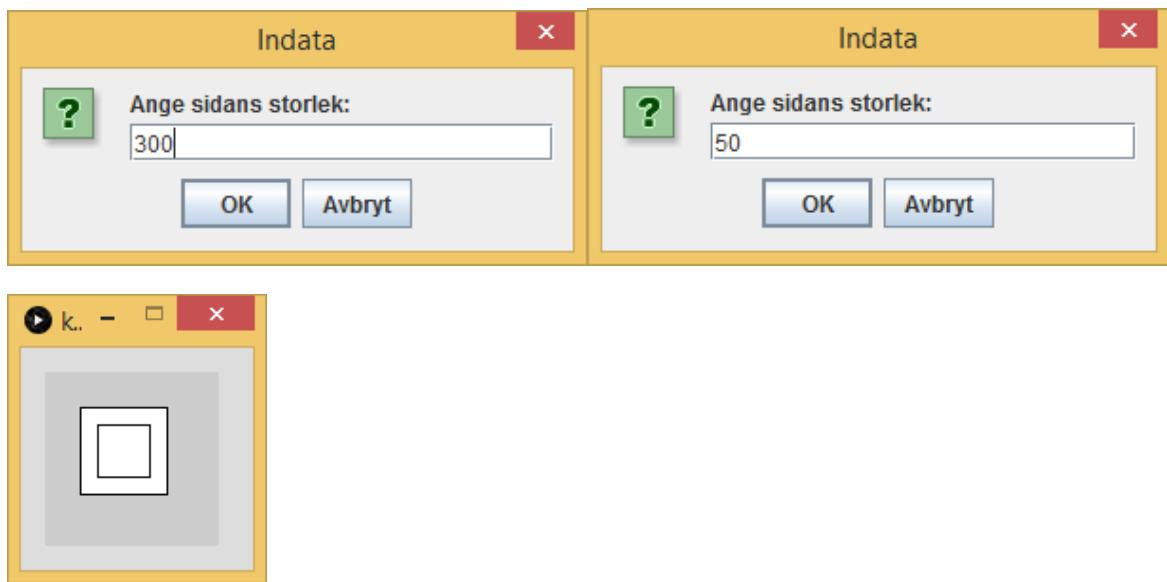
```
float d = random(5,50);
```

Vi inför en variabel x för att lagra vilken x-position som vi sist ritade ut en cirkel p.

```
whileex1 ▾  
1 size(600,60);  
2 float x=0;  
3 while(x<width){  
4   float d = random(5,50);  
5   x+=d/2;  
6   fill(0,0,4*d);  
7   ellipse(x,30,d,d);  
8 }
```

Övning 5-8 Övning kvadrat i kvadrat med inmatning och storlekskontroll

Ändra i övningen [kvadrat i kvadrat med inmatning](#) ovan så att användaren får möjlighet att ange den yttre rektangelns värde, och som upprepar inmatningen så länge som användaren skriver in ett så stort värde som gör att hela rektangeln inte syns, tex så tillåts inte ett värde större än 80 i exemplet nedan.



Övning 5-9 Slumpa fram cirklar så längre mitten är fri

a)

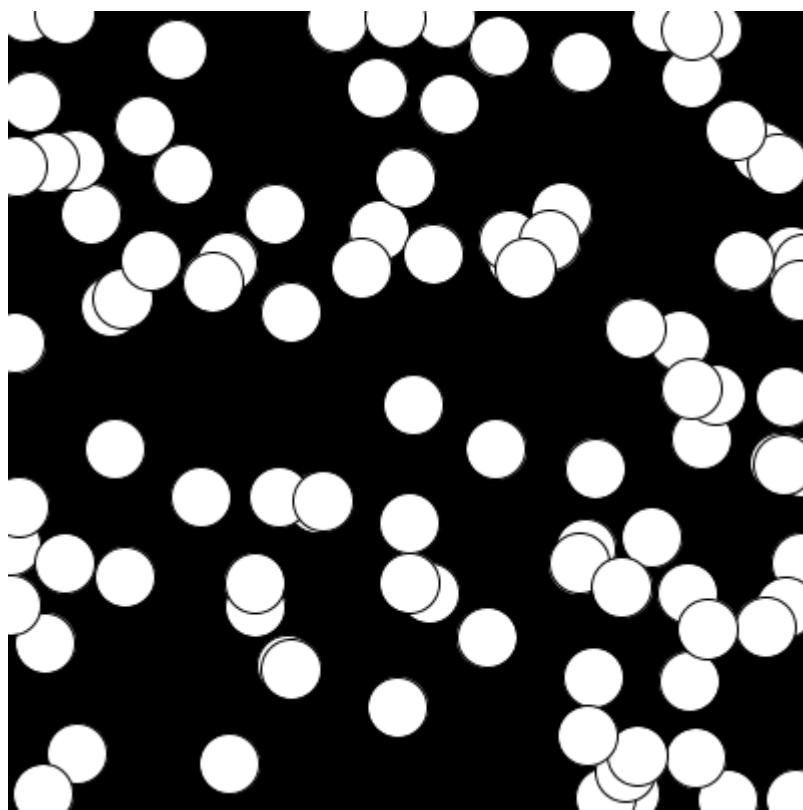
Skapa ett program som "slumper" fram cirklar fram till dess att mittpunkten inte längre är svart.

Du börjar med att sätta bakgrunden till svart och sedan slumper du fram cirklar fram till dess inte den mittersta punkten är svart För att kontrollera färgen på en viss punkt använder du `get(x, y)`, exempelvis om du vill kolla att punkten 20,20 är klarblå skriver du:

```
get(20,20)==color(0,0,255);
```

Storleken på ritytan ska vara 400x400. Cirkeln ska ha storleken 30 pixlar. Om du vill att bilden som du skapar ska se ut exakt som bilden lägger du i början av programmet till raden:

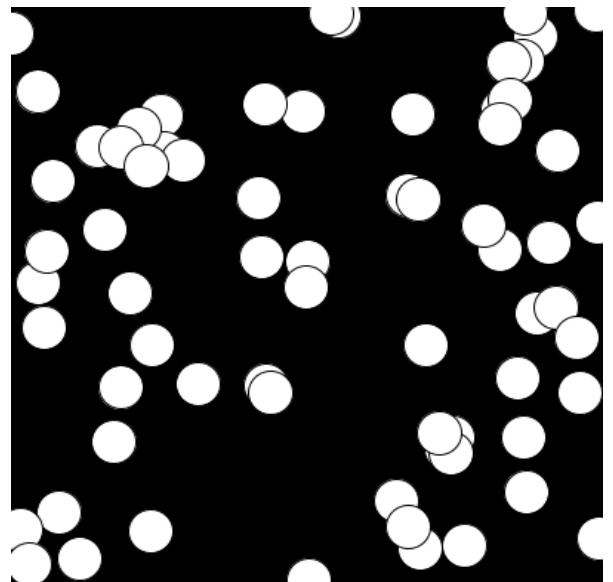
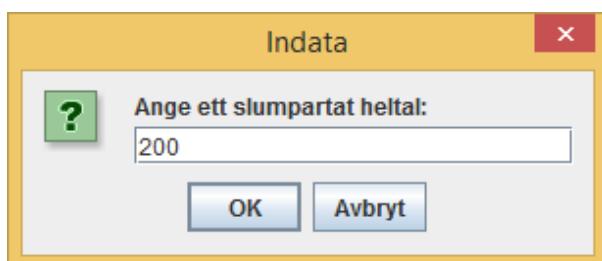
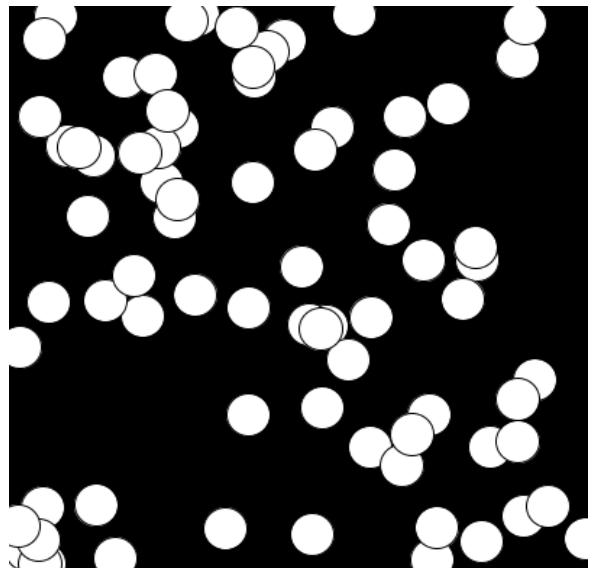
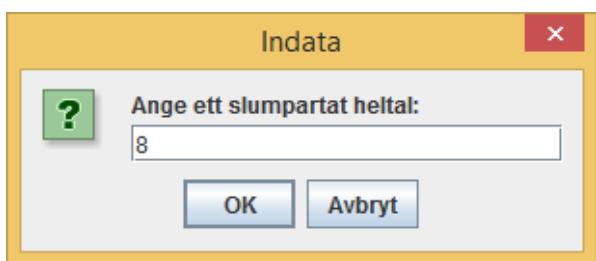
```
randomSeed(0);
```



b)

Den här gången ska du utöka programmet så att användaren får ange värdet till `randomSeed()`. `randomSeed` behöver köras i början av programmet. På detta sätt kan du skapa olika mönster som sedan kan återskapas genom att ange samma nummer igen.

Körexempel:



Övning 5-10 Numerisk lösning av ekvation

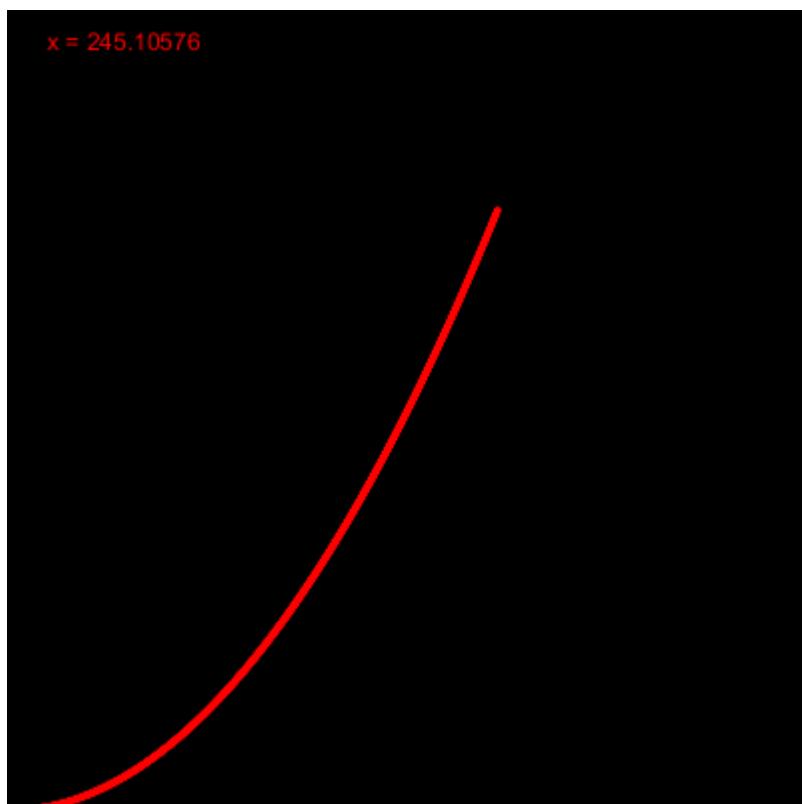
a)

Skapa ett program som räknar ut ekvationen

$$\frac{x^2}{200} = 300$$

genom att först sätta x till noll, och sedan öka x med 0.1 i taget och sedan sätta $y=x^2/200$ och köra så länge som $y < 300$. Rita också ut kurvan genom att för varje steg rita ut en klarröd cirkel med diametern tre och utan kantlinje. När loopen har stannat ska du skriva ut svaret på position 20,20, se bild nedan. Du börjar med att sätta bakgrunden till svart

Storleken på ritytan ska vara 400x400

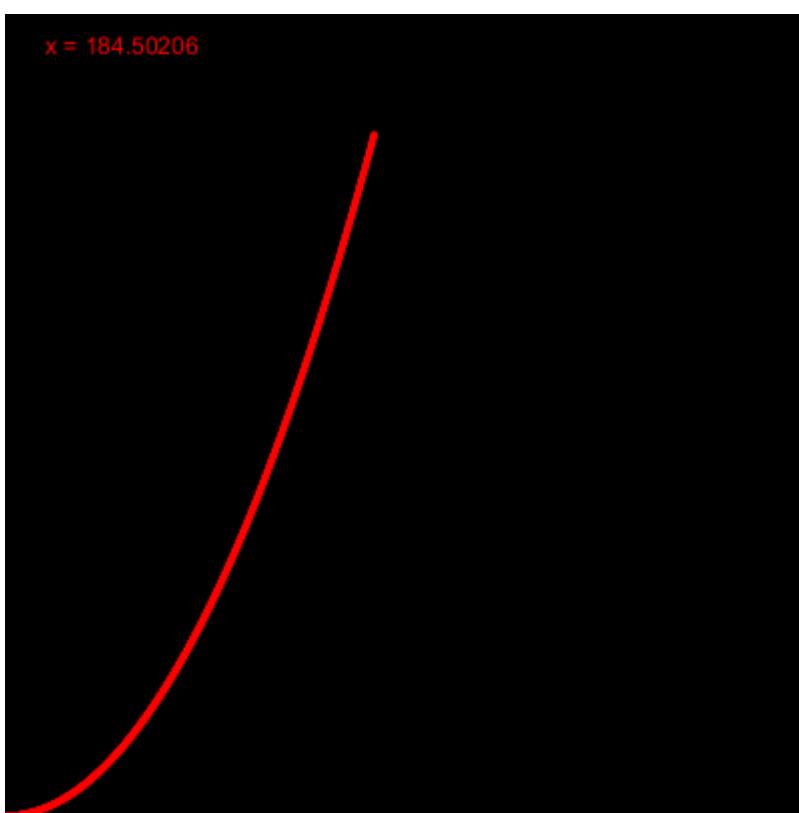


b)

Fortsätt på a-uppgiften och skapa ett program som räknar ut ekvationen

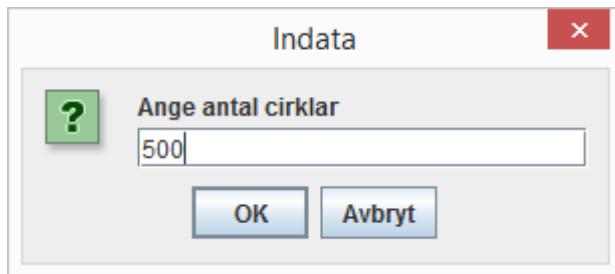
$$\frac{x^2}{n} = a$$

och ritar upp grafen. Men den här gången ska användaren få skriva in konstanterna a och n

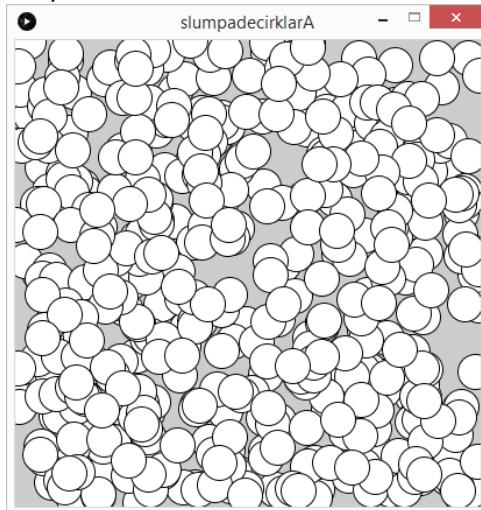


Övning 5-11 Övning på while och for-loop och slumptal

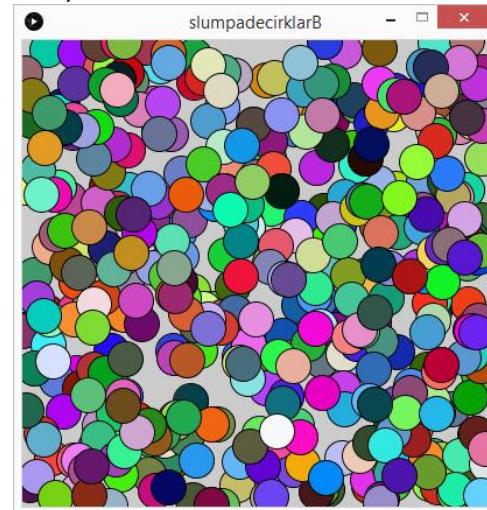
- Skapa ett program som frågar användaren hur många cirklar den vill rita ut och sedan ritar ut så många cirklar som användaren harbett om på en slumptad position, det vill säga x-värdet ska vara mellan 0 och width, och y-värdet är mellan 0 och height. cirkeln ska ha diametern 30.
- Ändra i programmet så att du använder while istället för for. (Detta är egentligen lite fult eftersom vi normalt brukar använda for när vi vet hur många gånger som något ska köra, men det kan vara en bra övning)
- Slumpa också vilken färg cirklarna ska ha. Det vill säga r,g och b-värdena ska sättas till ett slumptal mellan 0 och 255.



A)



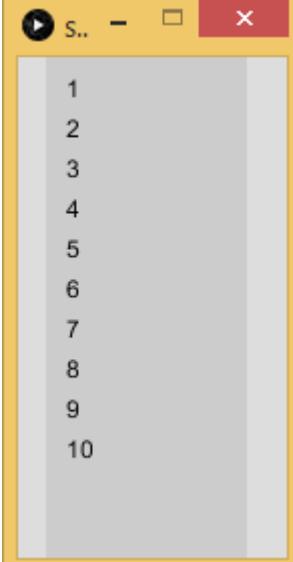
B)



7 Mer om variabler

7.1 Ett exempel med int, for och heltalsdivision

För att visa på ett fenomen som kan tyckas lite märkligt i Processing/Java ibland så ska vi börja med att dra oss till minnes hur man skriver en for-loop.

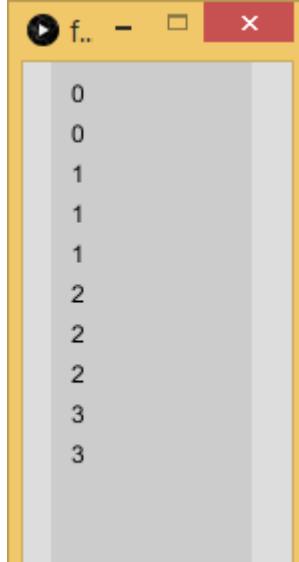


```
fordelat
1 size(100,250);
2 fill(0);
3 for(int i=1; i <= 10; i++){
4     text(i, 10, 20*i);
5 }
6
```

The code defines a sketch named "fordelat". It sets the size of the canvas to 100x250 pixels and fills the background with black. A for loop iterates from 1 to 10, printing each value at a y-position of 20 times its value. The output shows the numbers 1 through 10 in a vertical column.

Vi testar nu istället att skriva ut $i/3$. Om det hade fungerat som vi är vana vid från matten så skulle det först stå 0.333, sedan 0.667, och sedan 1, och sedan 1.333 osv. Men resultatet blir istället det som visas nedan.

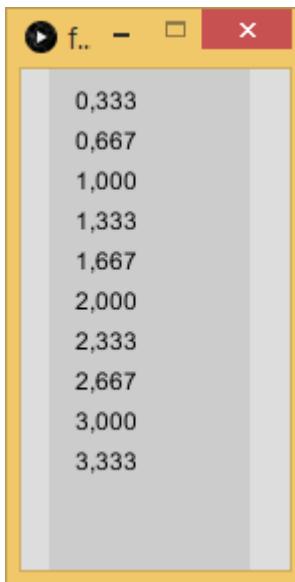
Vad är det nu som händer? Jo det är nämligen så att Processing och Java använder sig alltid av heltalsdivision när man försöker dividera två heltal.



```
fordelat
1 size(100,250);
2 fill(0);
3 for(int i=1; i <= 10; i++){
4     text(i/3, 10, 20*i);
5 }
6
```

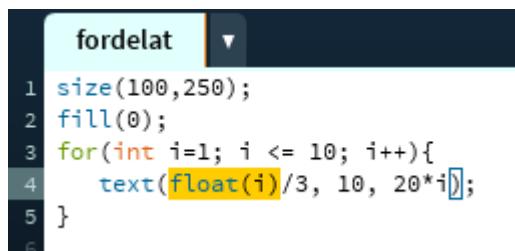
The code is identical to the previous one, except it divides i by 3 instead of multiplying it by 20. The output shows floating-point numbers 0.0, 0.333, 0.667, 1.0, 1.333, 1.667, 2.0, 2.333, 2.667, and 3.0.

Det enklaste sättet att åtgärda detta är att dela med 3.0 istället för 3. Så fort man skriver en decimalpunkt så tolkas talet som ett decimaltal.



```
fordelat
1 size(100,250);
2 fill(0);
3 for(int i=1; i <= 10; i++){
4     text(i/3.0, 10, 20*i);
5 }
6
```

Men man kan lika gärna omvandla variabeln `i` till float genom att skriva: `float(i)`



```
fordelat
1 size(100,250);
2 fill(0);
3 for(int i=1; i <= 10; i++){
4     text(float(i)/3, 10, 20*i);
5 }
6
```

7.2 Sammanfattning av skillnaden mellan heltal (int) och decimaltal (float)

Vi nämnde ovan att det är skillnad mellan decimaltal och heltal. Här kommer några frågor och svar angående skillnaden mellan heltal och decimaltal.

7.2.1 När ska vi använda heltal (int)?

- I for-loopar
- Vissa funktioner kräver att vi har ett heltal som parameter
 - Till exempel `get(x, y)`, som ger färgen på en viss pixel, där måste koordinaterna ges som heltal (int)
- När det handlar om att hålla reda ett antal saker.
 - Till exempel hur många gånger har användaren tryckt på musknappen.
- När vi vill använda heltalsdivision som till exempel $3/2=1$

7.2.2 När ska vi använda decimaltal (float)

- När vi vill kunna räkna med decimaltal som vi är vana från matematiken
- I Processing är det oftast enklast att använda `float` till koordinater i en animation eller spel trots att det handlar om uppräkningsbara punkter.
- När vi vill använda vanlig division, som till exempel $3/2=1.5$

7.2.3 Vad händer när vi omvandlar decimaltal till heltal?

- Decimaldelen kastas bort.

```
3  
4 float a=2.32;  
5 int b=int(a);  
6  
7 text(b, 20,60);  
8
```



7.2.4 Vad händer när man omvandlar från heltal till decimaltal.

- Vi kan aldrig vara säkra på att talet lagras exakt.
 - Vid stora tal så kommer vi aldrig att få exakt noggrannhet
 - Även vid ganska små tal kan vi få avrundningsfel på grund av omvandlingar från decimala talsystemet till binära talsystemet och tillbaka.

```
4 int a = 1234567890;  
5 text(a,20,40);  
6 float b = a;  
7 text(b, 20,80);  
8
```



Hur ser vi skillnaden mellan decimaltal och heltal?

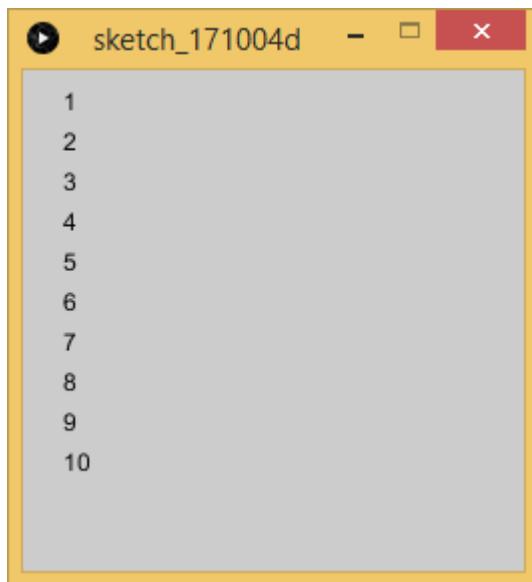
- I koden skriver vi en decimalpunkt för att ange ett decimaltal.
- Vid utskrift så visas alltid minst en decimal när vi skriver ut ett decimaltal (float), men aldrig några decimaler när man skriver ut ett heltal (int)

```
5 float a = 3.0;  
6 float b = 3;  
7 int c = 3;  
8 text("a är en float som är "+a, 20,30);  
9 text("b är en float som är "+b, 20,60);  
10 text("c är en int som är "+c, 20,90);  
11
```



8 Fler uppgifter på variabler och iteration

- 1 Skriv ett program som använder en for-loop och som genererar följande utskrift:



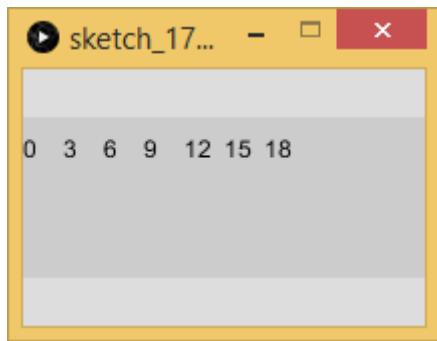
```
for [10 times] 
  say (1)
  next
end
```

- 2 Gör om programmet i 1) så att det använder en while-loop istället.
- 3 Skapa ett program som med hjälp av en for-sats skriver ut följande.

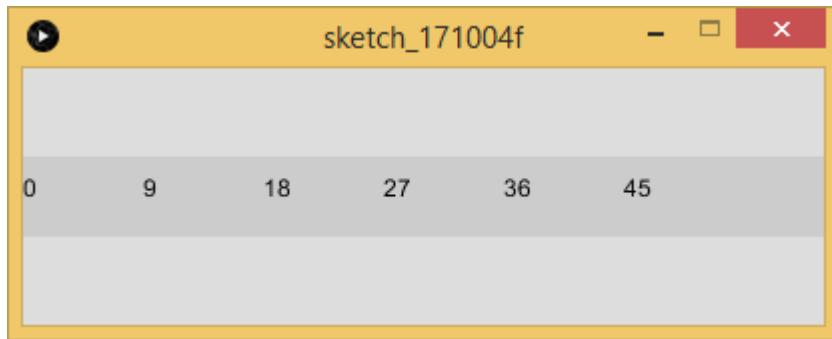


```
for [13 times] 
  say (3)
  next
end
```

- 4 Skapa ett program som med hjälp av en for-sats skriver ut följande.

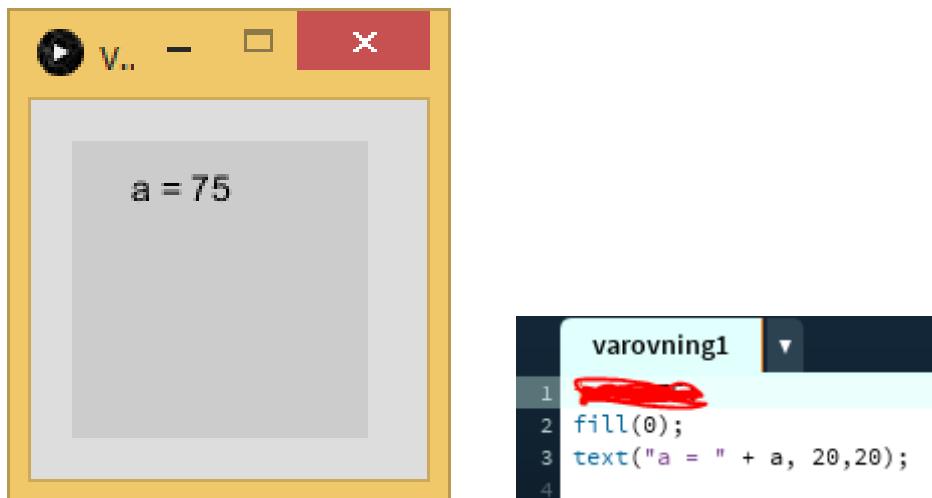


- 5 Skapa ett program som med hjälp av en for-sats skriver ut följande.



- 6 Gör om 5) så att den använder en while-sats istället.

- 7 Fyll i den överstrukna texten i exemplet nedan så du får utskriften enligt bilden nedan.



8 Fyll i den överstrukna texten i exemplet nedan så du får utskriften enligt bilden nedan.



```
varovning1
1 fill(0);
2 text("a = " + a, 20,20);
3
4
```

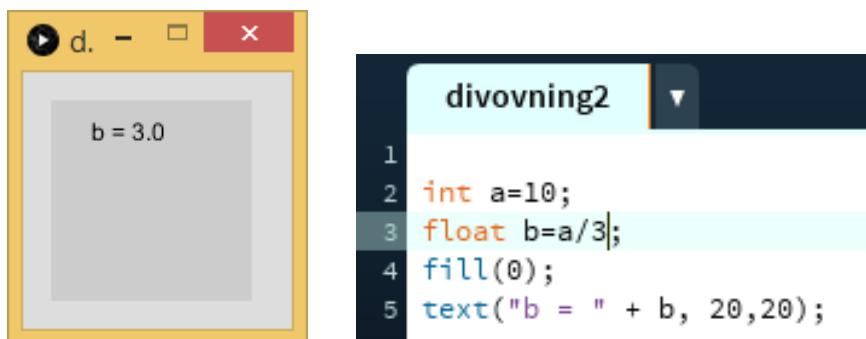
9 Ibland så väljer Processing/Java att använda heltalsdivision. Det kan bli lite tokigt. Ändra i exemplet så att utskriften istället blir: c = 1.8



```
divovning
1 int a=9;
2 int b=5;
3 float c=a/b;
4 fill(0);
5 text("c = " + c, 20,20);
6
```

10 Här har det också blivit fel. Du ska få utskriften att bli b = 3.3333 genom att:

- Ändra på rad 2
- Ändra nu tillbaks rad 2 så att exemplet ser ut som nedan, och ändra på rad 3 istället.



```
divovning2
1 int a=10;
2 float b=a/3;
3 fill(0);
4 text("b = " + b, 20,20);
5
```

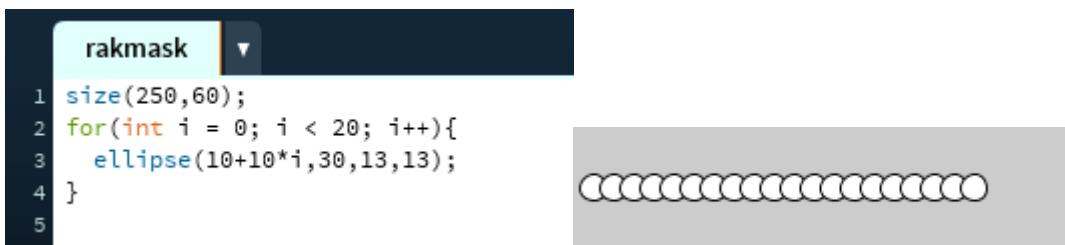
9 Arrayer

9.1 Vi skapar en mask med hjälp av en array med fördefinierade värden

Om vi tänker oss att vi vill rita ut en bild på en mask som ser ut som bilden nedan.



Skulle det ha varit en rak mask så hade det inte varit så svårt då hade programmet sett ut så här:



Nu vill vi istället rita ut en mask med en oregelbunden bana. Det vi vill åstadkomma är att det även i fortsättningen är exakt 10 pixlar mellan x-koordinaten på varje maskdel, men att y-koordinaten vill vi kunna sätta på ett mer oregelbundet sätt. Vi skapar en array för syftet:

```
int[] position = {20,22,25,30,35,40,42,40,35,30,25,22,20,17,15,14,13,14,15,17,19,20,20};
```

När vi deklarerar en array-variabel så skriver vi alltid datatypen följt av `[]`. Vill vi sedan deklarera startvärdet för arrayen så skriver dem innanför måsingar med komma emellan som i exemplet ovan.

Vill vi istället skapa en tom array så skriver vi:

```
int[] position = new int[23];
```

Och sedan får vi tilldela varje position för sig.

```
position[0] = 20;
position[1] = 22;
....
```

När vi sedan vill komma åt innehållet i en array så skriver vi på samma sätt som i vid tilldelning. Det vill säga indexet på platsen skrivet inom `[]`. Till exempel om vi vill skriva ut index 3 så skriver vi.

```
println(position[3]);
```

Vi kan också ha en variabel innanför hakparenteserna.

```

int index = 3;
println(position[index]);

```

Om vi nu vill skriva ut hela arrayen så kan vi skriva:

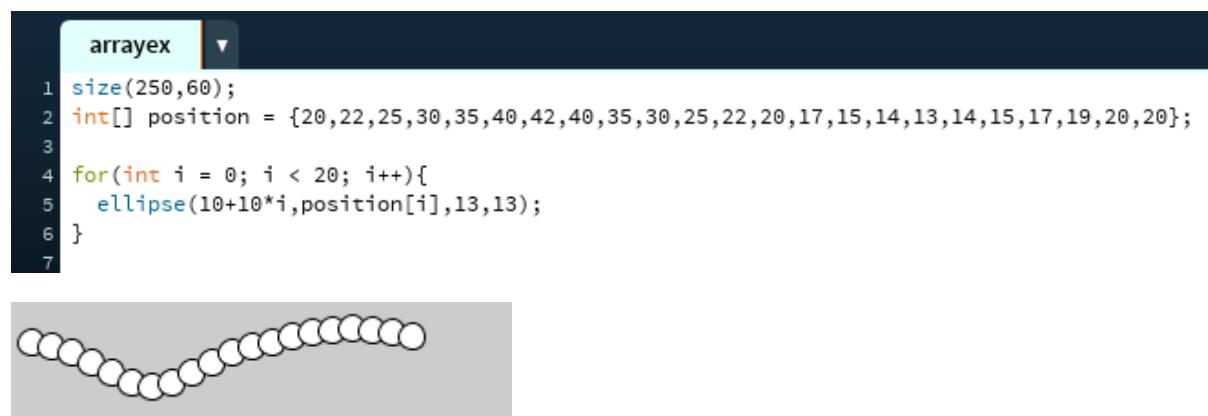
```

int[] position =
{20,22,25,30,35,40,42,40,35,30,25,22,20,17,15,14,13,14,15,17,19,20,20}
;
for(int i = 0; i < position.length; i++){
    println(position[i]);
}

```

Men nu ville vi ju rita ut cirklar så att det blev en mask.

Nu stoppar vi in arrayen i exemplet med den raka masken ovan, och vi byter ut det fasta y-värdet 30 till position[i],

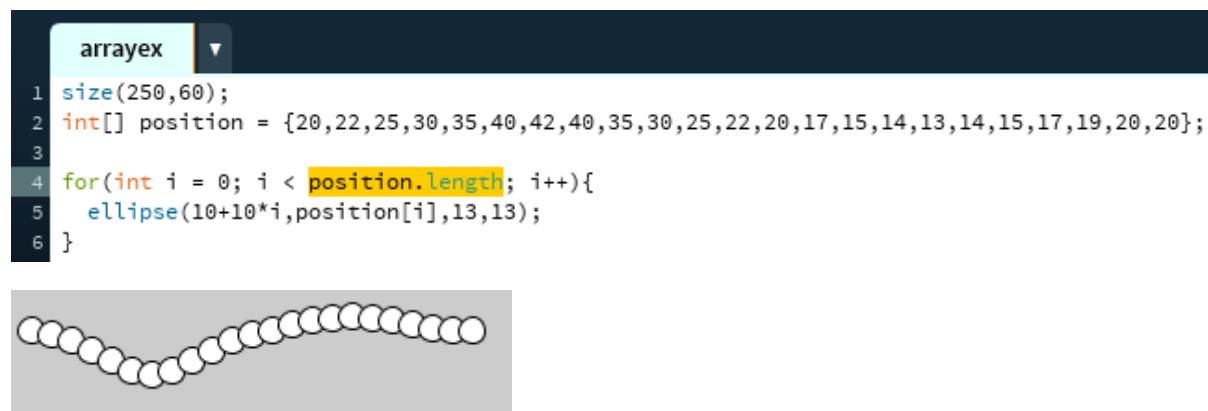


```

arrayex
1 size(250,60);
2 int[] position = {20,22,25,30,35,40,42,40,35,30,25,22,20,17,15,14,13,14,15,17,19,20,20};
3
4 for(int i = 0; i < 20; i++){
5     ellipse(10+10*i,position[i],13,13);
6 }
7

```

Nu blev det nästan rätt. Det visade dock att det var 23 värden i arrayen och inte 20. Om vi vill att for-loopen alltid ska köra lika många varv som arrayen är lång så byter vi ut slutvärdet 20 mot position.length.



```

arrayex
1 size(250,60);
2 int[] position = {20,22,25,30,35,40,42,40,35,30,25,22,20,17,15,14,13,14,15,17,19,20,20};
3
4 for(int i = 0; i < position.length; i++){
5     ellipse(10+10*i,position[i],13,13);
6 }
7

```

9.2 Vi skapar en tom array och låter användaren skriva in värdena

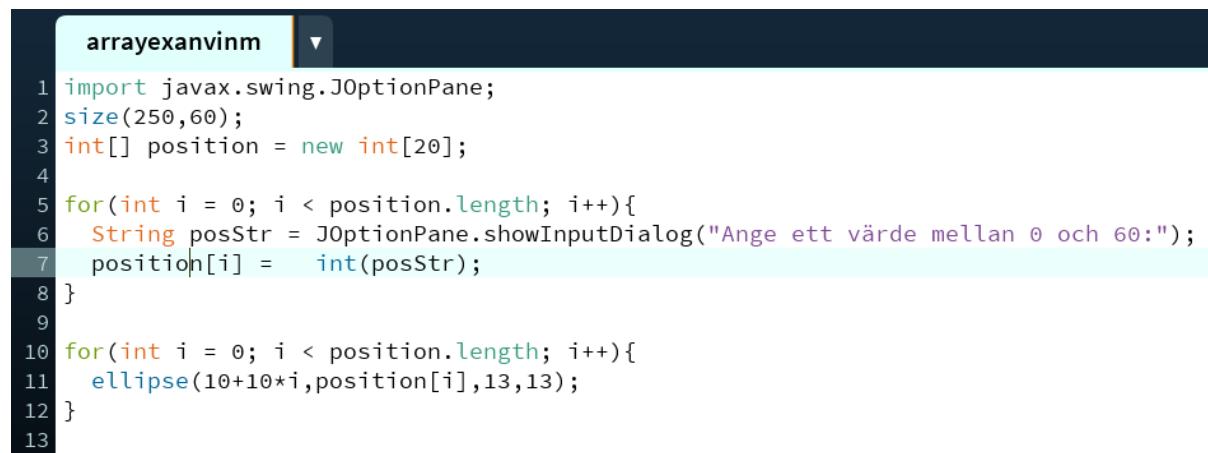
Fast vore det inte trevligt om användaren hade möjlighet att bestämma värdena istället? I så fall måste vi skapa arrayen först och sedan fylla den med innehåll. För att skapa arrayen skriver vi i så fall:

```
int[] position = new int[20];
```

Och sedan så behöver vi en for-loop för att mata in alla värdena, ett i taget:

```
for(int i = 0; i < position.length; i++){
    String posStr = JOptionPane.showInputDialog("Ange ett värde mellan 0 och 60:");
    position[i] = int(posStr);
}
```

Hela programmet blir då:



```
arrayexanvinm
1 import javax.swing.JOptionPane;
2 size(250,60);
3 int[] position = new int[20];
4
5 for(int i = 0; i < position.length; i++){
6     String posStr = JOptionPane.showInputDialog("Ange ett värde mellan 0 och 60:");
7     position[i] = int(posStr);
8 }
9
10 for(int i = 0; i < position.length; i++){
11     ellipse(10+10*i,position[i],13,13);
12 }
13
```

Övning 9-1 Övning stapeldiagram

- A. Skapa ett program som ritar ut ett stapeldiagram. Fortsätt på *Övning staplar* och ändra så att det skapas ett stapeldiagram med värden från en array.
Tips: Skapa först en array med de värden du ska visa i diagrammet.
- B. Utöka programmet så att användaren kan mata in värdena i diagrammet, och spara diagrammet varje gång det körs. (Spara gör man med kommandot
`save("diagram.png");`)
- C. * Flytta upp själva diagrammet och lägg till en rubrik för varje stapel under diagrammet. Skapa en till array för rubrikerna.

Övning 9-2 Övning olika nyanser av blått

Fortsätt på övningen *Större och större rektanglar* och gör att så att varje rektangel får olika nyanser av blått.

Övning 9-3 Övning bilder på rad

Skapa en array av filnamn på bilder, och skapa ett program som ritar ut bilderna på en rad efter varandra med 100 pixlars mellanrum.

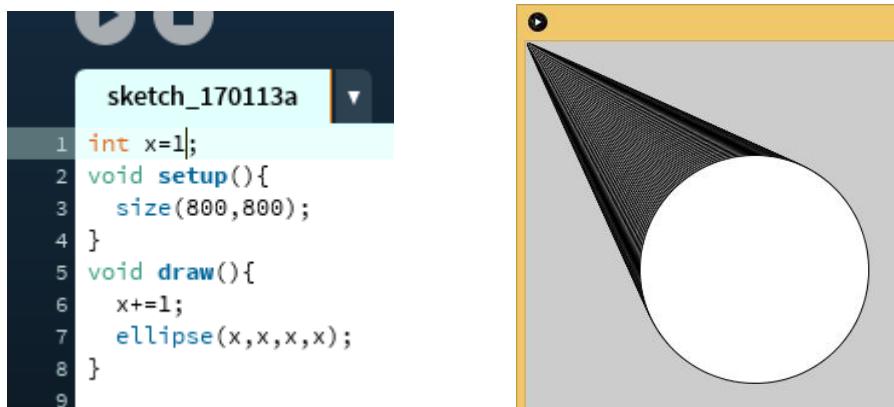
11 Tidsstyrda iterationer

I Processing är ett vanligt sätt att iterera att använda tidsstyrda iterationer. Det innebär att ett visst kodavsnitt körs med ett visst tidsintervall. I Processing är standardintervallet en sextiondels sekund. I Processing används denna teknik framförallt i animeringar och liknande för att kunna ändra bilden under körning. I de tekniker som används ovan så kan man enbart generera en stillbild. För att kunna använda tidsstyrda iterationer så delar man upp sin kod i två delar. Först den del som körs i början (`setup()`) och den som ska köras med ett intervall (`draw()`).

För att markera den första delen så skriver man `void setup() {` och sedan den kod som man vill ska köra följt av en `}`, och på motsvarande sätt skriver man `void draw() {` för den kod som man vill köra upprepade gånger, följt av `}`.

```
void setup(){
    //Kod som körs i början
}
void draw(){
    //Kod som körs sextio gånger i sekunden
}
```

Nedan finns ett litet exempel där vi skapar en variabel `x` och sätter den till ett, och sedan ökar variabeln med ett för varje iteration. Vi ritar också ut en cirkel med diametern `x`, på position `x=x` och `y=x`, för varje iteration. Lägg också märke till att vi definierar variabeln `x` först av allt utanför `setup` och `draw`. Variabler glöms alltid bort efter nästkommande `}`-tecknen.



11.2 Tidstydritation kombinerat med arrayer:

Vi kan nu använda kunskapen angående tidsstyrd iteration och tillämpa det på vårt arraymaskexempel ovan. Själva deklarationen av arrayen och vår x - variabel ska vara i början. I setup behöver vi bara anropa size-funktionen, och i draw så har vi vår for-loop som ritar ut masken. Lägg märke till att det nu står x istället för 10 först i anropet till ellipse-funktionen. Sist i draw ökar vi också x med ett.

```
arrayexanimerad
1 int[] position = {20, 22, 25, 30, 35, 40, 42, 40, 35, 30, 25, 22, 20, 17, 15, 14, 13, 14, 15, 17, 19, 20, 20};
2 int x = 0;
3 void setup(){
4     size(250, 60);
5
6 }
7
8 void draw() {
9     background(0);
10    for (int i = 0; i < position.length; i++) {
11        ellipse(x+10*i, position[i], 13, 13);
12    }
13    x++;
14 }
15
```

Om vi istället vill att användaren ska mata in värdena så kommer vi att först skapa en tom array i början av programmet, och sedan måste själva inmatningen ske i setup-funktionen eftersom den ska ske en gång i början. Funktionen draw behöver inte ändras.

```
arrayexanimeradmedinmatning
1 import javax.swing.JOptionPane;
2
3 int[] position = new int[20];
4 int x = 0;
5
6 void setup() {
7     size(250, 60);
8     for (int i = 0; i < position.length; i++) {
9         String posStr = JOptionPane.showInputDialog("Ange ett värde mellan 0 och 60:");
10        position[i] = int(posStr);
11    }
12 }
13
14 void draw() {
15     background(0);
16     for (int i = 0; i < position.length; i++) {
17         ellipse(x+10*i, position[i], 13, 13);
18     }
19     x++;
20 }
21
```

11.3 Tidsstyrda iteration kombinerat med slumptal

Tanken är att slumpra fram cirklar på en fast position med en slumptad storlek. På rad 6 så börjar vi med att sätta bakgrundens till svart och suddar därmed ut den cirkel som ritades ut förra gången. På rad 7 skapar vi en variabel som heter d, och slumper fram ett tal mellan 5 och 400 som lagras i d. Sedan ritar vi på rad 8 ut cirkeln på position 200,200 och diametern d. Detta upprepas nu 60 gånger per sekund.

```
slumpad_storlek_cirklar1 ▾  
1 void setup() {  
2     size(400, 400);  
3 }  
4  
5 void draw() {  
6     background(0);  
7     float d=random(5,400);  
8     ellipse(200,200,d,d);  
9 }  
0  
1
```

[Tryck här för att testa exemplet i webbläsare](#)

Det blir ett ganska flimrigt intryck när vi kör den. Vi kan förbättra programmet genom att minska hastigheten på animationen. Detta gör vi med kommandot frameRate() som vi kör i setup(). Med frameRate kan man ange hur ofta funktionen draw() ska köras, vilket i detta fall blir samma sak som antal gånger per sekund som det ritas ut en cirkel.

```
slumpad_storlek_cirklar1 ▾  
1 void setup() {  
2     size(400, 400);  
3     frameRate(7);  
4 }  
5  
6 void draw() {  
7     background(0);  
8     float d=random(5,400);  
9     ellipse(200,200,d,d);  
10 }
```

[Tryck här för att testa exemplet i webbläsare](#)

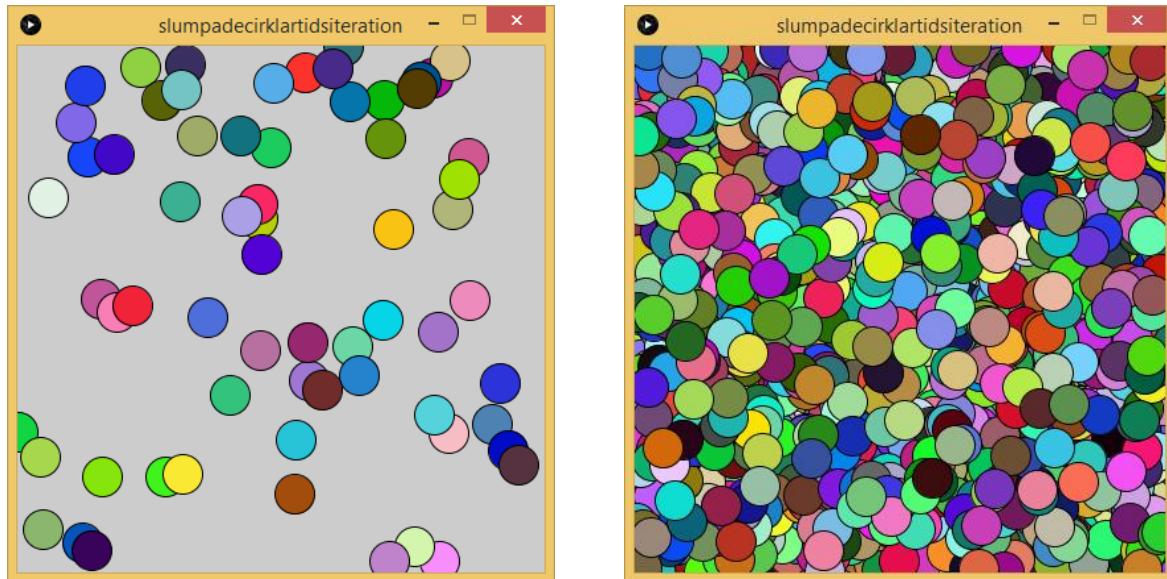
Som en sista övning så skulle jag också vilja sätta färg på cirklarna, genom att slumpa fram r, g och b värden, och sedan sätta färgen med fill.

```
slumpad_storlek_cirklar
1 void setup() {
2     size(400, 400);
3     frameRate(8);
4 }
5
6 void draw() {
7     background(0);
8     float d=random(5,400);
9     float r=random(0,255);
10    float g=random(0,255);
11    float b=random(0,255);
12    fill(r,g,b);
13    ellipse(200,200,d,d);
14 }
15 }
```

[Tryck här för att testa exemplet i webbläsare](#)

Övning 11-1 Slumpade animerade cirklar

Gör ett program som för varje gång draw körs, slumper färg och position för en cirkel, och ritar ut. Uppgiften blir ganska lik Övning på *while och for-loop och slumptal* ovan, men denna gång ska en cirkel i taget ritas ut.



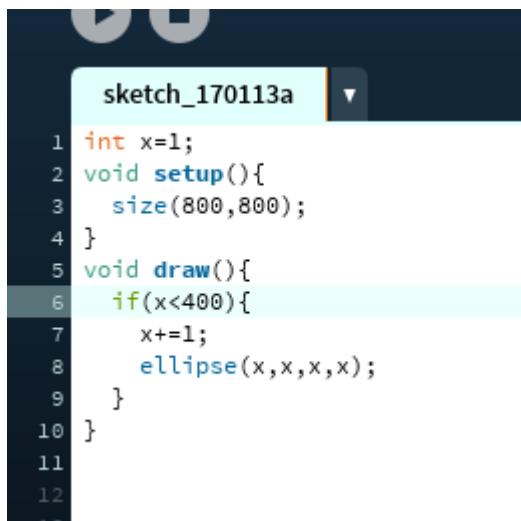
13 Selektion (if-satser)

Om vi vill att programmet ska sluta rita ut cirklar när vi har nått en viss storlek så kan vi använda en if-sats för att sätta ett villkor för när utritningen ska köras.

If-satser generellt:

```
if(villkor){  
    Satser som ska köras  
    .....  
}
```

Nu lägger vi till villkoret att cirkeln enbart ska ritas ut när x är mindre än 400, dvs $x < 400$



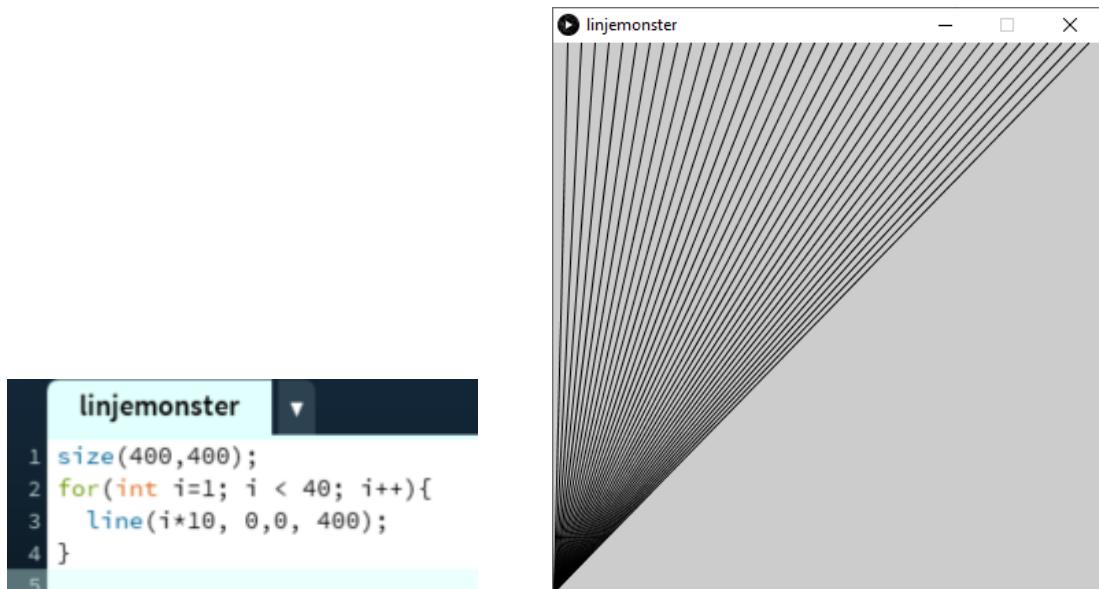
The screenshot shows the Arduino IDE interface with a sketch titled "sketch_170113a". The code is as follows:

```
1 int x=1;  
2 void setup(){  
3     size(800,800);  
4 }  
5 void draw(){  
6     if(x<400){  
7         x+=1;  
8         ellipse(x,x,x,x);  
9     }  
10 }
```

The line `if(x<400){` is highlighted in green, indicating it is the current line of code being edited or selected.

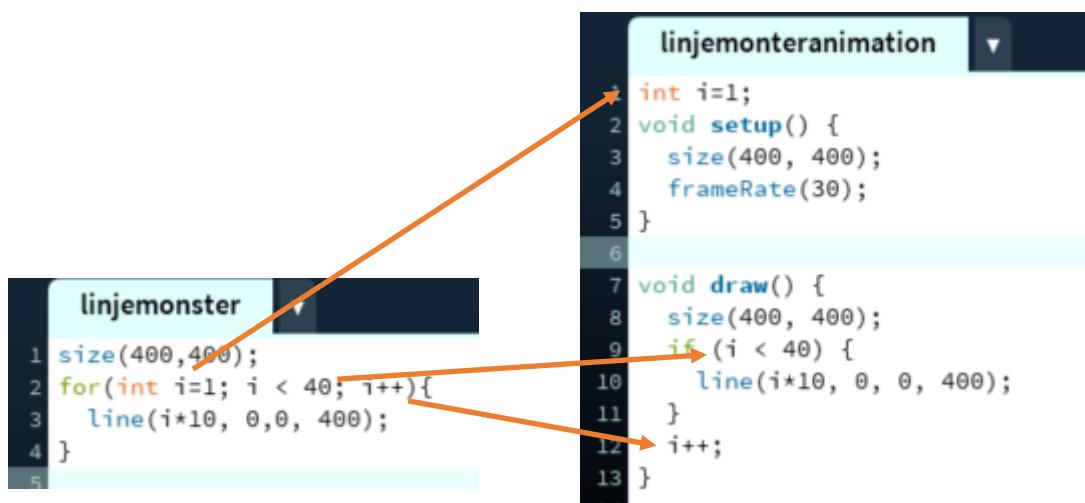
13.1.2 Konvertera for-loop till animering

Ibland så vill vi konvertera ett mönster som vi gjort med en for-loop i en stillbild till en animering. Tidigare så gjorde vi följande mönster med hjälp av en for-loop.



Vill vi nu göra samma mönster som en animering, det vill säga att en linje ritas ut i taget. Man skulle ju lätt kunna tro att man bara flyttar for-loopen till draw-metoden. Men det fungerar tyvärr inte. Det som skulle hända i så fall vore att hela for-loopen körs 60-gånger per sekund. Vi får samma stillbildsmönster ritat ut många gånger men resultatet blir ändå detsamma.

Ska vi nu skapa en animation får vi i stället koden nedan till höger.



Lägg märke till att deklarationen av variabeln `i` hamnar längst upp i koden. Om deklarationen av `i` hade legat i draw-metoden hade den istället skapats på nytt varje gång draw körs och sedan glömts bort när draw avslutas.

Övning 13-1 Övning slumpa ett visst antal animerade cirklar

Fortsätt på övning *Övning slumpade animerade cirklar* ovan men låt användaren bestämma hur många cirklar som ska ritas ut.



Övning 13-2 Antal frimärken 1

Skriv ett program som frågar efter en vikt och sedan svarar med antal frimärken utifrån tabellen nedan:

Max vikt i gram	Antal valörlösa frimärken
50	1
100	2

Om användaren matar in 49:



Om användaren matar in 50:



Antal: 1

Om användaren matar in 51:



Antal: 2

Övning 13-3 Övning stoppa bollen

Börja med att testköra följande program:

```
int speed=10;  
int x=200;  
int y=200;  
  
void setup() {  
    size(400, 400);  
}  
  
void draw() {  
    background(255);  
    x=x+speed;  
    fill(255,0,0);  
    ellipse(x,y,10,10);  
}
```

Som du ser så rör sig en boll åt höger oändligt länge. Försök få bollen att stanna när den kommer till 300.

Övning 13-4 Övning byt håll på bollen

Fortsätt på övningen ovan, men nu ska du få bollen att byta håll istället för att stanna

Övning 13-5 Boll fram och tillbaka

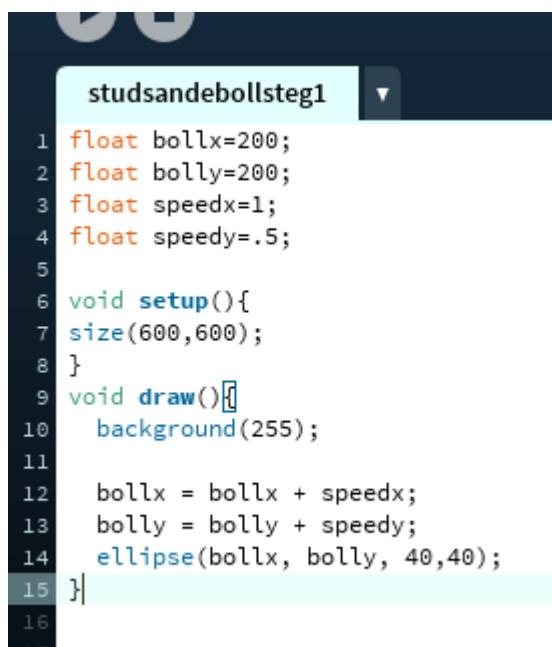
Fortsätt på Boll som byter riktning, men denna gång ska du få den att vända håll igen när bollen kommer till 200

13.2 Skapa en boll som rör sig

Ett sätt att använda tidsstyrda iterationen är att skapa en boll som rör sig över skärmen. Till detta behöver vi fyra variabler. Två för bollens position och två för bollens hastighet i x respektive y-led:

```
float bollx=200;  
float bolly=200;  
float speedx=1;  
float speedy=.5;
```

Sedan behöver vi i draw funktionen, bara lägga till speedx, och speedy varje gång draw körs. Lägg märke till att jag också har lagt till att bakgrunden målas vit varje gång draw körs!



The screenshot shows the Processing IDE with the sketch name 'studsandebollsteg1'. The code is as follows:

```
1 float bollx=200;  
2 float bolly=200;  
3 float speedx=1;  
4 float speedy=.5;  
5  
6 void setup(){  
7 size(600,600);  
8 }  
9 void draw(){  
10 background(255);  
11  
12 bollx = bollx + speedx;  
13 bolly = bolly + speedy;  
14 ellipse(bollx, bolly, 40,40);  
15 }  
16
```

Som ni kanske märker så är ett problem att bollen efter ett tag åker utanför fönstret. Nästa steg är att fixa detta med hjälp av if-satser.

13.3 Skapa en studsande boll

När vi skapade programmet ovan med bollen som rörde på sig så åkte den ganska snart ut genom högersidan på fönstret. Vi ska nu använda en if-sats för att få den att studsa på höger sida. I Processing finns en inbyggd variabel `width` som innehåller värdet på bredden. Det vi behöver kolla är alltså om `bollx` är större än `width`.

```
if(bollx >= width) {  
    Byt riktning  
}
```

Men vad innebär det att byta riktning i programmeringstermer. Eftersom det är x-värdet som är för stort så är det detta som vi måste göra något åt, och det gör vi genom att ändra hastigheten i x-led. Den ska vara negativ för att få bollen att gå åt andra hållet. Men storleken ska vara lika stor, dvs speedx=-speedx ;

```
if(bollx >= width) {  
    speedx=-speedx;  
}
```

Hela programmet blir då. (Jag har också gett bollen en fin blå färg)

```
studsandebollsteg2  
1 float bollx=200;  
2 float bolly=200;  
3 float speedx=1;  
4 float speedy=.5;  
5  
6 void setup(){  
7 size(600,600);  
8 }  
9 void draw(){  
10 background(255);  
11  
12 bollx = bollx + speedx;  
13 bolly = bolly + speedy;  
14  
15 if(bollx>=width){  
16     speedx=-speedx;  
17 }  
18 fill(0,0,255);  
19 ellipse(bollx, bolly, 40,40);  
20 }  
21 }
```

Problemet är nu istället att bollen åker ut nedtill istället, men det blir er uppgift att fixa:

13.3.1 Övning Studsande boll

- Som ni märkte så försvann bollen ut ur skärmen nedtill istället. Åtgärda det så att bollen studsar på nedersidan också. Det vill säga när bolly är större än height.
- Har ni klarat A-uppgiften försvinner fortfarande bollen ut ur bild så småningom. Fixa så att bollen studsar på alla sidor.
- Nu ska hastigheten öka varje gång som bollen studsar, så att hastigheten i x-led ökar när bollen studsar på höger och vänster kant och hastigheten i y-led ökar när den studsar på övre och nedre kant.

13.3.2 Övning Labyrint

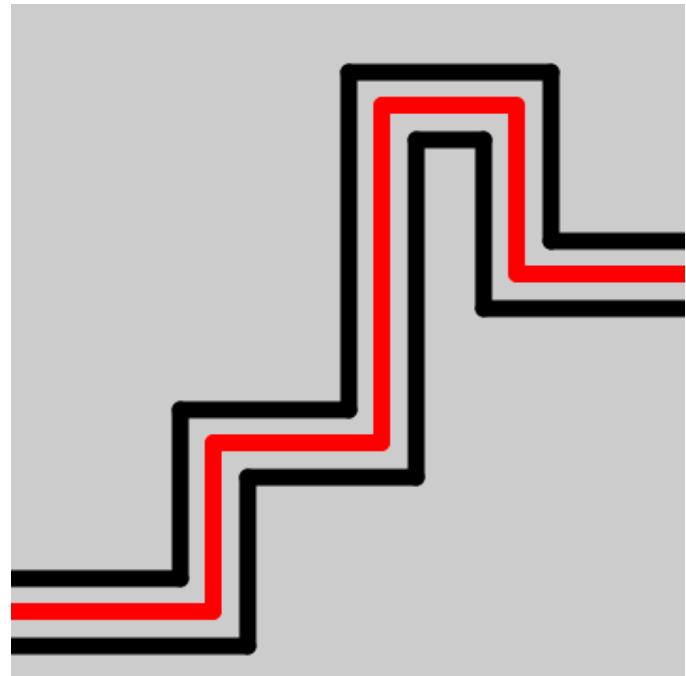
Använd koden nedan för att testköra exemplet. Gör klart programmet så att den röda masken tar sig igenom hela banan, så att slutbilden blir som bilden nedan. Alla x och y-värden som masken färdas över är jämna 10-tal.

```
int speedx=1;
int speedy=0;
int x=0;
int y=360;

void setup() {
    size(400, 400);
    strokeWeight(10);
    line(0, 380, 140, 380);
    line(0, 340, 100, 340);
    line(100, 340, 100, 240);
    line(100, 380, 140, 280);
    line(100, 240, 200, 240);
    line(140, 280, 240, 280);
    line(200, 240, 200, 40);
    line(240, 280, 240, 80);
    line(200, 40, 320, 40);
    line(240, 80, 280, 80);
    line(280, 80, 280, 180);
    line(320, 40, 320, 140);
    line(280, 180, 400, 180);
    line(320, 140, 400, 140);
}

void draw() {
    if (x==120 && y==360) {
        speedy=-1;
        speedx=0;
    }

    x=x+speedx;
    y=y+speedy;
    fill(255, 0, 0);
    noStroke();
    ellipse(x, y, 10, 10);
}
```



13.4 Tabell: Jämförelseoperatorer

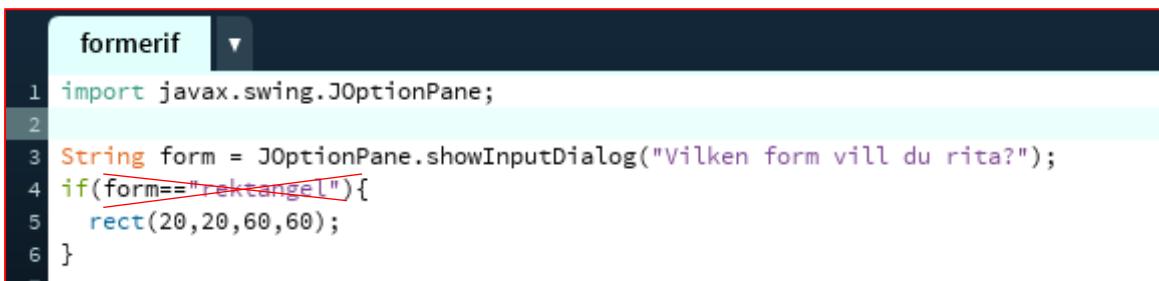
I exemplen ovan har vi använt olika jämförelseoperatorer till exempel < (mindre än), <= (mindre än eller lika med), för utföra jämförelserna i våra if-satser. Nedan ser du en tabell med alla dess operatorer som kan användas för jämförelse både i if-satser och while-satser, och för den delen och i for-satser, även om vissa operatorer som != och == inte är lika vanliga i for-satser.

Operator	Beskrivning	Exempel	prioritet
==	liko med	if(tal1==10)	5
!=	inte lika med	if(tal1!=10)	5
<	mindre än	if(x<width)	4
>	Större än	if(x>0)	4
<=	Mindre än eller lika med	if(x<=width)	
>=	Större än eller lika med	if(x>=0)	

13.5 if-else-satser och if-satser med strängar

Ibland vill vi jämföra två stycken strängar i en if-sats eller while-sats. Tyvärr kan vi inte jämföra strängar på samma sätt som vi kan jämföra tal.

Om vi tänker oss att vi vill skapa ett program som frågar efter vilken form vi vill rita ut och sedan ritar ut en rektangel om man skriver rektangel, så skulle det känna naturligt att skriva:



```
formerif ▼
1 import javax.swing.JOptionPane;
2
3 String form = JOptionPane.showInputDialog("Vilken form vill du rita?");
4 if(form=="rektagel"){
5     rect(20,20,60,60);
6 }
```

The code editor shows a red box highlighting the comparison line 'if(form=="rektagel")'. The word 'form' is also underlined in red, indicating it is being compared to the string 'rektagel'.

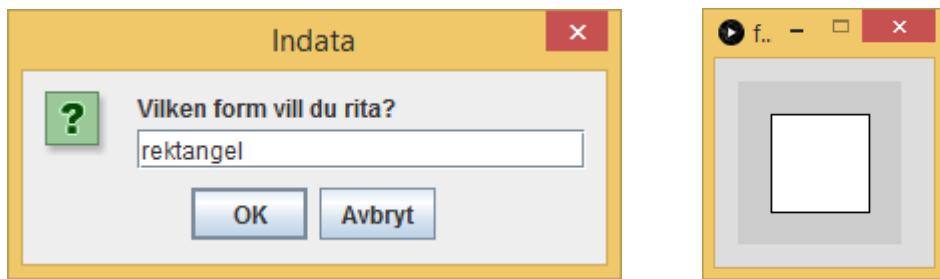
Detta kommer tyvärr inte att fungera eftersom det som jämförs är två minnesadresser. Det är nämligen så att variabeln `form` inte innehåller själva texten utan enbart information om var i minnet som texten finns.

Istället skriver vi `.equals` efter variabelnamnet så kommer `equals` att göra en jämförelse mellan de verkliga textsträngarna.

```

formerif
1 import javax.swing.JOptionPane;
2
3 String form = JOptionPane.showInputDialog("Vilken form vill du rita?");
4 if(form.equals("rekktangel")){
5     rect(20,20,60,60);
6 }

```



13.6 else

Om vi tittar igen på exemplet ovan så är det kanske inte så jättebra. Skriver vi *inte* exakt "rekktangel", så skrivs inte någonting ut alls. Jag skulle nu vilja att om användaren *inte* skriver in rekktangel så ska alltid en cirkel alltid skrivas ut.

Till detta behöver jag en *else*-sats.

If-else-satser generellt:

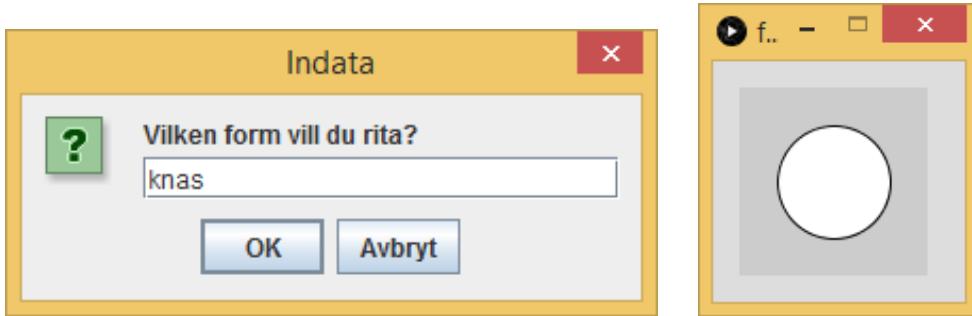
```

if(villkor) {
    Satser som ska köras om villkoret är uppfyllt
    .....
}
else {
    Satser som ska köras om villkoret inte är uppfyllt
    .....
}

```

Jag har alltså möjlighet att välja vad som ska hända när villkoret *inte* är uppfyllt, och

```
formerif | ▼  
1 import javax.swing.JOptionPane;  
2  
3 String form = JOptionPane.showInputDialog("Vilken form vill du rita?");  
4 if(form.equals("rektagel")){  
5     rect(20,20,60,60);  
6 }  
7 else {  
8     ellipse(50,50,60,60);  
9 }  
10
```

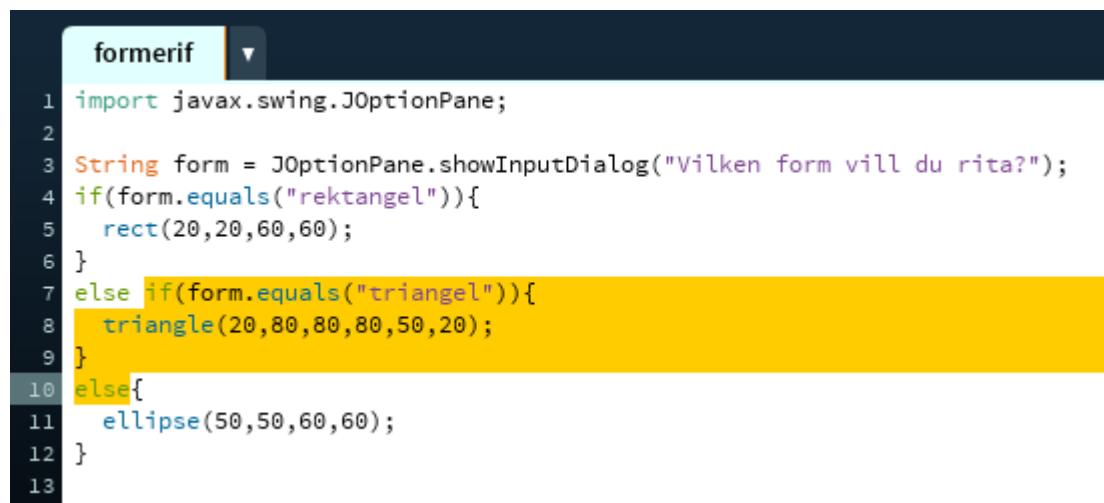


13.8 En kedja med flera if-satser

I bland vill testa flera olika saker efter varandra. Då staplar man ofta flera if-satser efter varandra.

```
if(villkor 1) {  
    Satser som ska köras när villkor 1 är uppfyllt  
    .....  
}  
else if(villkor 2) {  
    Satser som ska köras när villkor 2 är uppfyllt  
    .....  
}  
else {  
    Satser som ska köras när varken villkor 1 eller villkor 2 är  
    uppfyllt.  
    .....  
}
```

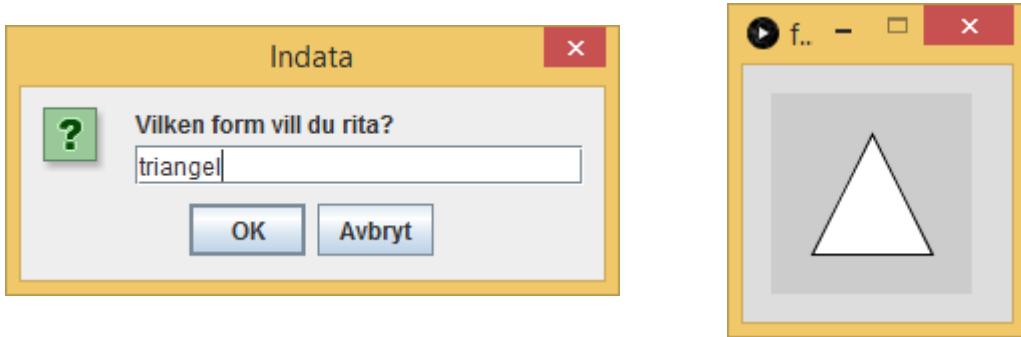
I vårt fall så kommer jag att skjuta in en till if-else-sats direkt efter den befintliga else satsen.



The screenshot shows a Java code editor with the following code:

```
formerif ▾  
1 import javax.swing.JOptionPane;  
2  
3 String form = JOptionPane.showInputDialog("Vilken form vill du rita?");  
4 if(form.equals("rektagel")){  
5     rect(20,20,60,60);  
6 }  
7 else if(form.equals("triangel")){  
8     triangle(20,80,80,80,50,20);  
9 }  
10 else{  
11     ellipse(50,50,60,60);  
12 }  
13
```

The code uses nested if statements to draw different shapes based on user input. The first if statement checks for "rektagel" (rectangle). The second if statement, highlighted in yellow, checks for "triangel" (triangle). The else block contains an ellipse drawing.



13.9 Logiska operatorer

Det finns tre stycken logiska operatorer som kan vara praktiska att använda tillsammans med if-satser.

Operator	Namn	Beskrivning	Exempel
&&	och	uttryck1 && uttryck2 Hela uttrycket är sant om både uttryck1 och uttryck2 är sanna	if(x>0 && x<width){
	eller	uttryck1 uttryck2 Hela uttrycket är sant om uttryck1 eller uttryck2 är sanna	if(x<0 x>width){
!	icke	!utryck1 Hela uttrycket är sant om uttryck1 är sant	if(!(x>0 && x < width)){

Eller operatorn kan till exempel användas om man vill byta håll när bollen antingen har nått vänster eller höger sida. I pseudokod blir det:

*Om bollen har nått höger sida eller bollen har nått vänster sida
Byt riktning*

Och i Processingkod:

```
if(bollx >= width || bollx <= 0) {
    speedx=-speedx;
}
```

13.10 Exempel logiska operatorer. Vem får åka Balder?

Nu kommer ett exempel på logiska operatorer. Om ni har gått på Liseberg någon gång så har ni sett att det finns olika slags krav för att man ska få åka en viss attraktion. Ska man till exempel åka Balder måste man vara 130 centimeter lång och vara minst 7 år gammal. Vi skriver ett program som kan avgöra om en viss person får åka Balder.

Vi har alltså två villkor:

längd >= 180 och alder >= 7

Uttrycket ovan kan vi nästan direkt omvandla till processingkod. Vi behöver bara byta ut ordet "och" med två stycken &-tecken (och sen tar jag bort de svenska tecknena), och får då:

langd >= 130 && alder >= 7

Vi stoppar sedan in uttrycket i en if-sats. Ett helt program kan då se ut så här:

```
balderlogiskaoperatorer ▾
1 import javax.swing.JOptionPane;
2 background(0);
3 int langd = int(JOptionPane.showInputDialog("Ange din längd i cm"));
4 int alder = int(JOptionPane.showInputDialog("Ange din ålder"));
5 if(langd >= 130 && alder >= 7){
6   text("Du får åka Balder",10,20,80,80);
7 }
8 else {
9   text("Du får inte åka Balder",10,20,80,80);
10 }
```

Vad händer då om man istället byter ut && mot ||? Då kommer villkoret att se ut så här:

langd >= 130 || alder >= 7

Det skulle motsvara

längd >= 180 eller alder >= 7

Det skulle i så fall innebära att man enbart skulle behöva uppfylla ett av kraven. Det räcker med att man är äldre än sju år eller längre än 130 centimeter, för att få åka med. Det skulle kunna bli riktigt farligt om vi lät en sjuåring som bara är 110 centimeter åka med, som troligtvis inte skulle sitta fast ordentligt. Det är alltså viktigt att inte blanda ihop && och ||.

Övning 13-6 Övning Studsande boll fortsättning

- D. Snygga nu till övningen studsande boll ovan så att du ändrar dina fyra if-satser i övningen till två med hjälp av eller-operatorn

13.11 Inbyggda variabler som kan användas med if-satser

Nu när vi har lärt oss if-satser kan vi börja att använda oss av musen och tangentbordet för att påverka programmet. Processingsystemet har ett antal variabler som uppdateras av systemet mellan varje gång som `draw()` körs.

Beskrivning	Variabelnamn	Värden som variabeln tilldelas av systemet	Exempel
Kontrollera om musen är nedtryckt	<code>mousePressed</code>	<i>true</i> : om musknappen är nedtryckt <i>false</i> : annars	<code>if(mousePressed){ ellipse(mouseX,mouseY,20,20); }</code>
Är någon tangent nedtryckt?	<code>keyPressed</code>	<i>true</i> : om någon tangent är nedtryckt <i>false</i> : annars	<code>if(keyPressed){ text(key,20,20); }</code>
Tecknet från nedtryckt tangent	<code>key</code>	Innehåller tecknet som den nedtryckta tangenten motsvarar.	<code>if(keyPressed){ text(key,20,20); }</code>
Koden för specaltangenter	<code>keyCode</code>	Innehåller koden för den nedtryckta specaltangenten. exempelvis: UP, DOWN, LEFT, RIGHT.	<code>if(keyCode==UP){ bolly--; }</code>
Musmarkörens x-position	<code>mouseX</code>	Musmarkörens x-position	<code>ellipse(mouseX, mouseY, 3, 3);</code>
Musmarkörens y-position	<code>mouseY</code>	Musmarkörens y-position	<code>ellipse(mouseX, mouseY, 3, 3);</code>

13.12 Exempel: ritprogram

Nu kan vi använda Processings inbyggda variablerna ovan för att skapa ett enkelt ritprogram. Det är bara att kolla om musknappen är nedtryckt och i så fall, rita en liten cirkel på den platsen. Med pseudokod blir det:

Om musknappen är nedtryckt

Rita en cirkel där musmarkören står

Det vill säga:

```
if (mousePressed) {  
    ellipse(mouseX, mouseY, 3, 3);  
}
```

Hela programmet blir:

```
ritprogsteg1 ▾  
1 void setup() {  
2     size(600, 600);  
3     fill(0);  
4 }  
5  
6 void draw() {  
7     if (mousePressed) {  
8         ellipse(mouseX, mouseY, 3, 3);  
9     }  
10 }
```

Vi vill också kunna spara. Vi lägger till att programmet ska spara bilden när vi trycker på s-tangenten.

Om någon tangent är nedtryckt

Om s är nedtryckt

Spara

```
if (keyPressed) {  
    if (key=='s') {  
        save("teckning.png");  
    }  
}
```

Hela programmet blir då:

```
ritprogsteg2 ▾  
1 void setup() {  
2     size(600, 600);  
3     fill(0);  
4 }  
5  
6 void draw() {  
7     if (mousePressed) {  
8         ellipse(mouseX, mouseY, 3, 3);  
9     }  
10    if (keyPressed) {  
11        if (key=='s') {  
12            save("teckning.png");  
13        }  
14    }  
15 }
```

Övning 13-7 Övning ritprogram

- A) När man trycker på knappen r så ska färgen bytas till röd
- B) När man trycker g, b, y, så ska programmet börja måla med grön, blå respektive gul.
- C) Om du hinner kan du göra fler tillägg, till exempel kan du låta användaren rita med en fyrkant istället för en cirkel.

Övning 13-8 Övning antal frimärken 2

Fortsätt på övningen *Antal frimärken 1* men den här gången med den kompletta portotabellen nedan:

Max vikt i gram	Antal valörlösa frimärken
50	1
100	2
250	4
500	6
1 000	8
2 000	10

Exempel på testfall:

Om användaren matar in 29:



Om användaren matar in 250:



Om användaren matar in 1500:

Antal: 10

13.13 Studsande boll med gravitation

Nu skulle vi vilja ha en boll som studsar på liknande sätt som en studsboll. Vi tänker oss att den har kastats åt höger vilket gör att hastigheten i x-led är positiv i början. Vi sätter hastigheten till 1 till att börja med. Hastigheten i y-led är noll från början. Bollen börjar på koordinaterna 200,100. Vi skapar också en variabel för bollens färg. Vi sätter ritytans storlek till 500,500

```
gravity1 ▾
1 int bollx=200;
2 int bolly=100;
3 int speedy=0;
4 int speedx=1;
5 color rod = color(255,0,0);
6
7 void setup() {
8   size (500, 500);
9 }
```

Sedan vill vi få bollen att rita ut sig och röra på sig. Detta sker i draw-funktionen. Vi ritar en svart bakgrund. Ritar ut en röd boll, och sedan flyttas bollen med det värde som anges i speedx, och speedy.

```
11 void draw() {
12   background(0);
13   fill(rod);
14   ellipse(bollx, bolly, 30, 30);
15   bollx+=speedx;
16   bolly+=speedy;
17 }
```

Nu kan vi testköra.

Vi har nu en boll som rör sig långsamt åt höger. Nu ville vi att den skulle studsa som en studsboll. Vi får börja med att få bollen att åka nedåt på ett sätt som liknar en studsboll. Det vill säga först långsamt och sedan snabbare.

Att en vanlig boll som man kastar åt höger beter sig som den gör beror på att vi har gravitation. I vårt program får simulera gravitationen. Gravitationen innebär att allting accelererar nedåt om det inte finns en motkraft. Vi inför en variabel för accelerationen och ger den värdet 0.1.

```
gravity1 ▾
1 int bollx=200;
2 int bolly=100;
3 float speedy=0;
4 float speedx=1;
5 float accy=.1;
6 color rod = color(255,0,0);
7
```

Acceleration innebär att hastigheten ökar proportionellt med accelerationen över tid. Därför så ökar vi speedy med värdet för accelerationen varje gång som draw-funktionen körs, se nedan.

```
12 void draw() {
13   background(0);
14   fill(rod);
15   ellipse(bollx, bolly, 30, 30);
16   bollx+=speedx;
17   bolly+=speedy;
18   speedy+=accy;
19 }
20
```

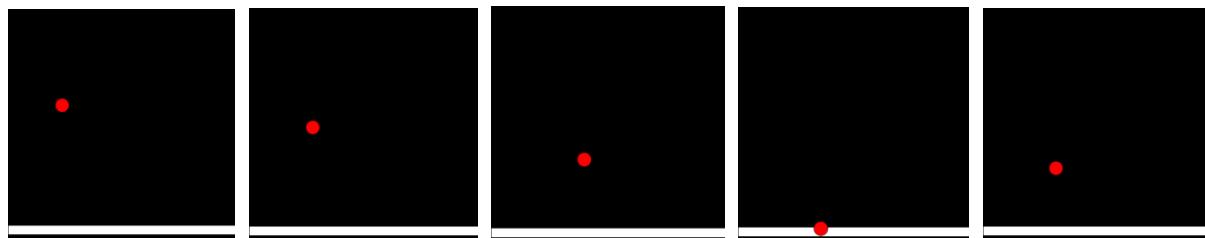
Testkör!

Nu ser vi att bollen faller på ett ganska naturligt sätt. Vi vill också att den ska studsa. Till skillnad från förra gången när vi använde bollens koordinater för att avgöra när bollen skulle studsa, så tänkte ja denna gången använda bakgrundsfärgen. Vi låter bollen studsa upp när den stöter på en vit färg. Vi målar därför en vit rektangel längst ner på bilden. Men vi börjar med att skapa en variabel för rektangelns färg.

```
5 float accy=.1;
6 color rod = color(255, 0, 0);
7 color vit = color(255);
8
9 void setup() {
...
13 void draw() {
14   background(0);
15
16   fill(vit);
17   rect(0, 470, 500, 20);
18 }
```

Direkt efter vi har ritat ut rektangeln så kontrollerar vi om bollen är i det vita området. Det gör vi genom att kolla färgen på den plats som pekas ut av bollens koordinater. Det finns en funktion `get(bollx, bolly)` som ger oss färgen på en punkt. Om punkten är vit så byter vi tecken på bollens hastighet i y-led.

```
13 void draw() {  
14     background(0);  
15  
16     fill(vit);  
17     rect(0, 470, 500, 20);  
18     if(get(bollx, bolly)==vit){  
19         speedy=-speedy;  
20     }  
21 }
```



Nu skulle det vara roligt att göra det här till något som mer liknar ett spel. Det gör vi enklast genom att göra rektangeln smalare och inför också en variabel för rektangelns x-koordinat, och styr sedan variabeln med vänster och höger piltangent.

```
5 float accy = 0.1;  
6 int rectx = 200;  
7 color rod = color(255,0,0);  
  
14 void draw() {  
15     background(0);  
16  
17     fill(vit);  
18     if (keyCode==LEFT) {  
19         rectx=rectx-10;  
20     }  
21     if (keyCode==RIGHT) {  
22         rectx=rectx+10;  
23     }  
24     rect(rectx, 470, 100, 20);  
25  
26     if (get(bollx, bolly)==vit) {  
27         speedy=-speedy;  
28     }  
29  
30     fill(rod);
```

Nu fungerar det i stort sett. Men bollen åker visst vidare ut ur bild bara man nuddar vänster eller höger pil. Vi lägger till en rad till för att kolla att en tangent verkligen är nedtryckt.

```
14 void draw() {  
15   background(0);  
16  
17   fill(vit);  
18   if (keyPressed) {  
19     if (keyCode==LEFT) {  
20       rectx=rectx-10;  
21     }  
22     if (keyCode==RIGHT) {  
23       rectx=rectx+10;  
24     }  
25   }  
26   rect(rectx, 470, 100, 20);  
27  
28   if (get(bollx, bolly)==vit) {  
29     speedy=-speedy;  
30   }  
31 }
```

Nu har vi ett spel som nästan fungerar. Nu är det er uppgift att göra det spelbart.

Övning 13-9 Gravitationsspel

- Fixa studs mot kant. Gör så att bollen inte åker utanför spelplanen för att den nuddar vänster eller höger sida, utan den istället studsar tillbaka.
- Lägg till poängräkning. Varje gång bollen studsar på rektangeln så ska poängen räknas upp med ett. Poängen skastå längst upp i vänstra hörnet.
- Du kanske har lagt märke till att bollen inte studsar förrän den har kommit till mitten av rektangeln. Det beror på att vi kollar om mitten på bollen är i den vita rektangeln. Ändra så att bollen studsar när den nedersta delen av bollen är i den vita rektangeln.

13.14 * Hur gör vi för att bollen inte ska studsa högre och högre?

Om ni får problem att bollen hela tiden studsar högre och högre så beror det på att vår modell för gravitationen är lite för enkel. I verkligheten så ökar hastigheten hela tiden, medan när vi jobbar i Processing så ökar den enbart 60 gånger per sekund. Det betyder att när vi ökar y-värdet med hastigheten så ökar vi lite för lite. Man skulle kunna tycka att det inte skulle ha så stor betydelse eller att det skulle jämma ut sig, men det gör det uppenbarligen inte. För att få det att fungera korrekt så behöver vi lägga till accelerationen delat med två varje gång vi ökar y-positionen med hastigheten.

```
35   bolly+=speedy + accy/2;  
36   speedy+=accy;
```

Vi måste också ändra variabeltypen på bollx och bolly till float annars så har ovanstående ändring ingen betydelse eftersom den kommer att försvinna i avrundningen.

```
1 float bollx=100;
2 float bolly=200;
3 float speedy=0:
```

När vi gör det så måste vi också ändra vår studsdetektering, så att vi omvandlar bollx och bolly till int direkt när vi anropar get-metoden.

```
27
28     if (get(int(bollx), int(bolly))==vit) {
29         speedy=-speedy;
30     }
31
```

13.15 * Hur hanterar man flera knapptryckningar samtidigt?

I bland vill vi kunna köra två spelare samtidigt på ett spel. För att ta ett enkelt exempel så har jag utgått från studsande boll med gravitation som vi gjorde ovan. Men tänker nu göra tillägg som gör att man kan styra bollen med 'a' repspektive 's'. På det sättet kan ena spelaren styra rektangeln där nere, och den andra spelaren kan styra bollen. Jag lägger därför till styrningen av bollen i koden genom två ytterligare if-satser.

```
17     fill(vit);
18     if (keyPressed) {
19         if (keyCode == LEFT) {
20             rectx = rectx-10;
21         }
22         if (keyCode == RIGHT) {
23             rectx = rectx+10;
24         }
25         if(key == 'a'){
26             bollx -= 5;
27         }
28         if(key == 'w'){
29             bollx += 5;
30         }
31     }
32 }
```

Lösningen ovan fungerar i princip, men man får problem om två tangenter trycks ned samtidigt.

För att lösa det problemet så kan man använda funktionerna keyPressed() , och keyReleased() i kombination med fyra booleanvariabler, det vill säga en för varje knapp

som vi vill hålla reda på. Respektive booleanvariabel ska sedan hålla reda om knappen är nedtrycket eller ej. Vi börjar med att skapa fyra boolean variabler.

```
7 color rod = color(255, 0, 0);
8 color vit = color(255);
9 boolean aDown = false;
10 boolean sDown = false;
11 boolean leftDown = false;
12 boolean rightDown = false;
13
14 void setup() {
```

Sedan lägger vi till funktionerna `keyPressed()` och `keyReleased()`. Funktionerna `keyPressed()` och `keyReleased()` fungerar på liknande sätt som `draw()`-funktionen. Men istället för att köras 60 gånger per sekund så körs de när en knapp har tryckts ned respektive en knapp har släppts upp.

```
54 void keyPressed() {
55
56 }
57
58 void keyReleased() {
59
60 }
```

Nästa steg är att fylla dessa funktioner med innehåll. Vi ska använda funktionerna till att sätta korrekta värden på boolean variablerna som vi skapade tidigare. När `keyPressed()` körs så har vi en if-sats för respektive knapp som vi är intresserad av, och om den knappen är nedtryck så sätter vi motsvarande variabel till `true`.

```
54 void keyPressed() {
55   if (keyCode == LEFT) {
56     leftDown = true;
57   }
58   if (keyCode == RIGHT) {
59     rightDown = true;
60   }
61   if (key == 'a') {
62     aDown = true;
63   }
64   if (key == 's') {
65     sDown = true;
66   }
67 }
```

På motsvarande sätt ser vi till att booleanvariablerna blir `false` när respektive knapp släpps upp.

```

69 void keyReleased() {
70   if (keyCode == LEFT) {
71     leftDown = false;
72   }
73   if (keyCode == RIGHT) {
74     rightDown = false;
75   }
76   if (key == 'a') {
77     aDown = false;
78   }
79   if (key == 's') {
80     sDown = false;
81   }
82 }
```

Nu återstår det bara att ändra i if-satserna som ändrar x-värdena på bollen beroende på vilka knappar som är nedtryckta. Vi behöver nu inte kolla om någon knapp är nedtryckt, och vi behöver inte ens ha en jämförelserna i if-satserna eftersom booleanvariablerna redan innehåller ett logiskt värde.

```

17 fill(vit);
18 if (keyPressed) {
19   if (keyCode == LEFT) {
20     rectx = rectx-10;
21   }
22   if (keyCode == RIGHT) {
23     rectx = rectx+10;
24   }
25   if(key == 'a'){
26     bollx -= 5;
27   }
28   if(key == 'w'){
29     bollx += 5;
30   }
31 }
32 }
```



```

21 fill(vit);
22 if (leftDown) {
23   rectx = rectx-10;
24 }
25 if (rightDown) {
26   rectx = rectx+10;
27 }
28 if (aDown) {
29   bollx -= 10;
30 }
31 if (sDown) {
32   bollx += 10;
33 }
34 }
```

13.16 Enkelt racerspel med Switch Case

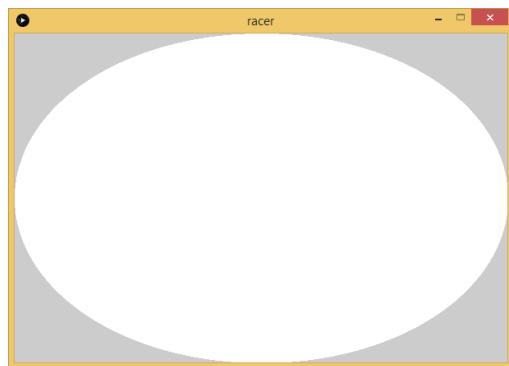
Vi börjar med att rita ut en bana. Detta gör vi genom att enklast rita en stor oval

```
ellipse(300,200,540,340);
```

med bredden 60:

```
strokeWeight(60)
```

```
8 void draw() {  
9   stroke(255);  
10  strokeWeight(60);  
11  ellipse(300,200,540,340);  
12 }
```



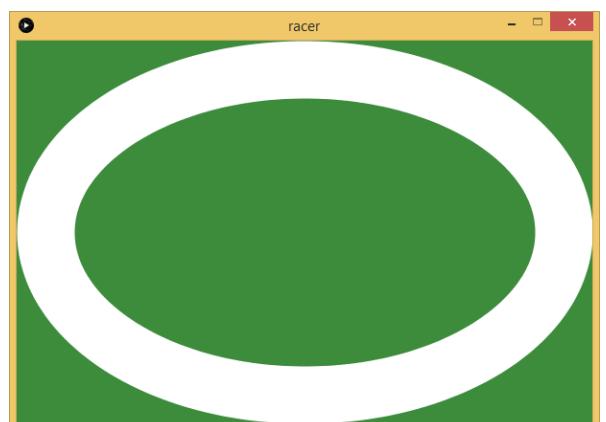
Men vi vill att bakgrunden och fyllningen ska vara gröna. vi börjar med att skapa en variabel för färgen grön, och placeras den först i hela programmet.

```
color gron = color(60,140,60);
```

Vi behöver sedan ange både bakgrunden och fyllnadsfärgen till grön

```
background(gron);  
fill(gron);
```

```
racer ▾  
1 color gron = color(60,140,60);  
2  
3 void setup() {  
4   size(600, 400);  
5 }  
6  
7 void draw() {  
8   background(gron);  
9   fill(gron);  
10  stroke(255);  
11  strokeWeight(60);  
12  ellipse(300,200,540,340);  
13 }
```

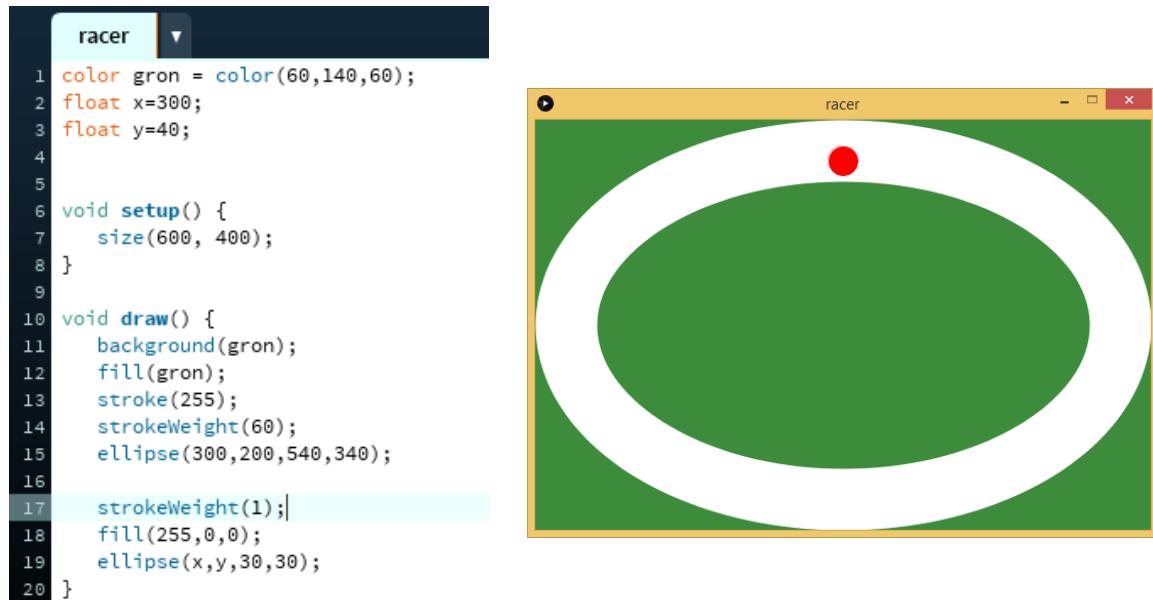


Nu vill vi skapa en liten bil som går att styra. Till att börja med så kan bilen symboliseras av en cirkel. Vi behöver variabler för variablerna x och y.

```
float x=300;
float y=40;
```

I draw metoden ska vi sedan rita ut cirkeln(bilen). Vi måste först sätta tillbaka strokeWidth till 1 och sätta fyllnadsfärgen.

```
strokeWeight(1);
fill(255,0,0);
ellipse(x,y,30,30);
```



```
racer
1 color gron = color(60,140,60);
2 float x=300;
3 float y=40;
4
5
6 void setup() {
7     size(600, 400);
8 }
9
10 void draw() {
11     background(gron);
12     fill(gron);
13     stroke(255);
14     strokeWeight(60);
15     ellipse(300,200,540,340);
16
17     strokeWeight(1);
18     fill(255,0,0);
19     ellipse(x,y,30,30);
20 }
```

Nästa steg är att styra bilen. Vi skapar till att börja med en if-sats innan vi ritar ut cirkeln.

```
if(keyPressed) {
    if(keyCode==RIGHT) {
        x=x + 5;
    }
}
```



```
racer
6 void setup() {
7     size(600, 400);
8 }
9
10 void draw() {
11     background(gron);
12     fill(gron);
13     stroke(255);
14     strokeWeight(60);
15     ellipse(300, 200, 540, 340);
16     if (keyPressed) {
17         if (keyCode==RIGHT) {
18             x=x + 5;
19         }
20     }
21
22     strokeWeight(1);
23     fill(255, 0, 0);
24     ellipse(x, y, 30, 30);
25 }
```

vi lägger sedan också till if-satser för vänster knapp

```
racer | ▾  
color gron = color(60, 140, 60);  
float x=300;  
float y=40;  
  
void setup() {  
    size(600, 400);  
}  
  
void draw() {  
    background(gron);  
    fill(gron);  
    stroke(255);  
    strokeWeight(60);  
    ellipse(300, 200, 540, 340);  
    if (keyPressed) {  
        if (keyCode==RIGHT) {  
            x=x + 5;  
        }  
        else if(keyCode==LEFT){  
            x = x - 5;  
        }  
        else if(keyCode==DOWN){  
            y = y + 5;  
        }  
        else if(keyCode==UP){  
            y = y - 5;  
        }  
    }  
  
    strokeWeight(1);  
    fill(255, 0, 0);  
    ellipse(x, y, 30, 30);  
}
```

Men nu finns det ett annat sätt att göra flera jämförelser med samma variabel: Switch - case.
Du kan få fram den grundläggande strukturen för switch-case genom att skriva switch och sedan trycka *ctrl+mellanslag* (om du har *Extended Code Completion* aktiverat)

<pre>16 if (keyPressed) { 17 if (keyCode==RIGHT) { 18 x=x + 5; 19 } 20 else if(keyCode==LEFT){ 21 x = x - 5; 22 } 23 else if(keyCode==DOWN){ 24 y = y + 5; 25 } 26 else if(keyCode==UP){ 27 y = y - 5; 28 } 29 }</pre>	<pre>16 if (keyPressed) { 17 switch (keyCode) { 18 case LEFT: 19 x=x + 5; 20 break; 21 case RIGHT: 22 x = x - 5; 23 break; 24 case DOWN: 25 y = y + 5; 26 break; 27 case UP: 28 y = y - 5; 29 break; 30 }</pre>
--	--

Sedan är det enbart själv detekteringen av kraschen kvar.

Övning 13-10 Övning Racerspel

Övningen är helt enkelt att göra färdigt och förbättra racerspelet. Några förslag:

- 1) Detektera krasch, och stanna hela spelet om man kraschar in i det gröna. Använd förslagsvis funktionen `get(x,y)` som vi använde i gravitationsspelet ovan. Till att börja med kan det räcka med att detektera när mittpunkten är utanför. För att få hela spelet att stanna kan man använda `noLoop()`
- 2) Detekterar när bilen har kört ett varv
- 3) Försök att mäta tiden som det tar för bilen att köra ett varv.
- 4) Försök göra kollisionsdetekteringen perfekt så du kollar flera punkter mot `get()`-funktionen.

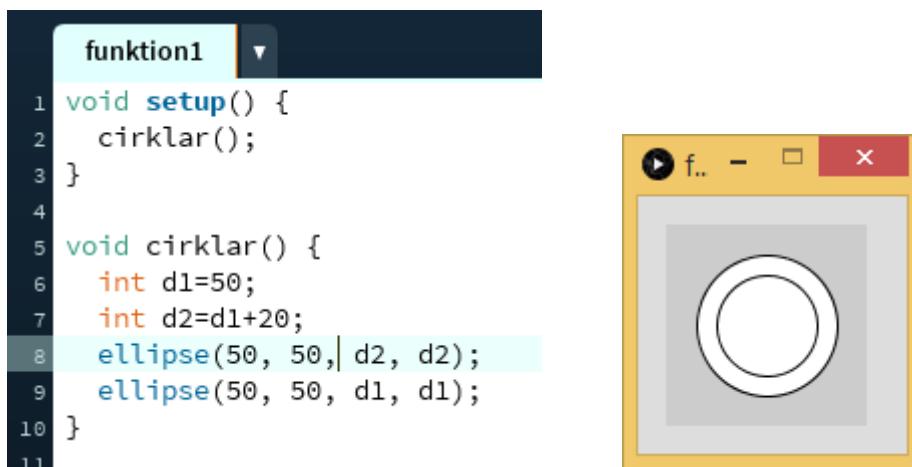
15 Funktioner

De flesta moderna programspråk ger programmeraren en möjlighet att bunta ihop en mängd satser och köra dem tillsammans. Det finns olika namn på detta i olika programmeringsspråk, men i Processing kallas det funktioner eller metoder.

I avsnittet om variabler ovan så hade vi ett exempel där vi ritade ut cirkel i cirkel. Det såg ut så här:

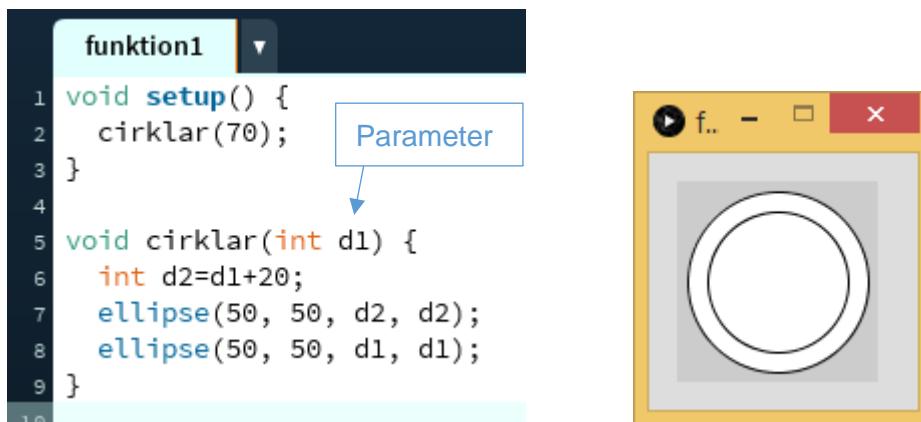
```
int d1=50;  
int d2=d1+20;  
ellipse(50, 50, d2, d2);  
ellipse(50, 50, d1, d1);
```

Den enklaste funktionen vi kan göra av det här är att stoppa raderna ovan i en funktioner som vi kallar för cirklar. Hela programmet blir då:



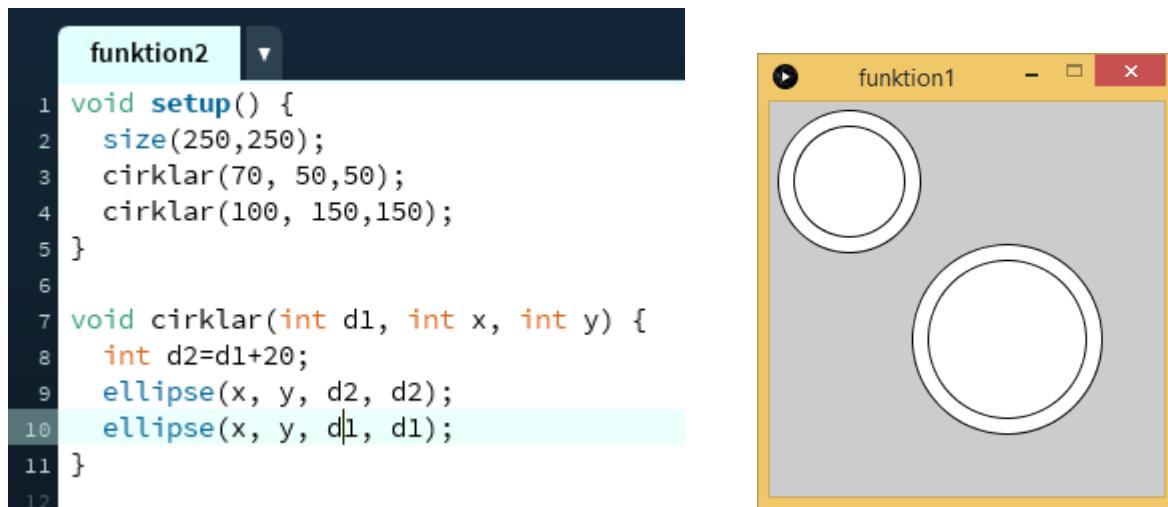
```
funktion1  
1 void setup() {  
2     cirklar();  
3 }  
4  
5 void cirklar() {  
6     int d1=50;  
7     int d2=d1+20;  
8     ellipse(50, 50, d2, d2);  
9     ellipse(50, 50, d1, d1);  
10 }  
11
```

Nu kommer samma sak hända varje gång vi kör funktionen. Men man kanske vill kunna ändra storleken på cirkeln när man kör funktionen. Då behöver vi en parameter till funktionen. En parameter (eller inparameter) är ett värde som vi skickar in i funktionen när den körs. Funktionen kan då bete sig olika vid olika tillfällen när den körs, till exempel rita ut cirklar med olika storlekar. Inuti funktionen så använder man parametern som en vanlig variabel.



```
funktion1  
1 void setup() {  
2     cirklar(70);  
3 }  
4  
5 void cirklar(int d1) {  
6     int d2=d1+20;  
7     ellipse(50, 50, d2, d2);  
8     ellipse(50, 50, d1, d1);  
9 }
```

Vi kanske också vill rita ut flera olika cirklar på olika ställen. Då behöver vi också ha koordinaterna som parametrar.



```
funktion2
void setup() {
    size(250,250);
    cirklar(70, 50,50);
    cirklar(100, 150,150);
}

void cirklar(int d1, int x, int y) {
    int d2=d1+20;
    ellipse(x, y, d2, d2);
    ellipse(x, y, d1, d1);
}
```

Nu börjar vi också kanske förstå vad vi ska med funktionerna till. En anledning till att använda funktioner är att kunna göra samma sak eller liknande saker många gånger, och vi behöver då bara ändra parametrarna till funktionen, varje gång vi vill göra dessa saker.

Den andra anledningen till att använda funktioner är att ha möjlighet att dela in sin kod i mindre bitar för att göra koden mer överblickbar. Men det blir kanske inte uppenbart i den här kursen eftersom våra program är ganska korta.

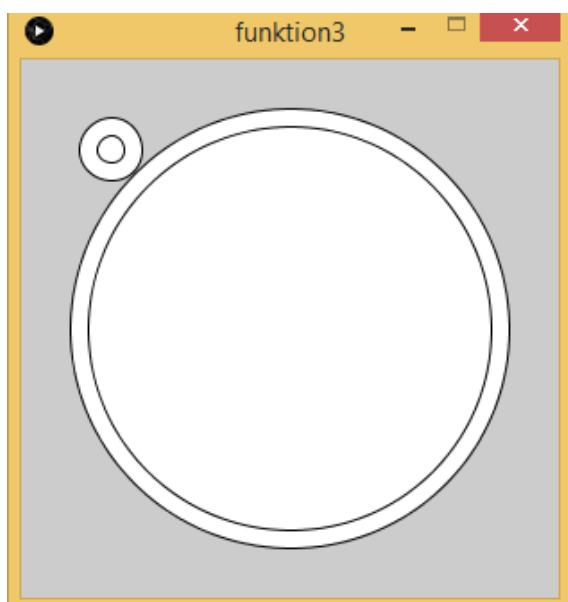
15.1 Funktioner med returvärde

I bland vill vi att en funktion räknar ut någonting åt oss. Ett enkelt exempel skulle kunna vara en funktion som kvadrerar två tal. Funktioner som räknar ut något returnerar ett värde, och den typ som det levererade värdet ska ha ska stå först i funktionsdefinitionen.

Typen på returvärdet
↓
`int kvadrera(int varde){
 int kvadrat = varde * varde;
 return kvadrat;
}`
↑
return skickar tillbaka returvärdet, och avslutar funktionen

Hela programmet skulle då kunna bli:

```
funktion3
1 void setup() {
2   size(300,300);
3   int d=15;
4   int dkvadrat = kvadrera(d);
5   cirklar(d, 50,50);
6   cirklar(dkvadrat, 150,150);
7 }
8
9 int kvadrera(int varde){
10   int kvadrat = varde * varde;
11   return kvadrat;
12 }
13
14 void cirklar(int d1, int x, int y) {
15   int d2=d1+20;
16   ellipse(x, y, d2, d2);
17   ellipse(x, y, d1, d1);
18 }
19 }
```

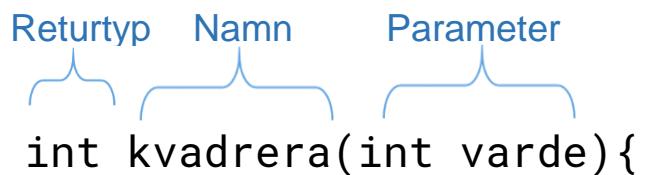


Teori om funktioner

Funktioner används alltså antingen till att göra beräkningar, och blir i detta fall ganska likt en funktion i matematiken, andra exempel från Processing är en funktion för att rita ut en mer komplex form, eller för att exempelvis avgöra om muspekaren är ovanför ett visst objekt.

En funktion har som vi såg i exemplet ovan alltid tre egenskaper

1. *Ett namn*
2. *En returtyp (även om returtypen kan vara "ingen returtyp" alltså void)*
3. *noll, en eller flera parametrar.*



Funktionens namn

Namnet får vara ska börja på en bokstav och kan sedan innehålla bokstäver blandat med siffror samt underscore (_)

Funktionens Returtyp

Returtypen anges antingen med void för att ange att man inte tänker returnera något alls eller så anges typen på värdet som ska returneras till exempel float, int eller String.

Funktionens Parametrar

En parameter är värden som man vill skicka med in till funktionen, och som antingen används för att beräkna returvärde, eller som på annat sätt används för att styra hur funktionen ska utföra sitt syfte.

Exempel 1

Returtyp Namn Parameter

int kvadrera(int varde){

- Returtyp: *int*
- Namn: *kvadrera*
- Parametrar:
 - *varde av typen int (heltal)*

Exempel 2

Returtyp Namn 3 Parametrar

void cirklar(int d1, int x, int y){

- Returtyp: *Ingen returtyp*
- Namn: *cirklar*
- Parametrar:
 - *d1 av typen int (heltal)*
 - *x av typen int (heltal)*
 - *y av typen int (heltal)*

Exempel 3

Returtyp Namn

boolean isMouseOverRect(){

- Returtyp: *boolean*
- Namn: *isMouseOverRect()*
- Parameter: *Ingen*

15.2 Hur vi skapar en funktion med returvärde. Steg för steg.

Hur ska man då tänka när man ska skapa en funktion med returvärde? Här kommer en guide på fyra steg som visar hur man kan skapa ett sklett för en funktion och sedan fylla den med innehåll.

1. Vad ska den hetा? När vi bestämt namnet kan vi skriva detta i vårt program och även passa på att skriva en start och slutparantes, samt en start och slutmåsminge i programmet. I vårt exempel döper vi funktionen till kvadrera.

```
9 kvadrera(){  
10 }
```

2. Vilken typ har värdet som ska skickas tillbaka.
 - a. Skriv den typen före namnet
 - b. Skapa en variabel först i funktionen med den typen
 - c. Skriv return och sedan namnet på variabeln som du skrev under b.

```
9 int kvadrera(){  
10     int kvadrat;  
11     return kvadrat;  
12 }
```

3. Vilka parametrar ska du ha, och vilken typ har de. Skriv typ och namn inom parentesen.

```
9 int kvadrera(int varde) {  
10     int kvadrat;  
11     // Här fyller vi på med resten av funktionen  
12     return kvadrat;  
13 }
```

Nu har vi skapat vårt funktionssklett. Nästa steg är nu att fylla på med själva uträkningen. Den börjar normalt på raden efter vi skapat vår returvaraibel. I detta fall så räcker det med en rad.

```
9 int kvadrera(int varde) {  
10     int kvadrat;  
11     kvadrat = varde * varde;  
12     return kvadrat;  
13 }
```

15.3 Exempel på boolean funktion. Vem får åka Balder?

Ibland är det bra att lägga jämförelser i en funktion som sedan returnerar ett boolean värde. Dessa funktioner kan man sedan använda direkt i en if-sats

```
7  if(farAkaBalder(langd, alder)){
8      text("Du får åka balder",20,30);
9  }
10 else {
11     text("Du får inte åka balder",20,30);
12 }
```

Vi behöver då skapa funktionen farAkaBalder(). Den ska returnera ett värde av typen boolean. Det vill säga antingen true eller false. Vi börjar med att skapa själva funktionshuvudet. Det börjar med typen som i detta fall är boolean sedan kommer namnet och sist parametrarna. Parametrarna ska vara av typen int, eftersom längden och åldern är heltal.

```
boolean farAkaBalder(int langd, int alder)
```

Och som med alla funktioner som ska returnera något så skapar vi först ett skelett på följande form:

```
17 boolean farAkaBalder(int l, int a){
18     boolean res;
19
20     return res;
21 }
```

Och sedan lägger vi till en if-sats:

```
17 boolean farAkaBalder(int l, int a){
18     boolean res;
19     if( a >= 7 && l >=130){
20         res = true;
21     }
22     else {
23         res = false;
24     }
25     return res;
26 }
```

Hela programmet blir då:

```
balderfunktion ▾
1 import javax.swing.JOptionPane;
2 void setup(){
3     size(200,100);
4     background(0);
5     int langd = int(JOptionPane.showInputDialog("Hur lång är du?"));
6     int alder = int(JOptionPane.showInputDialog("Hur gammal är du?"));
7     if(farAkaBalder(langd, alder)){
8         text("Du får åka balder",20,30);
9     }
10    else {
11        text("Du får inte åka balder",20,30);
12    }
13}
14
15 boolean farAkaBalder(int l, int a){
16     boolean res;
17     if( a >= 7 && l >=130){
18         res = true;
19     }
20     else {
21         res = false;
22     }
23     return res;
24 }
25
```

Övning 15-1 Funktion som ritar ut en blomma

Skapa en funktion som tar ett x och y värde och ritar ut en blomma liknande den i [Övning rita blomma](#) på den positionen som anges av x och y. Testa sedan att rita ut tre blommor, med hjälp av den funktion du skapat.

funktionblomma ▾

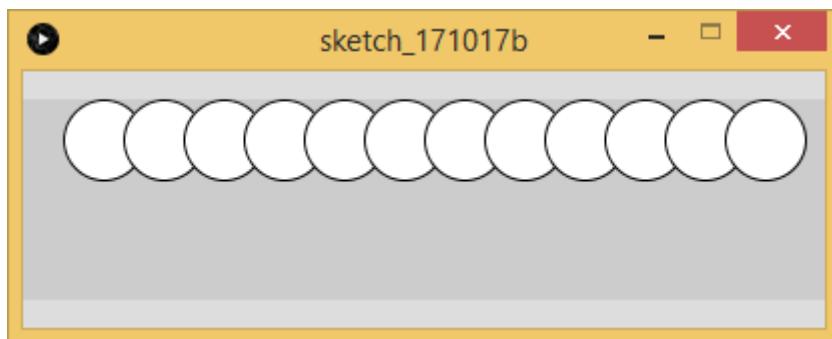
```
1 void setup() {
2     size(200, 200);
3     blomma(50, 60);
4     blomma(150, 60);
5     blomma(100, 140);
6 }
7 void blomma(x,y) {
8     ...
9     ...
10    ...
11    ...
12    ...
13    ...
14    ...
15 }
```

Övning 15-2 Övning funktion som ritar ut många cirklar

Skapa en funktion som tar `x`, `y`, `antal`, `diameter`, samt `avstand` som inparameter, och som ritar ut så många cirklar som anges av `antal`, men storlek och det avstånd som anges av parametrarna `avstånd` och `storlek`, på position `x,y`

```
sketch_171017b

1 void setup() {
2     size(400, 100);
3     cirklar(40,20,12, 40, 30);
4 }
5
6 void cirklar(6
7
8
9
10}
11
```



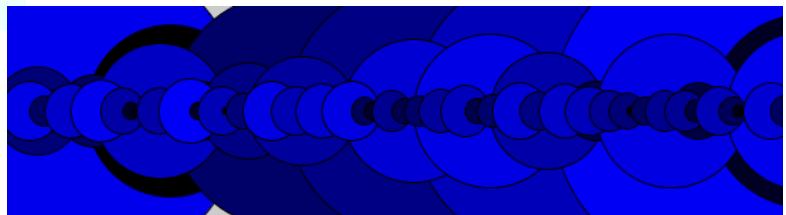
Övning 15-3 Övning slumpad cirkelrad-funktion

```
whileex1

1 size(600,60);
2 float x=0;
3 while(x<width){
4     float d = random(5,50);
5     x+=d/2;
6     fill(0,0,4*d);
7     ellipse(x,30,d,d);
8 }
```

Ändra while-exemplet som visas i bilden ovan och som vi [tidigare](#) gick igenom, så att du skapar en funktion som tar min och max diameter som inparameter, och ritar ut cirklar på en rad som i exemplet, med max och min-värdet på de slumpade radierna angivna av parametrarna . Kör funktionen tre gånger med olika inparametrar. Se exempel nedan.

```
void setup() {  
    size(600, 160);  
    cirkelrad(130,250);  
    cirkelrad(10, 130);  
    cirkelrad(5, 50);  
    save("slumpcirkelrad.png");  
}
```



För att få en större variation på färgerna har jag i bilden ovan också ändrat i raden som sätter färgen till: `fill(0, 0, map(d, min, max, 0,255));`

Övning 15-4 Övning upphöjt till-funktion

Skapa funktion potens som räknar ut upphöjt till

The screenshot shows the Processing IDE interface. On the left, the code for 'funktionpotens' is displayed:

```
1 void setup(){  
2     fill(0);  
3     float tal=potens(2,4);  
4     text(tal,20,20);  
5 }  
6  
7 float potens(float bas, float kraft){  
8     float resultat;  
9     resultat=bas*kraft;  
10    return resultat;  
11 }  
12  
13 }  
14  
15 }  
16 }
```

On the right, a preview window shows the result of the code execution: the value 16,000.

Övning 15-5 Funktion kontrollerar om muspekaren är ovanför en rektangel

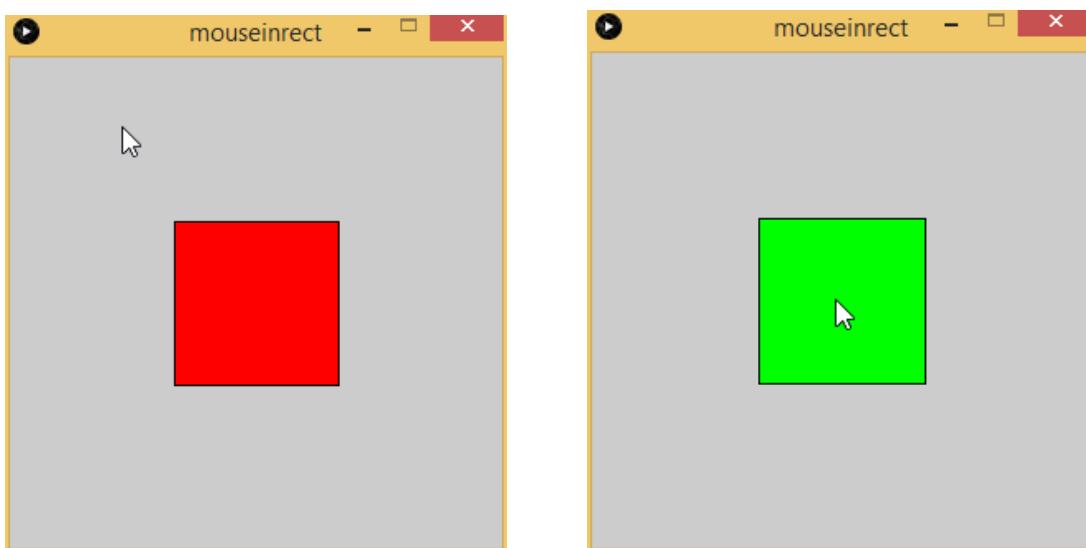
Skapa en funktion som tar fyra parametrar som representerar x, y, bredd och höjd, och som returnerar en boolean som ska vara sann om muspekaren befinner sig inom den angivna rektangeln. Använd koden nedan för att testa programmet.

```
int x=100;
int y=100;
int hojd=100;
int bredd = 100;

void setup() {
    size(300, 300);
}

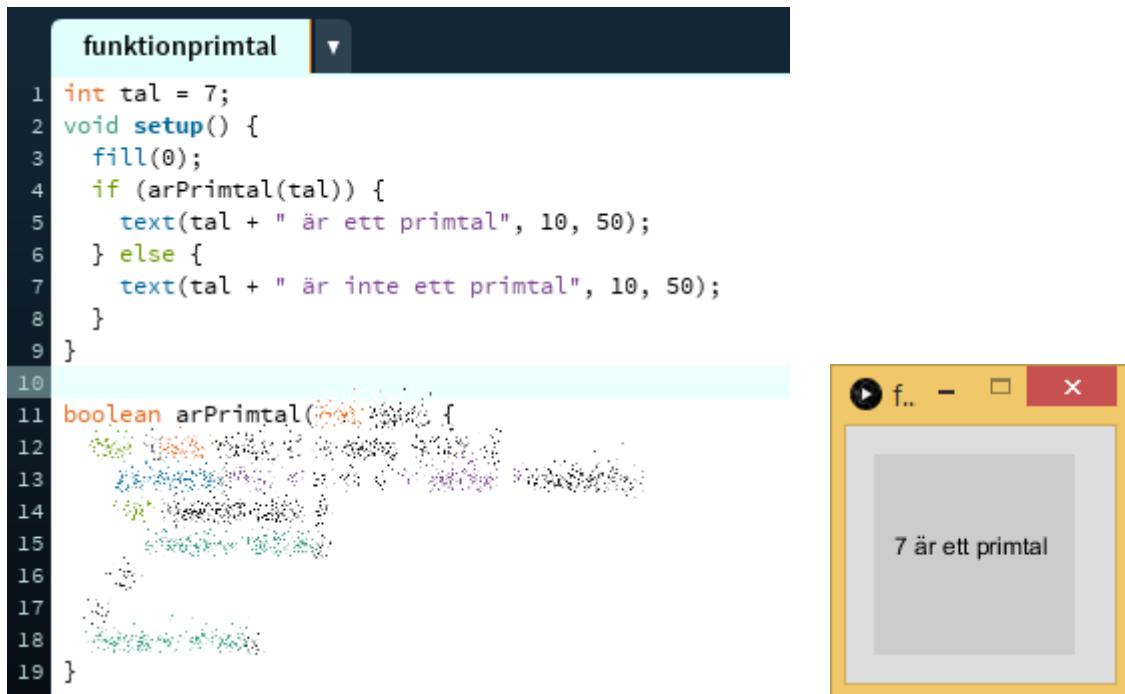
void draw() {
    if (mouseInRect(x, y, bredd, hojd)) {
        fill(0, 255, 0);
    } else {
        fill(255, 0, 0);
    }
    rect(x,y,bredd,hojd);
}

// Skriv din kod här
```



Övning 15-6 Övning primtalsfunktion

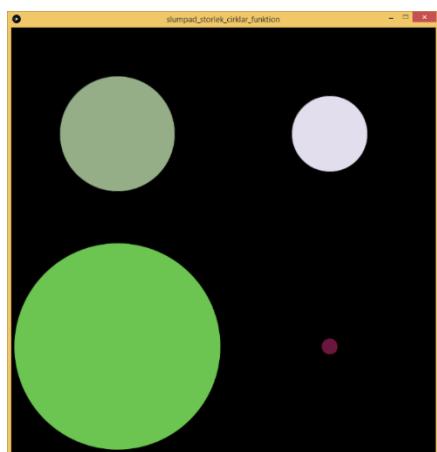
Skapa en funktion som undersöker om ett tal är ett primtal. För att kontrollera ifall ett tal är jämt delbart med ett annat tal används enklast modulo operatorn %, och du kontrollerar då ifall resultatet blir noll.



```
funktionprimtal
1 int tal = 7;
2 void setup() {
3     fill(0);
4     if (arPrimtal(tal)) {
5         text(tal + " är ett primtal", 10, 50);
6     } else {
7         text(tal + " är inte ett primtal", 10, 50);
8     }
9 }
10
11 boolean arPrimtal(int tal) {
12     for (int i = 2; i < tal; i++) {
13         if (tal % i == 0) {
14             return false;
15         }
16     }
17     return true;
18 }
19 }
```

Övning 15-7 Övning flera animerade slumpcirklar

Utgå från det sista exemplet som finns under Tidsstyrda iteration kombinerat med slumptal ovan, och skapa en funktion som ritar ut en cirkel med en slumpad diameter mellan 5 och 400 på en position som beror av inparametrarna x,y. Rita sedan ut fyra cirklar med hjälp av funktionen du skapade. Du får alltså inte kopiera koden som ritar ut ellipsen för att lösa uppgiften.



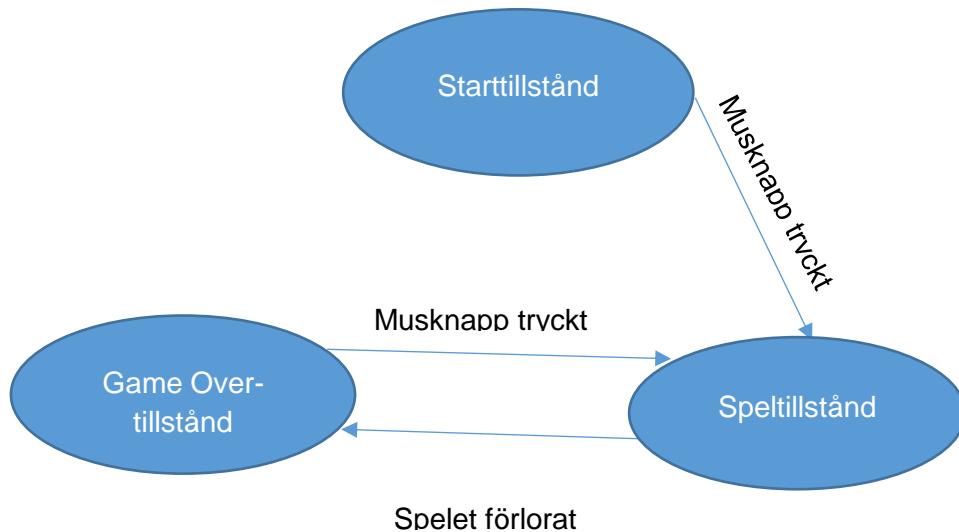
Du kan testa en webbversion av programmet här:

http://jonathan.dahlberg.media/exempel/slumpad_storlek_cirklar_funktion/

16 Tillståndsmaskiner

En tillståndsmaskin (på engelska *statemachine* eller *finite-state machine*) är ett koncept och inte något nytt vi ska lära oss om Processing i sig. Det går ut på tanken att vårt program kan hamna i olika tillstånd. När vi håller på med spel finns det oftast typiskt tre olika tillstånd. Ett startläge, där en startskärm visas, ett spelläge där man spelar själva spelet, samt ett gameover-läge, där ett *Game Over*-meddelande visas. Man använder normalt ett heltal (int) för att hålla reda på vilket läge som vi befinner oss i just nu. Den variabel kallar vi för vår tillståndsvariabel (statevariable).

Tillståndsmaskiner illustreras ofta med ett tillståndsdiagram. Bilden nedan visar hur ett sådant tillståndsdiagram skulle kunna se ut om vi ville utöka vårt gravitationsspel ovan med ett startläge och ett gameover-läge.



Det första som vi behöver lägga till vårt gravitationsspel är tillståndsvariabeln som jag kallar för state, samt också konstanta variabler som innehåller numret på varje tillstånd. Ordet final innan de tre översta variablerna betecknar att variabeln är en konstant.

```
final int START_STATE = 1;
final int PLAY_STATE = 2;
final int GAMEOVER_STATE = 3;
int state = START_STATE;
```

Jag skulle sedan vilja flytta över själva utritningen av spelet gravity som vi gjorde tidigare, det vill säga hela draw-funktionen till en annan funktion som vi kallar för drawPlay. Detta görs enklast genom att helt enkelt byta namn på den existerande draw-funktionen till drawPlay.

Nu har vi alltså inte längre någon draw-funktion i programmet, så nu skapar vi en ny draw-funktion. Det den ska göra är att i sin tur anropa andra funktioner beroende på vilket tillstånd som spelet befinner sig i.

```
29
30 void drawPlay() {
31   background(0);
32
33   fill(vit);
34   if (keyPressed) {
35     if (keyCode==LEFT) {
36       rectx=rectx-10;
37     }
38     if (keyCode==RIGHT) {
39       rectx=rectx+10;
40     }
41   }
42   rect(rectx, 470, 100, 20);
43
44   if (get((int)bollx, (int)bolly)==vit) {
45     speedy=-speedy;
46   }
47
48   fill(rod);
49   ellipse(bollx, bolly, 30, 30);
50   bollx+=speedx;
51   bolly+=speedy + accy/2;
52   speedy+=accy;
53 }
54
```

```
19 void draw() {
20   switch(state){
21     case START_STATE: drawStart();
22     break;
23     case PLAY_STATE: drawPlay();
24     break;
25     case GAMEOVER_STATE: drawGameOver();
26     break;
27   }
28 }
```

Som ni ser så är det röda träck under namnen drawStart och drawGameOver. Det beror naturligtvis på att de inte existerar än. Vi skapar funktionerna och lägger in kod för att rita ut det som ska ritas ut i respektive tillstånd.

```
55 void drawStart(){
56   background(0);
57   textAlign(CENTER,CENTER);
58   textSize(20);
59   text("Tryck på musknappen för att starta",width/2, height/2);
60 }
61
62 void drawGameOver(){
63   textAlign(CENTER,CENTER);
64   textSize(20);
65   text("Game Over\nTryck på musknappen för att start om",width/2, height/2);
66 }
```

Nu har vi nästan allt på plats. Men vi har inte skrivit någon kod för att växla mellan tillstånden.

Vi börjar med att lägga till kod för att komma från starttillståndet till speltillståndet:

```

55 void drawStart(){
56   background(0);
57   textAlign(CENTER,CENTER);
58   textSize(20);
59   text("Tryck på musknappen för att starta",width/2, height/2);
60   if(mousePressed){
61     state=PLAY_STATE;
62   }
63 }
64

```

Nu får ni gärna testköra och kontrollera så att startskärmen kommer upp och att vi sedan får börja spela när vi har klickat på fönstret.

Vi lägger sedan till följande sist i drawPlay för att detektera att spelaren har missat bollen och sedan ändrar vi tillstånd till game over-tillståndet.

```

53   if(bolly > height){
54     state = GAMEOVER_STATE;
55   }
56 }
57

```

Sist men inte minst så lägger vi till

```

69 void drawGameOver(){
70   textAlign(CENTER,CENTER);
71   textSize(20);
72   text("Game Over\nTryck på musknappen för att start om",width/2, height/2);
73   if(mousePressed){
74     state=PLAY_STATE;
75   }
76 }
77

```

Nu känns det ju som vi är klara, så nu testkör vi.

Om du testkör så märker du att det inte riktigt fungerar som vi tänkt oss. Problemet är att det blir *Game Over* direkt vi startat om eftersom vi inte ändrat tillbaka x och y-värdena för bollen till startvärderna.

Vi skapar en init-funktion som sätter alla startvärden. Vi kopierar helt enkelt de fem första raderna i programmet där variablerna skapas och initieras, oc h kopierar sedan dessa rader till init-funktionen, men tar bort själva skapandet av variablerna. Det vill säga vi raderar datatyperna som står innan.

```

1 float bollx=100;
2 float bolly=200;
3 float speedy=0;
4 float speedx=1;
5 int rectx=80;

```

Raderna kopieras
förutom typerna

```

82 void init() {
83   bollx=100;
84   bolly=200;
85   speedy=0;
86   speedx=1;
87   rectx=80;
88 }

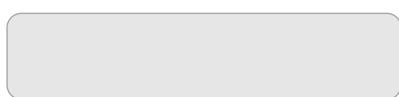
```

Vi kör sedan `init()` i samband med att vi sätter state till `PLAY_STATE`.

```
72 void drawGameOver() {  
73     textAlign(CENTER, CENTER);  
74     textSize(20);  
75     text("Game Over\nTryck på musknappen för att start om", width/2, height/2);  
76     if (mousePressed) {  
77         state=PLAY_STATE;  
78         init();  
79     }  
80 }
```

17 Aktivitetsdiagram och Pseudokod

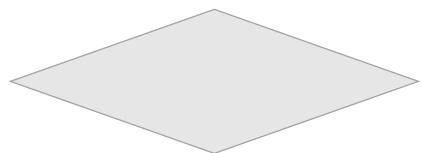
Det finns olika sätt för att planera sina programmeringsprojekt innan man börjar arbeta. Ett sätt är att använda Sekvensdiagram. I ett sekvensdiagram så använder man sig av följande tre symboler för att bygga upp ett sekvensdiagram.



Start eller stop



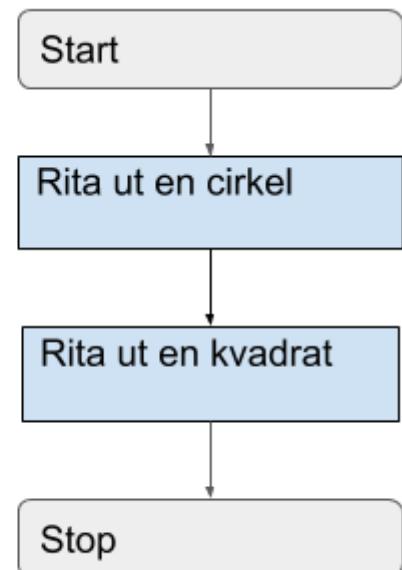
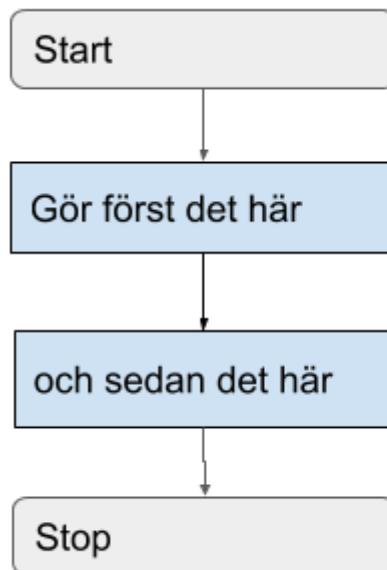
Något utförs (sekvens)



Ett val mellan två alternativ

Nedan till höger visas ett enkelt exempel på ett program som har planerats med hjälp av sekvensdiagram.

- Vi börjar med start
- Vi utför en eller flera satser efter varandra
- Avslutar med stop



17.1 Selektion

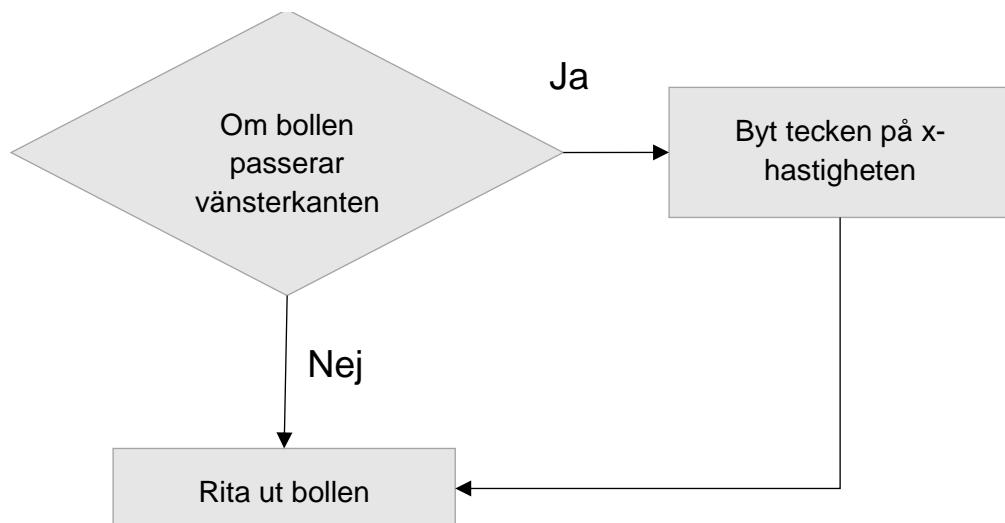
Selektion handlar om att göra val i koden, och alla val visas med den snedställda romben. Ut från symbolen går det alltid två stycken pilar, en som pekar på vad som ska hända om villkoret är uppfyllt och en som visar vad som händer om villkoret inte är uppfyllt.

För att exemplifiera hur ett aktivitetsdiagram respektive pseudokod fungerar så tar vi som exempel en liten del av exemplet/övningen med den studsande bollen som vi gjorde ovan. Nedan ser du ett diagram som visar studs mot vänsterkanten hanteras.

Pseudokod:

Om bollen passerar högerkanten
Byt tecken på x-hastigheten
Rita ut bollen

Aktivitetsdiagram:

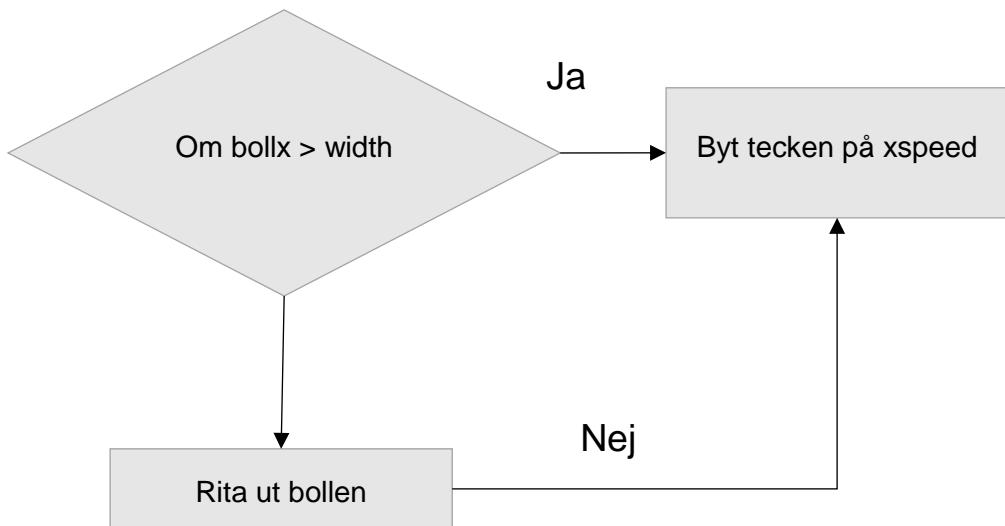


Man kan också göra beskrivning lite mer teknisk, det vill säga mer i detalj beskriva vad som ska utföras i koden.

Pseudokod:

Om bollx > width
 Byt tecken på xspeed
Rita ut bollen

Aktivitetsdiagram:



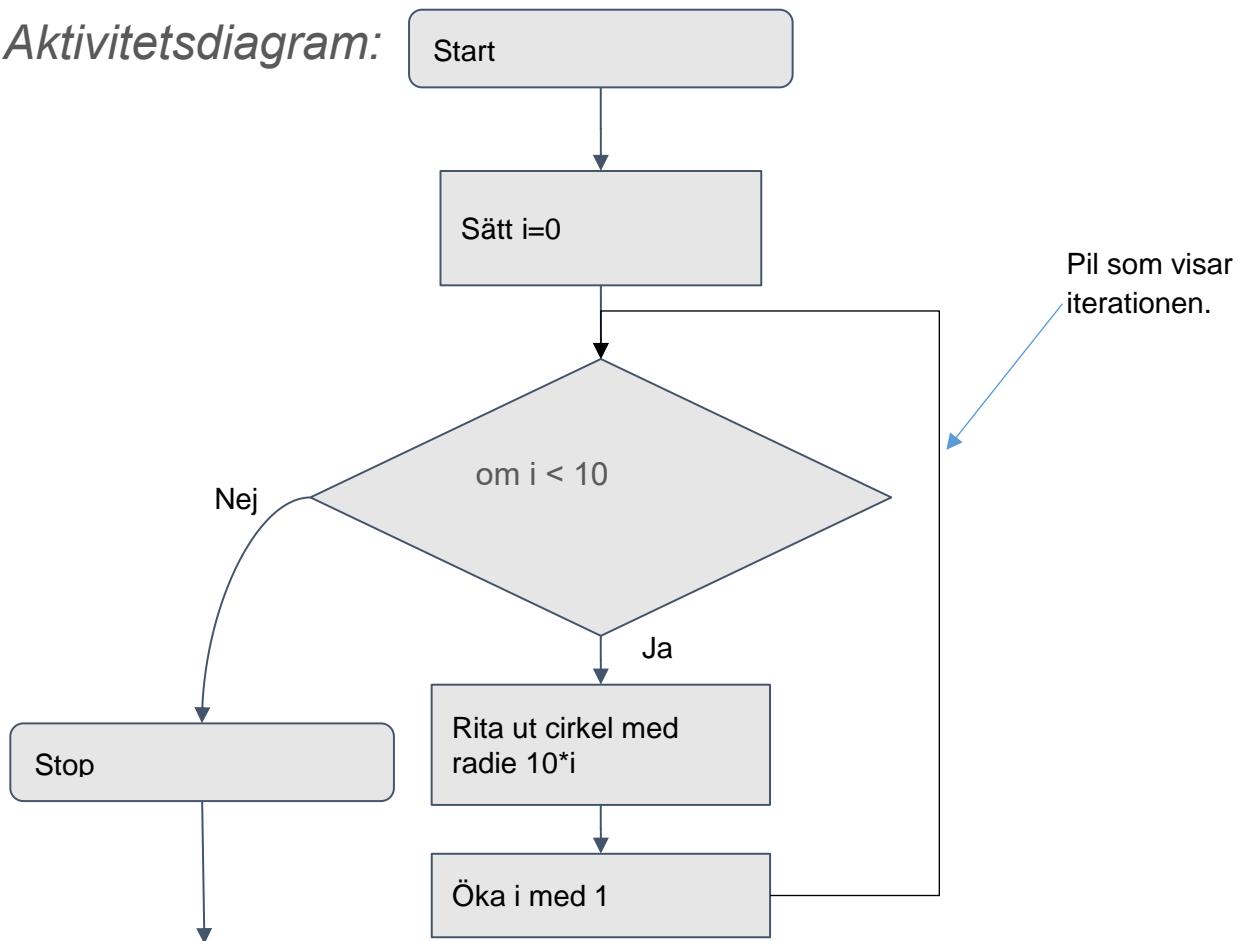
17.2 Iteration

Selektion ser ut på ett liknande sätt som selektionen. Med skillnaden att i iteration går alltid en pil tillbaka i diagrammet för att visa att det är en loop.

Pseudokod:

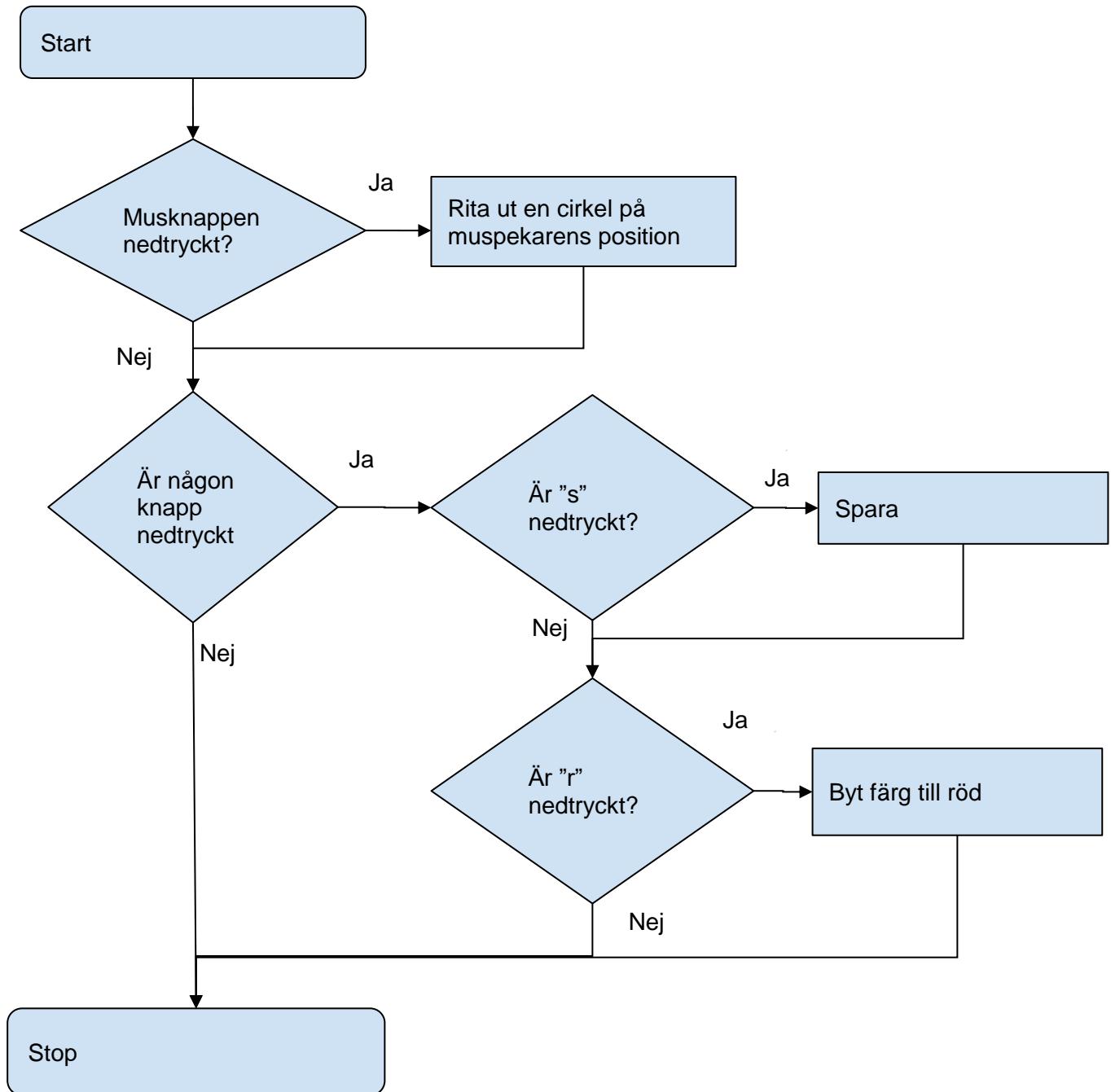
Om bollx > width
 Byt tecken på xspeed
Rita ut bollen

Aktivitetsdiagram:



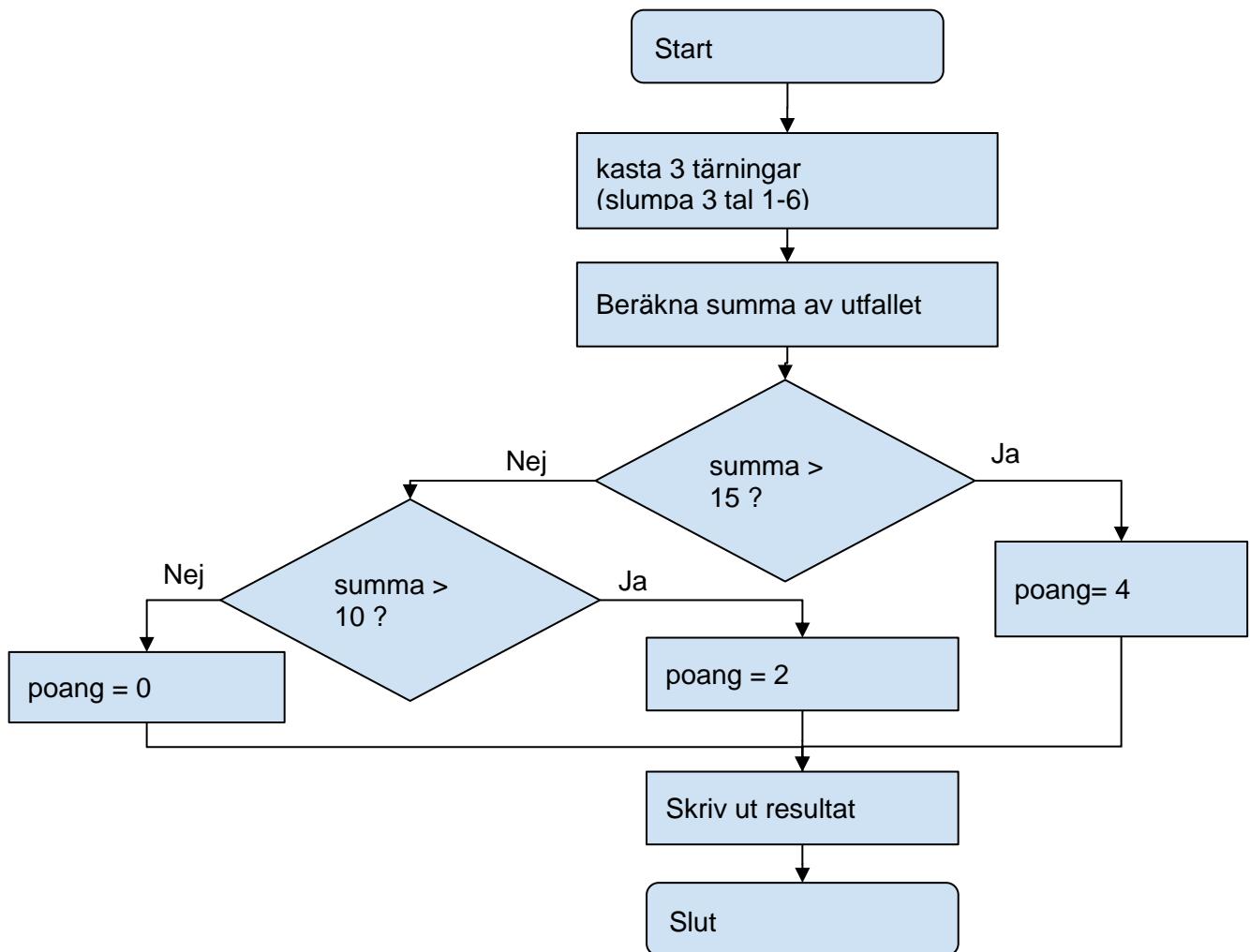
17.3 Aktivitetsdiagram för ritprogrammet

Nedan ser du ett aktivitetsdiagram över Övning 13-7 Övning ritprogram A).



17.4 Aktivitetsdiagram för tärningsspel

Nedan ser du ett exempel på ett aktivitetsdiagram över ett program som slumar tre tärningar och sedan får spelaren olika poäng beroende på summan på tärningarna.



17.5 Pseudokod för tärningsprogram

Pseudokod:

Kasta tre tärningar, det vill säga slumpa tre tal mellan 1-6

Beräkna summan av utfallet

Om summan > 15

Ska användaren få 4 poäng

annars

om användaren får > 10 poäng

Ska användaren få två poäng

annars

Användaren får 0 poäng

Skriv ut resultat

Alternativt

Pseudokod:

Kasta tre tärningar, det vill säga slumpa tre tal mellan 1-6

Beräkna summan av utfallet

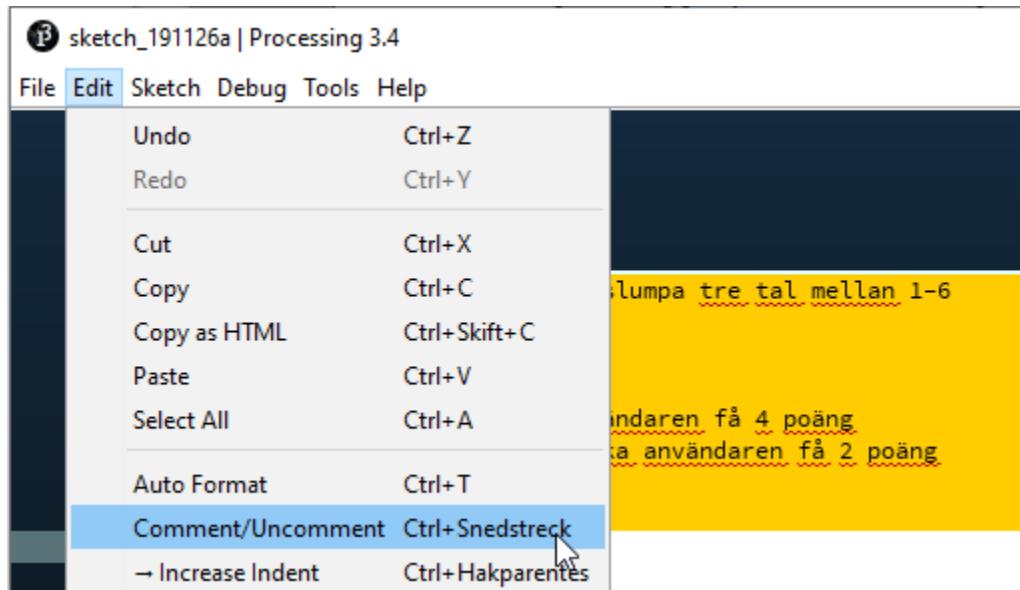
Om summan blir större än 15 ska användaren få 4 poäng

Om summan blir större än 10 poäng ska användaren få 2 poäng

Annars ska användaren få 0 poäng

Skriv ut resultatet.

Poängen med att skriva pseudokod är att vi sedan kan använda dem som kommentarer i projektet. Vi klistrar alltså in dem i Processing-miljön. Vi markerar sedan allt och väljer *Edit->Comment/Uncomment*.



```
1 //Kasta tre tärningar, det vill säga slumpa tre tal mellan 1-6
2
3 //Beräkna summan av utfallet
4
5 //Om summan blir större än 15 ska användaren få 4 poäng
6 //Om summan blir större än 10 poäng ska användaren få 2 poäng
7 //Annars ska användaren få 0 poäng
8
9 //Skriv ut resultatet.
```

Sedan börjar man helt enkelt att fylla i kod som motsvarar varje rad i pseudokoden, och man låter pseudokoden stå kvar som kommentarer.

```
1 //Kasta tre tärningar, det vill säga slumpa tre tal mellan 1-6
2 int t1 = int(random(1,7));
3 int t2 = int(random(1,7));
4 int t3 = int(random(1,7));
5
6 //Beräkna summan av utfallet
7
8 int summa = t1 + t2 + t3;
```

17.6 Övning Aktivitetsdiagram och Pseudokod

Gör ett aktivtetsdiagram och/eller pseudokod för ett program motsvarande *Övning 13-9 Gravitationsspel steg1* ovan. Det vill säga ett program som har en boll som faller med gravitationen, och som kan studsa mot en rektangel. En rektangel som användaren kan styra. Bollen ska också kunna studsa mot väggen.

Du kan antingen göra Aktivtetsdiagrammet/pseudokoden först och sedan sedan kolla om det stämmer överens med din lösning av uppgiften i Processing, eller också titta på koden och skapa Aktivtetsdiagrammet/pseudokoden

18 Projektidéer

När vi kommit så här långt i kursen så kan ni faktiskt så mycket så att ni kan börja göra egna projekt.

18.1 Sten, sax påse

Skapa ett enkelt *Sten Sax Påse* spel.

Steg 1

Skapa ett enkelt spel där användare får skriva sten sax eller påse som input i en sträng, och användaren får som utskrift vad han har valt, vad datorn har valt samt, vem som vann.

Steg 2

Illustrera användarens och datorns val med hjälp av bilder.

Steg 3

Ge användare möjlighet att köra flera gånger, och räkna poäng.

18.2 Plattformsspel

Skapa ett enkelt plattformsspel, liknande ett mycket förenklat mariospel, där du ritar ut en bana, och när man trycker en viss tangent t.ex. mellanslag så får spelaren en positiv y-hastighet, och åker uppåt, och använd sedan gravitation för att få spelaren att åka nedåt igen. Här kan du ha nytta att titta på gravitationsspelet som vi gjorde ovan.

Steg 1

Rita ut en bana

Steg 2

Skapa en spelare, som från början kanvara en cirkel, som hoppar upp när man trycker mellanslag, och sedan åker ner till själva marken igen med hjälp av gravitationen.

Steg 3

Se till att spelaren också kan förflytta sig med vänster och högerpilen

Steg 4

Inför objekt som spelaren kan plocka upp för att få poäng.

Steg 5

Skapa ett sätt som gör att spelet kan förloras, antingen att tiden går ut eller att det finns objekt som inte får nuddas.

18.3 Flappy bird

Skapa ett spel som lånar principerna från flappy bird, dvs när mellanslag trycks ned så hoppar fågeln upp, och den dras ned igen av gravitationen, samt spelaren ska väja för objekt som förflyttar sig från höger till vänster. Här har du nytta av att titta på gravitationsspelet som vi gjorde ovan.

Steg 1

Få själva fågeln, som till och börja med kan vara en cirkel, att bete sig som fågeln i flappy bird.

Steg 2

Försök få en rektangel att förflytta sig från höger till vänster, och när den kommer längst till vänster ska den börja om igen.

Steg 3

Fixa kollision med rektanglarna, det enklaste är att använda sig av get-metoden, dvs kolla färgen där fågeln strax ska ritas ut för att kontrollera om en kollision har skett.

Steg 4

Få rektanglarna att se ut som rör så att spelet ser ut mer som flappy bird.

Steg 5

Se till att kollisionsdetekteringen sker på mer än en punkt dvs inte enbart i mitten av fågeln (om du inte redan har gjort det), använd snygga bilder för fågeln och bakgrunden. Tänk bara på att om du använder färger för att detektera kollisionen med rören så får inte bakgrunden innehålla samma färg som rören.

18.4 Enkelt agar.io

Steg 1

Slumpa massa cirklar, koordinaterna behöver ligga i en array eller ArrayList. Antingen skapar du en klass för cirklarna, eller också skapar du flera arrayer för x och y-koordinat samt eventuellt storlek

Steg 2

Skapa en spelare som styrs med pil tangenterna

Steg 3

Nu kommer du till själva kollesionsdetekteringen. Processing har en inbyggd funktion för att räkna ut avståndet. Du skapar en forsats för att jämföra avståndet mellan spelaren och varje cirkel. Då kan du räkna ut om du fått en kollision.

Steg 4

När du väl fixat själva krocken så är det bara att se till att den inte ritas ut nästa gång.

18.5 Snake

Steg 1

Börja med att skapa en snake som enbart är ett segment lång, tex en enkel cirkel.

Steg 2

Slurpa ut mat på spelplanen

Steg 3

Detektera när spelaren träffar maten och ge spelaren poäng.

Steg 4

Gör masken lång genom att för varje frame lägga till koordinaterna till varje masksegment i en arraylist. Du behöver antingen två arraylist, eller en arraylist där du stoppar in ett Pvector objekt eller ett objekt från en egen klass. När masken nått sin maxlängd så ska du också börja med att ta bort det sista elementet från arralisten/arraylistorna.

18.6 21 Spel/black jack

Skapa ett enkelt 21 spel, eller ett mer avancerat black jackspel

Steg 1

I den enklaste varianten struntar du i hur många spelkort det finns i en kortlek och allt sådant utan slumar ett tal mellan 1 och 11 för användaren och ett för datorn, och frågar användaren om han vill fortsätta eller stanna. Om han vill fortsätta så slumar du ett till tal till användaren och ett för datorn, summerar datorn och användarens tal och frågar sedan användaren om han vill fortsätta. Datorn stannar automatiskt vid 16. Användaren slutar när användaren angivit att den slutar eller när den kommit över 21. Om båda har under 21 när båda slutat vinner den med mest poäng, om någon blivit "tjock" det vill säga fått mer än 21 poäng så vinner den andra.

Steg 2

Nu kan du försöka simulera en riktig kortlek istället.

Steg 3.

Visa upp fina bilder för varje kort som dras

18.7 Hänta gubbe

Skapa ett enkelt hänta gubbespel

Steg 1

Skapa en array med ett antal ord, och slumpa fram ett ord. Sedan får användaren ange en bokstav. Du får då kolla om bokstaven finns i ordet, om det finns i ordet så ska det synas på något sätt.

Steg 2

Fixa räkning av hur många gånger som användaren har kvar, och se till att användaren blir meddelad när hen förlorat.

Steg 3

Fixa så att användaren också kan vinna.

19 Blandade repetitionsuppgifter

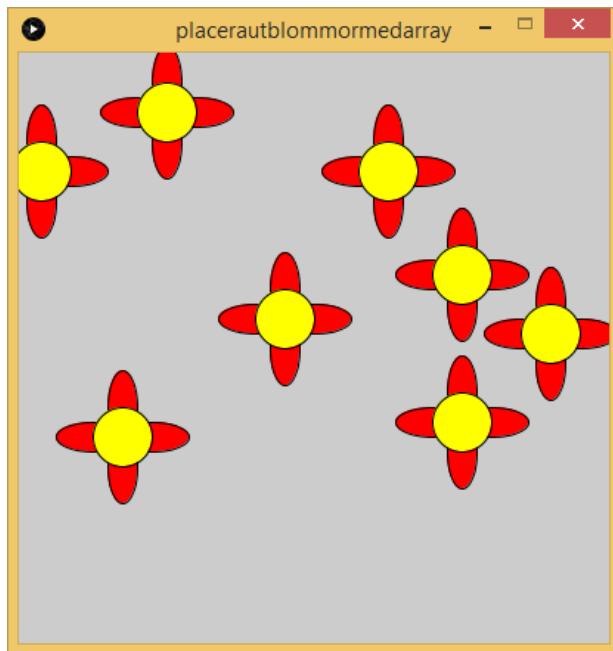
Övning 19-1 Placera ut dina fina blommor med hjälp av arrayer

Skapa en array för x-värdena och en array för y-värdena och stoppa in de först i exemplet nedan.

(Jag har använt x-värdena 100, 15, 300, 360, 180, 300, 250, 70, och y-värdena: 40, 80, 250, 190, 180, 150, 80, 260. Men du kan välja andra om du vill. Men det behöver vara lika många x-värden som y-värden.)

```
void setup() {
    size(400,400);
    for (int i=0; i < x.length; i++) {
        blomma(x[i],y[i]);
    }
}

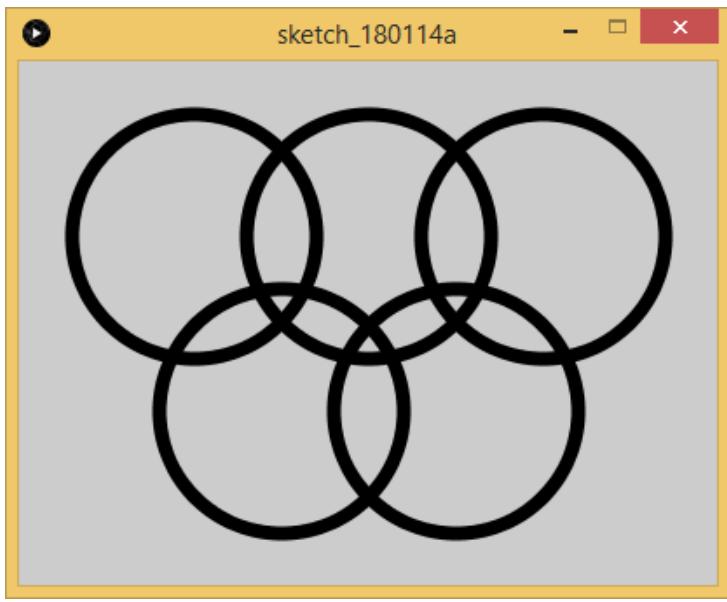
void blomma(float x, float y) {
    fill(255, 0, 0);
    ellipse(x-20, y, 50, 20);
    ellipse(x+20, y, 50, 20);
    ellipse(x, y-20, 20, 50);
    ellipse(x, y+20, 20, 50);
    fill(255, 255, 000);
    ellipse(x, y, 40, 40);
}
```



Övning 19-2 Olympiska ringarna

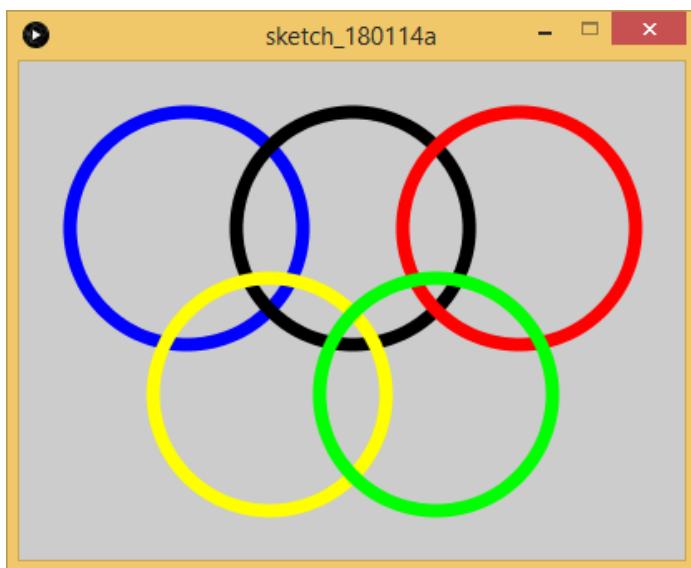
- A) Arrayerna nedan innehåller koordinaten till sex stycken cirklar. Använd en for-loop för att rita ut varje ring på sin plats.

```
int[] x={100, 200, 300, 150, 250}  
int[] y={100, 100, 100, 200, 200}
```



- B) Nu vill vi också att ringarna får färger. Använd arrayerna nedan för att fixa det.

```
int[] r ={0, 0, 255, 255, 0};  
int[] g={0, 0, 0, 255, 255};  
int[] b={255, 0, 0, 0, 0};
```



Övning 19-3 Animera din bokstav

Skriv ett program som på ett animerat sätt skriver ut din initial. Du kan utgå från programmet nedan och ändra det så att fler linjer animeras. Nedan finns början på koden för att rita ut ett J.

```
int count=0;
float x=50;
float y=50;
void setup() {
    size(400, 400);
    noStroke();
    fill(255,0,0);
}

void draw() {
    count++;
    if(count<100){
        x+=1;
    }
    //Lägg till fler satser här!! (förslagsvis else if)
    ellipse(x,y,20,20);
}
```



Du kan testa en webbversion av programmet som ritar ut ett J här:

<http://jonathan.dahlberg.media/exempel/animerabokstav/>

Övning 19-4 Funktion som summerar fyra tal

Skapa en funktion som tar fyra decimaltal.

```
float tal1 = 4;  
float tal2 = 6;  
float tal3 = 10;  
float tal4 = 20;  
  
void setup(){  
background(0);  
float summa = sum(tal1, tal2, tal3, tal4);  
text(summa, 20,20);  
summa = sum(25,100, 150, 175);  
text(summa, 20,40);  
}  
  
//Här ska ni skriva er funktion sum.
```

När ni kör programmet ska det se ut så här:



Övning 19-5 Funktion omkrets

Skapa en funktion som räknar ut omkretsen för en cirkel. Det vill säga som parameter ska den ta radien, och sedan ska den returnera omkretsen. Låt användaren mata in ett värde på radien och skriv ut resultatet.

Övning 19-6 Medelvärde

- Skapa en funktion `float medelvärde(float[] varden)` som tar en array som innehåller värde och som beräknar medelvärdet av värdena i arrayen. Lägg till enkel kod för att testa programmet.
- Utöka programmet så att användaren får mata in värdena som stoppas in i arrayen.

Övning 19-7 Omvänd ordning

Be användaren mata in tre ord i en array, och låt programmet sedan skriva ut orden i omvänt ordning.

Övning 19-8 Slumpat linjemönster

- Skapa ett program som först ritar en linje från 200,0 till height,0 och sedan varje gång slumpar fram en ny position som är i intervallet +-10 pixlar från den föregående både upp och ner.

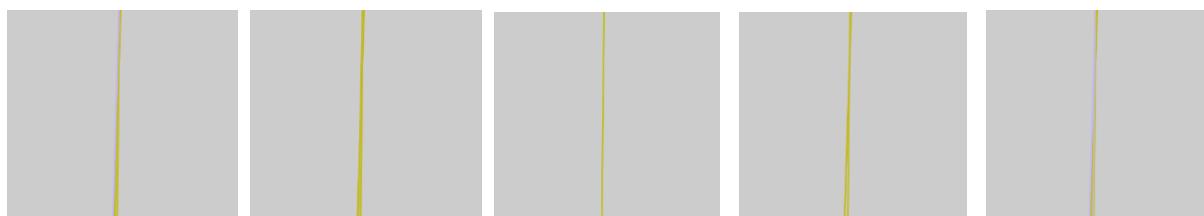
(I exemplet nedan har jag satt färgen så att red värdet är den övre x-koordinaten och green värdet är den nedre koordinaten, och sedan har jag slumpat fram ett värde till blue.

```
float b = random(256);  
color c = color(x1, x2, b);  
)
```

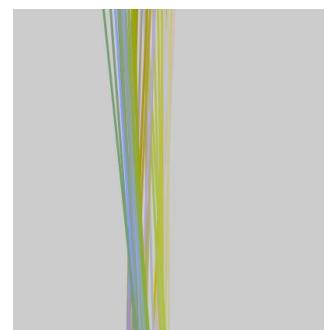
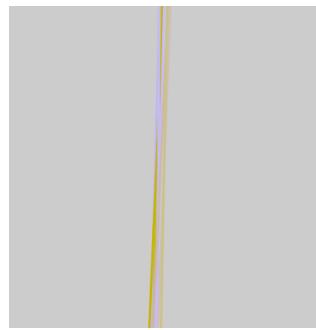
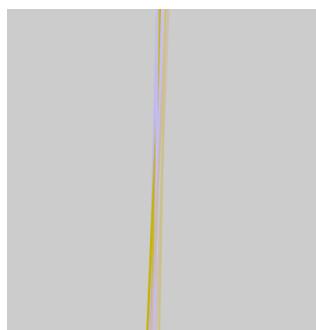
- Du märker att efter ett tag så kommer linjerna utanför fönstret. Fixa så att x-koordinaterna på linjerna aldrig är mindre än 0 eller större än width.

[Du kan köra en webbversion av programmet här:](#)

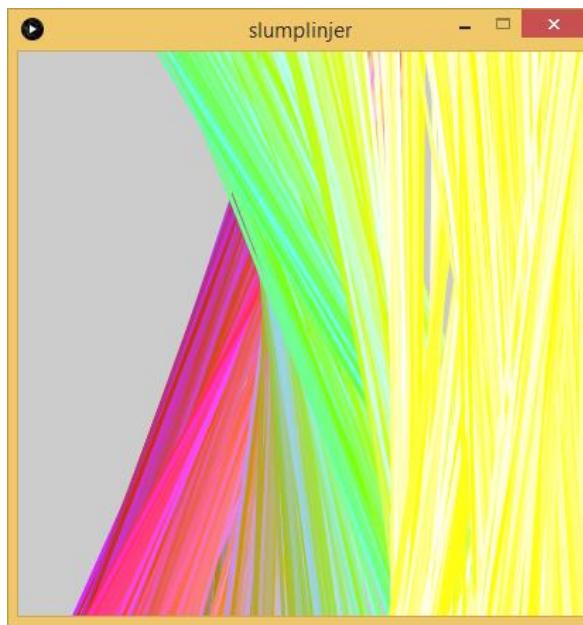
Så här kan programmet se ut den första sekunden:



....



Efter ca 10 sek kan det se ut så här:



Övning 19-9 Reaktionstest

Skapa ett program som vid fem tillfällen slumpar fram ett värde mellan 1 och 300. Använd en variabel för att hålla reda på hur många gånger som draw() har körts. Kolla när draw() har körts så många gånger som har angivits av det slumpade värdet, därefter ändras bakgrunden till en slumpvis färg.

Användaren ska sedan så snabbt som möjligt efter att färgen har bytts trycka på musknappen. Räkna hur många gånger som draw() har kört från det att cirkeln visades till dess musknappen var nedtryckt och stoppa värdet i en array.

Detta ska upprepas fem gånger, sedan ska alla reaktioner visas på skärmen och största och minsta värde ska skrivas ut.



Trycker man för tidigt så ska man förlora direkt:



Du kan köra en webbversion av programmet här:

Övning 19-10 *Geometry dash*

- A. Skapa en funktion som ritar en geometrisk form och tar inparametrarna:
 - form - som är ett heltal där 0 betyder kvadrat, 1 betyder triangel, 2 betyder cirkel
 - x - som är x-koordinaten för den geometriska formen
 - y - som är y-koordinaten för den geometriska formen
- B. Nu ska du använda funktionen som du skapade ovan. Slumpa fram ett värde mellan 0 och 2 för att bestämma formen. Låt sedan den geometriska formen förflytta sig mellan höger till vänster, och när den kommit till vänster ska en ny form slumpas fram och den ska sedan röra sig från höger till vänster etc.
- C. Skapa en spelare som ska vara en röd boll, och när man trycker ner mellanslag ska den hoppa upp.
- D. Detektera krasch. Skriv ut game over samt stoppa spelet vid krasch.

21 Klasser och Objekt

Klasser är ett trevligt sätt att kunna samla ihop metoderna och de variablerna som hör ihop med dessa metoder till en gemensam enhet. Det kan handla om till exempel en grafisk form som en boll eller rektangel, det kan handla om en spelarfigur i ett spel, eller också kan det vara en knapp i ett grafiskt gränssnitt.

Vi kommer att titta på hur man kan använda klasser och objekt för att rita ut grafiska objekt, till att börja med en boll. De variabler som behövs för att skapa en boll är dess position (x,y), samt dess diameter d. Om vi vill skapa en klass med enbart dessa variabler så skriver vi:

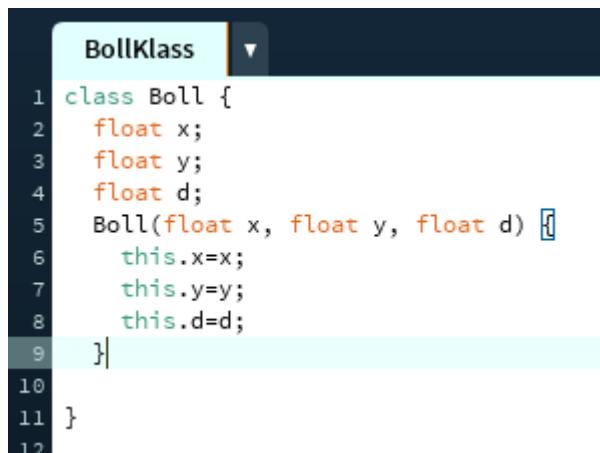
```
class Boll {  
    float x;  
    float y;  
    float d;  
}
```

För att skapa ett nytt objekt av klassen så kan vi skriva:

```
Boll b = new Boll();
```

Vi kan tänka oss att Klassen är som en mall och att objektet är de kopior som skapas utifrån mallen.

Vi kommer (nästan) alltid att vilja ha flera metoder för att kunna göra något med vår boll. En speciell slags metod kallas för konstruktör. Den körs i samband med att objektet skapas. Nedan ser vi boll-klassen med en konstruktör:



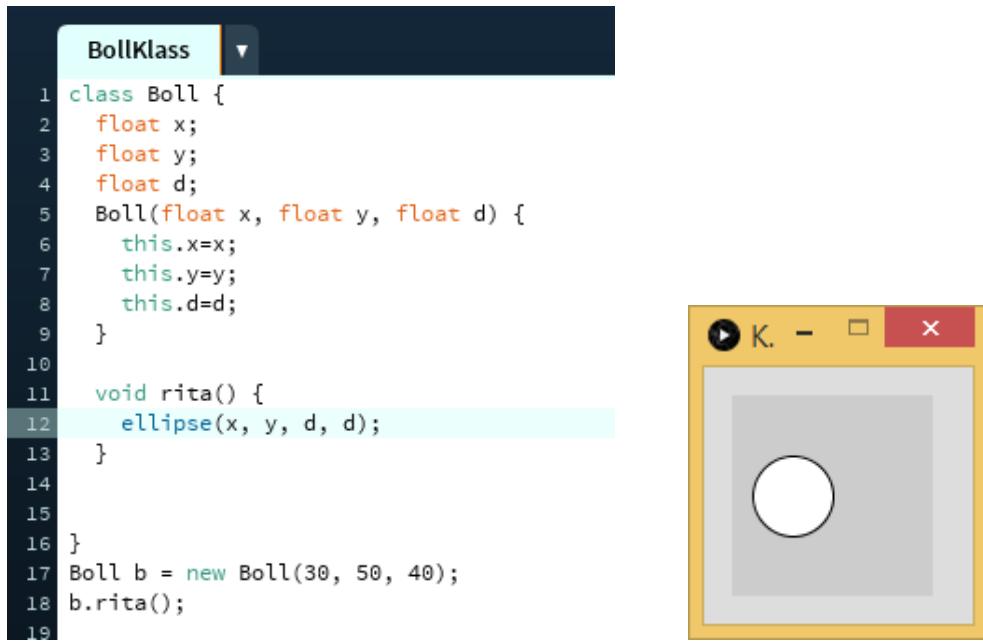
```
BollKlass  
1 class Boll {  
2     float x;  
3     float y;  
4     float d;  
5     Boll(float x, float y, float d) {  
6         this.x=x;  
7         this.y=y;  
8         this.d=d;  
9     }  
10 }  
11 }  
12 }
```

Nu när vi skapat en konstruktör med tre inparametrar måste vi använda dessa när ett objekt skapas. Vi skriver då:

```
Boll b = new Boll(30, 50, 40);
```

Men något mer vill ju göra med klassen. Framförallt skulle jag vilja att den skulle kunna rita ut sig själv. Vi lägger till en *rita* metod.

```
BollKlass ▾  
1 class Boll {  
2     float x;  
3     float y;  
4     float d;  
5     Boll(float x, float y, float d) {  
6         this.x=x;  
7         this.y=y;  
8         this.d=d;  
9     }  
10    void rita() {  
11        ellipse(x, y, d, d);  
12    }  
13}  
14  
15}  
16}  
17 Boll b = new Boll(30, 50, 40);  
18 b.rita();  
19
```



För att sedan köra metoden rita så skriver vi bara objektets namn i detta fall *b* följt av en punkt(.) och sedan namnet på metoden följt av parenteser, dvs *b.rita()* ;, som ni ser i exemplet ovan.

Sedan skulle jag också vilja kunna flytta på bollen med hjälp av ett enkelt metodanrop. Vi lägger till två variablerna speed, och speedy till klassen, samt en metod *flytta*. Hastighetessvariablerna anger helt enkelt hur långt bollen ska förflyttas mellan varje gång *flytta()* körs.

```

BollKlass2
1 class Boll {
2     float x;
3     float y;
4     float d;
5     float speedx=10;
6     float speedy=10;
7     Boll(float x, float y, float d) {
8         this.x=x;
9         this.y=y;
10        this.d=d;
11    }
12
13    void flytta() {
14        x = x + speedx;
15        y = y + speedy;
16    }
17    void rita() {
18        fill(0, 0, 255);
19        ellipse(x, y, d, d);
20    }
21 }
22
23 Boll b = new Boll(20, 20, 40);
24 b.rita();
25 b.flytta();
26 b.rita();
27 b.flytta();
28 b.rita();
29

```

Övning

- Skapa ett en klass som representerar en rektangel med en konstruktör för att ge objektet dess initiala värden.
- Komplettera med rita ut klass- och flyttametod på samma sätt som i exemplet ovan.

Vi skulle nu vilja att bollen skulle kunna flytta sig av sig själv. Vi ändrar hastigheten till några mindre värden:

```

8
9 class Boll {
10    float x;
11    float y;
12    float d;
13    float speedx=2;
14    float speedy=1;

```

Sedan skapar vi en draw metod, som anropar flytta och rita.

```
BollKlass4 ▾  
1 Boll b = new Boll(30, 20, 40);  
2 void draw(){  
3     background(0);  
4     b.flytta();  
5     b.rita();|  
6 }  
7  
8 class Boll {  
9     float x;  
10    float y;  
11    float d;  
12    float speedx=10;  
13    float speedy=10;  
14    Boll(float x, float y, float d) {  
15        this.x=x;  
16        this.y=y;  
17        this.d=d;  
18    }  
19  
20    void setSpeed(int xs, int ys) {  
21        speedx=xs;  
22        speedy=ys;  
23    }  
24    void flytta() {  
25        x = x + speedx;  
26        y = y + speedy;  
27    }  
28    void rita() {  
29        fill(0, 0, 255);  
30        ellipse(x, y, d, d);  
31    }  
32 }  
33 }
```

Det skulle ju vara trevligt om vi också kunde få bollen att studsa som i exemplet studsande boll ovan. Gå gärna tillbaka till det exemplet och se hur vi gjorde där.

```

BollKlass5 ▾

1 Boll b = new Boll(30, 20, 40);
2 void draw(){
3     background(0);
4     b.flytta();
5     b.rita();
6 }
7
8 class Boll {
9     float x;
10    float y;
11    float d;
12    float speedx=10;
13    float speedy=10;
14    Boll(float x, float y, float d) {
15        this.x=x;
16        this.y=y;
17        this.d=d;
18    }
19
20    void setSpeed(int xs, int ys) {
21        speedx=xs;
22        speedy=ys;
23    }
24    void flytta() {
25        x = x + speedx;
26        y = y + speedy;
27        if(x>=width || x<=0){
28            speedx=-speedx;
29        }
30        if(y>=height || y<=0){
31            speedy=-speedy;
32        }
33    }
34    void rita() {
35        fill(0, 0, 255);
36        ellipse(x, y, d, d);
37    }
38 }

```

Nu borde det nya programmet fungera precis som det gamla exemplet med studsande bollar ovan. Det kan tyckas att detta var mycket jobb för ingenting, men det vackra uppstår när vi vill köra flera bollar samtidigt.

Vi skapar helt enkelt en array av bollar. Vi byter ut den första raden till:

```
Boll[] bollar = new Boll[60];
```

Vi skapar en setup-metod och använder en for-sats i setup-metoden för att skapa många bollar

```
BollKlass3 ▾  
1 Boll[] bollar = new Boll[30];  
2  
3 void setup() {  
4     size(600, 600);  
5     for (int i = 0; i < bollar.length; i++) {  
6         bollar[i] = new Boll(300,300,30);  
7     }  
8 }  
9 |
```

Sedan blir vi också tvungen att använda en till for-loop i draw metoden för att se till att flytta och rita-metoderna körs i draw-metoden.

```
10 void draw() {  
11     background(0);  
12     for (int i=0; i < bollar.length; i++) {  
13         Boll b = bollar[i];  
14         b.flytta();  
15         b.rita();  
16     }  
17 }  
18 |
```

Om ni nu testkör programmet kommer ni att märka att ni inte märker så stor skillnad. Det skapas visserligen 30 stycken bollar och flyttas och ritas ut. Men eftersom de startar på samma ställe och har samma hastighet så kommer de att rita ut på samma ställe ovanpå varandra...

Vi behöver se till att hastigheten inte är samma för alla bollar. Vi ändrar så att hastigheten slumpas fram för varje boll:

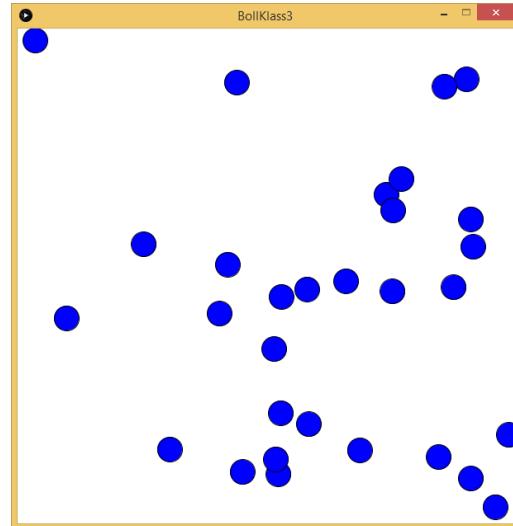
```
17 class Boll {  
18     float x;  
19     float y;  
20     float d;  
21     float speedx=random(-10,10);  
22     float speedy=random(-10,10);  
23 }
```

Nu kan vi testköra och se vad som händer. Nedan finns också koden till hela programmet.

```

BollKlass6 ▾
1 Boll[] bollar = new Boll[30];
2
3 void setup() {
4   size(600, 600);
5   for (int i=0; i < bollar.length; i++) {
6     bollar[i] = new Boll(30, 20, 40);
7   }
8 }
9
10 void draw() {
11   background(0);
12   for (int i=0; i < bollar.length; i++) {
13     Boll b = bollar[i];
14     b.flytta();
15     b.rita();
16   }
17 }
18
19 class Boll {
20   float x;
21   float y;
22   float d;
23   float speedx=random(-10,10);
24   float speedy=random(-10,10);
25
26   Boll(float x, float y, float d) {
27     this.x=x;
28     this.y=y;
29     this.d=d;
30   }
31
32   void setSpeed(int xs, int ys) {
33     speedx=xs;
34     speedy=ys;
35   }
36
37   void flytta() {
38     x = x + speedx;
39     y = y + speedy;
40     if (x>=width || x<=0) {
41       speedx=-speedx;
42     }
43     if (y>=height || y<=0) {
44       speedy=-speedy;
45     }
46   }
47
48   void rita() {
49     fill(0, 0, 255);
50     ellipse(x, y, d, d);
51   }
52 }
53

```



21.1 Skapa en mask som rör sig med klasser, objekt och ArrayList

Jag tänkte nu att vi skulle kunna utnyttja våra nya kunskaper om klasser och objekt för att göra en mask som rör sig över skärmen. Det skulle kunna vara början av ett snakespel, men vi ska inte göra hela spelet. Det ett så bra projekt att jobba med, så det nöjet vill jag inte ta ifrån er.

Till denna uppgift behöver vi använda ArrayList. Det är en av Processings inbyggda klasser. Tabellen nedan visar hur vi kan skapa, lägga till, ta bort och ta ut ett objekt från en ArrayList:

Beskrivning	Syntax	Exempel
Skapa ny ArrayList	ArrayList<Klass> namn = new ArrayList();	ArrayList<String> namn = new ArrayList();
Lägga till objekt sist i listan	namn.add(objekt);	namn.add("Hej");
Ta bort objekt från listan	namn.remove(index)	namn.remove(0); //Ta bort det första objektet från listan.
Plocka ut ett objekt ur listan	namn.get(index)	String n = namn.get(1);

Om ni vill veta allt man kan göra med listan så titta på:

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

Det som vi framförallt har nyttta av när det gäller ArrayList är att det är så lätt att lägga till objekt sist i listan och ta bort objekt först i listan.

Vi börjar med att skapa en klass liknande den klass Boll som vi gjorde ovan. Men denna gång ska klassen hålla koll på läget för ett masksegment. Vi behöver alltså enbart hålla ordning på x och y-värdet på maskdelen, och klassen ska dessutom kunna rita ut sig själv.

```
Mask1 ▾
1 class MaskDel {
2     float maskx;
3     float masky;
4
5     MaskDel(float x, float y) {
6         maskx=x;
7         masky=y;
8     }
9
10    void rita() {
11        fill(0, 0, 255);
12        ellipse(maskx, masky, 30,30);
13    }
14 }
15
```

Varje maskdel ska alltid ritas ut på samma ställe, och istället ska vi rita ut en ny del på ena sidan och ta bort en på andra sidan för att få det se ut som att den rör på sig. Därför skapar vi längst upp i vårt program globala variabler. Vi behöver variabler för maskens hastighet, och för x och y värdena på positionen där nästa maskdel ska ritas ut, och vi skapar också en ArrayList där vi ska lagra våra maskdelar.

```
Mask2 ▾
1 ArrayList<MaskDel> maskDelar = new ArrayList();
2 float speedx=5;
3 float speedy=0;
4 float x=30;
5 float y=300;
6
```

Vi skapar en ny maskdel varje gång som drawmetoden körs, och vi ritar också ut varje maskdel varje gång drawmetoden körs.

```
7 void setup() {
8     size(600, 600);
9     frameRate(20);
10 }
11
12 void draw() {
13     background(0);
14     x = x + speedx;
15     y = y + speedy;
16     MaskDelar.add(new MaskDel(x, y));
17     for (int i=0; i < MaskDelar.size(); i++) {
18         MaskDel b = MaskDelar.get(i);
19         b.rita();
20     }
21 }
22
```

Med vårt program så här långt så får vi en mask som blir längre och längre och tillslut åker ut ur rutan. Vi behöver alltså ta bort baken på masken så fort den har nått full längd. Så vi lägger till följande kod för att ta bort den maskdel som vi lade till först.

```
12 void draw() {  
13     background(0);  
14     x = x + speedx;  
15     y = y + speedy;  
16     maskDelar.add(new MaskDel(x, y));  
17     if (maskDelar.size()>20) {  
18         maskDelar.remove(0);  
19     }  
20     for (int i=0; i < maskDelar.size(); i++) {  
21         MaskDel b = maskDelar.get(i);  
22         b.rita();  
23     }  
24 }
```

Hela programmet blir då:

```
Mask3 ▾  
1 ArrayList<MaskDel> maskDelar = new ArrayList();  
2 float speedx=5;  
3 float speedy=0;  
4 float x=30;  
5 float y=300;  
6  
7 void setup() {  
8     size(600, 600);  
9     frameRate(20);  
10 }  
11  
12 void draw() {  
13     background(0);  
14     x = x + speedx;  
15     y = y + speedy;  
16     maskDelar.add(new MaskDel(x, y));  
17     if (maskDelar.size()>20) {  
18         maskDelar.remove(0);  
19     }  
20     for (int i=0; i < maskDelar.size(); i++) {  
21         MaskDel b = maskDelar.get(i);  
22         b.rita();  
23     }  
24 }  
25  
26  
27 class MaskDel {  
28     float maskx;  
29     float masky;  
30  
31     MaskDel(float x, float y) {  
32         maskx=x;  
33         masky=y;  
34     }  
35  
36     void rita() {  
37         fill(0, 0, 255);  
38         ellipse(maskx, masky, 30,30);  
39     }  
40 }
```

22 Rekursiva funktioner

Med rekursiva så brukar vi mena funktioner som anropar sig själv. Vi demonstrerar det hela med ett exempel.

Vi skapar en funktion som tar en diameter som inparameter som som ritar ut en cirkel, och som sedan anropar sig själv med diametern delat med två.

```
void cirkel(float d){  
    ellipse(width/2, height/2, d, d);  
    cirkel(d/2);  
}
```

Vi behöver då enbart en setup-metod som anropar funktionen en första gång.



Men ni kanske märker att programmet hänger sig och efter ett tag så dyker ett felmeddelande upp:

```
A StackOverflowError means that you have a bug that's causing a function  
to be called recursively (it's calling itself and going in circles),  
or you're intentionally calling a recursive function too much,  
and your code should be rewritten in a more efficient manner.
```

Felet beror på att metoden kommer att anropa sig själv i all oändlighet. Vi måste ha något villkor som stoppar rekursionen. Jag väljer att cirkel enbart ska anropa sig själv om diametern är större än en pixel. Det blir en enkel if-sats.



22.1 Kvadrater i kvadrat

Ett litet mer avancerat exempel skulle kunna vara att försöka rita rektanglar i rektanglar i rektanglar

Vi börjar med att skapa ett mycket enklare program. Vi skapar en funktion som ritar ut en rektangel.

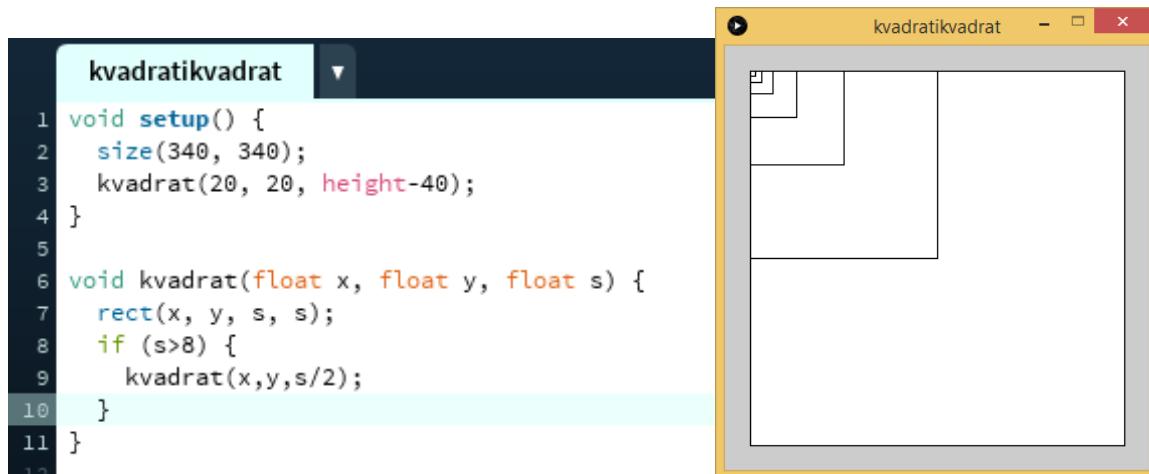


The screenshot shows the Processing IDE with a sketch titled "kvadratikvadrat". The code is as follows:

```
kvadratikvadrat
1 void setup() {
2     size(340, 340);
3     kvadrat(20, 20, height-40);
4 }
5
6 void kvadrat(float x, float y, float s) {
7     rect(x, y, s, s);
8 }
9
```

The output window shows a single square centered at approximately (170, 170) with a side length of 30 pixels.

Men vi ville ju rita ut kvadrater i kvadraterna, så vi kör funktionen inifrån funktionen, fast med halva sidstorleken.



The screenshot shows the Processing IDE with a sketch titled "kvadratikvadrat". The code has been modified to include a recursive call:

```
kvadratikvadrat
1 void setup() {
2     size(340, 340);
3     kvadrat(20, 20, height-40);
4 }
5
6 void kvadrat(float x, float y, float s) {
7     rect(x, y, s, s);
8     if (s>8) {
9         kvadrat(x,y,s/2);
10    }
11 }
12
```

The output window shows a fractal-like pattern of nested squares. The main square is divided into four quadrants, with the top-left quadrant further subdivided into four smaller squares, and so on, creating a recursive structure.

Men det blir enbart rektanglar i övre vänstra hörnet. Vi vill ha mer rektanglar, så vi ritar ut rektanglar i först i det övre högra hörnet, och sedan i det nedre vänstra hörnet. Då blir det lite coolare.

The screenshot shows a Processing sketch window titled "kvadratikvadrat". On the left is the code editor with the following code:

```

1 void setup() {
2   size(340, 340);
3   kvadrat(20, 20, height-40);
4 }
5
6 void kvadrat(float x, float y, float s) {
7   rect(x, y, s, s);
8   if (s>8) {
9     kvadrat(x,y,s/2);
10    kvadrat(x+s/2, y, s/2);
11    kvadrat(x, y+s/2, s/2);
12  }
13 }

```

On the right is the preview window showing a fractal square pattern composed of smaller squares.

Ännu vackrare blir det om vi sätter färg på rektanglarna. Att jag multiplicerar sidans längd med just 5 och sedan lägger till 50 är något som jag testat mig fram till.

The screenshot shows a Processing sketch window titled "kvadratikvadrat2". On the left is the code editor with the following code:

```

1 void setup() {
2   size(340, 340);
3   kvadrat(20, 20, height-40);
4 }
5
6 void kvadrat(float x, float y, float s) {
7   fill(0, 0, 5*s+50);
8   rect(x, y, s, s);
9   if (s>8) {
10    kvadrat(x,y,s/2);
11    kvadrat(x+s/2, y, s/2);
12    kvadrat(x, y+s/2, s/2);
13  }

```

On the right is the preview window showing a fractal square pattern where each square is filled with a blue color determined by the formula $5*s+50$.

Testa gärna att öka storleken till 1000×1000 .

23 Animerad rekursion

Det skulle vara effektfullt om man skulle kunna rita ut bilden efter varje steg i den rekursiva algoritmen ovan. Problemet är att bilden bara ritas ut en gång efter att setup-metoden har körts och sedan varje gång draw-metoden körs. Vi skulle vilja köra ett steg i rekursionen för varje gång draw metoden körs. Då kan vi inte använda rekursion på det enkla sättet som visades ovan. Istället så vill vi köra metoden och sedan spara undan information om nästa steg på en Stack.

Det vi behöver spara på stacken är både x, y och sidan (s). Hur gör vi då för att spara alla dessa tre i en enda Stack. Jo vi har ju lärt oss att använda klasser och objekt och detta är en bra tillämpning. Vi har tre egenskaper och en metod som ritar ut kvadraten och sedan lägger tre nya objekt på stacken.

Vi börjar med att skapa en klass med egenskaperna x, y och s

```
15 class Kvadrat {  
16     float x;  
17     float y;  
18     float s;  
19     Kvadrat(float x, float y, float s) {  
20         this.x=x;  
21         this.y=y;  
22         this.s=s;  
23     }  
24 }
```

Sedan tar vi och klistrar in vår metod kvadrat ovan in i vår klass, och döper om den till rita

```
15 class Kvadrat {  
16     float x;  
17     float y;  
18     float s;  
19     Kvadrat(float x, float y, float s) {  
20         this.x=x;  
21         this.y=y;  
22         this.s=s;  
23     }  
24  
25     void rita(float x, float y, float s) {  
26         int b = (int)(5*s+50);  
27         fill(0, 0, b);  
28         rect(x, y, s, s);  
29         if (s>8) {  
30             rektangel(x,y,s/2);  
31             rektangel(x+s/2, y, s/2);  
32             rektangel(x, y+s/2, s/2);  
33         }  
34     }  
35 }
```

Nu vill vi skapa nya Kvadratobjekt och lägga dem på en stack istället för att köra rektangel-metoden.

Vi börjar med att skapa en Stack längst upp i filen

```
kvadratikvadrat2animStack ▾  
1 import java.util.Stack;  
2 Stack<Kvadrat> stack = new Stack<Kvadrat>();
```

Sedan ser vi till att skapa Kvadrat objekt och lägga dem på stacken.

```

16 class Kvadrat {
17     float x;
18     float y;
19     float s;
20     Kvadrat(float x, float y, float s) {
21         this.x=x;
22         this.y=y;
23         this.s=s;
24     }
25
26     void rita() {
27         fill(0, 0, 5*s+50);
28         rect(x, y, s, s);
29         if (s>8) {
30             stack.push(new Kvadrat(x, y, s/2));
31             stack.push(new Kvadrat(x+s/2, y, s/2));
32             stack.push(new Kvadrat(x, y+s/2, s/2));
33         }
34     }
35 }
36

```

I vår setup metod måste vi skapa vår första Kvadrat och köra rita metoden på den. (rita-metoden kommer nu att lägga tre nya kvadrater på stacken).

```

5 void setup() {
6     size(1000, 1000);
7     Kvadrat k = new Kvadrat(20, 20, height-40);
8     k.rita();
9 }

```

Det enda som vi behöver göra i draw-metoden är att plocka ut dvs pop:a en i taget och köra rita-metoden på den.

```

11 void draw() {
12     Kvadrat k = stack.pop();
13     k.rita();
14 }

```

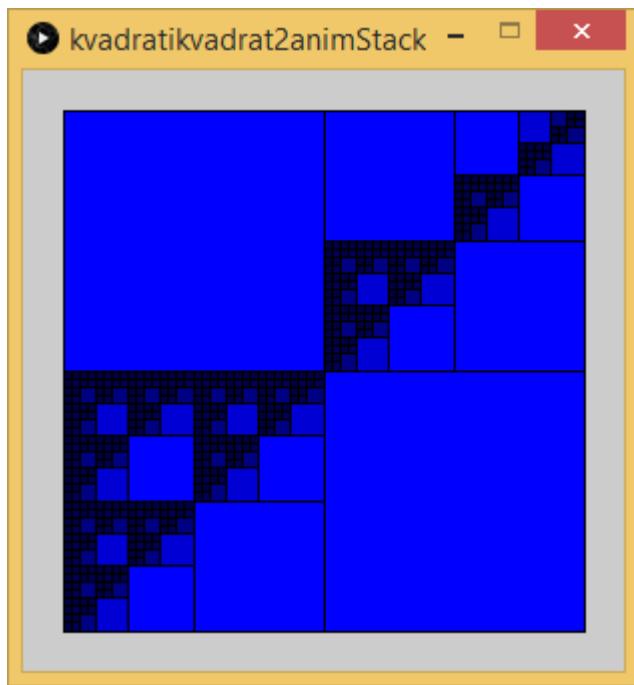
Testkör programmet.

Allt fungerar bra till dess ett **EmptyStackException** dyker upp. Stacken blir tom. Vi borde kolla så att stacken inte är tom när vi försöker poppa ett objekt.

```

11 void draw() {
12     if (!stack.empty()) {
13         Kvadrat k = stack.pop();
14         k.rita();
15     }
16 }

```



Hela programmet blir:

```

kvadratikvadrat2animStack ▾
1 import java.util.Stack;
2 Stack<Kvadrat> stack = new Stack<Kvadrat>();
3
4
5 void setup() {
6     size(300, 300);
7     Kvadrat k = new Kvadrat(20, 20, height-40);
8     k.rita();
9 }
10 void draw() {
11     if (!stack.empty()) {
12         Kvadrat k = stack.pop();
13         k.rita();
14     }
15 }
16 class Kvadrat {
17     float x;
18     float y;
19     float s;
20     Kvadrat(float x, float y, float s) {
21         this.x=x;
22         this.y=y;
23         this.s=s;
24     }
25
26     void rita() {
27         fill(0, 0, 5*s+50);
28         rect(x, y, s, s);
29         if (s>8) {
30             stack.push(new Kvadrat(x, y, s/2));
31             stack.push(new Kvadrat(x+s/2, y, s/2));
32             stack.push(new Kvadrat(x, y+s/2, s/2));
33         }
34     }
35 }

```

Övning 23-1 Övning gör din egen stack

Programmet ovan fungerar bra i den vanliga processingmiljön. Men vill man köra den på webben får man problem eftersom processing.js inte har någon inbyggd stack. Gör därför en klass Stack som fungerar på samma sätt som javas Stack. Den behöver minst ha metoderna push, pop och empty.

24 Att gå till Java från Processing - Skillnader mellan Java och Processing

24.1 Mainklass och main-metod

I Java ligger all kod i en klass, och varje program har en main(metod).

All kod i java måste ligga i en klass. Men när man är ny i java så har det ingen större praktisk betydelse. Det blir mest som en rubrik man sätter i början på sitt program.

```
public class MittForstaJavaProjekt {  
    //Här kan du skriva ditt program  
}
```

Varje program har en main-metod, som ligger inuti en av klasserna. main-metoden måste alltid deklareras som `public static void main(String[] args)`

```
public class MittForstaJavaProjekt {  
    public static void main(String[] args) {  
        //Här kan du skriva din kod.  
    }  
}
```

24.2 Skriva ut någonting

I bland vill man skriva ut någonting till *Output-fönstret* respektive *Kommandotolken*, beroende på om programmet körs från Netbeans eller direkt från *Kommandotolken*. Det gör man i Java genom att skriva `System.out.println(<Text>);`

Till exempel:

```
System.out.println("Hej på er alla glada elever");
```

Vill man göra det enkelt för sig så kan man skriva sout och sedan trycka tab-tangenten så fyller den automatiskt i resten.

24.3 Datatyper

24.3.1 double istället för float

Eftersom Processing i princip är Java så är det samma datatyper i Java och Processing. Däremot använder man nästan alltid double istället för float när man programmerar Java. Främst för att alla inbyggda matematiska funktioner kräver att man använder double.

Typen color som finns i Processing finns inte i Java.

24.3.2 Omvandlingar

Omvandling	Processing	Java och Processing
Från String till int	int(str)	Integer.parseInt(str) alt. Integer.valueOf(str)
Från String till float	float(str)	Float.parseFloat(str) alt. Float.valueOf(str)
Från String till double	double(str)	Double.parseDouble(str) alt. Double.valueOf(str)
Från float till int	int(tal)	(int) tal exempelvis: int a=(int)b;
Från till double till int	int(tal)	(int) tal exempelvis: int a=(int)b;

24.4 Funktioner och Metoder

Funktioner kallas oftast för metoder i java eftersom de alltid hör ihop med en klass. Vill man använda metoder/funktioner på liknande sätt som vi gjort hittills och anropa den direkt från main-metoden behöver lägga till ordet static framför.

```
int kvadrera(int varde){
    int kvadrat = varde * varde;
    return kvadrat;
}
```

blir

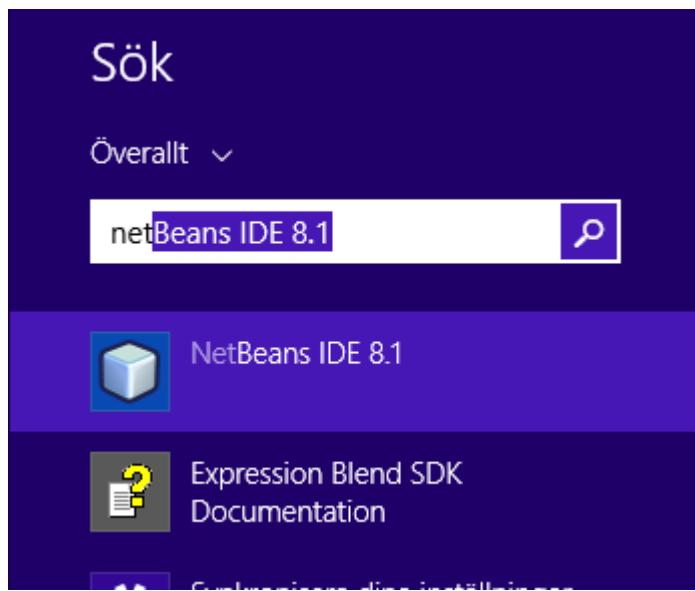
```
static int kvadrera(int varde){
    int kvadrat = varde * varde;
    return kvadrat;
}
```

25 Dina första Java program

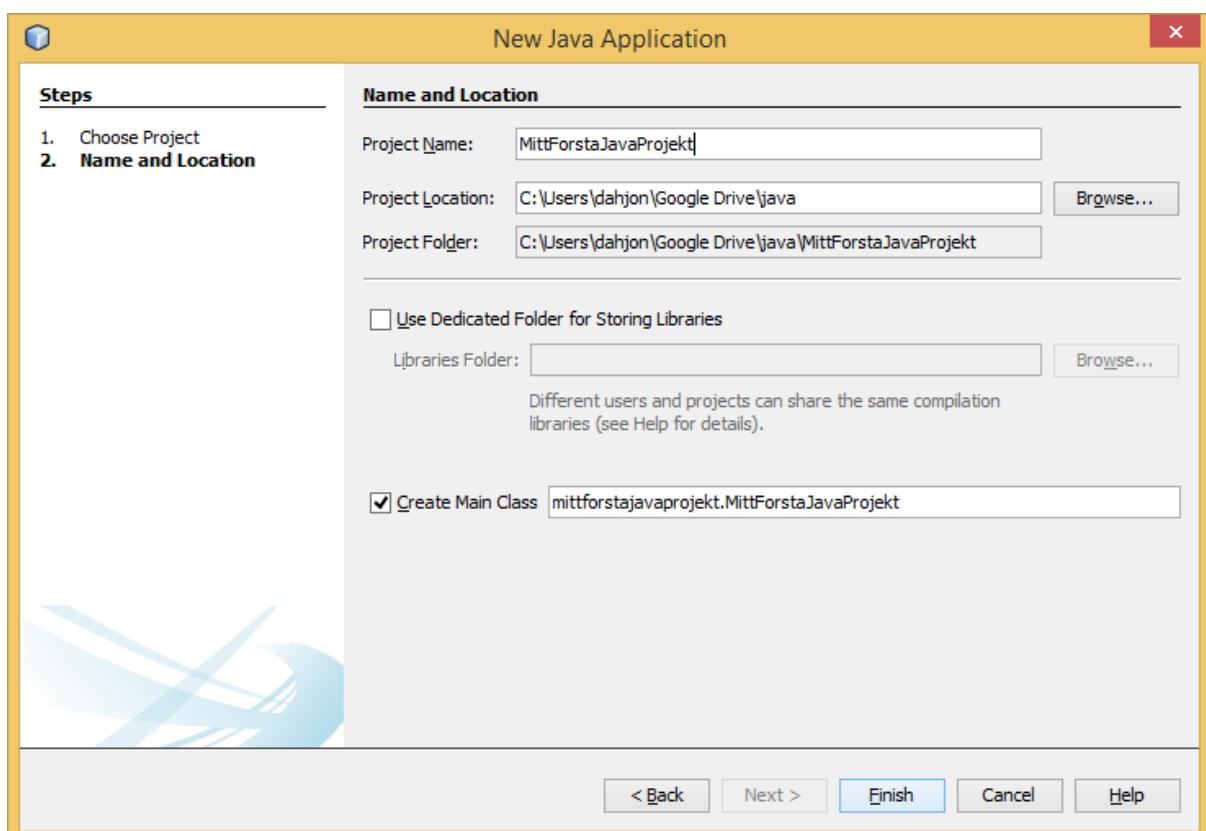
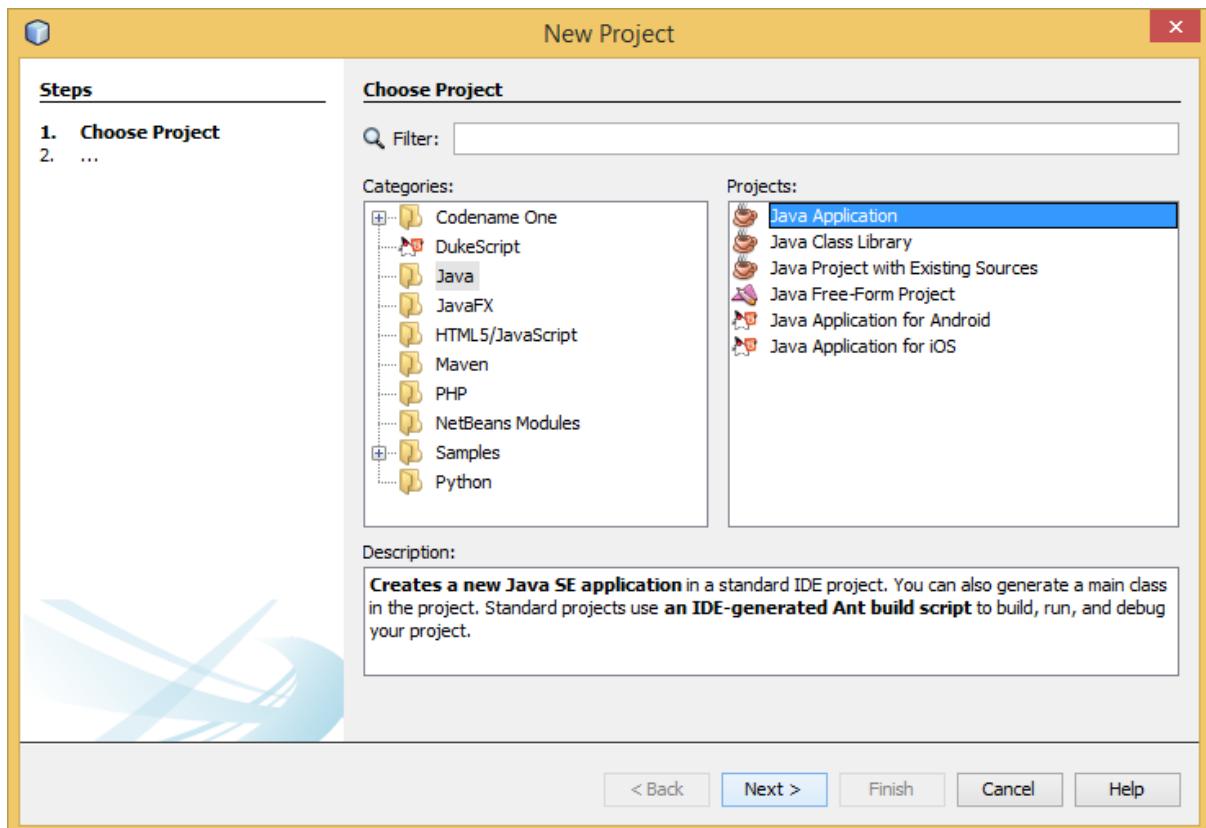
25.1 Att skapa ett enkelt program som enbart skriver ut några ord.

Börja med att starta netbeans

Tryck på startknappen och sök på netbeans



Skapa ett nytt projekt



Tryck på finish

Du kommer då att få upp följande:

```
5  L  */
6  package mittforstajavaprojekt;
7
8  /**
9   *
10  * @author dahjon
11  */
12 public class MittForstaJavaProjekt {
13
14  /**
15   * @param args the command line arguments
16  */
17  public static void main(String[] args) {
18      // TODO code application logic here
19  }
20
21 }
22
```

För att sedan få programmet att skriva ut någonting så får vi lägga till

```
System.out.println("Hello World");
```

```
12  public class MittForstaJavaProjekt {
13
14  /**
15   * @param args the command line arguments
16  */
17  public static void main(String[] args) {
18      System.out.println("Hello World");
19  }
20
21 }
22
```

25.2 Program med enkel inmatning med JOptionPane

Ni kommer ihåg JOptionPane som vi använde med Processing. Vi använder den för att fråga efter ett namn, och skriver sedan Hej: *Namn*.

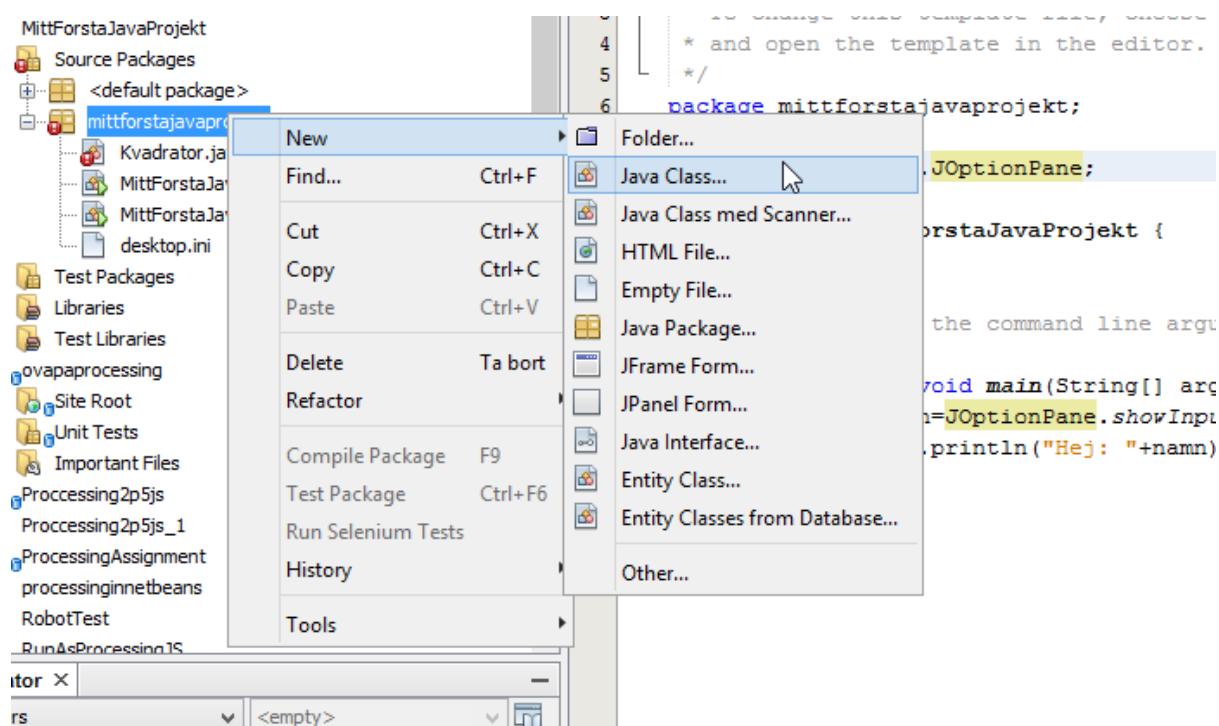
```

8  import javax.swing.JOptionPane;
9
10 public class MittForstaJavaProjekt {
11
12     /**
13      * @param args the command line arguments
14     */
15     public static void main(String[] args) {
16         String namn=JOptionPane.showInputDialog("Ange ditt namn");
17         System.out.println("Hej: "+namn);
18     }
19
20 }

```

Ett enkelt program med inmatning av tal

Vi högerklickar på paketet (mappen) mittforstajavaprojekt, och väljer New->Java Class



och väljer att skapa en klass Kvadrator.

Vi skriver *psvm* och trycker *tab*.

```

public class Kvadrator {
    psvm
}

```

```
11 |  /* */
12 |  public class Kvadrator {
13 |      public static void main(String[] args) {
14 |      }
15 |  }
```

Nu har vi fått vår main-metod som där vi kan skriva vår kod. Vi använder JOptionPane för att mata in en sträng som vi sedan konverterar till ett heltal med hjälp av Integer.valueOf() som gör samma sak som int() gjorde i Processing.

```
public class Kvadrator {
    public static void main(String[] args) {
        String str = JOptionPane.showInputDialog("Ange ett tal:");
        int tal = Integer.valueOf(str);
    }
}
```

Vi lägger sedan till en rad som utför själva kvadreringen och skriver sedan ut den med System.out.println()

```
public class Kvadrator {
    public static void main(String[] args) {
        String str = JOptionPane.showInputDialog("Ange ett tal:");
        int tal = Integer.valueOf(str);
        int produkt = tal*tal;
        System.out.println(tal+" kvadrerat blir "+produkt);
    }
}
```

25.3 Enkel inmatning i konsolen.

Det finns en färdig klass som heter Scanner som vi kan använda för att mata in information till vårt program.

Vi måste först skapa vårt scanner objekt genom att skriva.

```
Scanner scan = new Scanner(System.in);
```

Sedan kommer vi till själva inläsningen:

```
String namn = scan.next();
```

```

14  public class MittForstaJavaProjekt {
15
16      /**
17      * @param args the command line arguments
18      */
19      public static void main(String[] args) {
20          Scanner scan = new Scanner(System.in);
21          String namn=scan.next();
22          System.out.println("Hej: "+namn);
23      }
24
25  }

```

Vi kommer också behöva beskriva för användaren vad hen ska göra, så vi lägger till en rad:

```

14  public class MittForstaJavaProjekt {
15
16      /**
17      * @param args the command line arguments
18      */
19      public static void main(String[] args) {
20          Scanner scan = new Scanner(System.in);
21          System.out.println("Ange ditt namn");
22          String namn=scan.next();
23          System.out.println("Hej: "+namn);
24      }
25
26  }

```

25.3.1 Sammanfattning om Scanner:

Skapa scannerobjekt:

```
Scanner scan= new Scanner(System.in);
```

Inmatning

Typ	Kodrad
int	int tal=scan.nextInt();
float	float tal=scan.nextFloat();
double	double tal=scan.nextDouble();
String (ett ord)	String str = scan.next();

String (en rad)	String str = scan.nextLine();
-----------------	-------------------------------

25.4 Trevliga förkortningar i Netbeans

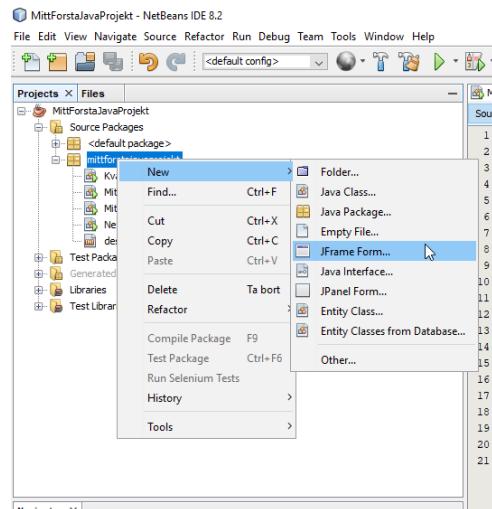
```
sout + tab -> System.out.println("");  
psvm + tab -> public static void main(String[] arg){  
    }  
}
```

Shift+F6 För att köra aktuell fil

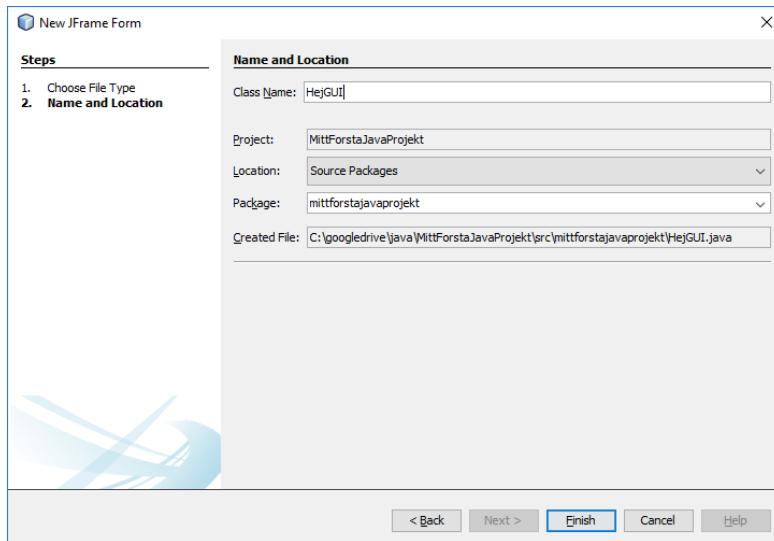
26 Skapa program med grafiska användargränssnitt i Java och netbeans

En sak som är trevlig med netbeans är att man enkelt kan skapa grafiska användargränssnitt med hjälp av en inbyggd editor. Man kan helt enkelt rita upp användargränssnittet.

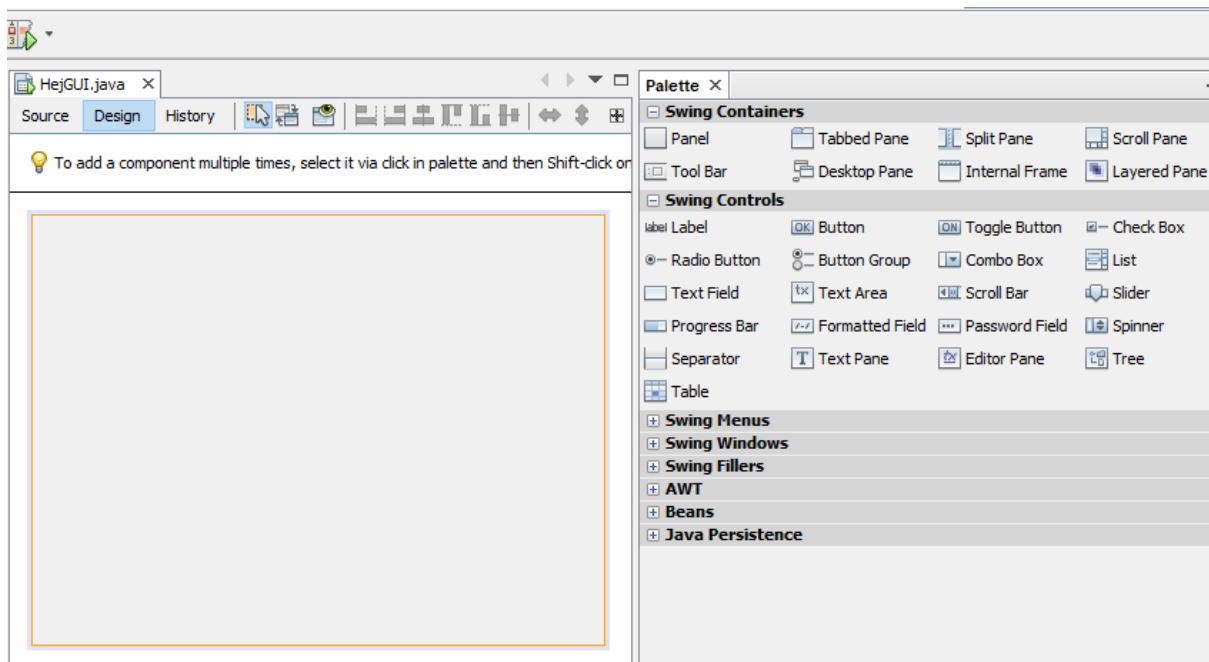
För att börja skapa ett användargränsnittsfönster så högerklickar du paketet *mittforstaprojekt* i projektet *MittForstaProjekt*, och väljer *New->JFrame Form*. (Se till att inte förväxla den med JPanel form)



Vi kallar filen för HejGUI.

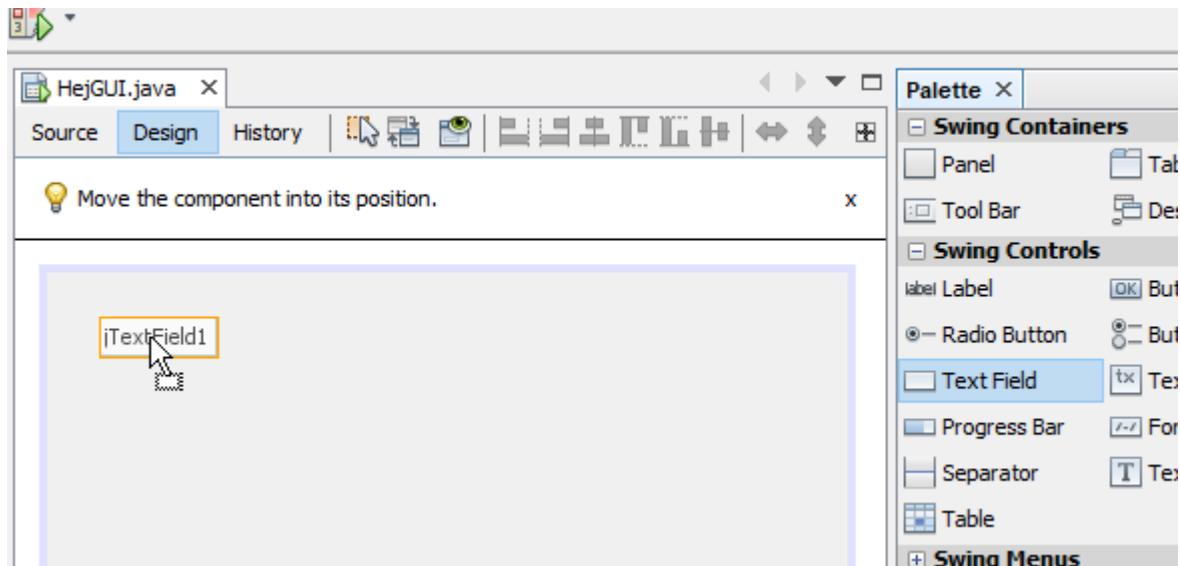


Då dyker följande upp:

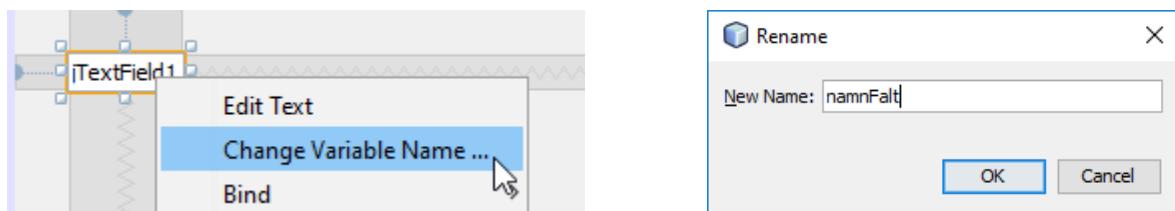


Till vänster har vi vårt fönster som fungerar som en rityta och till höger en palett med komponenter som vi kan dra in i vårt fönstret.

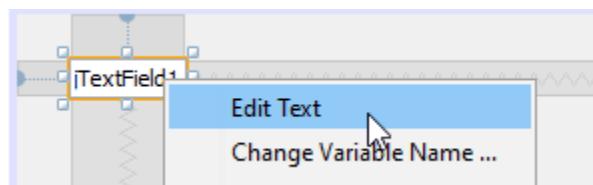
Nu tar vi och drar ett TextField in i fönstret:



Vi högerklickar sedan och väljer *Change variable name*. Vi kan sättat namnet till *namnFält*.



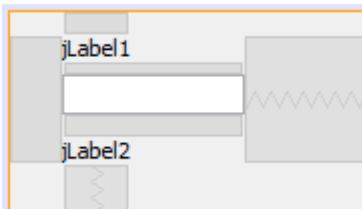
Och sedan högerklickar vi igen och väljer *Edit text*, och tömmer textfältet.



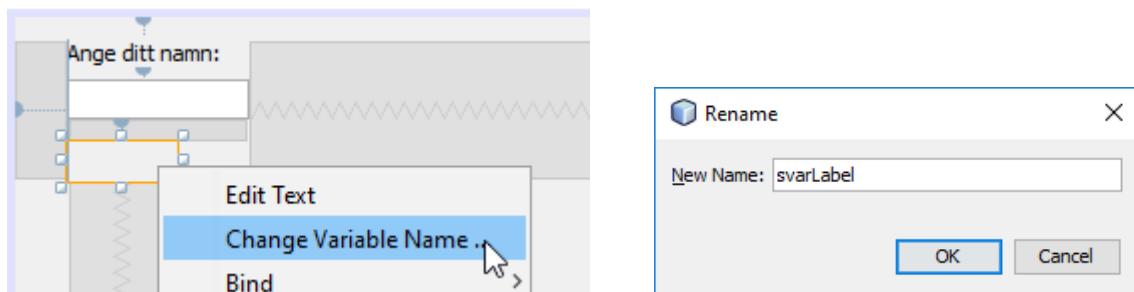
Men då blev det visst pyttelitet så vi får ta och dra ut textfältet.



För att visa text i vår applikation så använder vi labels. Label betyder helt enkelt Etikett på engelska.



Den översta etiketten ska enbart innehålla en instruktion som inte ska ändras, medan den nedre ska innehålla programmets respons på inmatningen, så den kan vara tom från början. Variabelnamnet på den översta labeln som inte ska ändra spelar ingen roll, medan namnet på den nedre ska ändras till något vettigt till exempel svarLabel.



För att koppla kod till Textfältet så tar vi nu och dubbelklickar på textfältet. Då skapas en metod som kommer att köras när användaren trycker *Enter* i textfältet.

```

70
71  private void namnFaltActionPerformed(java.awt.event.ActionEvent evt) {
72      // TODO add your handling code here:
73  }
74
75 /**

```

Nu har vi skapat större delen av vårt program utan att skriva en rad. Men nu ska vi hämta texten från vårt textfält och skriva ett svar till användaren i vår label.

För att hämta namnet använder vi `getText()`

```

71  private void namnFaltActionPerformed(java.awt.event.ActionEvent evt) {
72      String namn = namnFalt.getText();
73
74 }

```

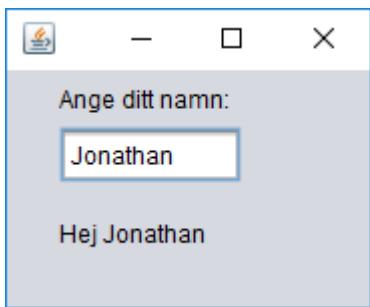
Och för att sätta text använder vi metoden `setText()`

```

71  private void namnFaltActionPerformed(java.awt.event.ActionEvent evt) {
72      String namn = namnFalt.getText();
73      svarLabel.setText("Hej " + namn);
74
75 }

```

När det är färdigt ser det ut så här!



Övning 26-1 Övning Fahrenheit till Celsius i java

Gör om övningen som vi hade tidigare som handlade om att omvandla från fahrenheit till Celsius. Programmet ska ha ett textfält där användaren kan mata in en temperatur i fahrenheit och sedan ska temperaturen skrivas ut i Celsius. Formeln är $^{\circ}\text{C} = (^{\circ}\text{F} - 32) \cdot 5/9$. Använd typen `double` för att lagra temperaturen i fahrenheit

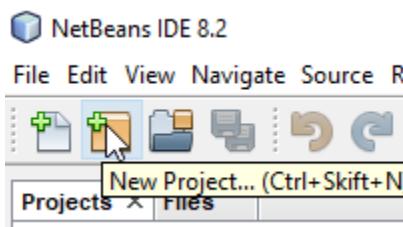
Övning 26-2 Övning summera två tal.

Skapa ett program som har två textfält och två eller tre labels, där den nedersta labeln ska visa summan av två tal så fort man trycker på enter på det nedre textfältet.

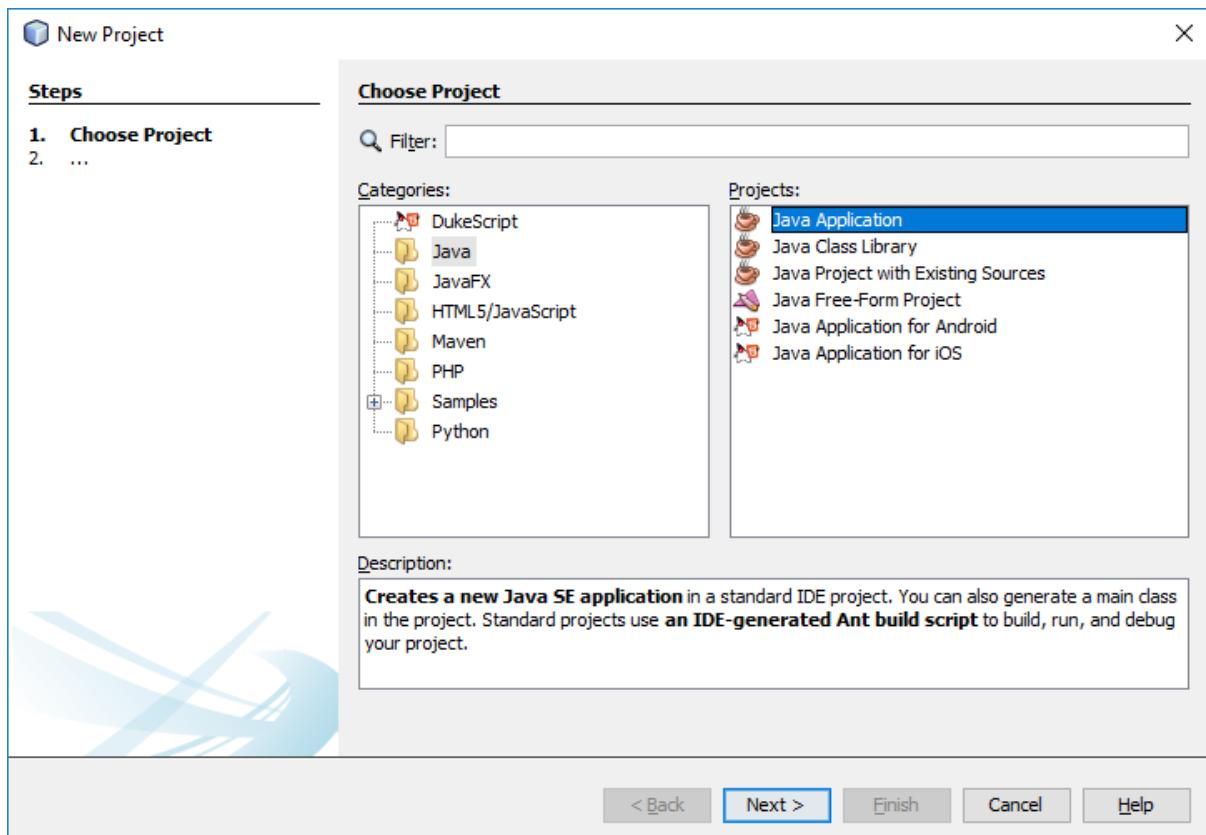
26.1 Sten Sax Påse-spel

För att träna mer på netbeans och Java så ska vi skapa ett enkelt Sten, Sax och påse spel.

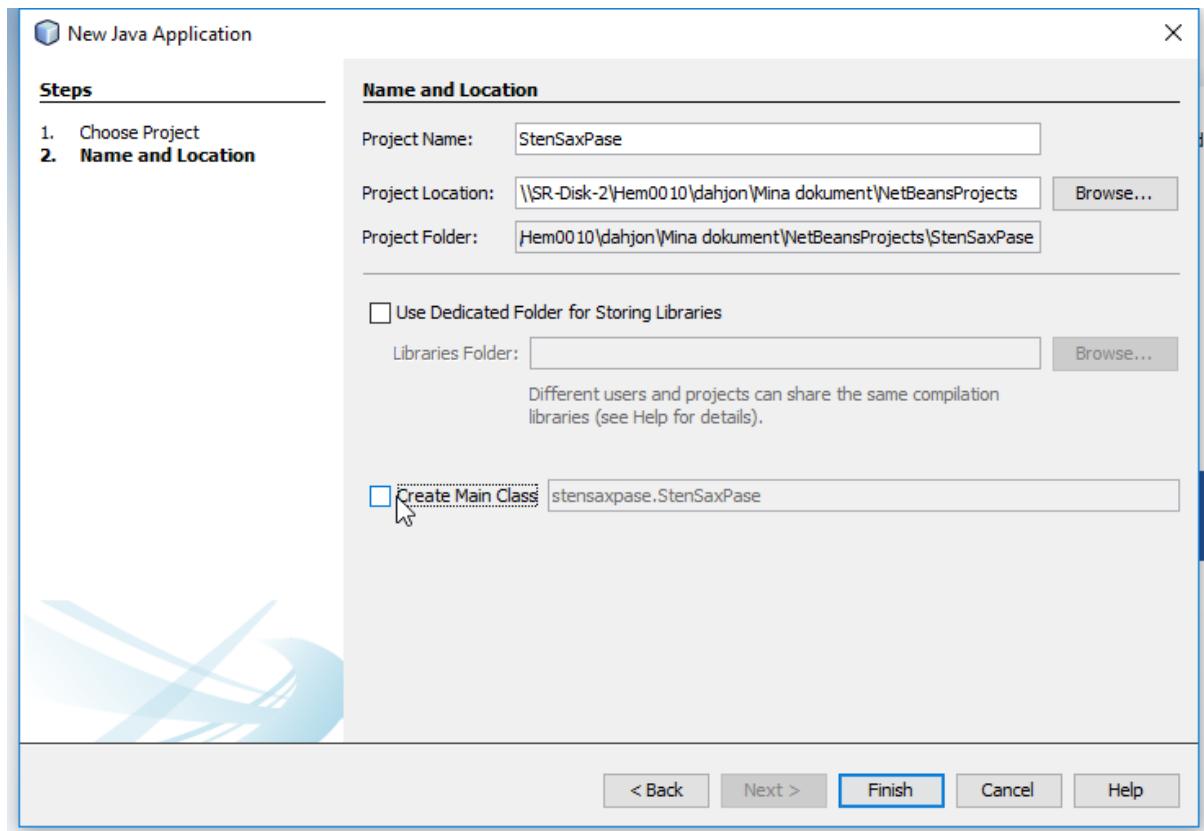
Börja med att skapa ett nytt projekt



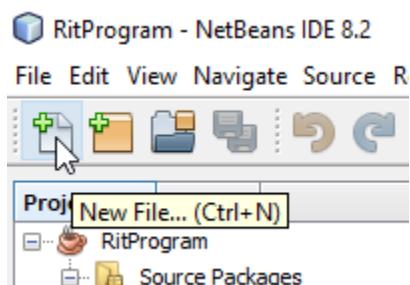
Välj Java Application



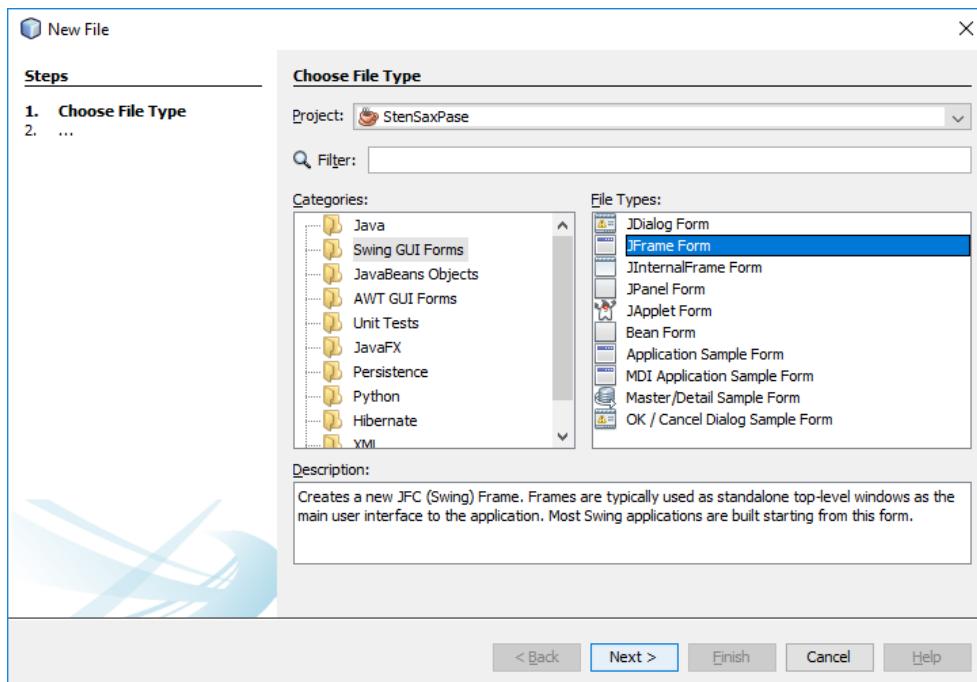
Döp projektet till *StenSaxPase* och klicka bort *CreateMainClass*.



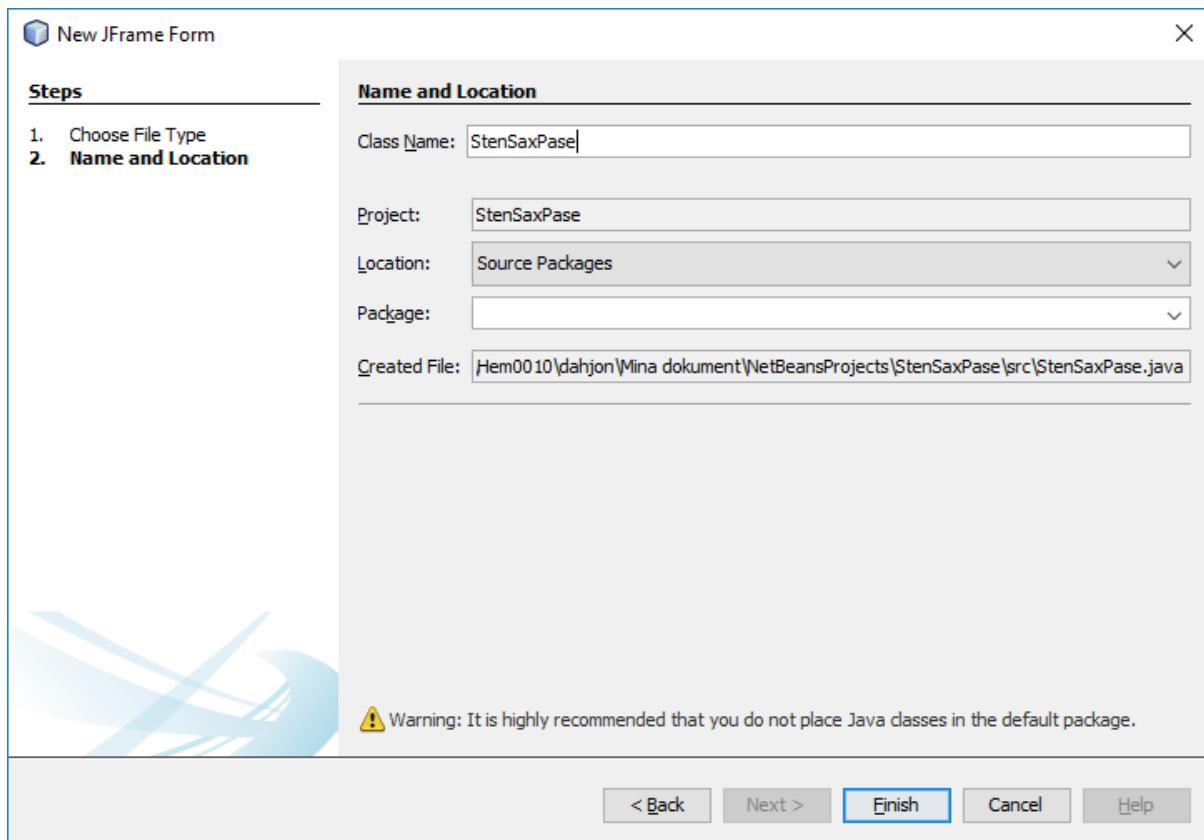
Vi trycker ni på knappen *New File* igen



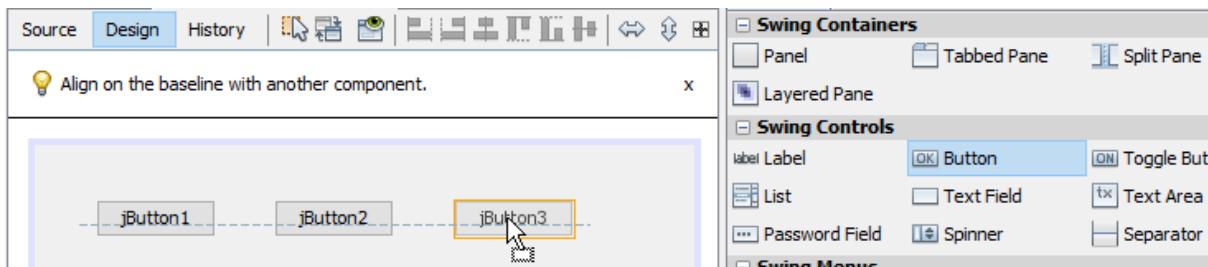
Väljer typen *JFrame Form*



Och döper filen till samma som projektet: StenSaxPase

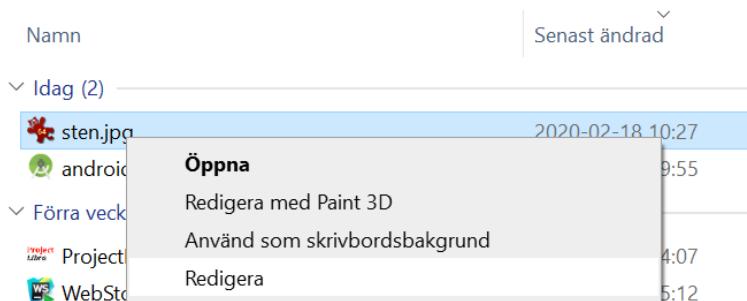


Vi drar in tre knappar till fönstertytan.

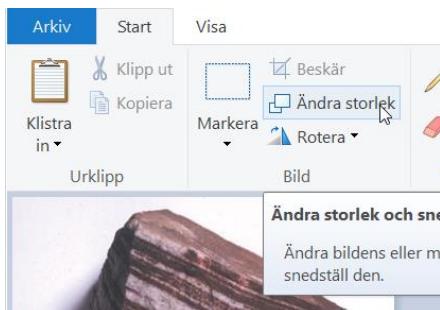


Nu skulle vi vilja lägga till bilder till knapparna, och då behöver vi för det första bilder. Alla bilder som jag använder här är hämtade från Wikipedia¹.

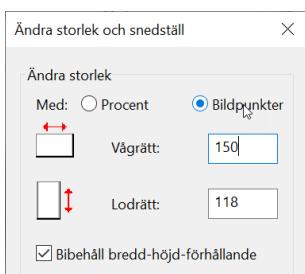
Vi öppnar bilderna i Paint för att göra dem lagom stora.



Vi trycker på *Ändra storlek*



Vi väljer att ändra storlek med *Bildpunkter*, och fyller i 150 i vågrätt.

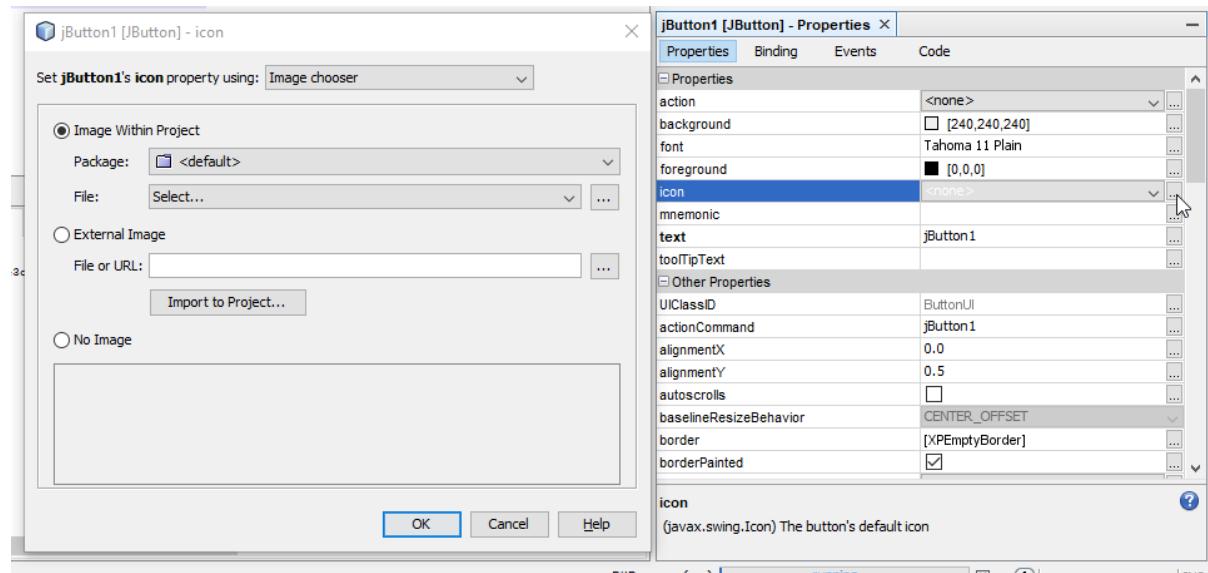


Vi sparar sedan bilden med ett bra namn, det vill säga *sten.jpg*, för stenbilden, *sax.jpg*, för

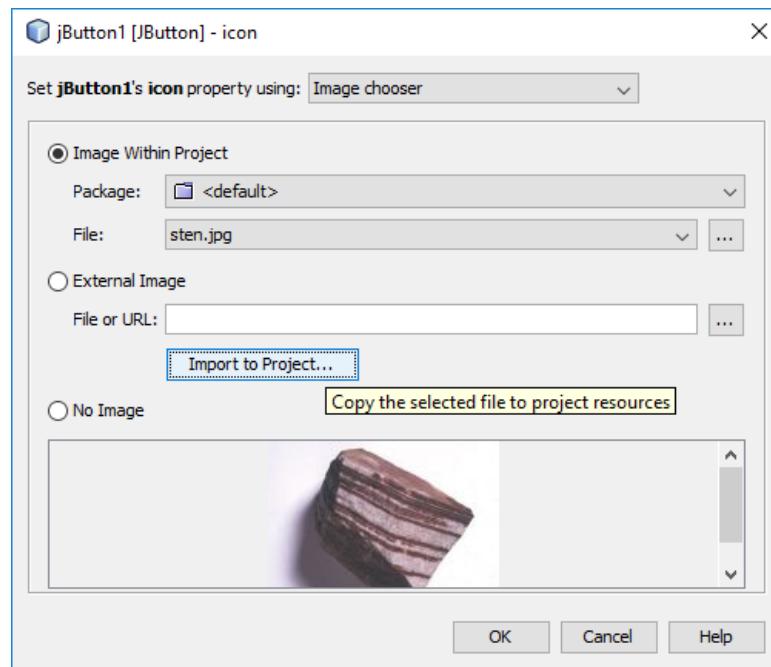
¹ <https://sv.wikipedia.org/wiki/B%C3%A4rkasse>,
[https://en.wikipedia.org/wiki/Rock_\(geology\)#/media/File:SandstoneUSGOV.jpg](https://en.wikipedia.org/wiki/Rock_(geology)#/media/File:SandstoneUSGOV.jpg)
https://sv.wikipedia.org/wiki/Sax#/media/File:Handsax,_Nordisk_familjebok.png

saxbilden och pase.jpg för påsebilden.

Gå till propertiespanelen i Netbeans som normalt finns i det nedre högra hörnet. Leta rätt på *icon* och tryck på knappen med de tre punkterna (se bild nedan)



Tryck på *Import to project*, och välj din stenbild.



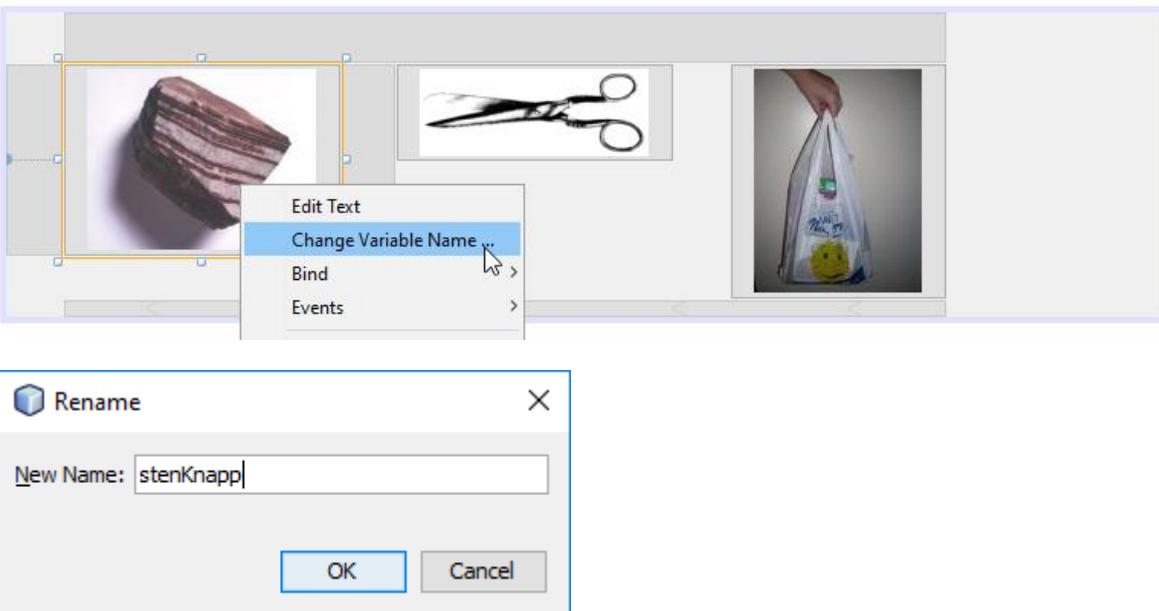
Gör på motsvarande sätt för saxen och påsen. Det vill säga ändra storleken i Paint och lägg sedan till den till de andra knapparna i Netbeans.



Vi ska nu ta bort texten på varje knapp.

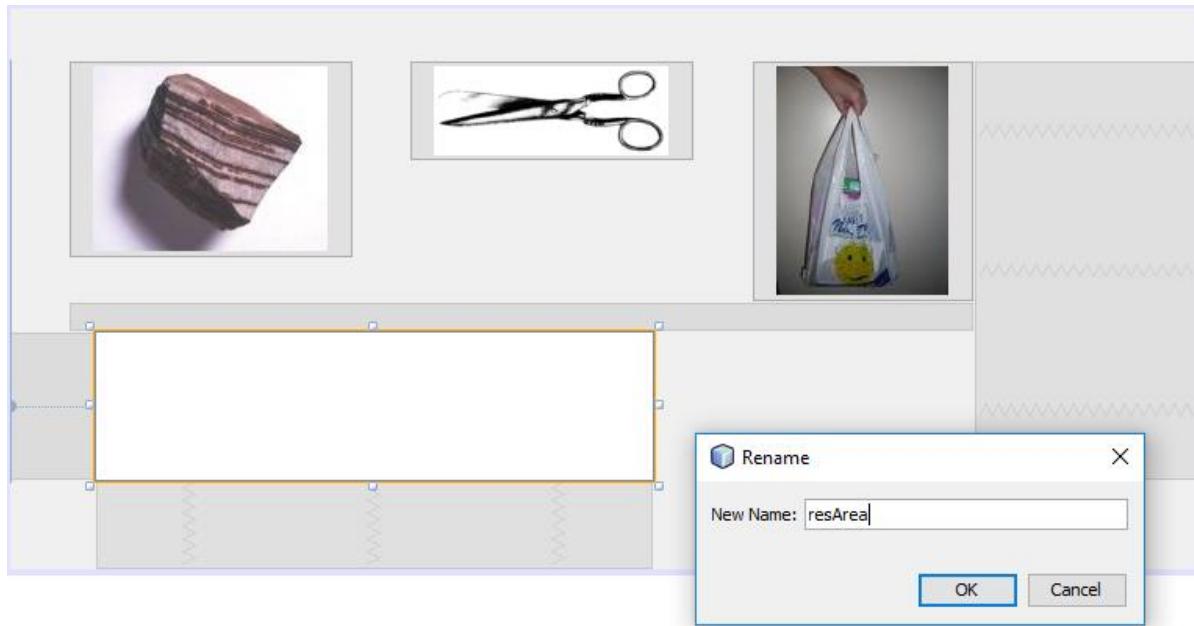


Och sätta bra variabelnamn på knapparna, så att knapparna heter *stenKnapp*, *saxKnapp* och *paseKnapp*.

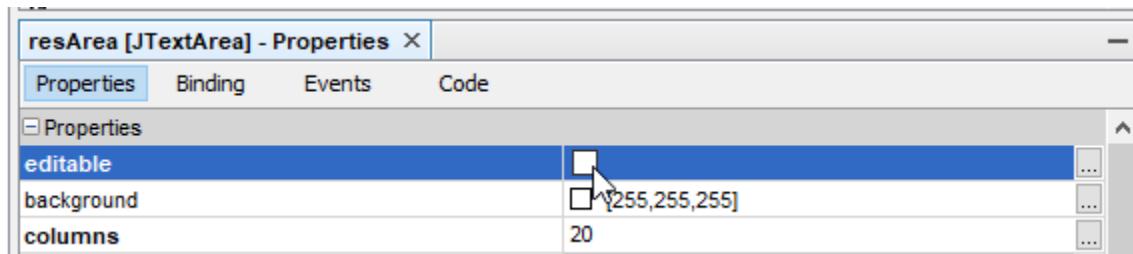


Den sista komponenten som vi behöver i användargränssnittet är en textarea, där vi kan skriva ut resultatet. En JTextarea skiljer sig från JTextField, och JLabel genom att den kan hantera flera rader, och precis som JTextField så kan den användas för input. Men denna

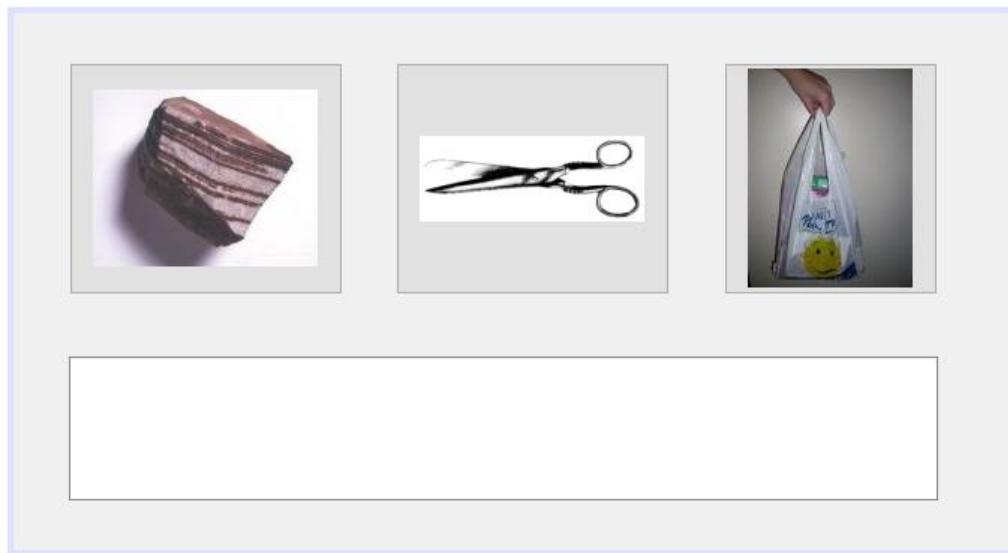
gång ska vi enbart använda den för output, och ska vi sätta editable till false.



Vi vill inte att man ska kunna skriva i resultat-textarean. Det löser vi enklast genom att sätta editable till false, vilket görs genom att gå till propertiespanelen och tar bort bocken framför editable.



Det sista jag gör på gränssnittet är att snygga till knapparna, och textfältet. Detta kan man göra genom att helt enkelt dra i sidan på knapparna, respektive textfältet.



Nu är vi färdiga med gränssnittet. Nu är det dags att börja skriva kod. Då börjar vi med att dubbelklicka på stenknappen, för att skapa en metod som kommer att köras när vi klickar på knappen.

```
84  
85  private void stenKnappActionPerformed(java.awt.event.ActionEvent evt) {  
86      // TODO add your handling code here:  
87  }  
88
```

Vi börjar med att skapa en funktion för datorns slump. För att slumpa fram ett tal så börjar vi att skapa ett Random-objekt. Detta objekt skapar vi först i klassen, eftersom vi ska använda det flera gånger.

```
11  public class StenSaxPase extends javax.swing.JFrame {  
12  
13      Random rand = new Random();
```

Vi trycker på glödlampan och väljer *Add import for java.util.random*.

Sedan skriver vi själva funktionen `datorVal()`, där skapar vi en ny heltalsvariabel som vi tilldelar genom att använda metoden `nextInt()`, och returnerar sedan värdet i variabeln.

```
94  int datorVal() {  
95      int val = rand.nextInt(3);  
96      return val;  
97  }
```

Vi går sedan tillbaka till att skriva kod i *stenSaxPaseActionPerformed*.

```
88  private void stenKnappActionPerformed(java.awt.event.ActionEvent evt) {  
89      resArea.append("Du valde Sten\n");  
90      int datorval = datorVal();  
91      if(datorval==0){  
92          resArea.append("Datorn valde Sten\n");  
93          resArea.append("Oavgjort\n");  
94      }  
95  }  
96
```

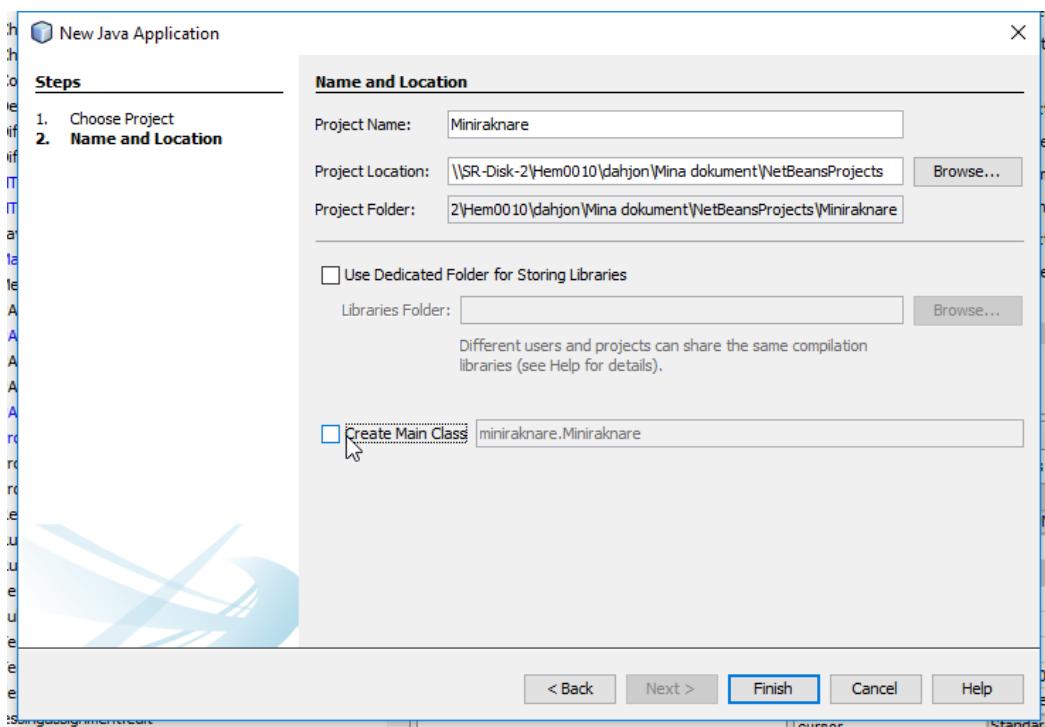
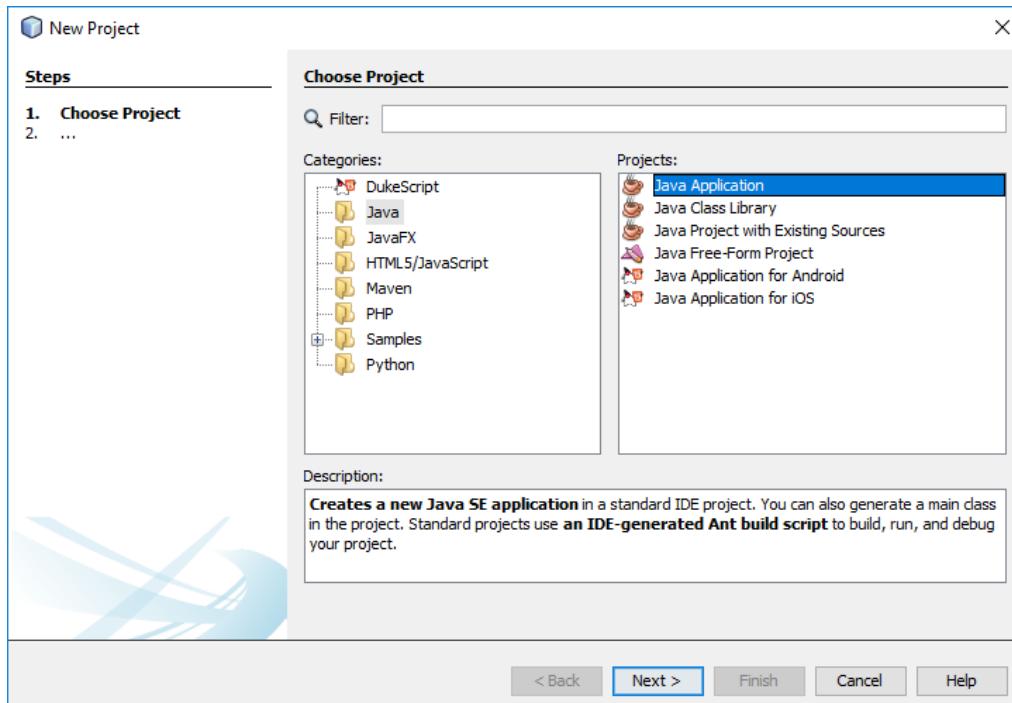
Nu blir det din uppgift att göra klart programmet. Börja med att skriva klart koden för stenkнопpen, och sedan för de andra knapparna.

Övning 26-3 Övning Sten Sax Påse

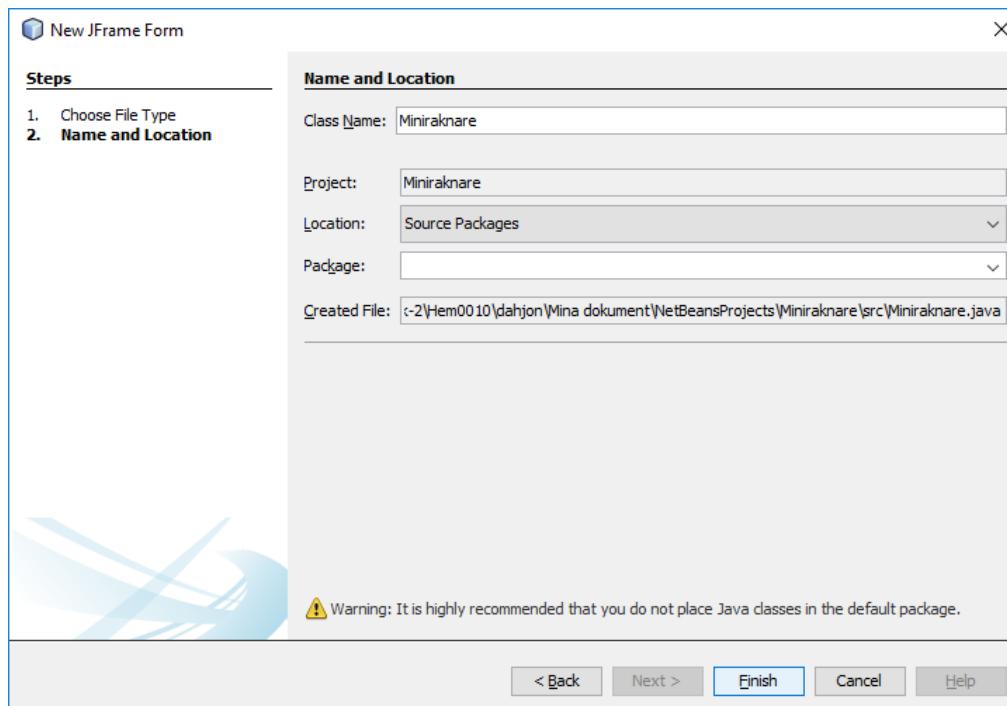
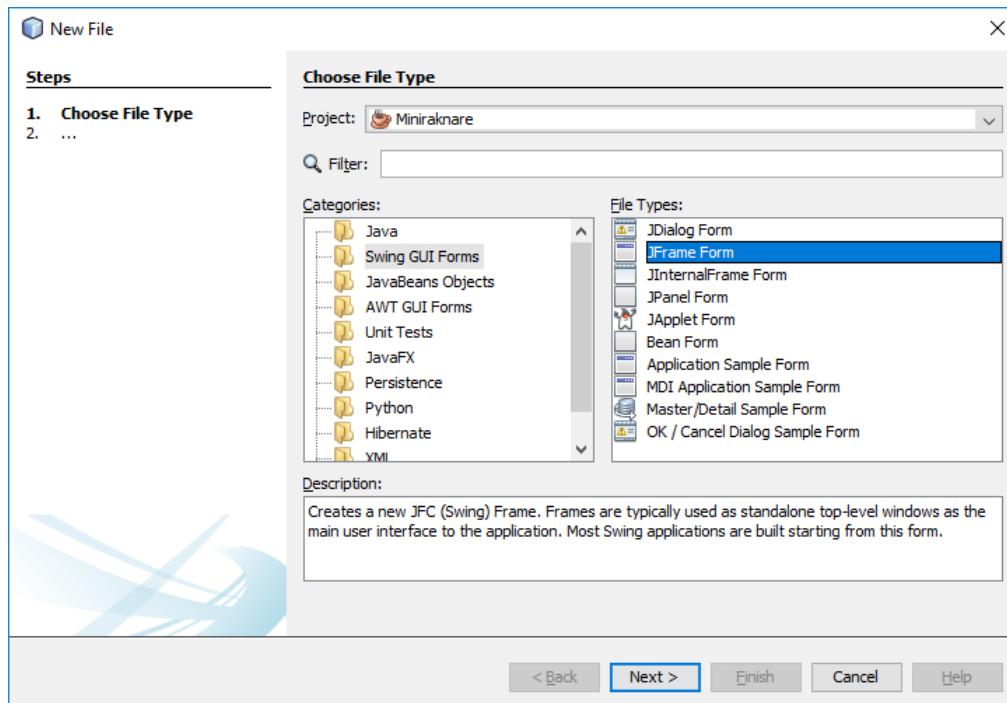
1. Börja med att få Sten-knappen att fungera som den ska.
2. Skapa koden för de andra två knapparna så du får ett fungerande spel
3. Börja räkna hur många vinster spelaren respektive datorn har. Skapa en till label där du skriver aktuell poängställning.
4. Skapa en knapp som nollställer spelet. Det vill säga sätter tillbaka poängställningen till noll för både dator och spelare, samt tömmer resultatarean.

26.2 Miniräknarexempel

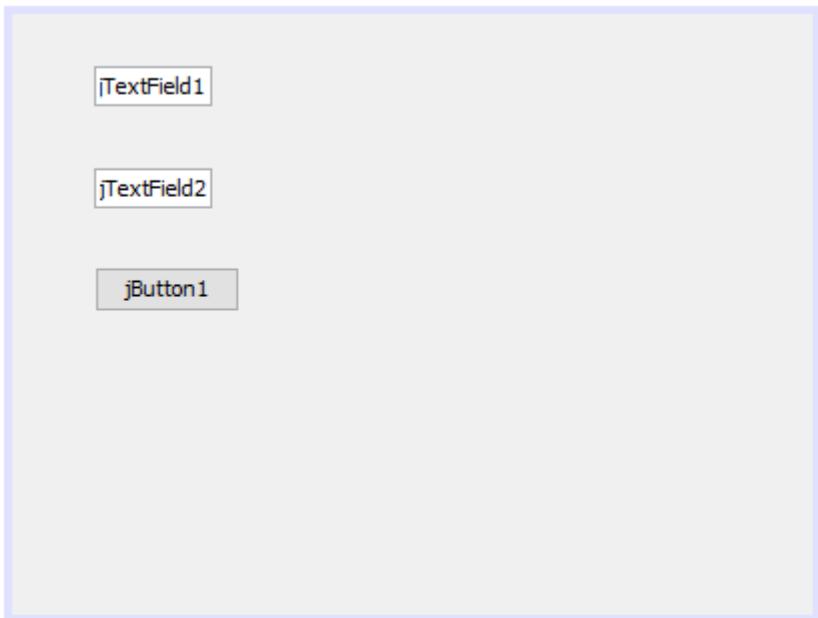
Börja med att skapa ett nytt projekt, som du kallar *Miniräknare*. Klicka även denna gång bort *Create Main Class*.



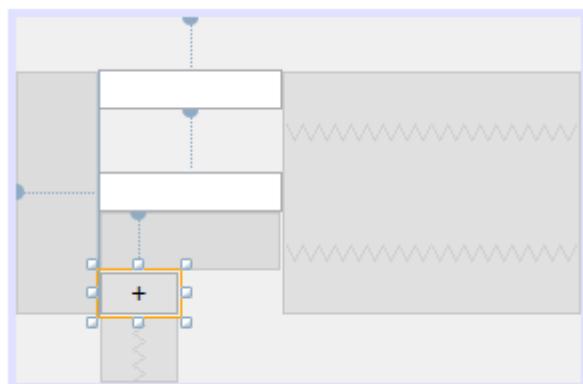
Välj *Ny File...* och välj *Swing GUI Forms/JFrame Form*.



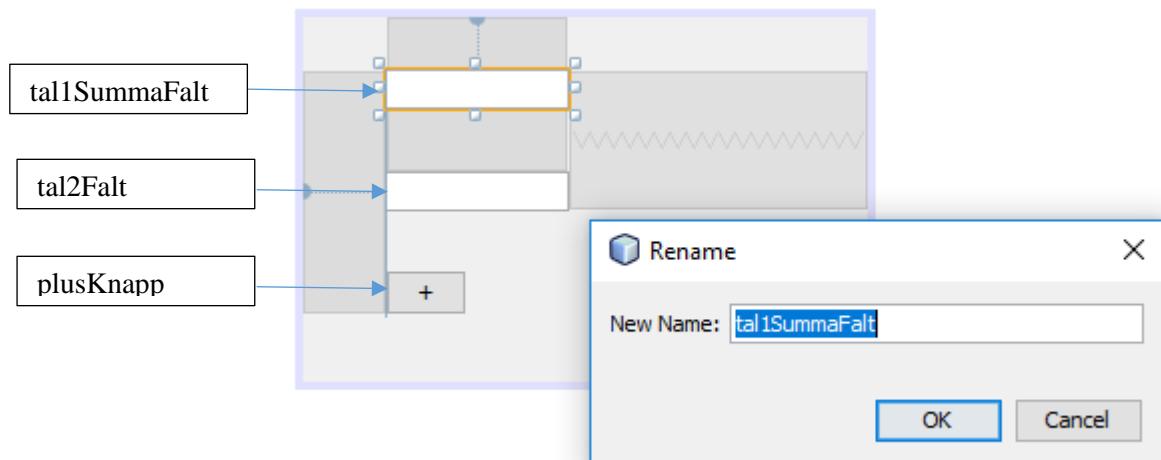
Rita upp formulär enligt bilden nedan, genom att dra in två textfält (TextField) och en knapp (Button).



Snygga till formuläret, ta bort texten i textfälten, samt skriv "+" i knappen.



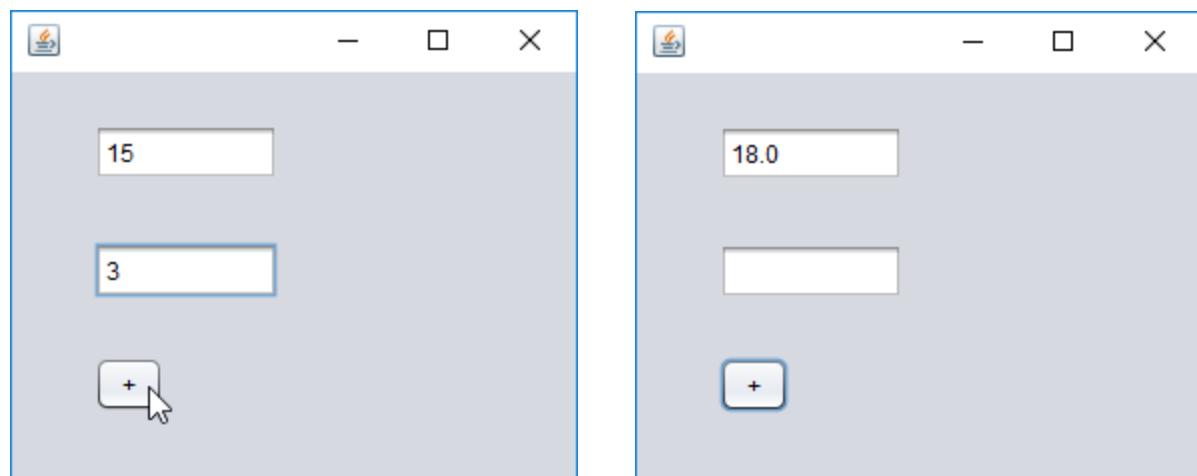
Sätt bra namn på textfälten, se bilden nedan



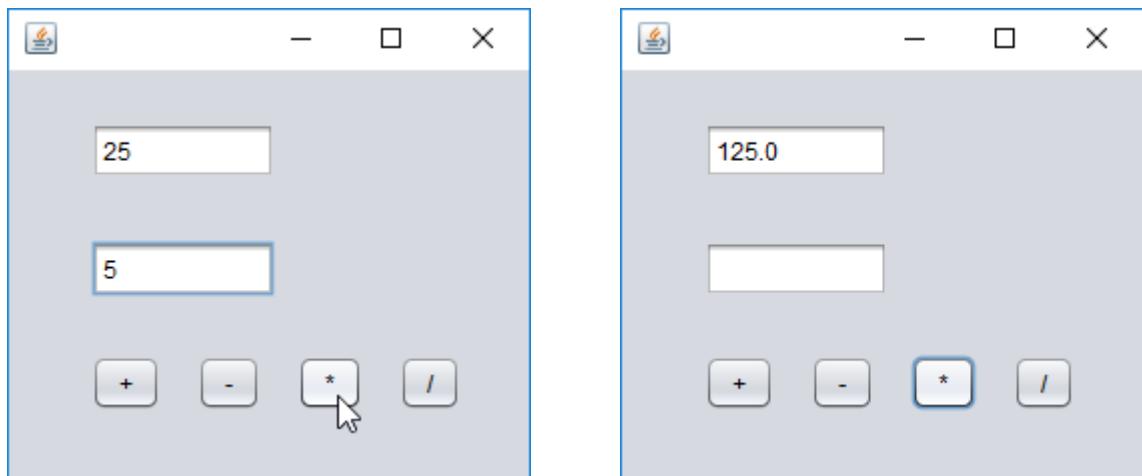
Dubbelklicka sedan på plusknappen för att skapa funktionen som kommer att köras när vi klickar på plusknappen.

```
69  I  private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {  
70      // TODO add your handling code here:  
71  }  
72
```

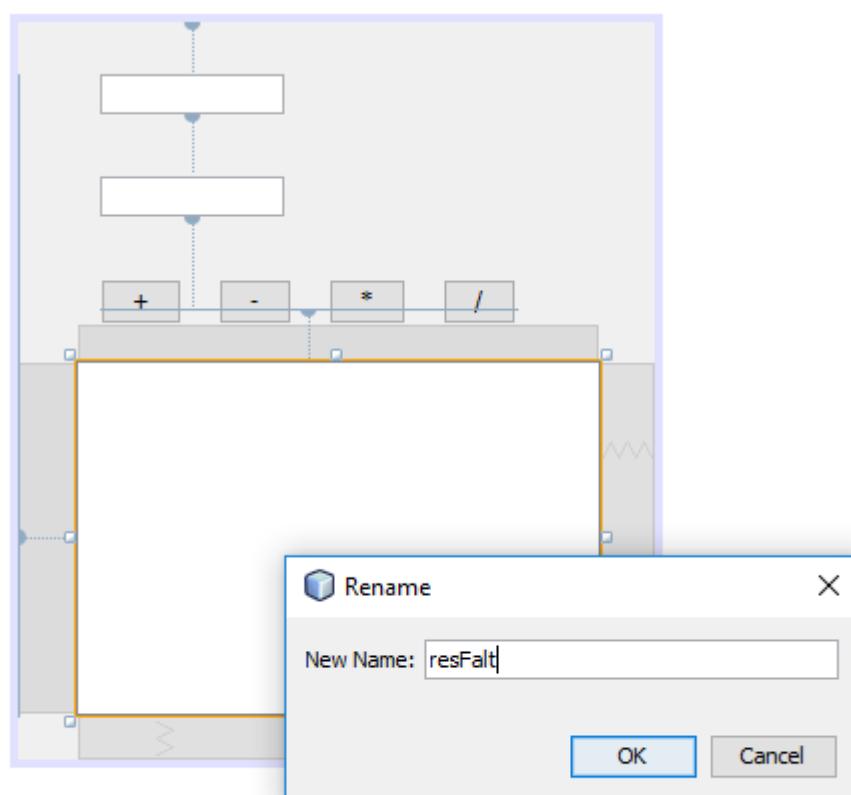
Skriv koden för att hämta in texten från de två textfälten, omvandla texterna till double värden, summera och skriv resultatet i *tal1SummeraFalt*. (Om du inte kommer på hur man gör som finns koden sist i uppgiftsbeskrivningen).



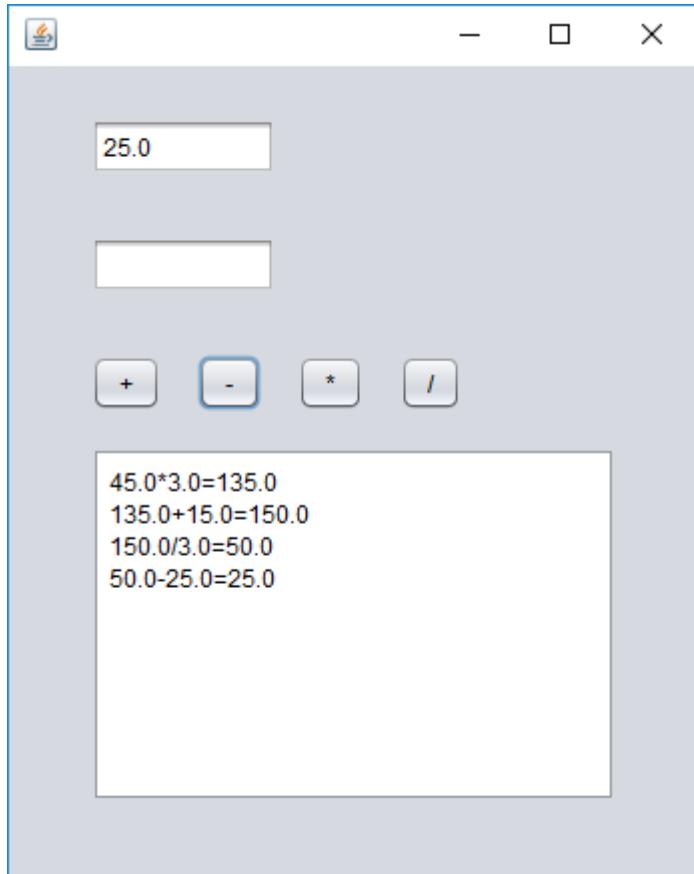
Lägg till tre knappar till. För de andra tre räknesätten, och se till att de också fungerar.



Lägg till ett resultatfält, som du döper till resFalt.



Lägg till kod för att alla uträkningar även visas i resultatfönstret på det sätt som bilden nedan visar.



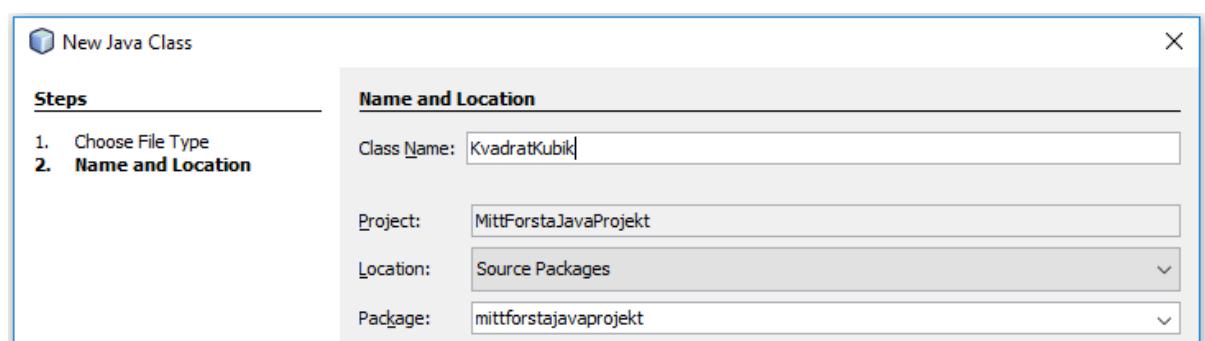
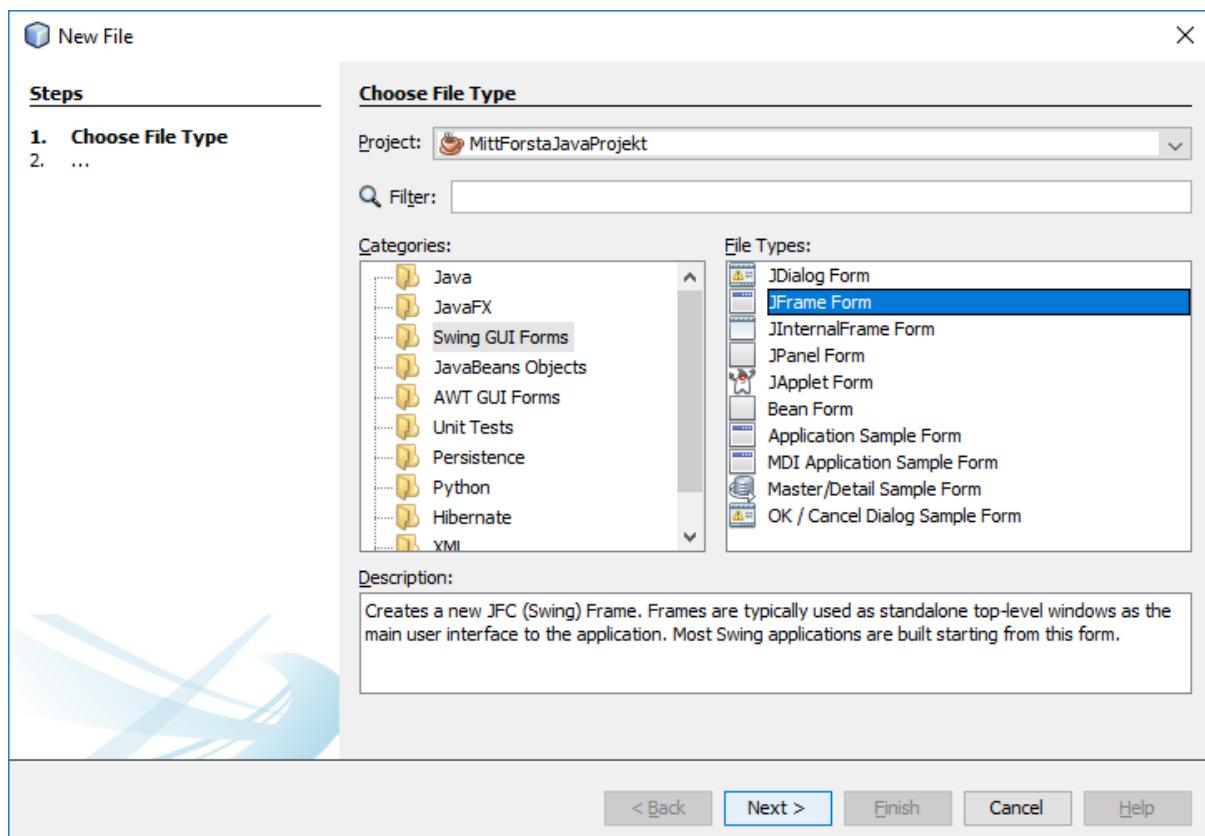
```
69  private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {  
70      String tallstr=tallSummaFalt.getText();  
71      String tal2str=tal2Falt.getText();  
72      double tall = Double.parseDouble(tallstr);  
73      double tal2 = Double.parseDouble(tal2str);  
74      double summa = tall+tal2;  
75      tallSummaFalt.setText(""+summa);  
76      tal2Falt.setText("");  
77  }  
78 }
```

Övning 26-4 Miniräknarövning fortsättning

1. Lägg till funktion för att nollställa miniräknaren
2. Lägg till knapp för upphöjt till. Alla matematiska funktioner finns i klassen Math (<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>), och du skriver till exempel Math.pow(2,3); för att räkna ut 2 upphöjt till tre.
3. Lägg till knappar för roten ur, sinus och cossinus, och gärna andra matematiska operatorer
4. Lägg till knapparna 0-9 plus decimalpunkt, så att man kan trycka på dessa istället för tangentbordet för att mata in tal i tal2fältet.
5. Du märker kanske att du också skulle behöva två piltangenter för att växla mellan fälten. För att byta fält så anropar du metoden requestFocus() i din komponent.

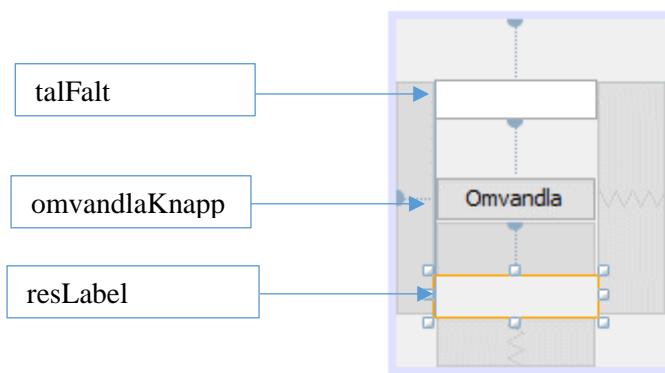
26.3 Kvadrat och Kubikexempel

Vi vill skapa ett program som tar ett tal och sedan får användaren välja om det ska kvadreras eller tas i kubik.





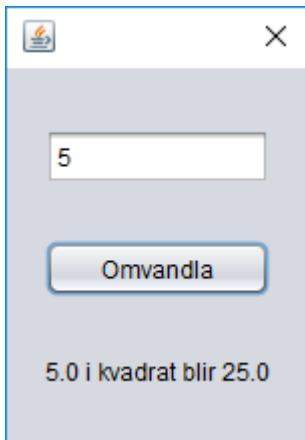
Vi tar bort texten i textfältet, och lägger till text till knappen, samt sätter namn enligt figuren nedan.



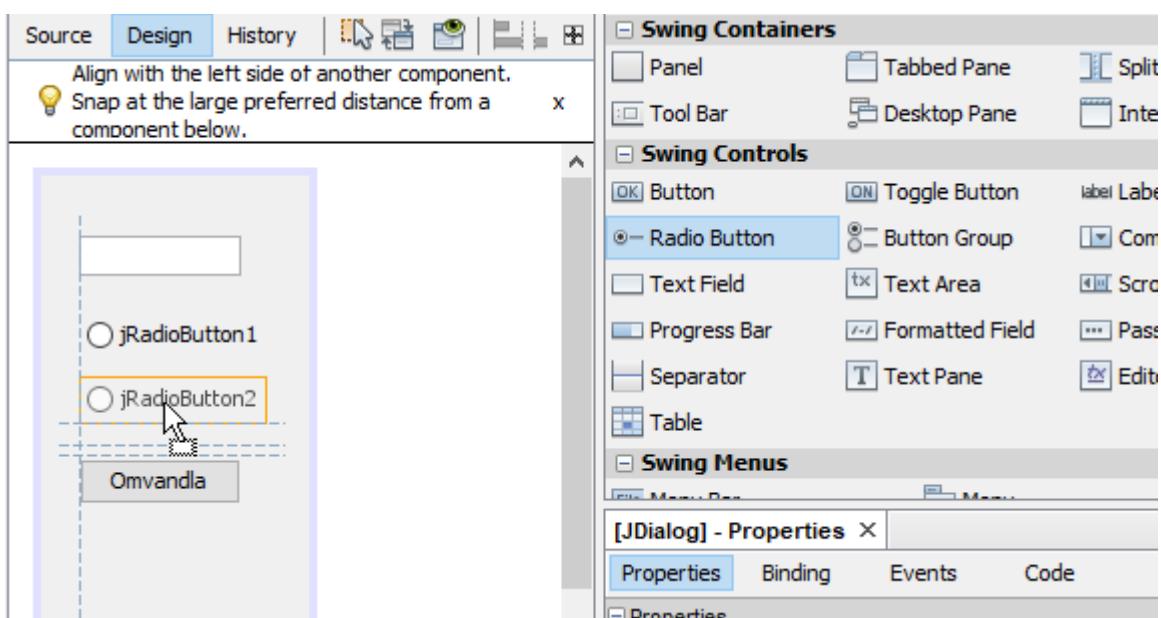
Nu dubbelklickar vi på omvandlaknappen för att skapa en metod som körs när man klickar på knappen. Förhoppningsvis dyker följande tomta metod upp:

```
66
67  private void omvandlaKnappActionPerformed(java.awt.event.ActionEvent evt) {
68      // TODO add your handling code here:
69  }
70
```

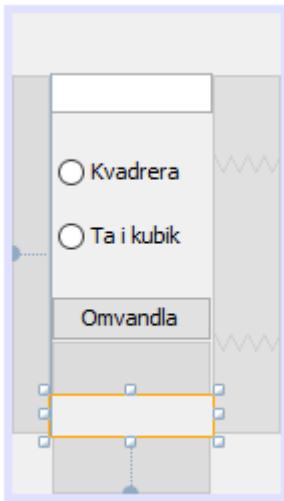
```
private void omvandlaKnappActionPerformed(java.awt.event.ActionEvent evt) {
    String talStr = talFalt.getText();
    double tal = Double.parseDouble(talStr);
    double kvadrat = tal*tal;
    resLabel.setText(tal + " i kvadrat blir " + kvadrat);
}
```



Nu har vi ett fungerande program som kvadrerar, men vi ville att användaren skulle få välja mellan att kvadrera och att ta i kubik.



Vi ändrar texterna för knapparna till Kvadrerar respektive Ta i kvadrat, och sätter namnen till kvadreraKnapp respektive kubikKnapp

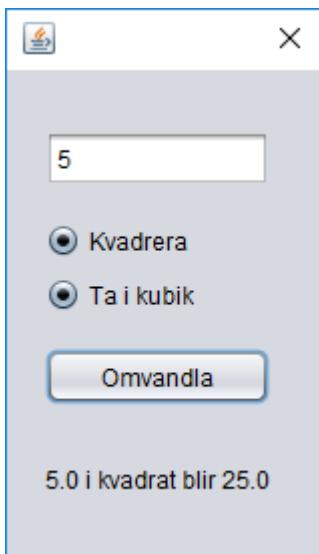


Vi går tillbaka till vår metod och skriver kod som kollar om kvadreraKnapp är förkryssad i så fall ska vi köra koden som vi redan skrivit som räknar ut kvadraten på användarens tal, annars ska den istället räkna ut kubiken på talet. För att få reda på om kvadreraKnapp är förkryssad använder vi `kvadreraKnapp.isSelected()`

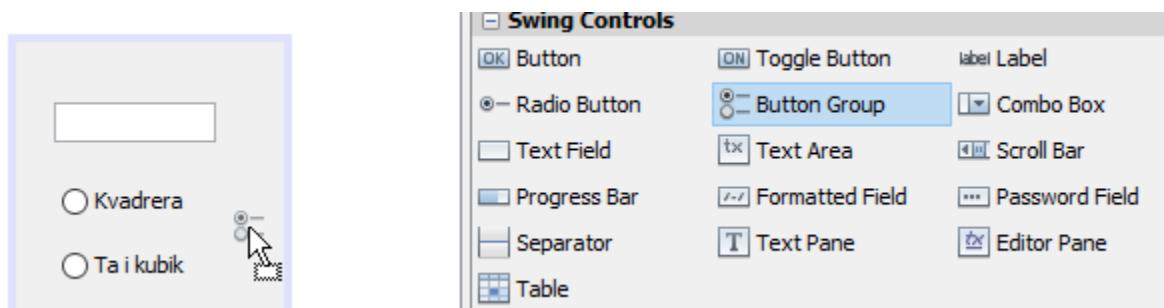
Hela koden för metoden blir då:

```
84     private void omvandlaKnappActionPerformed(java.awt.event.ActionEvent evt) {  
85         String talStr = talFält.getText();  
86         double tal = Double.parseDouble(talStr);  
87         if (kvadreraKnapp.isSelected()) {  
88             double kvadrat = tal * tal;  
89             jLabell.setText(tal + " i kvadrat blir " + kvadrat);  
90         } else {  
91             double kubik = tal * tal * tal;  
92             jLabell.setText(tal + " i kubik blir " + kubik);  
93         }  
94     }
```

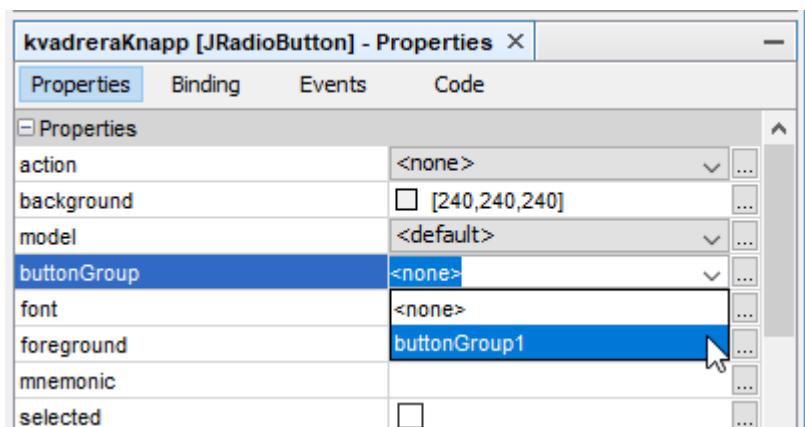
Det vi märker när vi testkör programmet är att man kan ha båda knapparna intryckta samtidigt, vilket inte var vad vi tänkt oss.



För att lösa det problemet så behöver vi använda oss av en *Button Group*. Vi tar och drar en *Button Group* till vårt fönster.



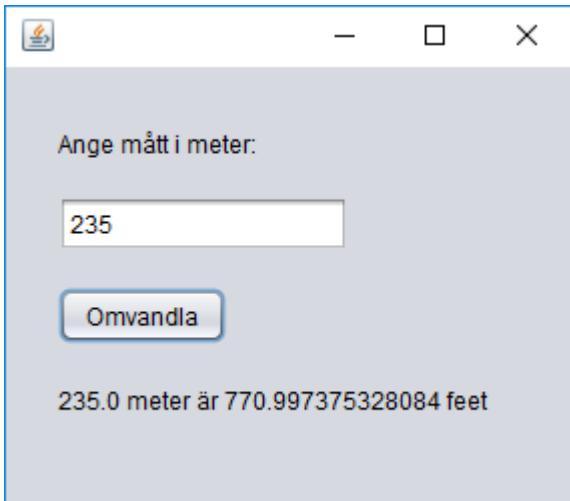
Vi måste sedan gå in under *properties* på vardera knappen och sätta *buttonGroup* till *buttonGroup1*, istället för *<none>*.



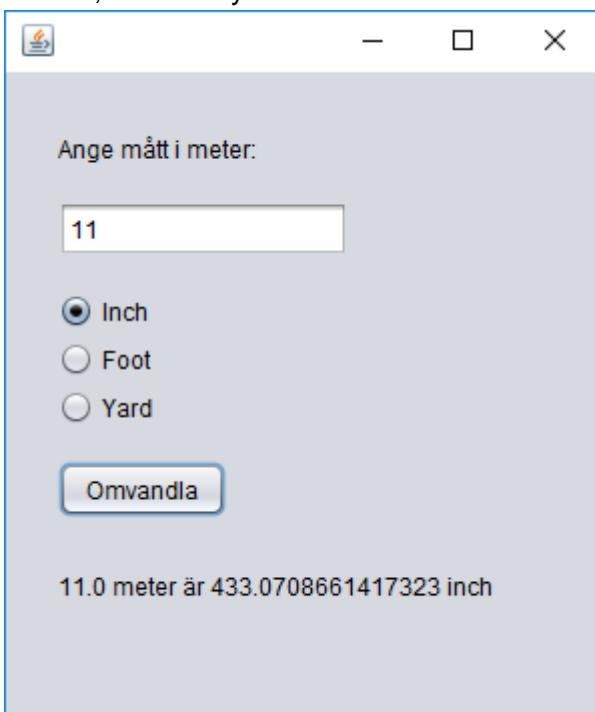
Om vi nu kör igen så ska allt fungera som det är tänkt.

Övning 26-5 Omvandling till brittiska mått enheter

- a) Skapa ett program där användaren får mata in antalet meter och sedan ska det omvandlas till foot när användaren trycker på knappen omvandla.



- b) Lägg till tre radioknappar som ska ha texten inch, foot respektive yard och stoppa dessa i en buttongroup. Användaren ska nu alltså ha möjlighet att välja att omvandla till inch, foot eller yard.



Från wikipedia kommer följande information:

inch (in) = 25,4 mm

foot (ft) = 12 in = 0,3048 m

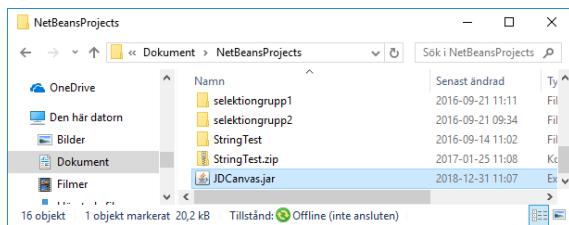
yard (yd) = 3 ft = 36 in = 0,9144 m

26.4 Rita i Java på Processingvis

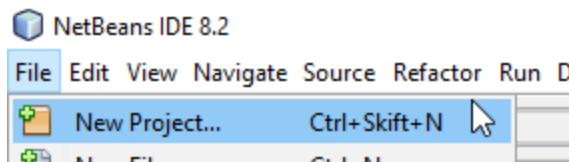
Även i java kan man ju vilja rita ibland, och inte minst skulle man vilja ha tillgång till det enkla sättet att rita från Processing kombinerat med enkelheten att skapa ett användargränssnitt som finns i Java kombinerat med Netbeans. Tyvärr är det ganska krångligt att rita med Javas inbyggda bibliotek. Det krävs ungefär 15 rader kod för att enbart rita ut en liten rektangel. Därför har ett Processingliknande bibliotek utvecklats till denna bok för detta syfte, klassen heter JDCanvas, och kan laddas ner från:

<http://jonathan.dahlberg.media/JDCanvas/JDCanvas.jar>.

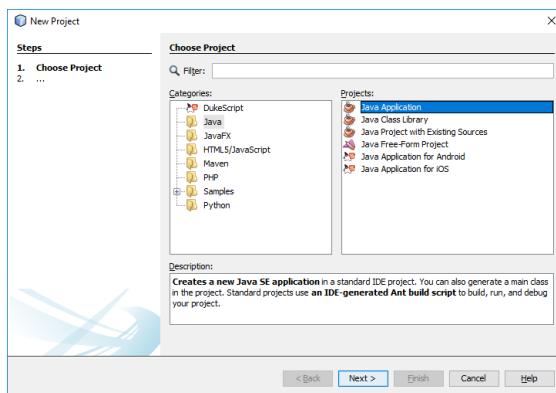
Börja med att ladda ner JDCanvas och placera på lämpligt ställe, till exempel i din NetbeansProject-Mapp.



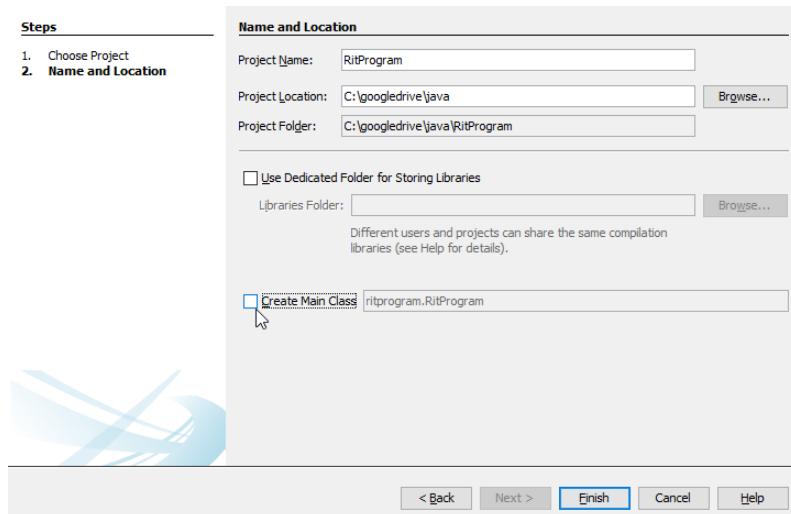
För att visa hur man kan jobba med JDCanvas tänkte jag att vi överföra det enkla ritprogrammet som vi gjorde i Processing till Java. Vi börjar med att skapa ett nytt projekt i Netbeans.



Välj *Java Application*.

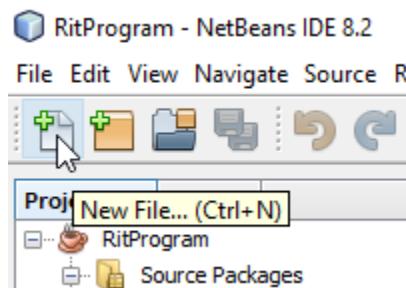


Döp Programmet till RitProgram och kryssa bort rutan, *Create main Class*, och välj sedan *Finish*.

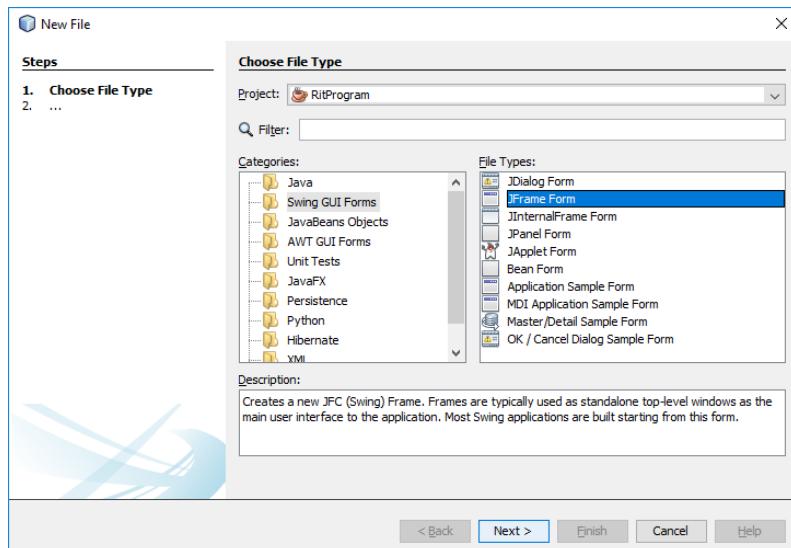


Nu har vi skapat själva projektet. Nu ska vi skapa en ny javafil som vi vill kunna redigera användargränssnittet på i Netbeans GUI-editor. Den typen av fil kallas Netbeans för *JFrame Form*.

Tryck på knappen för ny Fil:

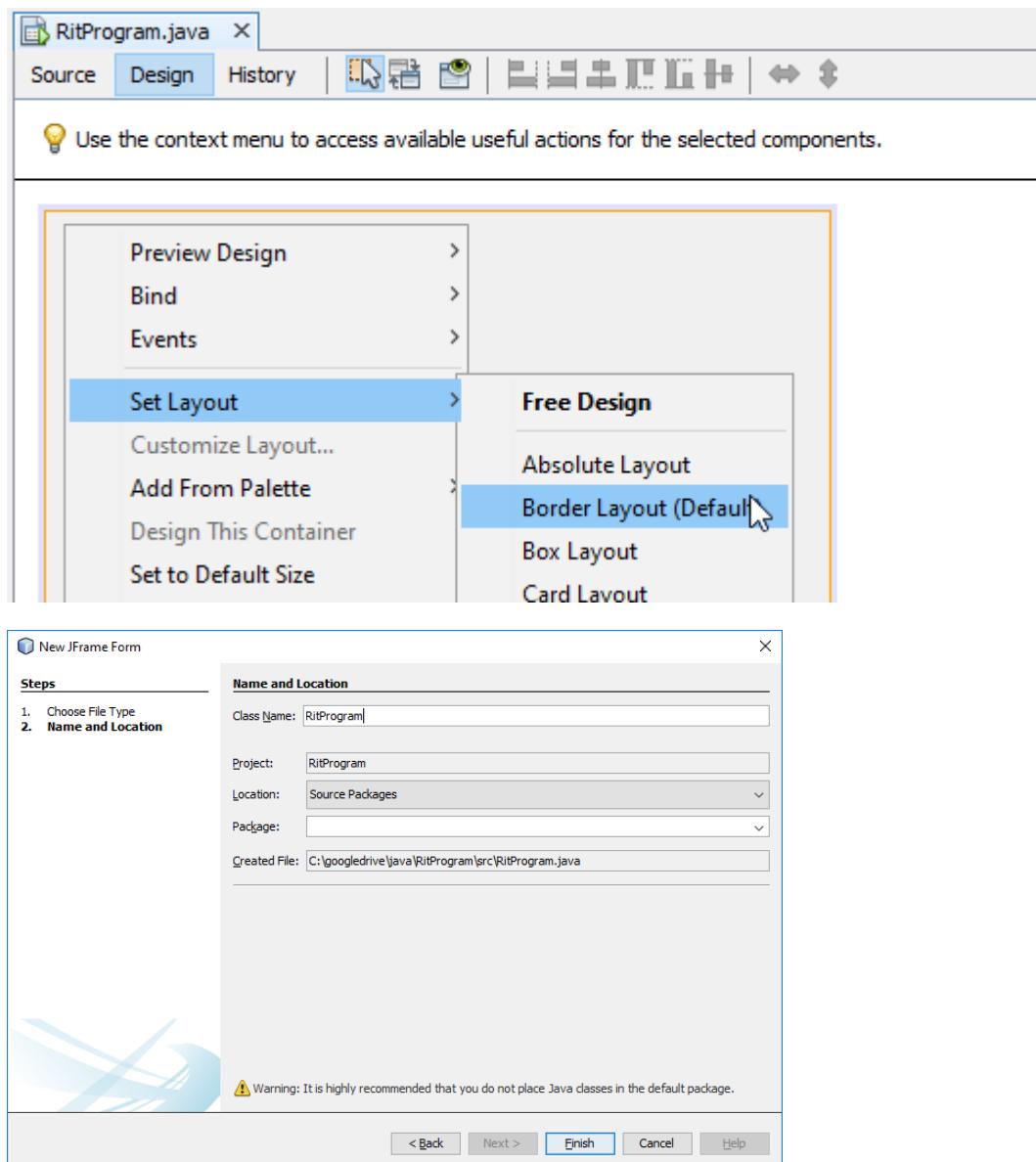


Markera Swing GUI Forms i vänstra vyn och välj sedan JFrame Form i den högra.



Nu kommer programmet att visas i Designläge. Vi högerklickar nu på själva fönstrets yta, och väljer *Set Layout/BorderLayout*. Med borderlayout har man möjlighet att placera ut en komponent i mitten och en komponent på varje sida som man anger med konstanterna

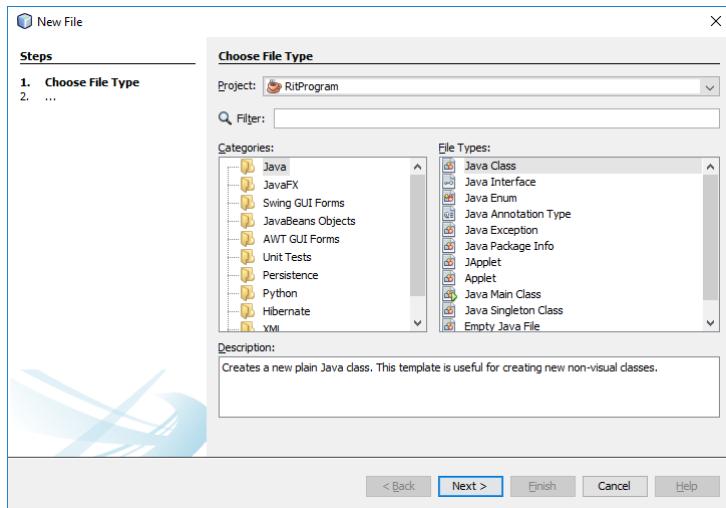
BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, eller BorderLayout.WEST. Anger man inget så hamnar komponenten i mitten.



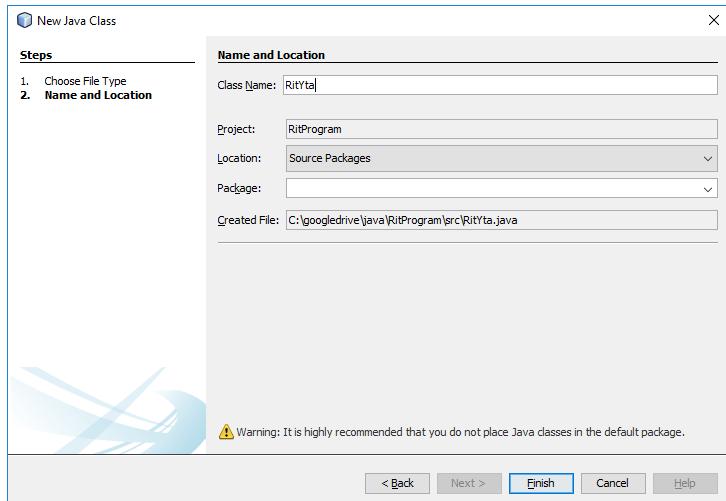
Vi trycker ni på knappen *New File* igen



och väljer *Java och Java Class*



Vi väljer namnet *Rityta*.



Du får då upp klassen Rityta. Vi ska nu lägga till *extends JDCanvas*

```

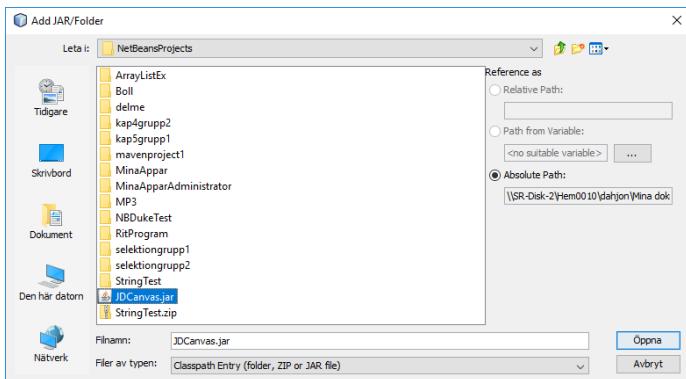
12  /*
13  * 
14  public class Rityta extends JDCanvas{
15  }
16

```

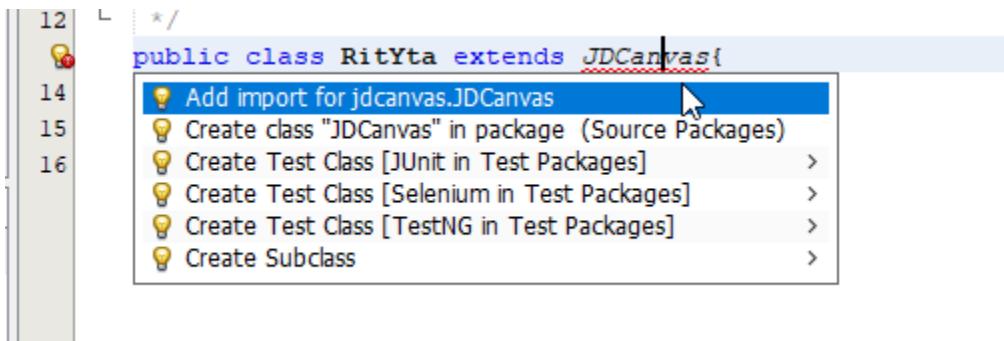
Vi ser att JDCanvas blir rött. Det beror på att JDCanvas inte hittas. Vi måste då börja med att lägga till JDCanvas.jar till vårt projekt. Det gör vi genom att högerklicka på Libraries i Projektvyn till vänster i Netbeans, och välja *Add JAR/Folder*.



Vi letar sedan reda på filen JDCanvas.jar och väljer den.



Nu går vi tillbaka till filen RitYta.java och trycker på glödlampan tillvänster om klassdeklarationen, och väljer *Add import for jdcanvas.JDCanvas*.



Nu kan vi gå tillbaka till *Exempel: ritprogram* ovan, och kopiera in koden. När vi har kopierat över det enkla ritprogrammet från Processing så ser det ut så här:

```
1 import jdcanvas.JDCanvas;
2
3 public class RitYta extends JDCanvas {
4
5     void setup() {
6         size(600, 600);
7         fill(0);
8     }
9
10    void draw() {
11        if (mousePressed) {
12            ellipse(mouseX, mouseY, 3, 3);
13        }
14    }
15
16 }
```

Nu måste vi bara skriva `public` framför metoderna `setup` och `draw` för att det ska bli rätt. Ordet *public* betyder att metoden ska vara tillgänglig i alla andra klasser, och det måste våra metoder `setup` och `draw` vara.

Nu kommer vår färdiga `RitYta`-klass att se ut så här:

```
1 import jdcanvas.JDCanvas;
2
3 public class RitYta extends JDCanvas {
4
5     public void setup() {
6         size(600, 600);
7         fill(0);
8     }
9
10    public void draw() {
11        if (mousePressed) {
12            ellipse(mouseX, mouseY, 3, 3);
13        }
14    }
15
16 }
```

Nu går vi tillbaka till vår huvudklass `RitProgram` och väljer `Source` för att se koden

```
RitProgram.java
Source Design History
```

```
1
2 class RitProgram {
```

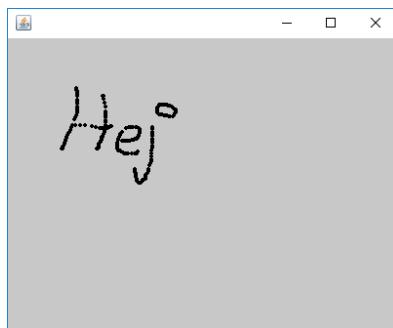
Vi börjar med att skapa ritytan först i vår klass genom att skriva:

```
RitYta yta = new RitYta();
```

Och sedan lägger vi till add(yta) i konstruktorn för att lägga till ritytan i mitten på vårt program, och vi lägger till setSize() sist i konstruktorn för att sätta storleken på programfönstret.

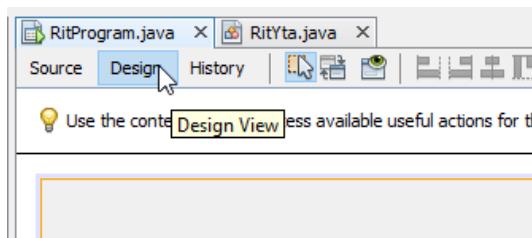
```
3  public class RitProgram extends javax.swing.JFrame {  
4  
5      RitYta yta = new RitYta();  
6      /**  
7      * Creates new form RitProgram  
8      */  
9      public RitProgram() {  
10         initComponents();  
11         add(yta);  
12         setSize(610,750);  
13     }  
14 }
```

Nu kan du trycka F6 för att köra programmet, och det borde se ut ungefär så här:

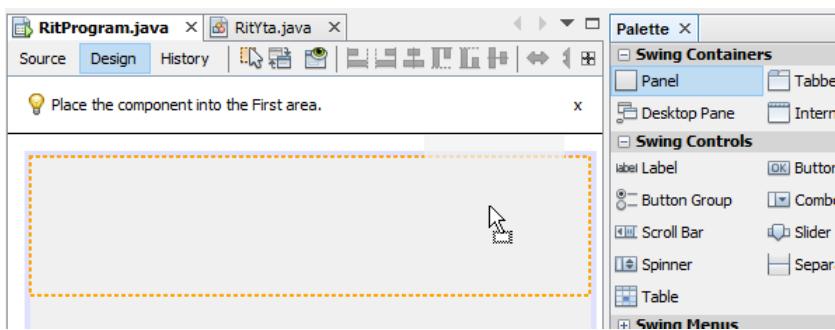


Detta är kanske inte så fantastiskt eftersom det är samma sak som vi åstadkom i Processing långt tidigare, och är det exakt detta som man vill göra så är det förstås bättre att använda Processing. Men vi ska nu testa att skapa några knappar för att till exempel byta färg.

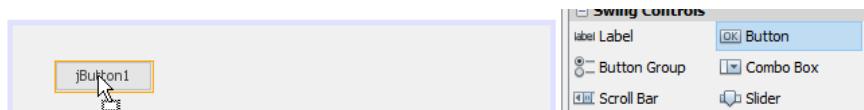
Vi klickar på Design för att komma till designläget.



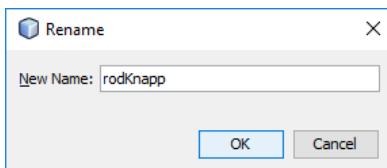
Även om det inte syns så bra så har vi vår rityta i mitten. Vi vill nu lägga en panel i övre kanten. I den kan vi sedan lägga knappar och andra komponenter som vi vill ha för att styra ritprogrammet.



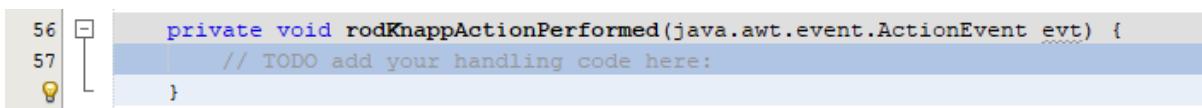
Till den ytan kan vi sedan dra in en knapp.



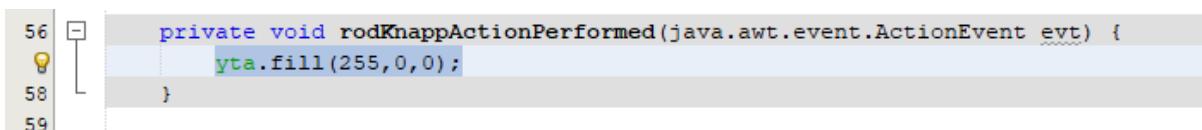
Vi ändrar texten på knappen till "röd" genom att högerklicka på knappen och högerklickar också och väljer *change variable name*, och väljer namnet *rodKnapp*.



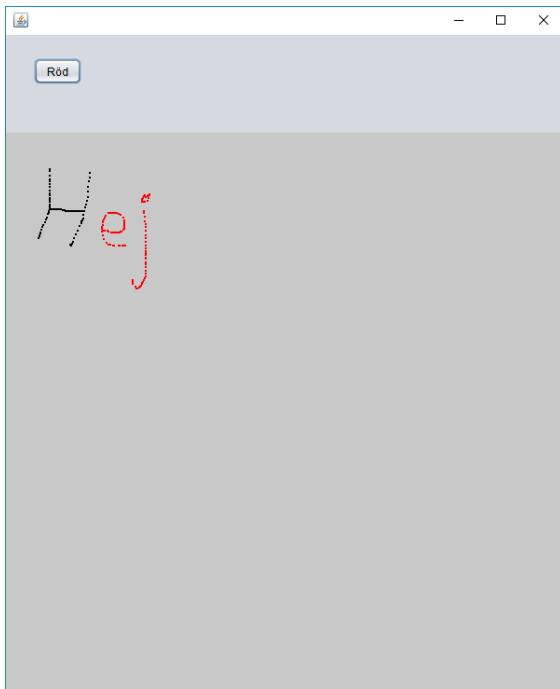
Nu dubbelklickar vi på knappen för att skapa en metod som kommer att köras när man trycker på knappen.



Nu behöver vi bara skriva `yta.fill(255,0,0)`, för att ändra färgen på det som ska ritas ut.



Nu bör programmet fungera.



Övning 26-6 Övning Ritprogram i Java

1. Lägg till fler knappar för färger
2. Skapa en knapp för att tömma ritytan
3. Lägg till en knapp och ett textfält för att spara en bild. Lägg dem gärna i en panel och sätt en titled border på panelen.



4. Skapa knappar för att ändra former att rita med
5. Skapa textfält för R, G och B, värden så att användaren fritt kan välja färger.
6. Lägg till ett textfält där man kan skriva in en text som man sedan kan placera ut

26.5 Förteckning över metoderna i JDCanvas

En mer detaljerad förteckning finns på:

<http://jonathan.dahlberg.media/JDCanvas/jdcanvas/JDCanvas.html>

void	background (java.awt.Color c)	Sätta bakgrundsfärgen till värdet i färgobjektet som anges som parameter
void	background (int gray)	Sätta bakgrundsfärgen till gråskalevärde
void	background (int r, int g, int b)	Sätta bakgrundsfärgen till värdet i färgobjektet som anges som parameter
void	draw()	En metod som körs ett visst antal gånger i sekunden, som standard 60 gånger per sekund.
void	ellipse (double x, double y, double width, double height)	Rita en ellips
void	ellipse (int x, int y, int width, int height)	Rita en ellips
void	fill (java.awt.Color c)	Sätta fyllningsfärgen
void	fill (int gray)	Sätta fyllningsfärgen
void	fill (int r, int g, int b)	Sätta fyllningsfärgen
void	frameRate (int frameRate)	Sätter framerate
int	get (int x, int y)	

	Hämta färgen på en viss position.
int	getEllipseMode()
java.awt.Graphics2D	getGraphics2D()
int	getRectMode()
	Hämtar rektangelmode
void	image(java.awt.image.BufferedImage img, double x, double y)
	Rita ut en bild av typen BufferedImage som är ett standardformat i java
void	image(java.awt.image.BufferedImage img, double x, double y, double w, double h)
	Skala om bilden och rita ut en bild av typen BufferedImage som är ett standardformat i java.
void	image(java.awt.image.BufferedImage img, int x, int y)
	Rita ut en bild av typen BufferedImage som är ett standardformat i java
void	image(java.awt.image.BufferedImage img, int x, int y, int w, int h)
	Skala om bilden och rita ut en bild av typen BufferedImage som är ett standardformat i java.
void	keyPressed()
	Överlagringsbar metod som körs när en knapp trycks ner
void	keyReleased()
	Överlagringsbar metod som körs när en knapp släpps upp
void	line(double x1, double y1, double x2, double y2)
	Rita en linje

void	line(int x1, int y1, int x2, int y2)
	Rita en linje
java.awt.image.BufferedImage	loadImage(java.lang.String file)
	Ladda in en bild av typen BufferedImage som är ett standardformat i java
void	mouseClicked()
	Överlagringsbar metod som körs när en musknappen klickats
void	mouseDragged()
	Överlagringsbar metod som körs när en musknappen hålls nere samtidigt som musen flyttas
void	mousePressed()
	Överlagringsbar metod som körs när en musknappen tryck ned
void	mouseReleased()
	Överlagringsbar metod som körs när en musknappen släpps upp
void	noLoop()
	Stoppar draw från att köras
void	noStroke()
	Från och med nu ska vi inte ha någon kantlinje på de objekt som ritas ut
void	rect(double x, double y, double width, double height)
	Rita en rektangel
void	rect(int x, int y, int width, int height)
	Rita en rektangel
java.awt.image.BufferedImage	resize(java.awt.Image img, int w, int h)

Skala om en bild av typen BufferedImage som är ett standardformat i java

void **save**(java.lang.String fileName)

Spara Canvasen i en fil.

void **saveFrame**(java.lang.String fileName)

Spara Canvasen i en fil.

void **setEllipseMode**(int ellipseMode)

Sätter ellipse mode till CENTER, CORNERS, eller RADIUS

void **setRectMode**(int rectMode)

Används för att sätta rectMode alltså
rektangelmode till CENTER, CORNER, eller
RADIUS

void **setup()**

Överlagringsbar metod som där det är
meningen att man ska stoppa initieringskod

void **size**(int width, int height)

Skapa och sätta storleken på ritytan.

void **stroke**(java.awt.Color c)

Sätta kantlinjefärgen

void **stroke**(int gray)

Sätta kantlinjefärgen

void **stroke**(int r, int g, int b)

Sätta kantlinjefärgenfärgen

void **strokeWeight**(int weight)

Sätter kantlinjebredden

void **text**(java.lang.String txt, double x,
double y)

Skriva ut text på en viss plats på skärmen

void	text (java.lang.String txt, int x, int y)	Skriva ut text på en viss plats på skärmen
void	textAlign (int alignX)	Sätta textalign till LEFT, RIGHT eller CENTER
void	textAlign (int alignX, int alignY)	Sätta textalign till LEFT, RIGHT eller CENTER, respektive BASELINE, UP eller DOWN Obs!
void	textSize (double textSize)	Sätta textstorlek
void	triangle (double x1, double y1, double x2, double y2, double x3, double y3)	Rita en triangel
void	triangle (int x1, int y1, int x2, int y2, int x3, int y3)	Rita en triangel

26.6 Fler saker att tänka på när man använder JDCanvas

26.6.1 Färgobjekt och get()-metoden

En av de största skillnaderna mellan Processing och Java med JDCanvas är att färgtypen som finns i Processing inte finns i Java. Men du kan skapa ett färgobjekt på följande sätt.

```
Color rod = new Color(255, 0, 0);
```

eller

```
Color vit = Color.white;
```

Men det som kommer ut från get()-metoden är av typen int.

Så för att jämföra en färg med det som kommer ut från get-funktionen så måste man först göra om färgbobjektet till int, vilket görs med getRGB()-metoden exempelvis:

Följande kod i processing:

```
if(get(int(bollx), int(bolly))==vit){  
    speedy=-speedy;  
}
```

Blir på följande sätt i java:

```
if(get((int)bollx, (int)bolly)==vit.getRGB()) {  
    speedy=-speedy;  
}
```

Lägg också märke till att omvandlingen från float eller double till int är annorlunda i Java från det vi använde i Processing. (Men man kan ändå göra på Javasettet i Processing)

26.6.2 Bilder i JDCanvas

När det gäller att ladda in bilder i JDCanvas är det två saker som man ska tänka på. För det första så används inte typen Pimage som var Processings specielklass för bilder istället används BufferedImage-klassen som är standard i Java.

```
laddabild1  
void setup() {  
    size(640, 490);  
    PImage img = loadImage("Blomma.jpg");  
    image(img, 20, 20);  
}
```

Blir:

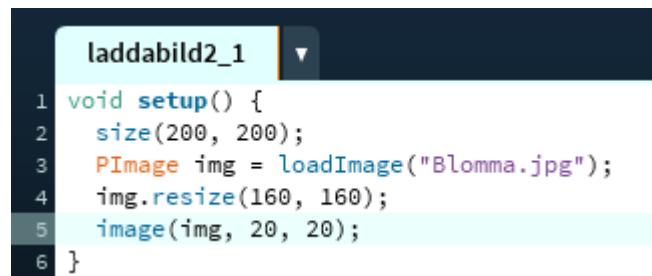
```
public void setup() {  
    size(640, 490);  
    BufferedImage img = loadImage("Blomma.jpg");  
    image(img, 20, 20);  
}
```

Men var ska nu programmet leta efter *Blomma.jpg* i exemplet ovan? Processing letar efter bilder dels i samma mapp som själva sketchen och dels i undermappen data. I Java kommer den automatiskt att leta i den mapp där programmet körs. I Netbeans kommer det att vara själva Projektmappen, alltså själva mappen som i sin tur innehåller src-mappen där själva koden ligger.

26.6.3 Ändra storleken på en bild

Den javaklass som JDCanvas använder sig av är BufferedImage. Den har tyvärr ingen enkel metod för att ändra storleken på den. Men i JDCanvas finns en metod som skalar om en

BufferedImage. Den tar en BufferedImage som inparameter och returnerar en BufferedImage. Se exemplet nedan.



```
laddabild2_1
1 void setup() {
2   size(200, 200);
3   PImage img = loadImage("Blomma.jpg");
4   img.resize(160, 160);
5   image(img, 20, 20);
6 }
```

Blir

```
public void setup() {
    size(200, 200);
    BufferedImage img = loadImage("Blomma.jpg");
    img=resize(img,160,160);
    image(img, 20, 20);
}
```

26.6.4 Använda Javas standardritmetoder.

I JDCanvas finns metoden getGraphics2D(), som returnerar ett Graphics2D objekt. Detta objekt ger tillgång till alla standardjava-ritverktyg. Du kan då använda dessa utöver de mycket begränsade urval av ritmetoder som kommer direkt med JDCanvas. Det är i själva verket så att det är dessa ritverktyg som JDCanvas använder sig av.

Övning 26-7 Testa att själv konvertera en uppgift från Processing till Java och JDCanvas

Välj ett av programmen som vi gjorde i Processing och försök att få det att fungera i Java med JDCanvas. Det kan vara ett eget projekt du gjort i Processing eller någon av exemplen eller övningarna ovan, till exempel Studsande boll med gravitation.

27 Klasser i Java - Steg för steg

Klasser är ett trevligt sätt att kunna samla ihop metoderna och de variablerna som hör ihop med dessa metoder till en gemensam enhet. Det kan handla om till exempel ett grafiskt form som en cirkel eller rektangel, det kan handla om en spelarfigur i ett spel, eller också kan det vara en knapp i ett grafiskt gränssnitt.

Vi kommer att använda sex stycken steg när vi skapar vår klass:

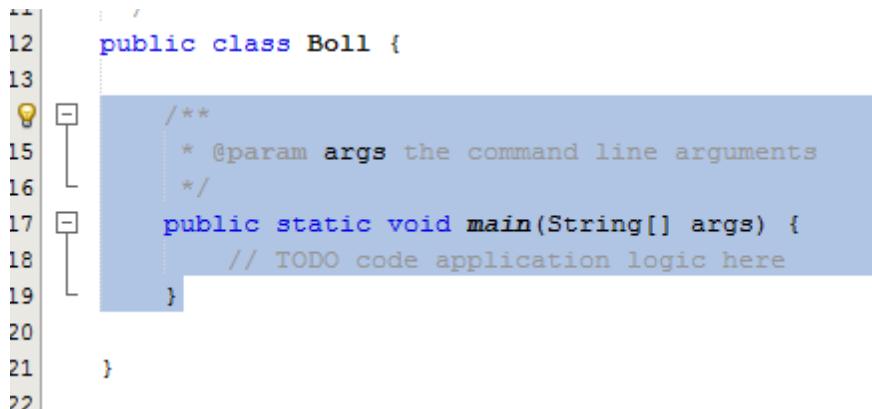
- Steg 1 Skapa klassfilen
- Steg 2 Skapa dina variabler
- Steg 3 Infoga konstruktör.
- Steg 4 Infoga Setters and Getters
- Steg 5 Skapa övriga metoder

27.1 Exempel Klass som representerar en Boll

Vi kommer att titta på hur man kan använda klasser och objekt för att rita ut grafiska objekt, till att börja med en boll. Vi börjar alltså med att skapa själva filen som klassen ska finnas i och sätta namnet på klassen. Vi börjar sedan att fundera på vilka egenskaper vårt objekt har, för en boll kan det tex vara dess x, och y position samt storlek, dessa blir de variabler som ska finnas i klassen, sedan skapar vi metoder för att komma åt egenskaperna, och sist skapar vi övriga metoder, tex för att Flytta bollen.

27.1.1 Steg 1 Skapa klassfilen

Börja med att skapa ett nytt projekt Boll. Nu skapas förmöglichen också en klass Boll, och den kan vi nu fortsätta jobba med. Om det inte skapas en klass Boll så ska du nu skapa klassen.



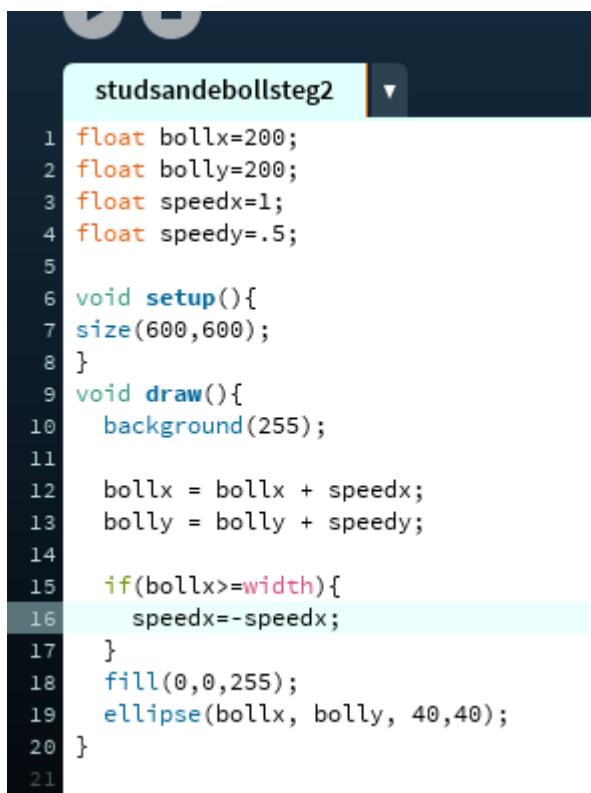
```
12 public class Boll {  
13  
14     /**  
15      * @param args the command line arguments  
16      */  
17     public static void main(String[] args) {  
18         // TODO code application logic here  
19     }  
20  
21 }  
22
```

Om det skapas en mainmetod i klassen ska du nu ta bort den.

27.1.2 Steg 2 Skapa dina variabler

Vilka variabler behöver vi då för att hålla reda på en boll. Om vi vill kan vi gå tillbaka till vårt tidigare exempel studerande boll och titta. Här har vi fyra variabler. Det är x-positionen, y-

positionen, hastighet i x-led och hastighet i y-led. Så dessa fyra variabler vill jag ha med. Dessutom skulle jag vilja lägga till diametern som en ytterligare variabel.



```
studsandebollsteg2
1 float bollx=200;
2 float bolly=200;
3 float speedx=1;
4 float speedy=.5;
5
6 void setup(){
7 size(600,600);
8 }
9 void draw(){
10 background(255);
11
12 bollx = bollx + speedx;
13 bolly = bolly + speedy;
14
15 if(bollx>=width){
16     speedx=-speedx;
17 }
18 fill(0,0,255);
19 ellipse(bollx, bolly, 40,40);
20 }
21 }
```

Det första du ska tänka på när du skapar en ny klass är vilka egenskaper som dina objekt ska ha. När det gäller en boll i ett spel eller annat program på datorn, så kommer jag att tänka på åtminstone fem egenskaper. Det är x-positionen, y-positionen, diametern, hastighet i x-led och hastighet i y-led

```
public class Boll {
    private float x;
    private float y;
    private float d;
    private float speedx;
    private float speedy;
}
```

För att skapa ett nytt objekt av klassen så kan vi skriva:

```
Boll b = new Boll();
```

Vi kan tänka oss att Klassen är som en mall och att objektet är de kopior som skapas utifrån mallen.

27.1.3 Steg 3 Infoga konstruktör.

En konstruktör är som en vanlig metod som körs när man skapar ett objekt. Den används ofta för att kopiera in startvärdet i klassen.

I Netbeans kan man enkelt generera en konstruktör genom att trycka på *alt-insert* och välja *Constructor..*

Sedan väljer vi vilka variabler som vi ska kopiera in i klassen genom konstruktorn, och trycker sedan på *Generate...*

The screenshot shows the NetBeans IDE interface. On the left, there is a code editor window with the following Java code:

```
public class Boll {  
    private float x;  
    private float y;  
    private float d;  
    private float speedx;  
    private float speedy;  
}
```

Below the code editor, a context menu is open over the closing brace of the class definition. The menu items include: Generate, Constructor..., Logger..., Getter..., Setter..., Getter and Setter..., equals() and hashCode()..., toString()..., Override Method..., Add Property..., and a separator line followed by comments. The 'Constructor...' item is highlighted with a mouse cursor.

To the right of the code editor, a 'Generate Constructor' dialog box is displayed. The title bar says 'Generate Constructor'. The main area is titled 'Select fields to be initialized by constructor:' and contains five checkboxes, each with a lock icon, corresponding to the variables defined in the class: x : float, y : float, d : float, speedx : float, and speedy : float. All checkboxes are checked. At the bottom of the dialog are 'Select All', 'Select None', 'Generate', and 'Cancel' buttons. The 'Generate' button is highlighted with a blue border.

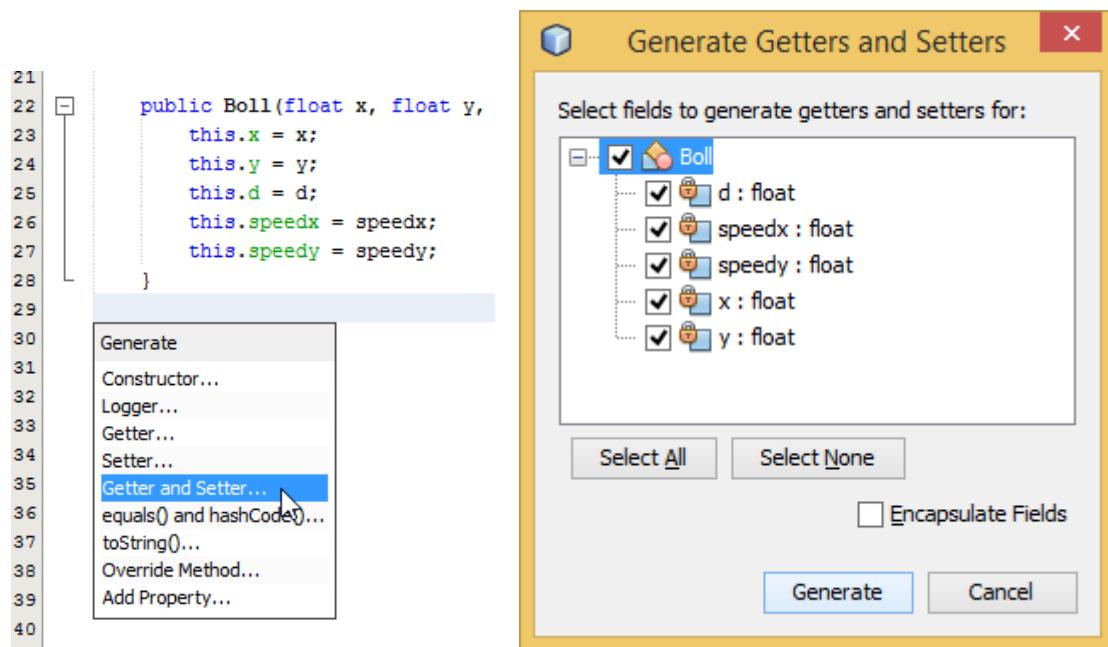
At the bottom of the screen, the code editor shows the generated constructor:

```
14 public class Boll {  
15  
16     private float x;  
17     private float y;  
18     private float d;  
19     private float speedx;  
20     private float speedy;  
21  
22     public Boll(float x, float y, float d, float speedx, float speedy) {  
23         this.x = x;  
24         this.y = y;  
25         this.d = d;  
26         this.speedx = speedx;  
27         this.speedy = speedy;  
28     }  
29 }
```

27.1.4 Steg 4 Infoga Setters and Getters

Vi har skrivit `private` före våra medlemsvariabler, `x`, `y` och `z`. Det betyder att vi inte kommer åt dem från någon annan klass. Ofta skapar man därför speciella metoder för att komma åt dessa. Dessa kallas man oftast *Getters* och *Setters*. Dessa kan också genereras genom Netbeans. Vi trycker bara *alt-insert* igen, och väljer *Getter and Setters* denna gång. Vi väljer vilka variabler vi ska skapa setters och getters för, och trycker på *Generate*. Om man är nybörjare kan man gott skapa setters och getters för alla variabler. Man kan alltid ta

bort dem senare.



27.1.5 Steg 5 Skapa övriga metoder

Ofta vill man ha egna metoder i klassen. I detta fall vill jag åtminstone ha två. En för att flytta bollen, och en för att rita ut den.

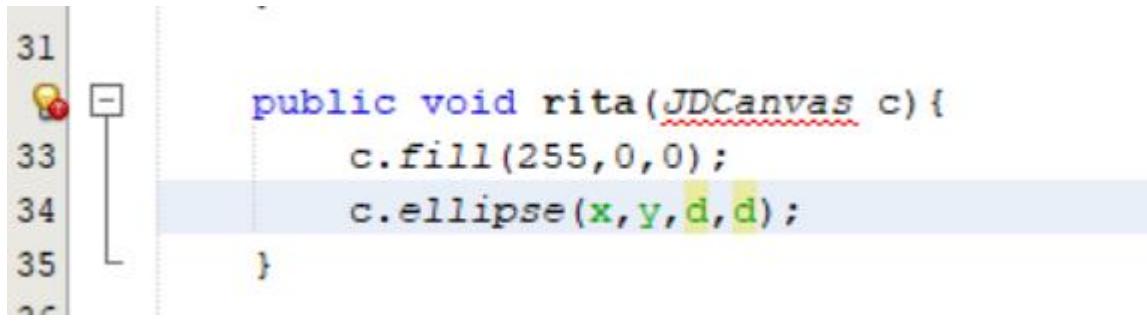
27.1.6 Metod för att flytta bollen.

Metoden ska lägga till speedx till vår x-variabel, och på motsvarande sätt lägga till speedy till y.

```
30  [-]  public void flytta() {  
31      x = x + speedx;  
32      y = y + speedy;  
33  }
```

27.1.7 Metod för att rita ut bollen

Vi skulle också vilja ha en metod som ritar ut Bollen. Metoden är tänkt att köras från Draw-metoden, och måste som inparameter ha själva Canvas klassen eftersom vi vill rita i canvasen.

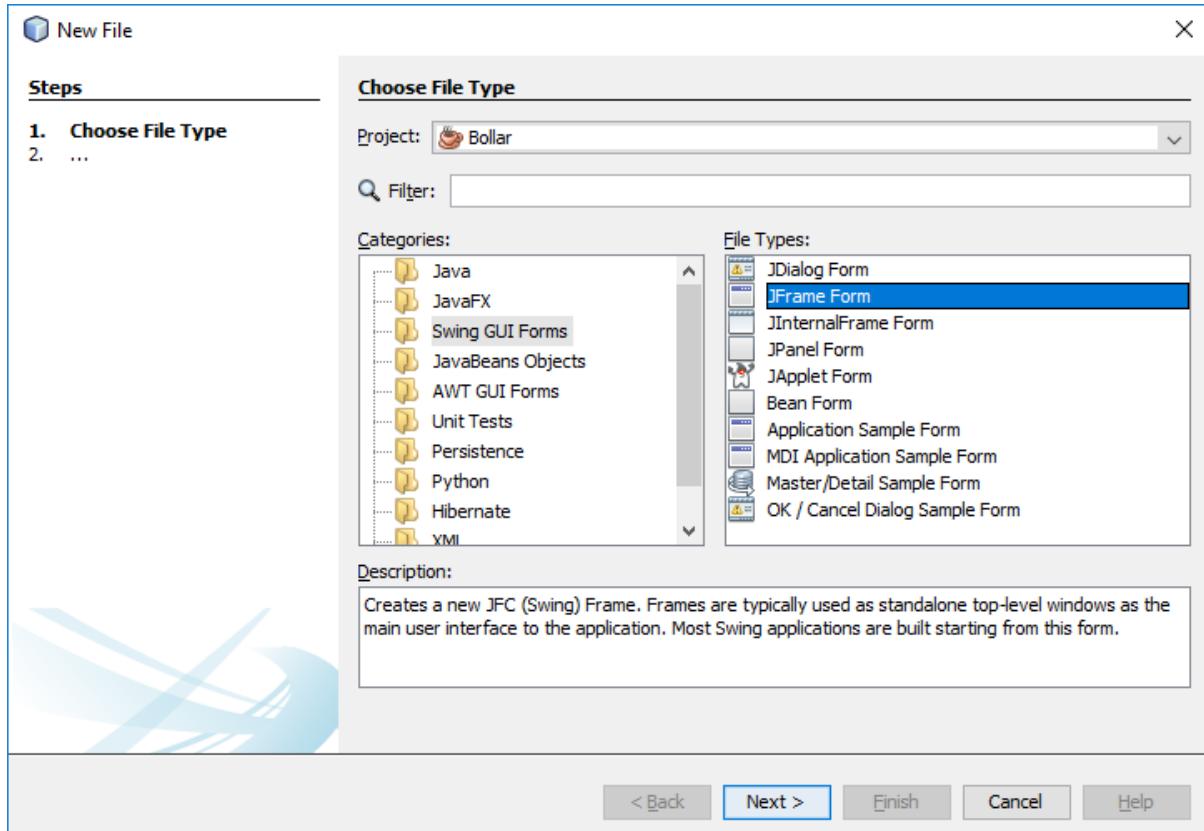


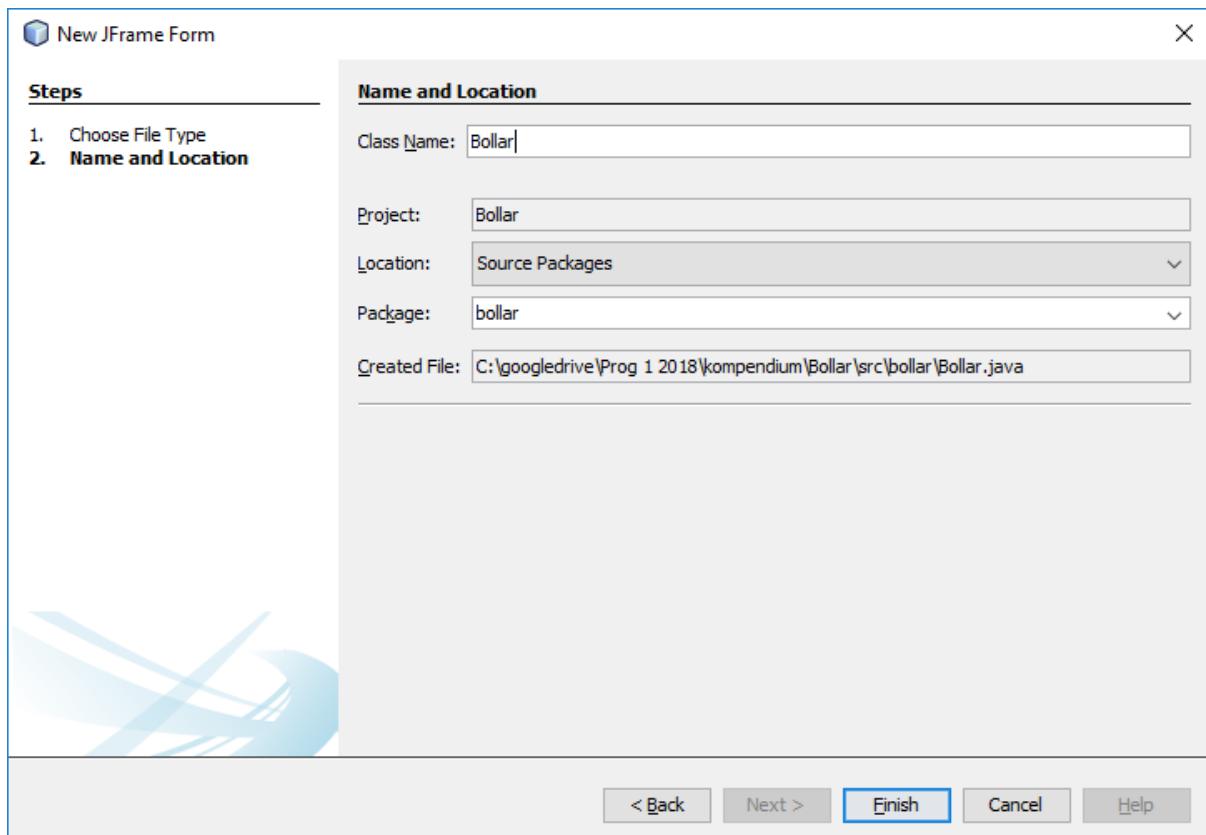
```
31
32      public void rita(JDCanvas c) {
33         c.fill(255,0,0);
34         c.ellipse(x,y,d,d);
35     }
36
```

Nu ser vi att det lyser rött på JDCanvas. Det beror på att vi inte lagt till JDCanvas libbet.

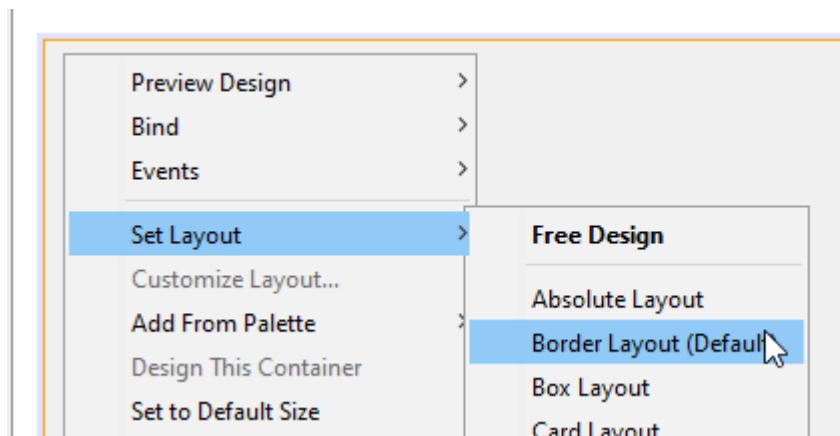
28.1 Skapa en JDCanvasklass som använder din Bollklass

Om du inte har kört kapitlet Rita i Java på Processingvis, så är det dags att göra det nu. Du behöver först skapa en klass



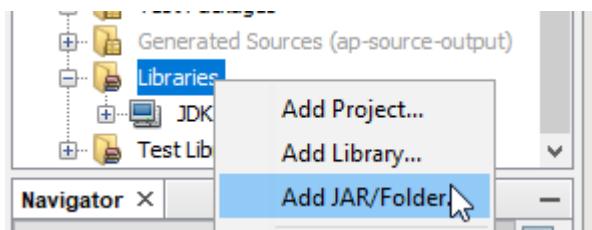


Vi högerklickar och väljer borderlayout som tidigare.

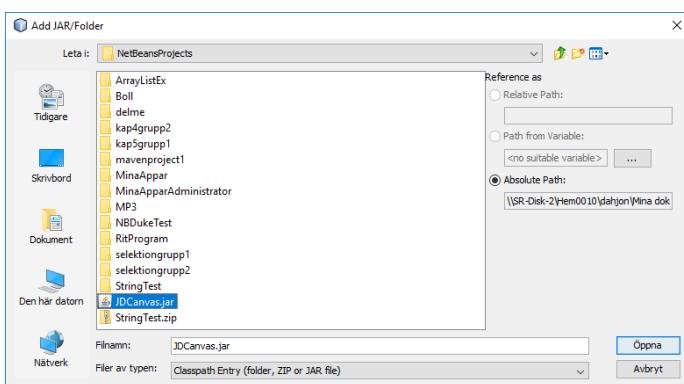


28.2.1 Lägga till JDCanvas libbet

Högerklicka på Libraries i ditt projekt och välj *Add Library...*

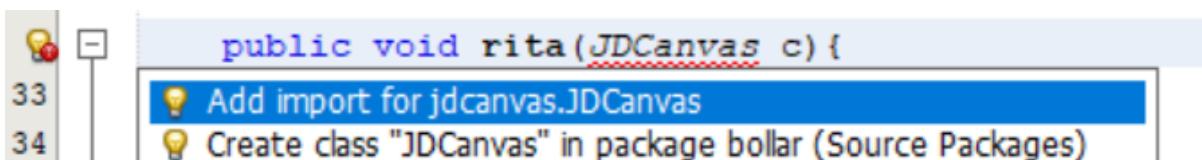


Vi letar sedan reda på filen JDCanvas.jar och väljer den.



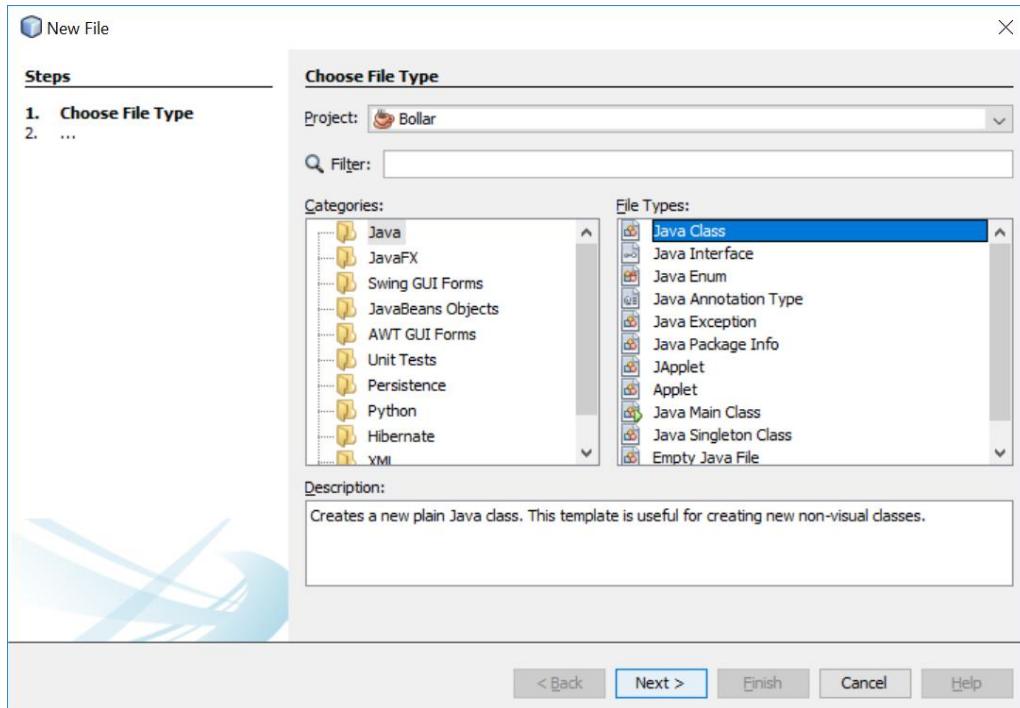
Nu kan vi börja använda klassen.

Nu behöver du gå in i Boll-klassen och importera libbet. Det gör du enklast genom att trycka på glödlampan bredvid din rita-metod och välja *Add import for jdcanvas.JDCanvas*.

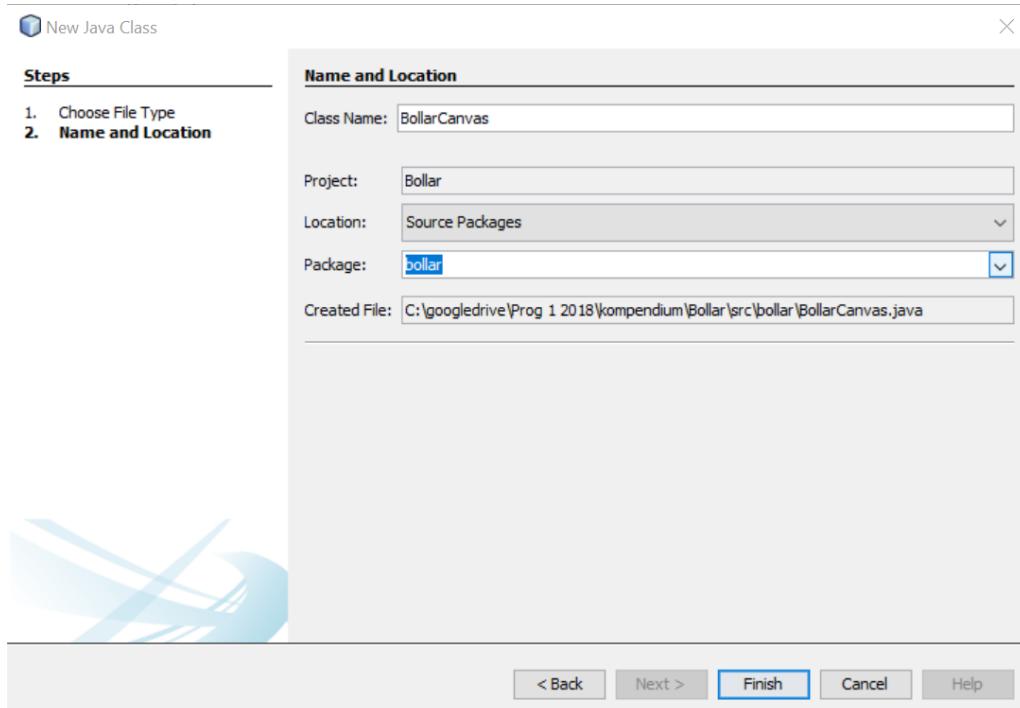


28.2.2 En boll som rör på sig

Vi måste börja med att skapa en klass för vår sketch/Canvas. Så tryck på ny fil igen, och välj Java Class



Döp klassen till BollarCanvas. Det är också viktigt att BollarCanvas ligger i samma paket som de andra klasserna, så i vårt fall ser vi till att det står *bollar* efter *Package*.



Nu behöver vi skriva extends JDCanvas för att göra klassen till en rityta. Vi behöver också trycka på glödlampan för att

```
1 package bollar;
2
3 public class BollarCanvas extends JDCanvas{
4
5 }
```

The code editor shows a warning icon next to the word "JDCanvas" in the line "public class BollarCanvas extends JDCanvas{". A tooltip appears with two options: "Add import for jdcanvas.JDCanvas" and "Create class "JDCanvas" in package bollar (Source Packages)".

Vi skulle nu vilja att bollen skulle kunna flytta sig av sig själv

Vi anropar flytta och rita i draw-metoden.

```
1 package bollar;
2
3 import jdcanvas.JDCanvas;
4
5 public class BollarCanvas extends JDCanvas{
6     Boll b = new Boll(200,200,30,2,1);
7
8     public void setup() {
9         size(400,400);
10        fill(0);
11    }
12
13     public void draw() {
14         background(0);
15         b.flytta();
16         b.rita(this);
17     }
18 }
```

Det skulle ju vara trevligt om vi också kunde få bollen att studsa som i exemplet studsande boll som vi gjorde i Processing. Gå gärna tillbaka till det exemplet och se hur vi gjorde där.

Men för att det ska köras behöver du också lägga till din Canvas i själva huvudklassen *Bollar*.

```

1 package bollar;
2
3 public class Bollar extends javax.swing.JFrame {
4     BollarCanvas canvas = new BollarCanvas();
5     /**
6      * Creates new form Bollar
7      */
8     public Bollar() {
9         initComponents();
10        add(canvas);
11        pack();
12    }
13
14
15     public void flytta() {
16         x = x + speedx;
17         y = y + speedy;
18         if(x>=400 || x<=0) {
19             speedx=-speedx;
20         }
21         if(y>=400 || y<=0) {
22             speedy=-speedy;
23         }
24     }
25 }
```

Nu borde det nya programmet fungera precis som det gamla exemplet med studsande bollar ovan. Det kan tyckas att detta var mycket jobb för ingenting, men det vackra uppstår när vi vill köra flera bollar samtidigt.

Vi skapar helt enkelt en array av bollar. Vi byter ut den första raden i klassen ProcessingBollar till:

```
Boll[] bollar = new Boll[60];
```

Vi skapar en setup-metod och använder en for-sats i setup-metoden för att skapa många bollar

```

15     Boll[] bollar = new Boll[20];
16
17     public void setup() {
18         for (int i = 0; i < bollar.length; i++) {
19             bollar[i] = new Boll(200, 200, 30,2,1);
20         }
21     }
```

Sedan blir vi också tvungen att använda en till for-loop i draw-metoden för att se till att flytta och rita-metoderna körs i draw-metoden.

```

public void draw() {
    background(0);
    for (int i = 0; i < bollar.length; i++) {
        Boll b = bollar[i];
        b.flytta();
        b.rita(this);
    }
}

```

Om ni nu testkör programmet kommer ni att märka att ni inte märker så stor skillnad. Det skapas visserligen 30 stycken bollar som flyttas och ritas ut. Men eftersom de startar på samma ställe och har samma hastighet så kommer de att rita ut på samma ställe ovanpå varandra...

Vi behöver se till att hastigheten inte är samma för alla bollar. Vi ändrar så att hastigheten slumpas fram för varje boll:

Nu kan vi testköra och se vad som händer. Nedan finns också koden till hela programmet.

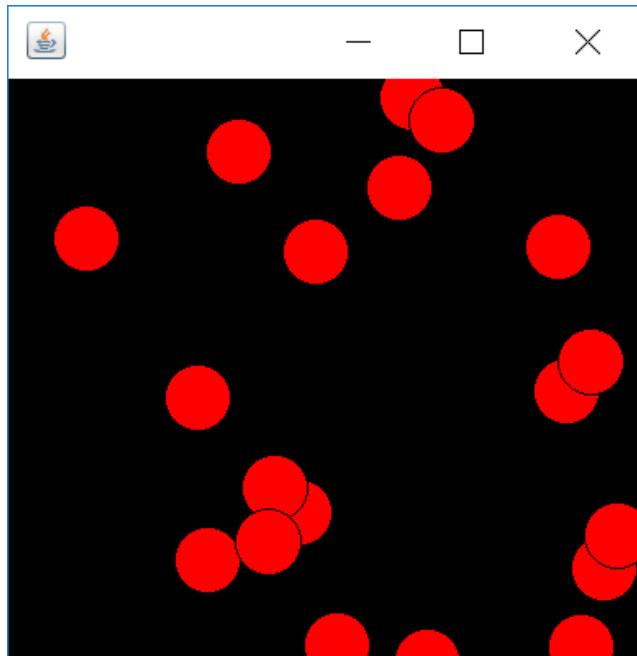
```

public class BollarCanvas extends JDCanvas {

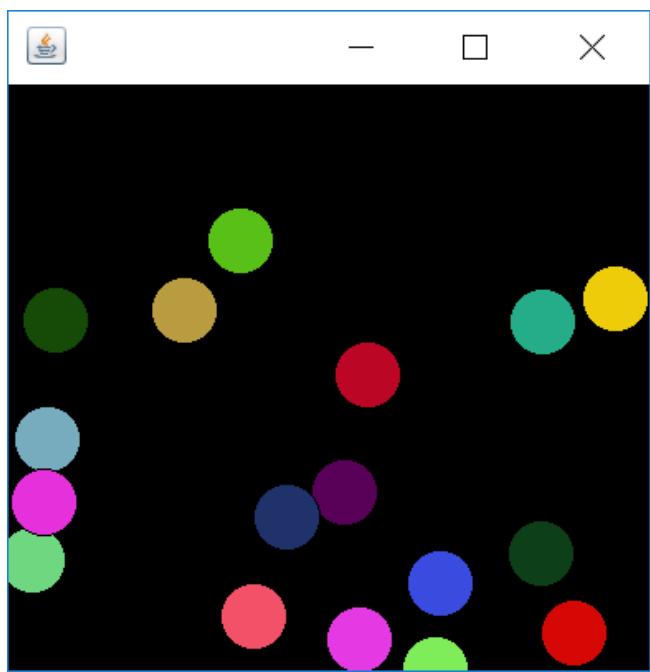
    Boll[] bollar = new Boll[20];
    Random rand = new Random();

    public void setup() {
        size(400, 400);
        for (int i = 0; i < bollar.length; i++) {
            bollar[i] = new Boll(200, 200, 40, rand.nextDouble(), rand.nextDouble());
        }
    }
}

```

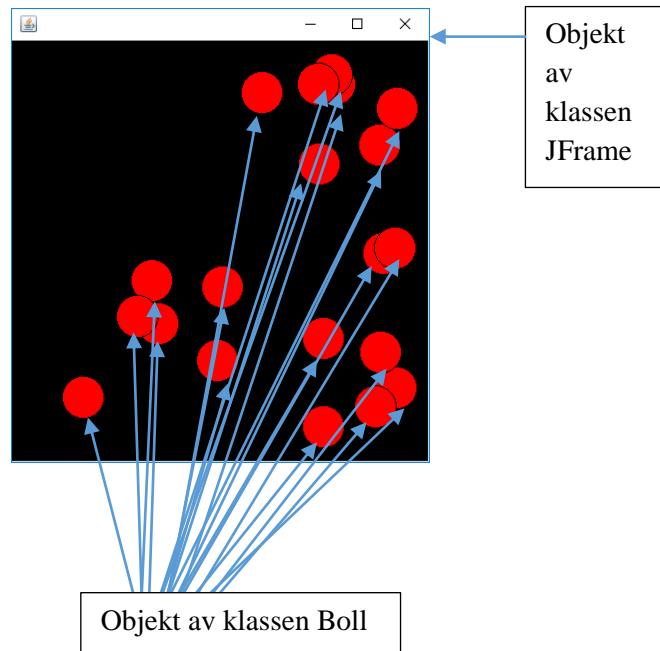


Du kan nu på egen hand lägga till egenskaperna r, g och b till Boll-klassen, skapa setters och getters, och anropa detta från setup-metoden i ProcessingBollar-klassen.

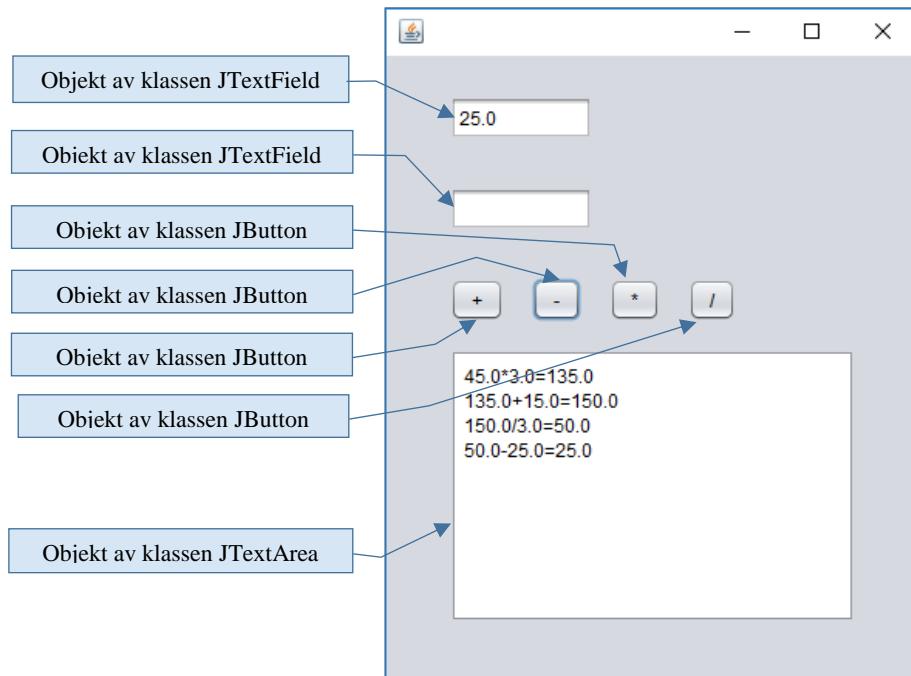


29 Teori om objektorientering

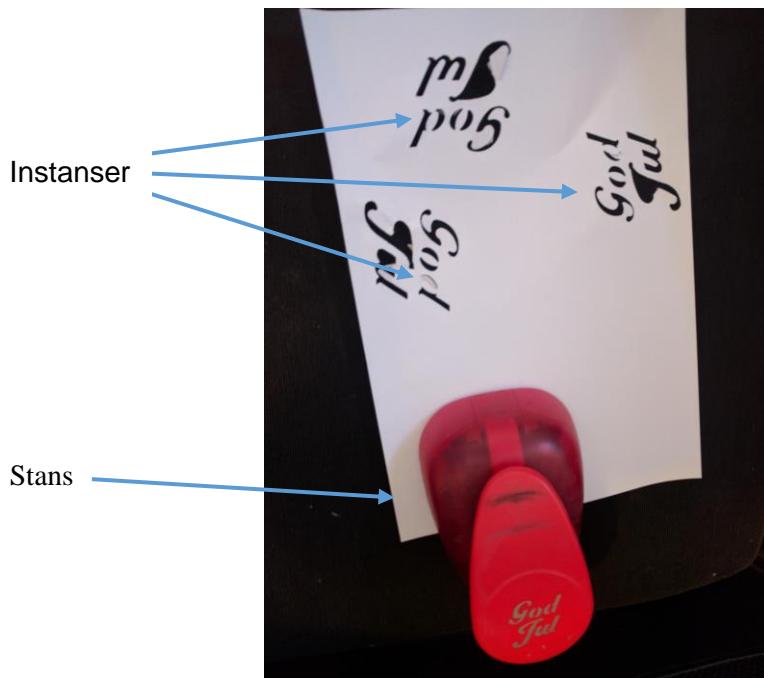
Java är ett objektorienterat språk. Det innebär att programkoden beskriver ett antal klasser. När programmet körs skapas ett antal objekt utifrån klasserna. Till exempel så skapade vi ovan en klass för en boll, och sedan vid körning skapades ett antal objekt från den klassen. Men vi hade också en klass som representerade själva ritytan, och en klass som representerade själva fönstret.



I miniräknarexemplet hade vi i stället bara en klass som var specifik för vårt program och det var själva klassen som representerade själva programfönstret. Men vi använde här många av Javas inbyggda klasser, till exempel *JTextField*, *JButton* och *JTextArea*



Man liknar ofta en klass vid en stans och objektet vid en instans, se bilden nedan. För er som inte vet vad en stans är för något så kan jag berätta att en stans är en form för klippning. Det mest vardagliga exemplet är en håslagare som är en stans och de hålade pappret är en instans. Det finns också andra typer av instanser för hemmabruk. Till exempel god jul-stansen på bilden nedan



På motsvarande sätt så är alltså klassen till exempel vår bollklass ovan en stans och objekten till exempel bollarna en instans.

```
import java.awt.Color;
import jdccanvas.JDCCanvas;

public class Boll {
    float x;
    float y;
    float d;
    float speedx;
    float speedy;

    public Boll(float x, float y, float d, float speedx, float speedy) {
        this.x = x;
        this.y = y;
        this.d = d;
        this.speedx = speedx;
        this.speedy = speedy;
    }

    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }

    public void setY(float y) {
        this.y = y;
    }

    public float getD() {
        return d;
    }

    public void setD(float d) {
        this.d = d;
    }

    public float getSpeedx() {
        return speedx;
    }

    public void setSpeedx(float speedx) {
        this.speedx = speedx;
    }

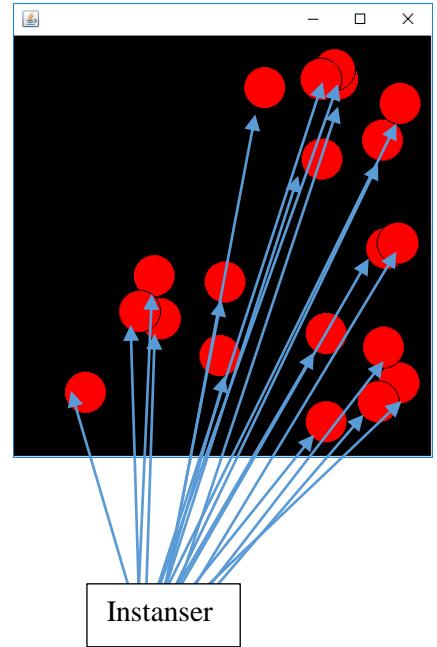
    public float getSpeedy() {
        return speedy;
    }

    public void setSpeedy(float speedy) {
        this.speedy = speedy;
    }

    public void flytta(){
        x+=speedx;
        y+=speedy;
        if(x<=0 || x>=400){
            speedx=-speedx;
        }
        if(y<=0 || y>=400){
            speedy=-speedy;
        }
    }

    public void vita(JDCCanvas c){
        c.fill(Color.red);
        System.out.println("x=" + x + ", y=" + y + ", d=" + d);
        c.ellipse(x, y, d, d);
    }
}
```

Stans



Instanser

Instansiering

```
Boll[] bollar = new Boll[20];
Random rand = new Random();

@Override
public void setup() {
    size(400, 400);
    for (int i = 0; i < bollar.length; i++) {
        bollar[i] = new Boll(200, 200, 40, 0.2 * rand.nextDouble(), rand.nextDouble());
    }
}
```

29.1 Tillgänglighet

- Public
 - Betyder offentlig
 - Medlemmarna kan användas i alla andra klasser i programmet.
- Private
 - Betyder att variabeln är gömd i klassen och oåtkomlig för andra klasser.

30 ArrayList igen

När vi jobbar med klasser och objekt är det ofta praktiskt att lagra många objekt i en ArrayList. Därför vill jag påminna er om hur ArrayList fungerar. Det är samma ArrayList som vi hade i Processing. Processing använder sig av Javas vanliga ArrayList så det är bokstavligen ingen skillnad, förutom att vi i vanlig java måste ha med importdeklarationen:

```
import java.util.ArrayList;
```

Det som är så praktiskt med ArrayList i förhållande till en vanlig array är att vi kan lägga till och ta bort objekt på ett enkelt sätt, vilket vi inte kunde göra i en array.

Tabellen nedan visar hur vi kan skapa, lägga till, ta bort och ta ut ett objekt från en ArrayList:

Beskrivning	Syntax	Exempel
Skapa ny ArrayList	ArrayList<Klass> namn = new ArrayList();	ArrayList<MP3> lista = new ArrayList();
Lägga till objekt sist i listan	namn.add(objekt);	MP3 mp3 = new MP3(titel, artist, fil); lista.add(mp3);
Ta bort objekt från listan	namn.remove(index)	lista.remove(0); //Ta bort det första objektet från listan.
Plocka ut ett objekt ur listan	namn.get(index)	MP3 mp3 = lista.get(1);
Hämta storleken på listan	namn.size()	int langd = lista.size();

Om ni vill veta allt man kan göra med listan så titta på:

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

31 Klasser i Java steg för steg - MP3

[Det finns en video till denna övning](#)

Klasser är ett trevligt sätt att kunna samla ihop metoderna och de variablerna som hör ihop med dessa metoder till en gemensam enhet. Det kan handla om till exempel ett grafiskt form som en cirkel eller rektangel, det kan handla om en spelarfigur i ett spel, det vara en knapp i ett grafiskt gränssnitt, eller också så handlar det helt enkelt om att lagra information om något fysiskt objekt, eller varför inte en musikfil som i exemplet nedan.

Vi ska skapa ett program som håller ordning på dina MP3Filer.

De fem stegen:

- Steg 1 Skapa klassfilen
- Steg 2 Skapa dina variabler
- Steg 3 Infoga konstruktör.
- Steg 4 Infoga Setters och Getters-metoder
- Steg 5 Skapa övriga metoder

31.1 Steg 1 Skapa klassfilen

Vi börjar med att skapa ett nytt projekt och då kommer också en fil som heter MP3 att skapas. Vi tar bort main-metoden i den.

31.2 Steg 2 Skapa dina variabler

Det första man måste fundera på är vilka variabler som man behöver lagra i vår klass.

Vi ska börja med att skapa en klass för MP3-skivor. Klassen ska ha följande fält

- titel - som innehåller låtens titel
- artist - som innehåller låtens artist
- fil - som lagrar den absoluta sökvägen för din fil.

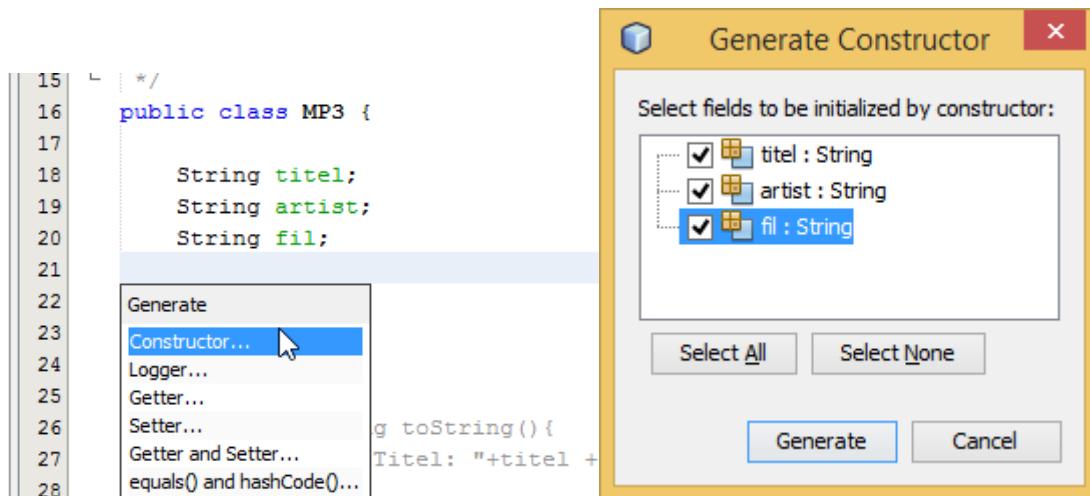
```
public class MP3{  
    private String titel;  
    private String artist;  
    private String fil;  
}
```

31.3 Steg 3 Infoga konstruktör.

En konstruktör är som en vanlig metod som körs när man skapar ett objekt. Den används ofta för att kopiera in startvärden i klassen.

I Netbeans kan man enkelt generera en konstruktör genom att trycka på *alt-insert* och välja *Constructor..*.

Sedan väljer vi vilka variabler som vi ska kopiera in i klassen genom konstruktorn, och trycker sedan på *Generate...*



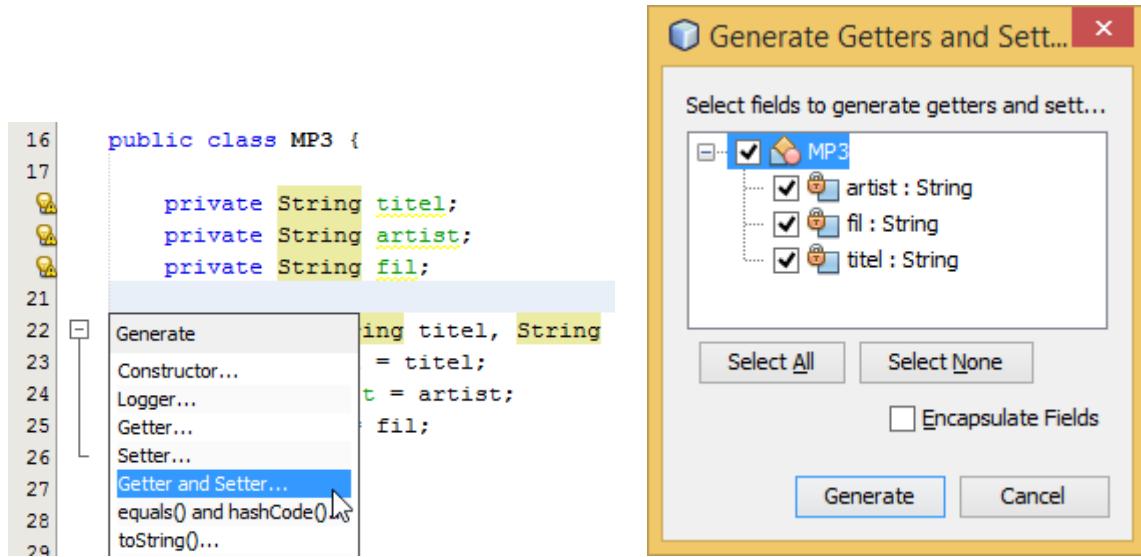
Vi kan tänka oss att Klassen är som en mall och att objektet är de kopior som skapas utifrån mallen.

```
public class MP3 {  
  
    private String titel;  
    private String artist;  
    private String fil;  
  
    public MP3(String titel, String artist, String fil) {  
        this.titel = titel;  
        this.artist = artist;  
        this.fil = fil;  
    }  
}
```

31.4 Steg 4 Infoga Setters and Getters

Vi har skrivit `private` före våra medlemsvariabler, titel, artist och fil. Det betyder att vi inte kommer åt dem från någon annan klass. Ofta skapar man därför speciella metoder för att komma åt dessa. Dessa kallas man oftast *Getters* och *Setters*. Dessa kan också genereras genom netbeans. Vi trycker bara *alt-insert* igen, och väljer *Getter and Setters* denna gång. Vi

väljer vilka variabler vi ska skapa setters och getters för, och trycker på *Generate*. Om man är nybörjare kan man gott skapa setters och getters för alla variabler. Man kan alltid ta bort dem senare.



```

16 public class MP3 {
17
18     private String titel;
19     private String artist;
20     private String fil;
21
22     public MP3(String titel, String artist, String fil) {
23         this.titel = titel;
24         this.artist = artist;
25         this.fil = fil;
26     }
27
28     public String getTitel() {
29         return titel;
30     }
31
32     public void setTitel(String titel) {
33         this.titel = titel;
34     }
35
36     public String getArtist() {
37         return artist;
38     }
39
40     public void setArtist(String artist) {
41         this.artist = artist;
42     }
43
44     public String getFil() {
45         return fil;
46     }
47
48     public void setFil(String fil) {
49         this.fil = fil;
50     }
51
52 }
53

```

31.5 Steg 5 Lägg till övriga metoder

Nästa steg i skapandet av vår MP3-klass är att fundera om vi behöver några övriga funktioner.

31.5.1 `toString`

Vi kommer att behöva en funktion som ger oss en sträng med all information om objektet. Den kommer vi att kalla för `toString()` som är ett standardnamn för sådana metoder i java.

31.5.2 play

Vi skulle också vilja ha en metod som spelar upp mp3 filen. Jag har aldrig testat det här själv tidigare så jag google på det och hittade följande sida:

<https://www.daniweb.com/programming/software-development/threads/475808/how-to-play-mp3-files-in-java-using-eclipse>

Det här förslaget fick jag när det gäller att spela upp MP3filer i java.

yes, it's really that easy, and is 100% mainstream Oracle-supported Java.

Now the "gotcha". If you run this as part of a non-JavaFX application then the JavaFX € properly. However, you can force JavaFX to initialise as part of an ordinary applicatio

```
1. new javafx.embed.swing.JFXPanel(); // forces JavaFX init
```

so that's it. In summary, a trivial player looks like:

```
1. void playMP3(String fileName) {
2.     new javafx.embed.swing.JFXPanel();
3.     String uriString = new File(fileName).toURI().toString();
4.     new MediaPlayer(new Media(uriString)).play();
5. }
```

Jag ändrar lite på förslaget för att passa vår situation.

```
55 //https://www.daniweb.com/programming/software-development/threads/475808/1
56     void play() {
57         new javafx.embed.swing.JFXPanel();
58         String uriString = new File(fil).toURI().toString();
59         new MediaPlayer(new Media(uriString)).play();
60     }

[-] public String toString(){
    return "Titel: "+titel + " Artist: "+artist + " Fil: "+fil;
}
```

Nu är vi faktiskt färdiga med vår javaklass. Nu behöver vi en användargränssnittsklass.

31.6 Användargränssnitt till MP3Player

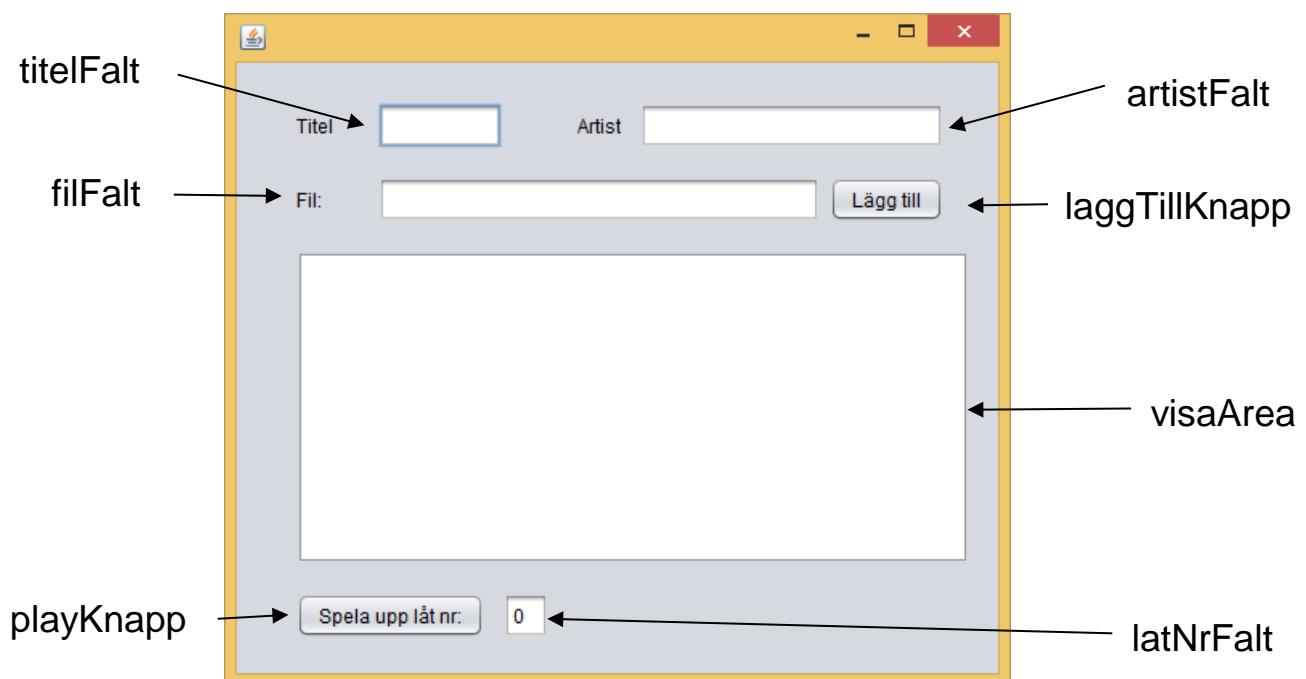
[Här börjar video 2](#)

31.7 Steg 6 Skapa användargränssnittsfilen

Skapa sedan en JFrame Form-fil som heter MP3Player.

31.8 Steg 7 Rita ut användargränssnittet

, och rita ut användargränssnittet, enligt nedan. Komponenternas variabelnamn ska namnges enligt bilden nedan.



31.9 Steg 8 skapa en ArrayList

Vi har inte använt arraylist förut. ArrayList fungerar som en array men man slipper hålla reda på hur många objekt som man skapat, samt ArrayList ser också till att alltid finns plats för ett till element. För att skapa array-listan skriver du:

```
15  public class MP3Player extends javax.swing.JFrame {  
16  
17      ArrayList<MP3> lista = new ArrayList<>();  
18
```

För att lägga till ett mp3-objekt i listan skriver du:

```
MP3 mp3 = new MP3(titel, artist, fil);  
lista.add(mp3);
```

31.10 Steg 9 dubbelklicka på knappen...

Dubbelklicka på knappen "Lägg till"-knappen för att skapa en funktion som körs när användaren klickar på knappen.

31.11 Steg 10 Skapa kod i händelsehanteraren

När man trycker på Lägg till knappen ska ett objekt skapas och en nytt objekt skapas, och läggas till en arraylist. Tips:

- skapa en String-variabel för förnamn och kopiera dit texten i artistFalt

```
private void laggTillKnappActionPerformed(java.awt.event.ActionEvent evt) {  
    String titel = titelFalt.getText();  
    String artist = artistFalt.getText();
```

- Gör samma sak med titel och fil
- Skapa ett nytt objekt av klassen MP3.
- Skapa en arraylist (se nedan för hjälp)
- Lägg till mp3-objektet i en arraylist. (se nedan för hjälp)

- Så här ser fela funktionen ut när den är klar:

```
116  
117  
118  
119  
120  
121  
122  
123  
...  
private void laggTillKnappActionPerformed(java.awt.event.ActionEvent evt) {  
    String titel = titelFalt.getText();  
    String artist = artistFalt.getText();  
    String fil = filFalt.getText();  
    MP3 mp3 = new MP3(titel, artist, fil);  
    lista.add(mp3);  
    visaLista();  
}
```

- När man har tryckt på Lägg till så ska hela listan skrivas ut. Skapa en metod, visaAllt(), som skriver ut hela listan och som körs varje gång som läggTill-knappen trycks.

För att skriva ut hela listan behöver vi en for loop. Det påminner om hur man gör hör att skriva ut en array. Men storleken på en arraylist får man med hjälp av size() - metoden, och varje objekt får man tag på genom get()-metoden. Man kan få hjälp med att skriva en for-loop som går igenom en lista. Genom att skriva forl och sedan tryck på tabbknappen.

Då får du upp något i denna stilén:

```
for (int i = 0; i < lista.size(); i++) {  
    MP3 m = lista.get(i);
```

Men från början stod det get istället för m. Sedan är det bara att skriva ut cd

Hela metoden blir:

```
146     private void visaLista() {
147         txaLista.setText("");
148         for (int i = 0; i < lista.size(); i++) {
149             MP3 m = lista.get(i);
150             txaLista.append(i+": "+m.toString() + "\n");
151         }
152     }
153 }
```

När vi gjort detta så kan vi testa att lägga till en mp3 låt och se att den dyker upp i listan.

Nästa steg skulle nu vara att kunna spela upp en låt. Vi går in i designläget och dubbelklickar på *Spela upp låt nr*-knappen. Sedan behöver vi hämta strängen från *latNrFalt*-fältet och sedan göra om det till integer. Sedan hämtar vi in valt mp3-objekt från listan, och slutligen kör vi play-metoden på mp3-objektet.

```
146     private void playKnappActionPerformed(java.awt.event.ActionEvent evt) {
147         String nrStr = latNrFalt.getText();
148         int nr = Integer.parseInt(nrStr);
149         MP3 mp3 = lista.get(nr);
150         mp3.play();
151     }
152 }
```

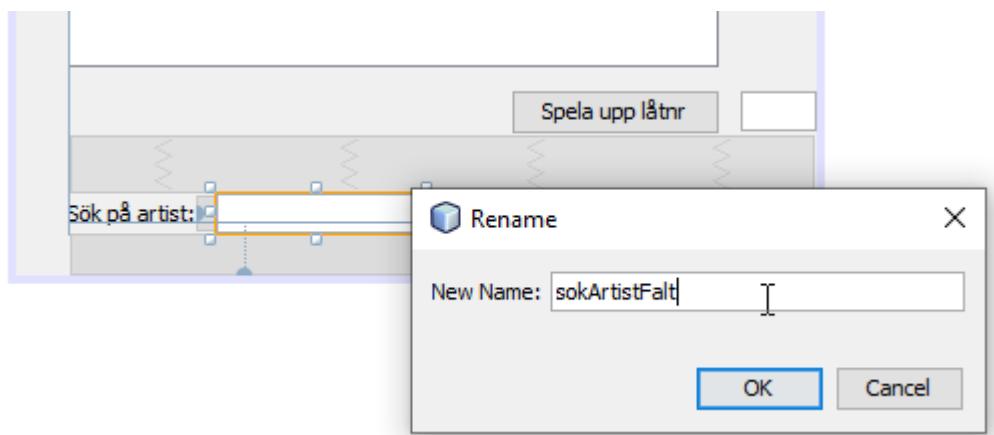
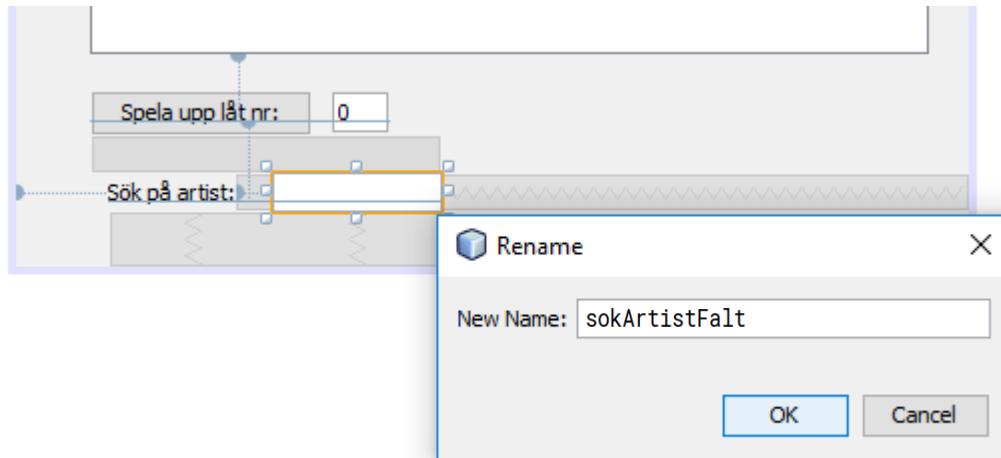
Förslag till utökningar.

- Använd JFileChooser för att välj mp3fil.
- Inför extra textfält och sökknapp för att söka på mp3fil
- Inför möjlighet att spara lista: Du kan använda ArrayListSaveable som finns i libbet sparare.jar. [Se intruktioner här.](#)

32 Sökning

När man har gjort ett program med något register av något slag som i MP3-såpelaren ovan vill man ofta kunna söka i registret. I vårt fall med en MP3-spelare så är det mest uppenbart att man vill kunna söka på artist, så det tänkte jag visa här.

Vi börjar med att skapa ett textfält där man ska ha möjlighet att skriva in den artist man vill söka på. Vi kallar textfältet *sokArtistFalt*. Sökningen ska ske när man trycker på *enter*-tangenten.



Vi dubbelklickar på textfältet för att skapa en metod som kommer att köras när man trycker på enter-tangenten när vi står i textfältet.

Det som ska stå i metoden är faktiskt ganska likt det som står i metoden `visaAllt()`, därför börjar vi med att kopiera innehållet i `visaAllt()`.

```
private void sokArtistFaltActionPerformed(java.awt.event.ActionEvent evt) {
    visaArea.setText("");
    for (int i = 0; i < lista.size(); i++) {
        MP3 m = lista.get(i);
        visaArea.append(i + ": " + m.toString() + "\n");
    }
}
```

Först måste vi också hämta texten från sökfältet.

```
private void sokArtistFaltActionPerformed(java.awt.event.ActionEvent evt) {
    visaArea.setText("");
    String sok = sokArtistFalt.getText();
    ...
```

Den största skillnaden i denna metod jämfört med `visaAllt()` är att vi här bara ska visa de MP3-objekt med artisten som vi skrivit in i sökfältet. Vi ska med andra ord lägga till en if-sats.

Hela metoden med if-sats blir:

```
private void sokArtistFaltActionPerformed(java.awt.event.ActionEvent evt) {  
    visaArea.setText("");  
    String sok = sokArtistFalt.getText();  
    for (int i = 0; i < lista.size(); i++) {  
        MP3 m = lista.get(i);  
        if (sok.equals(m.getArtist())) {  
            visaArea.append(i + ": " + m.toString() + "\n");  
        }  
    }  
}
```

33 Kundlistaexempel

Vi ska skapa en kundlista till en skoaffär.

Vi börjar med att skapa en klass för en kund:

Den ska lagra:

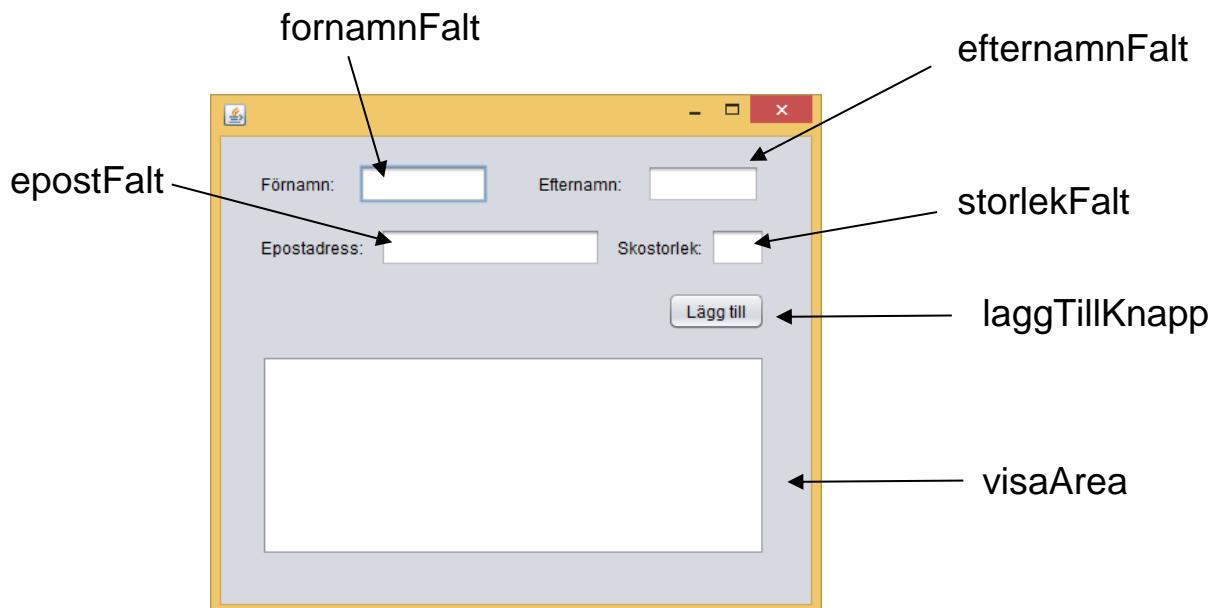
- Förfannm
- Efternamn
- Epostadress
- Skostorlek

```
6  package kundlista;  
7  
8  /**  
9  *  
10 * @author dahjon  
11 */  
12 public class Kund {  
13     private String fornamn;  
14     private String efternamn;  
15     private String epost;  
16     private int storlek;  
17  
18 }
```

Vi infogar konstruktör, setters och getter, samt en `toString`-metod.

```
12 public class Kund {
13     private String fornamn;
14     private String efternamn;
15     private String epost;
16     private int storlek;
17
18     public Kund(String fornamn, String efternamn, String epost, int storlek) {
19         this.fornamn = fornamn;
20         this.efternamn = efternamn;
21         this.epost = epost;
22         this.storlek = storlek;
23     }
24
25     public String getFornamn() {
26         return fornamn;
27     }
28
29     public void setFornamn(String fornamn) {
30         this.fornamn = fornamn;
31     }
32
33     public String getEfternamn() {
34         return efternamn;
35     }
36
37     public void setEfternamn(String efternamn) {
38         this.efternamn = efternamn;
39     }
40
41     public String getEpost() {
42         return epost;
43     }
44
45     public void setEpost(String epost) {
46         this.epost = epost;
47     }
48
49     public int getStorlek() {
50         return storlek;
51     }
52
53     public void setStorlek(int storlek) {
54         this.storlek = storlek;
55     }
56     public String toString() {
57         return fornamn + " " + efternamn + " " + epost + " Storlek: " + storlek;
58     }
59 }
60
```

Vi skapar en ny JFrame Form som får heta *KundLista*. Vi ritar upp följande gränssnitt och sätter också namnen på textfälten och knappen enligt nedan.



Vi skapar vår arraylist

```
12  */
13  public class KundLista extends javax.swing.JFrame {
14      ArrayList<Kund> lista = new ArrayList<>();
15  /**
16  * Creates new form KundLista
17  */
```

Lägger också till import för ArrayList

```
12  */
13  public class KundLista extends javax.swing.JFrame {
14      ArrayList<Kund> lista = new ArrayList<>();
```

A tooltip appears over the line 'ArrayList<Kund> lista = new ArrayList<>();'. It contains the following options:

- Add import for java.util.ArrayList
- Create class "ArrayList" in package Kundlista (Source Packages)
- Create class "ArrayList" in kundlista.KundLista
- Move initializer to constructor(s)

```
15  /**
16  * Creates new form KundLista
17  */
18      initComponents();
19  }
```

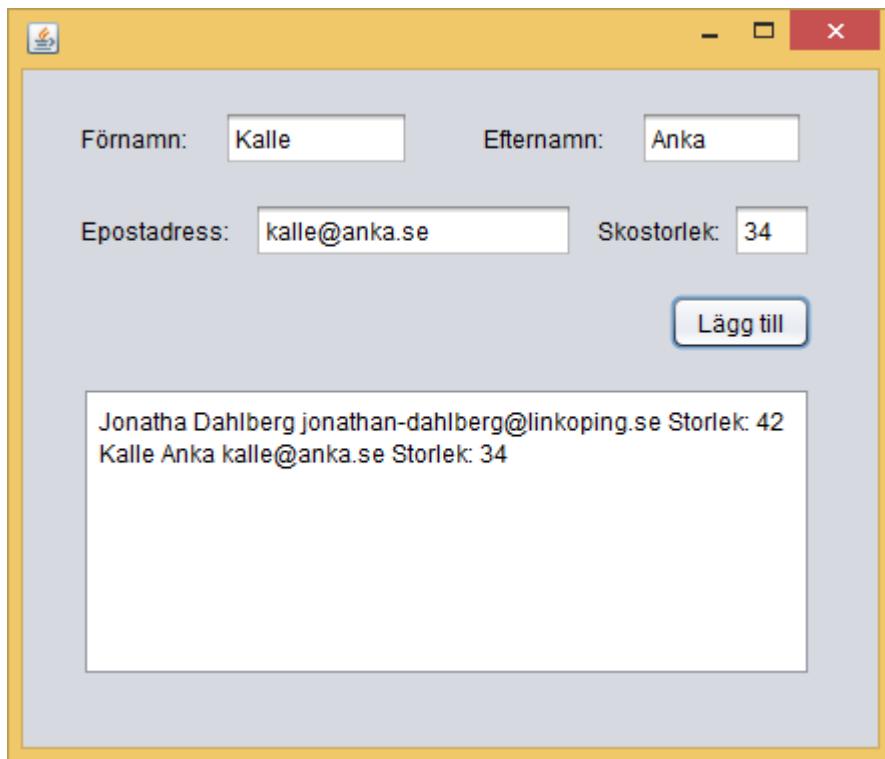
Vi dubbelklickar på *Lägg till*-knappen så kommer en actionperformed-metod kopplad till knappen att skapas att skapas:

```
private void laggTillKnappActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

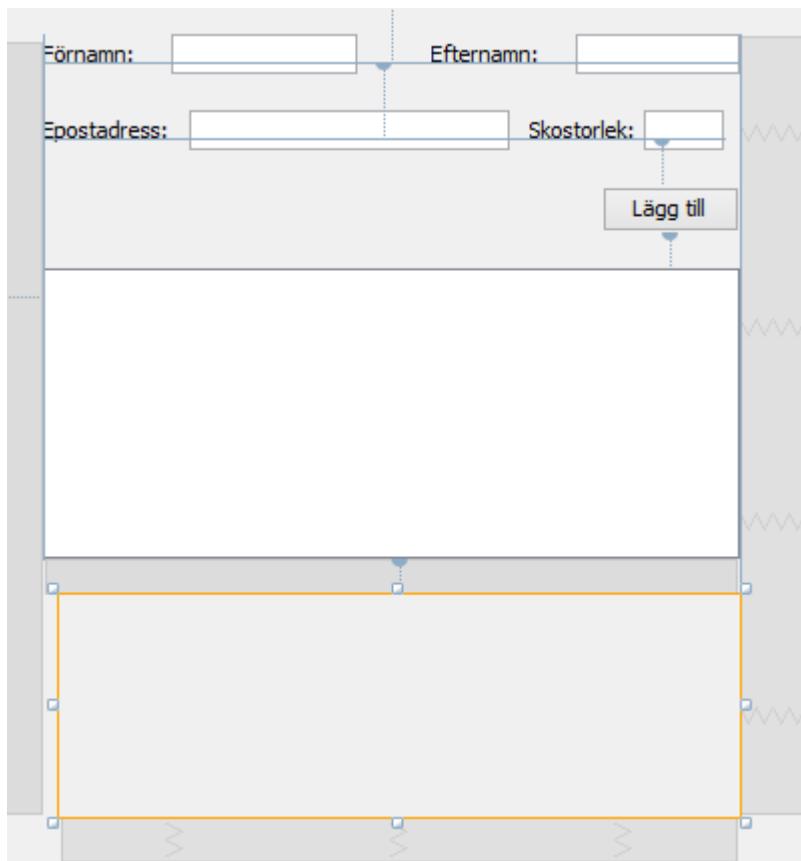
Vi infogar sedan kod för att lägga till en kund i listan när vi tryckt på knappen. Vi skapar dessutom en *visaAllt*-metod som först tömmer textarean och sedan visar upp allt innehåll i listan.

```
234 private void laggTillKnappActionPerformed(java.awt.event.ActionEvent evt) {  
235     String fornamn = fornamnFalt.getText();  
236     String efternamn = efternamnFalt.getText();  
237     String epost = epostFalt.getText();  
238     String storlekStr = storlekFalt.getText();  
239     int storlek = Integer.parseInt(storlekStr);  
240     Kund k = new Kund(fornamn, efternamn, epost, storlek);  
241     lista.add(k);  
242     visaAllt();  
243  
244 }  
245  
246 private void txfSokFornamnActionPerformed(java.awt.event.ActionEvent evt) {  
247     txaVisa.setText("");  
248     for (int i = 0; i < lista.size(); i++) {  
249         Kund k = lista.get(i);  
250         String sok = txfSokFornamn.getText();  
251         if (sok.equalsIgnoreCase(k.getFornamn())) {  
252             txaVisa.append(k.toString() + "\n");  
253         }  
254     }  
255 }  
256
```

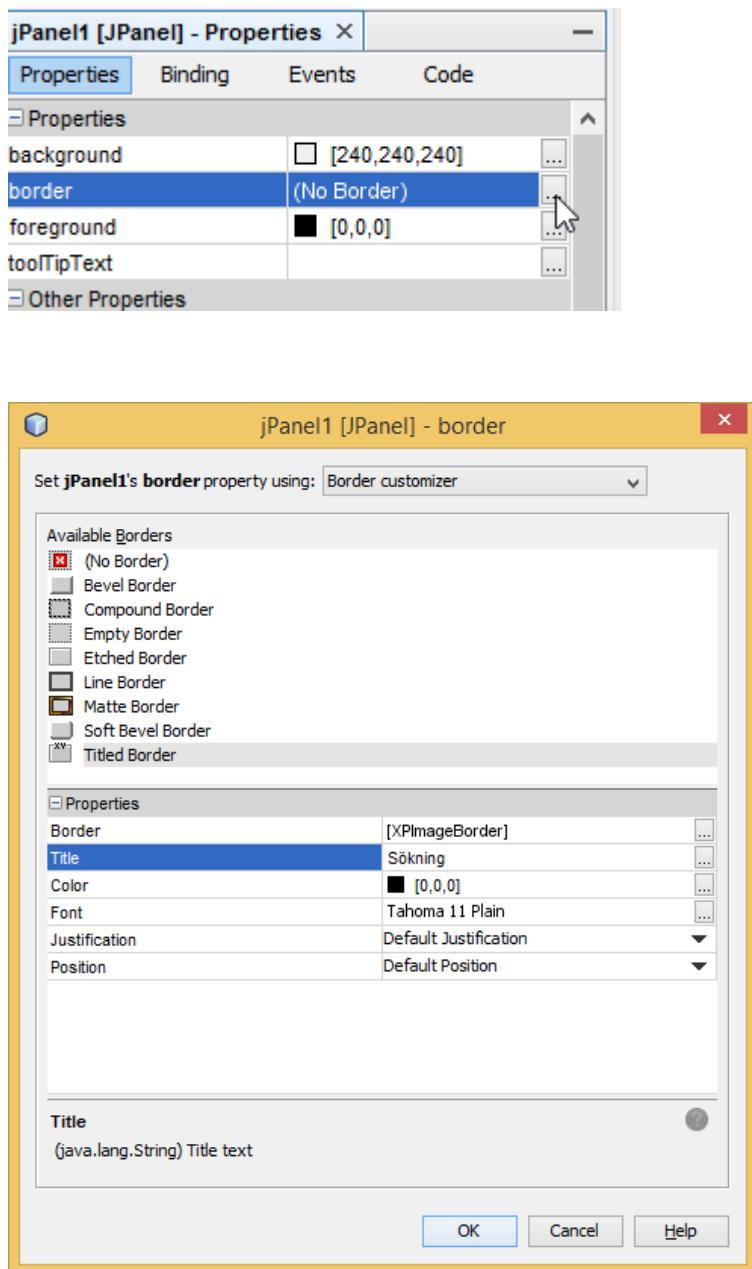
Nu borde programmet fungera:



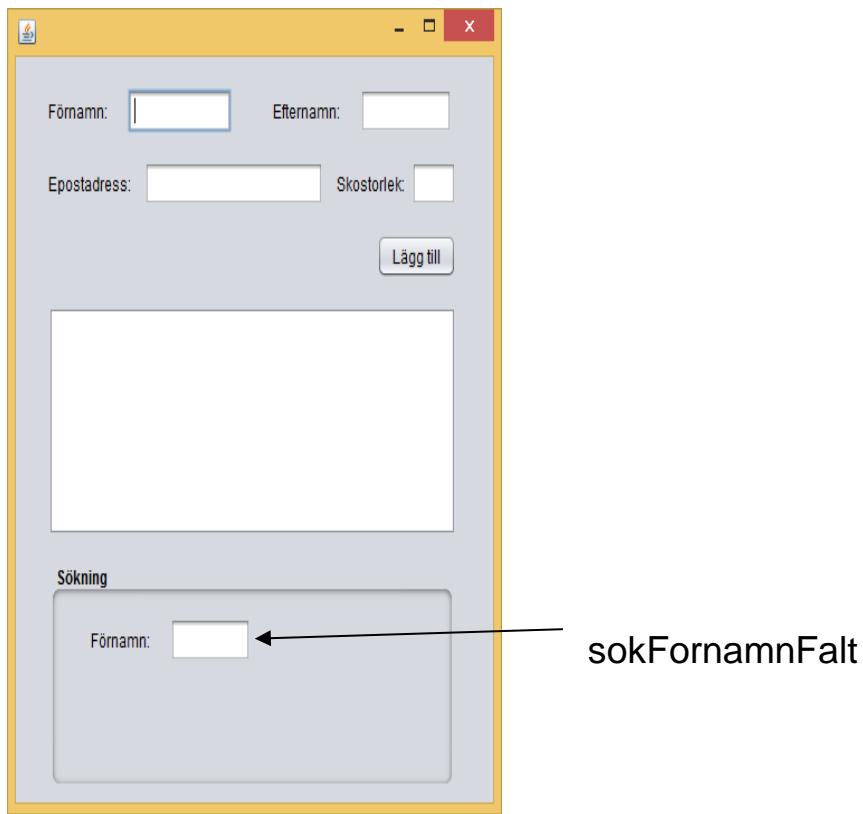
Jag lägger nu till en panel nedanför textrutan.



På panelen sätter jag en *Titled border*.



Vi lägger till ett textfält i panelen:



Nu dubbelklickar vi på textfältet. Då kommer en actionPerformed-metod att skapas för textfältet. Metoden kommer att köras när vi trycker på `enter` när vi står i textfältet.

```
246 |  private void sokFornamnFaltActionPerformed(java.awt.event.ActionEvent evt) {  
247 |     // TODO add your handling code here:  
248 | }
```

Vi skapar själva sökningen. Lägg märke till att den är identisk med `visaAllt`-metoden förutom att det finns en if-sats.

```
235 |  private void sokFornamnFaltActionPerformed(java.awt.event.ActionEvent evt) {  
236 |     txaVisa.setText("");  
237 |     for (int i = 0; i < lista.size(); i++) {  
238 |         Kund k = lista.get(i);  
239 |         String sok = sokFornamnFalt.getText();  
240 |         if (sok.equalsIgnoreCase(k.getFörnamn())) {  
241 |             txaVisa.append(k.toString() + "\n");  
242 |         }  
243 |     }  
244 | }
```

Fortsätt själva med att göra så att användaren kan:

1. Lägg till en knapp *Visa Allt*, visar allt innehåll när den trycks in.
2. Söka på Efternamn
3. Söka på Epost
4. Söka på storlek större än
5. Skapa knapp för att söka på förnamn och efternamn samtidigt, dvs hela namnet.
6. Söka på storlek mindre än
7. Skapa knapp för att söka på storlek större än och mindre än samtidigt.
8. Fixa så att ett felmeddelande kommer upp om användaren skriver in en bokstav istället för siffra i storleksfälten.

Så här skulle gränssnittet på det färdiga programmet kunna se ut. (men ni kan säkert göra det snyggare)

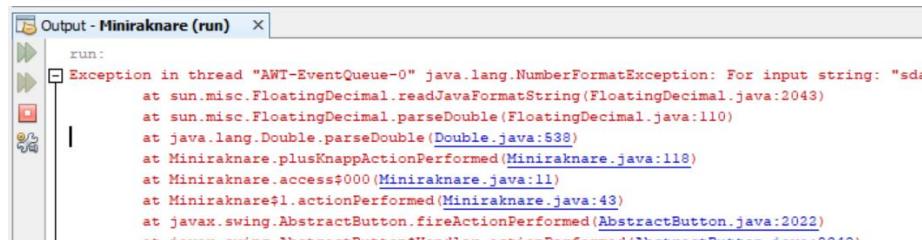
The screenshot shows a Windows application window with a yellow title bar and a red close button. The main area contains several input fields and buttons. At the top, there are four input fields: 'Förnamn:' with an empty text box, 'Efternamn:' with an empty text box, 'Epostadress:' with an empty text box, and 'Skostorlek:' with an empty text box. Below these are two buttons: 'Visa Allt' and 'Lägg till'. A large empty rectangular area is positioned below the buttons. At the bottom, there is a section titled 'Sökning' (Search) containing three input fields: 'Förnamn:' with an empty text box, 'Efternamn:' with an empty text box, and 'Hela namnet' with an empty text box. Below these are two more input fields: 'Epost' with an empty text box and 'Storlek större än' with an empty text box. To the right of the 'Epost' field is a button labeled 'Storlek mindre än' and to the right of the 'Storlek större än' field is a button labeled 'Större och mindre än'.

Tips för vissa uppgifter

34 Undantagshantering

I java finns en teknik för att fånga upp fel och hantera dem. Det kallas för undantagshantering. Grundtanken är att man utgår från att allting går bra, men om det går fel så har man möjlighet att fånga upp det. Det kan till exempel handla om att läsa från en fil eller omvandla en String till double, som skulle kunna gå fel. I andra språk som C, använder man returvärden för att skicka felmeddelanden, men det blir väldigt mycket kod för enbart felmeddelanden. Med undantagshantering kan man istället lägga felhantering samlat på ett ställe. Man säger att man kastar (throws) ett undantag (exception), när något går fel, som sedan hanteras genom att fånga (catch) undantaget. Vi kommer att visa hur det fungerar för omvandling mellan text till tal.

Vi tittar på vårt miniräknarexempel ovan. Vi vet att det fungerar om vi gör rätt, men om vi skriver in något annat än ett tal vad händer då?



Vi ser i Output-fönstret i Netbeans att det har kastats ett undantag. Och namnet på undantaget står också utskrivet: `Java.lang.NumberFormatException`, vi får också reda på vilken rad i koden som har orsakat felet, det vill säga rad 118.

Vi går tillbaka och tittar på koden som körs när vi trycker på plusknappen, och mycket riktigt så var det på rad 118 som omvandlingen från text till tal skedde.

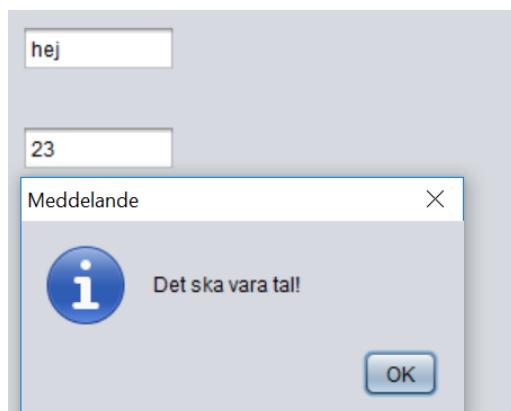
```
115     private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {  
116         String tallstr = tallSummaFalt.getText();  
117         String tal2str = tal2Falt.getText();  
118         double tall = Double.parseDouble(tallstr);  
119         double tal2 = Double.parseDouble(tal2str);  
120         double summa = tall + tal2;  
121         tallSummaFalt.setText(" " + summa);  
122         tal2Falt.setText("");  
123         resArea.append(tall+"+"+tal2+"="+summa+"\n");  
124     }  
}
```

Det vi nu behöver göra är att lägga en try-stas före den plats det kan gå fel och en catch sats

efteråt.

```
118     private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {  
119         try {  
120             String tallstr = tallSummaFalt.getText();  
121             String tal2str = tal2Falt.getText();  
122             double tall = Double.parseDouble(tallstr);  
123             double tal2 = Double.parseDouble(tal2str);  
124             double summa = tall + tal2;  
125             tallSummaFalt.setText("" + summa);  
126             tal2Falt.setText("");  
127             resArea.append(tall + "+" + tal2 + "=" + summa + "\n");  
128         } catch (java.lang.NumberFormatException ex) {  
129             JOptionPane.showMessageDialog(this, "Det ska vara tal!");  
130         }  
131     }  
132 }
```

Catch stasen liknar en metod definition på det sättet att den har en typ följt av ett variabelnamn. Men i en *catch* sats är typen alltid det undantag som man vill fånga i detta fall *java.lang.NumberFormatException*. Vet vi inte vilken typ av exception vi behöver fånga kan vi försöka provocera fram felet, vilket i detta fall betyder att vi skriver in ett felaktigt värde det vill säga bokstäver istället för siffror. Då kommer, som vi såg ovan, java att skriva ut vilket undantag som inträffade. I vissa fall till exempel vid fylläsning finns det krav på att det ska finnas undatagshantering, då kan man använda glödlampan i Netbeans för att få hjälp att skapa try-catch-satser.



34.1 Sammanfattning om undantag (exceptions)

Om fel av exceptiontyp dyker upp:



```
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "sda"
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
    at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
    at java.lang.Double.parseDouble(Double.java:538)
    at Miniraknare.plusKnappActionPerformed(Miniraknare.java:118)
    at Miniraknare.access$000(Miniraknare.java:11)
    at Miniraknare$1.actionPerformed(Miniraknare.java:43)
    at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2022)
    at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2348)
```

Kan man alltid infoga try...catch

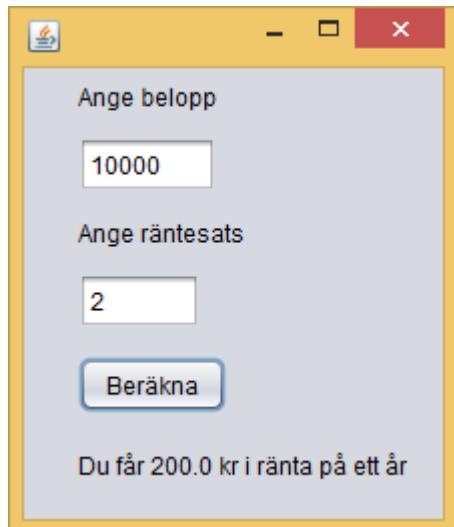
```
private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {
    String tallstr = tallSummaFalt.getText();
    String tal2str = tal2Falt.getText();
    double tall = Double.parseDouble(tallstr);
    double tal2 = Double.parseDouble(tal2str);
    double summa = tall + tal2;
    tallSummaFalt.setText(" " + summa);
    tal2Falt.setText(" ");
    resArea.append(tall+"+"+tal2+"="+summa+"\n");
}
```

```
private void plusKnappActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String tallstr = tallSummaFalt.getText();
        String tal2str = tal2Falt.getText();
        double tall = Double.parseDouble(tallstr);
        double tal2 = Double.parseDouble(tal2str);
        double summa = tall + tal2;
        tallSummaFalt.setText(" " + summa);
        tal2Falt.setText(" ");
        resArea.append(tall+"+"+tal2+"="+summa+"\n");
    } catch (java.lang.NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Det ska vara tal!");
    }
}
```

35 Blandade övningar

Övning 35-1 Räntekalkylator

Skapa ett enkelt program som räknar ut hur mycket ränta du får på dina pengar efter ett år om du har ett visst belopp och en viss ränta. Du behöver ett textfält för beloppet och en för räntan och en label för resultatet.



Övning 35-2 Färgvisare

Skapa ett program som har tre textfält där man kan mata in värden på röd, grön respektive blått. När man trycker på Visa-knappen ska färgen visas upp i en panel i den nedre delen av programmet. Programmet ska se ut som bilden nedan.



För att sätta färgen på en Panel som heter *panel* så skriver man:

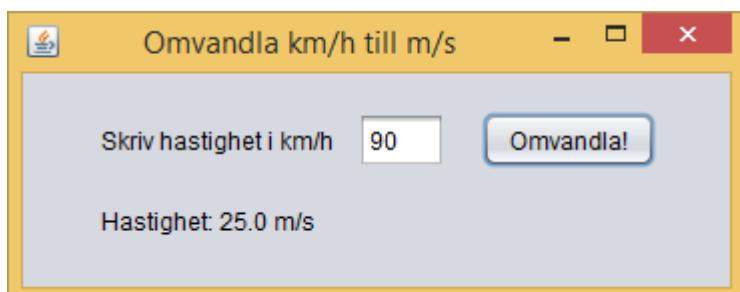
```
Color farg=new Color(255,0,0);  
panel.setBackground(farg);
```

Det finns också ett antal färdiga färger, och då kan man skriva:

```
panel.setBackground(Color.red);
```

Övning 35-3 Omvandlare från kilometer i timmen till meter per sekund

Skapa ett program som omvandlar km/h till m/s. Programmet ska ha ett textfält och en knapp, se nedan när man trycker på knappen ska resultatet visas.



Om hastigheten är över 24,5 m/s så är det storm och då ska bakgrunden på textfältet byta färg till röd.

