Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Att organisera Javakod
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

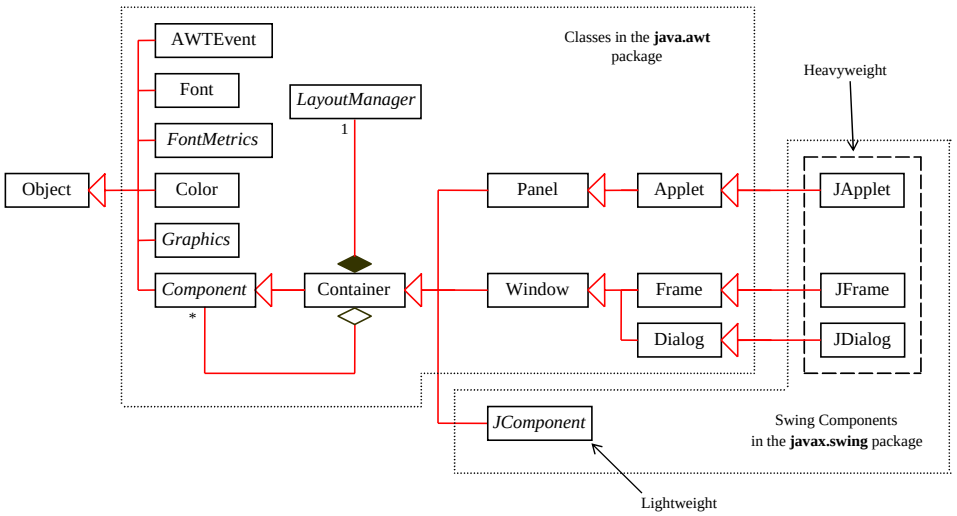
Grafik

- Grafik används i de flesta moderna programvarusystem
- Java stödjer grafik genom två paket/ klassbiblioteket

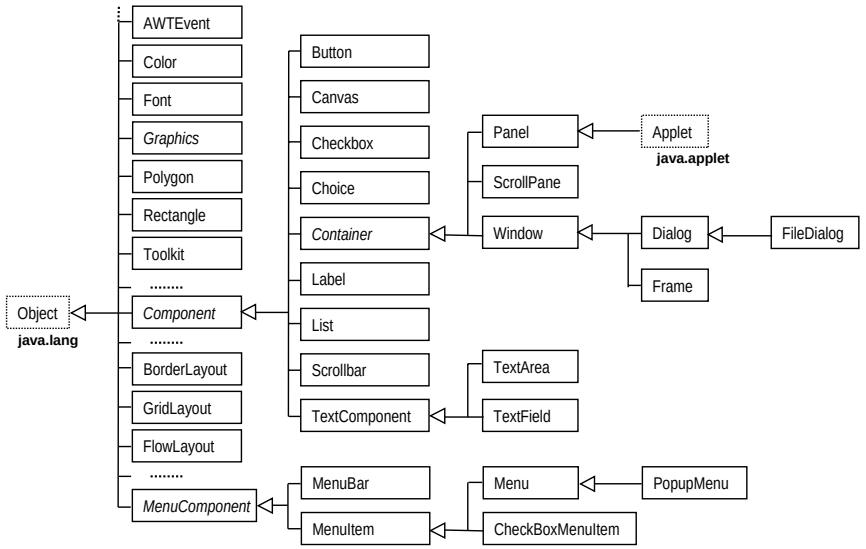
- AWT (**A**bstract **W**indowing **T**oolkit, `java.awt`)
  - Bygger på "native" kod
  - ➔Utseendet beror på platform

- `Swing` (`javax.swing`)
  - Bygger på AWT
  - Skriven helt i Java
  - "Pluggable look and feel" (`javax.swing.plaf`)
  - ➔Utseendet bestäms av programmeraren

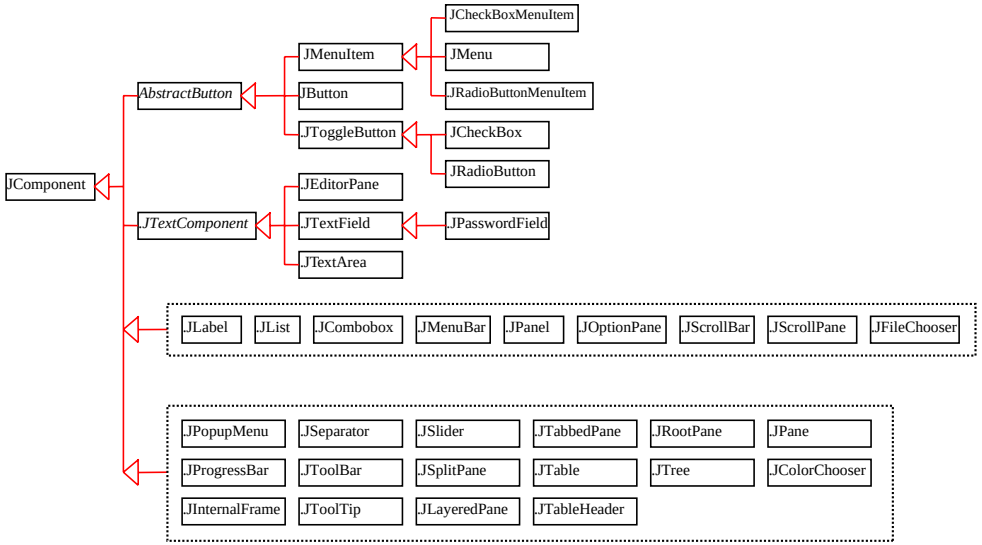
AWT och Swing



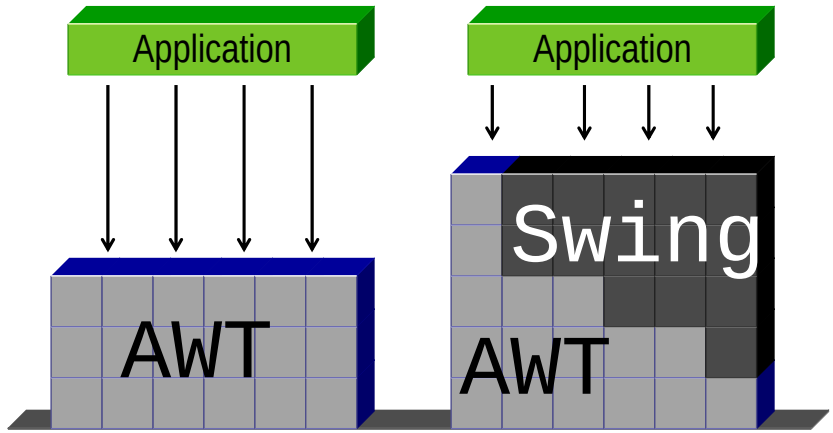
AWT



JComponent (javax.swing)



AWT och Swing (igen)



Top-Level Container

- Varje Swing/AWT program måste ha en top-level container
  - `JFrame/Frame`
  - `JDialog/Dialog`
  - `JApplet/Applet`
- Top-level containern är behållaren till alla GUI komponenter
- `JFrame/Frame` klassen kan användas för att skapa fönster

Skapa ett enkelt Fönster

```
import javax.swing.*;

public class SimpleFrame
{
   public static void main (String args[])
   {
      // skapa ett "top-level" fönster
      JFrame f = new JFrame ("Ett fönster");

      f.setSize (300, 200);
      f.setVisible (true);
   }
}
```

Fönstrets Position

- Fönstren hamnar alltid längst upp till vänster (position `<0,0>`)
- Positionen kan ändras med `setLocation` methoden (i `java.awt.Component`)

```
setLocation (x, y)
```

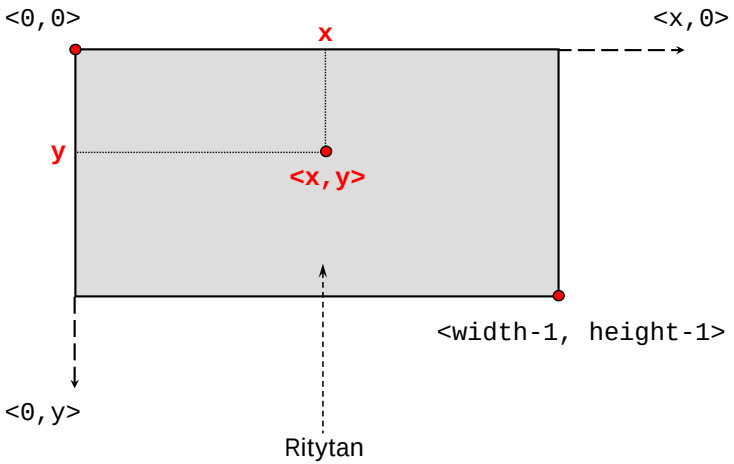
- Flyttar fönstrets (komponentens) “vänster-upp” hörn till positionen `<x,y>`

Koordinatsystemet

- Positionerna representeras genom ett koordinatsystem
- Varje punkt `<x,y>` i koordinatsystemet representerar en bildpunkt (pixel)
- **OBS!** Koordinatsystemet är upp och ner, dvs. `<0,0>` ligger längst upp till vänster

- Varje komponent har en egen rityta med ett eget koordinatsystem
- Varje rityta har en bredd (width) och en höjd (height)
- Det som ritas utanför ritytan kan inte ses (men man kan/får rita där)

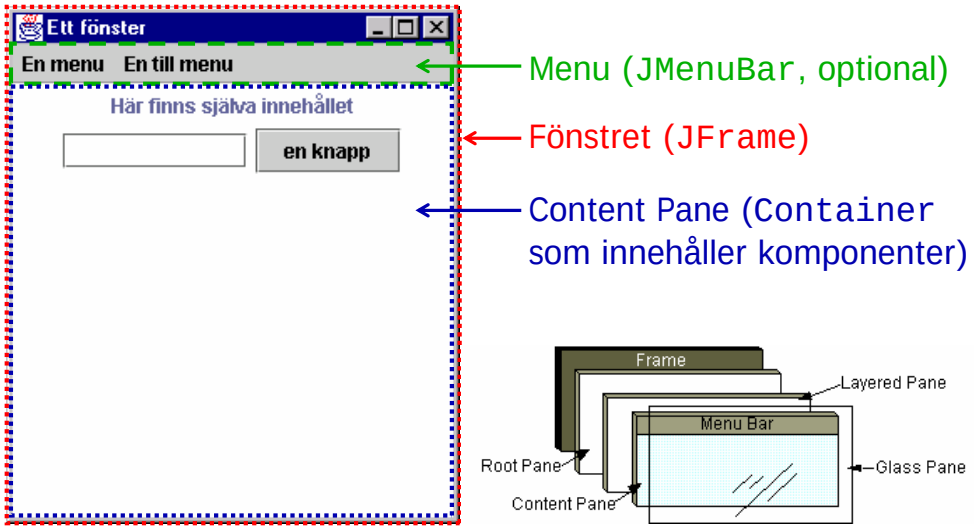
Koordinatsystemet



Ett “Standard” Fönster

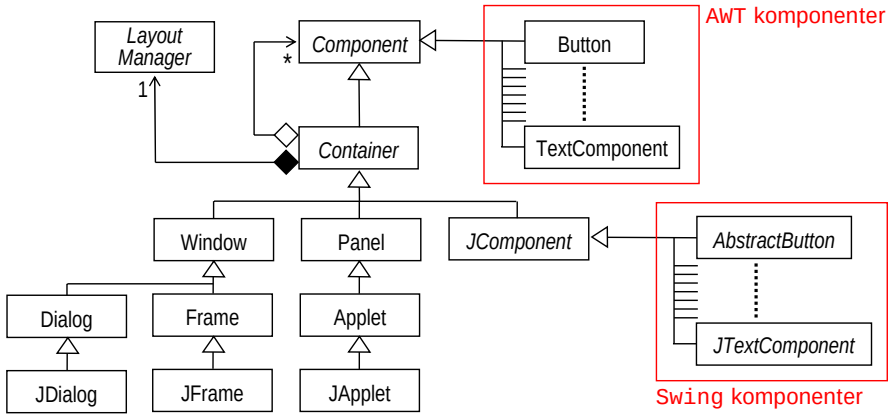
```
import javax.swing.*;

public class SimpleFrame
{
   public static void main (String args[])
   {
      // skapa ett "top-level" fönster
      JFrame f = new JFrame ("Ett fönster");

      // sätta operationen för att stänga fönstret
      f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      f.setSize (300, 200);
      f.setLocation (400, 200);
      f.setVisible (true);
   }
}
```

Fönstrets Anatomi

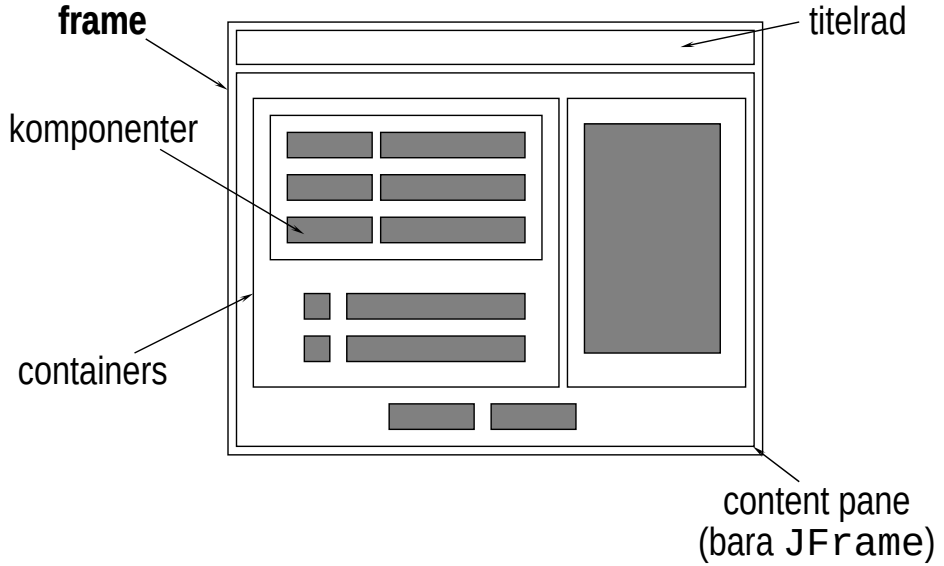


Fönster, Komponenter, Container och Layout

- Ett fönster är en behållare för (GUI) komponenter
- En komponent är också en container



Ett `JFrame` objekt



Container

- En container ***grupperar komponenter***
- Containrar är också komponenter
- I `Swing` är komponenter också container
- Alla komponenter måste placeras i en container (annars kan de inte visas)
- `Applet`, `Frame` eller `Dialog` är exempel på container

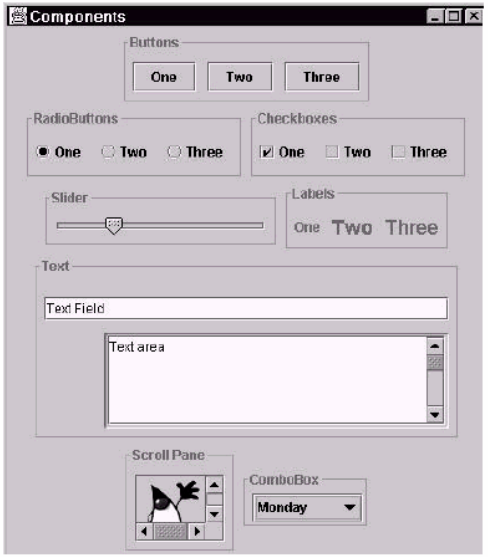
➔ Knappar osv. kan visas i applets, fönster och dialoger

- Varje container har en ***layout manager*** som lägger fast hur komponenterna i containern organiseras
- Det finns 7 standard layout manager

Komponenter

- Komponenter är de GUI element som användare interagera med, som t. ex.
  - Knappar
  - Menuer
  - Listor
  - Inmatningsrutor
  - ...
- Det finns många komponent klasser definierade i AWT och Swing
- Swing komponenterna är lite mera avancerade

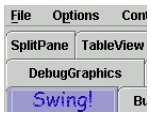
Några AWT komponenter



Några Swing komponenter

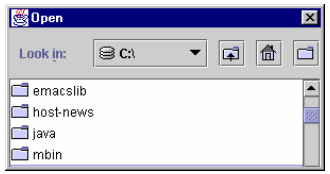
Frame

Dialog



TabbedPane


Toolbar



File Chooser

Tool tip

Color Chooser

Plus Swing versioner av AWT komponenter

Layout Manager

- AWT
  - Flow layout
  - Border layout
  - Card layout
  - Grid layout
  - GridBag layout
- Swing
  - Box layout
  - Overlay layout

Flow Layout 1

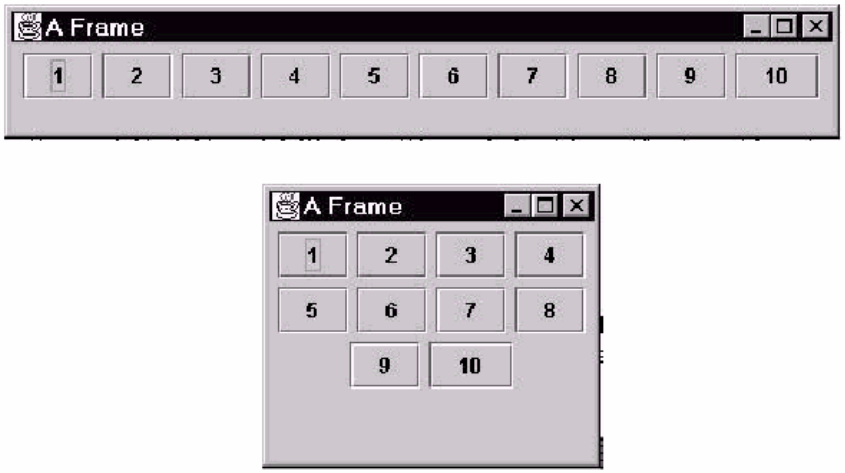
- Komponenter ordnas från vänster till höger i översta raden
- Om raden är full påbörjas en ny rad
- Komponenterna centreras i varje rad (CENTER)
- Programmeraren kan specificera mellanrum (gap) mellan komponenter och rader
- Flow layout är default för `Panel` och `Applet`

```
new FlowLayout (
   alignment, verticalGap, horizontalGap)
```

CENTER, LEFT eller RIGHT

mellanrum i pixel

Flow Layout 2



- Se också `Flow.java` och `JFlow.java`

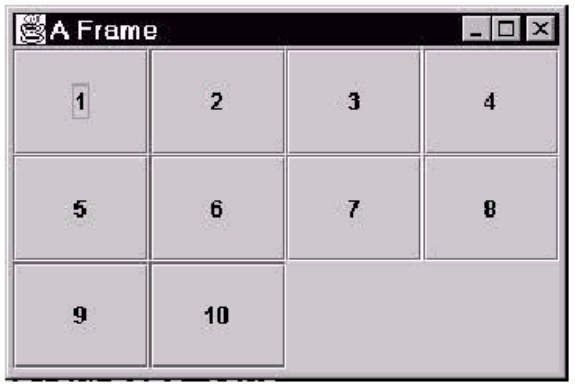
Grid Layout 1

- Komponenter ordnas i rader och kolumner
- Alla celler har samma storlek
- I varje cell ryms precis en komponent
- Cellerna ifylls från vänster till höger och “top-down”

```
new GridLayout (
   rows, columns, verticalGap, horizontalGap)
```

- **OBS!** Antal kolumnerna väljs så att alla komponenter får plats
  - `Rows` är den dominanta parametern (omm != 0)

Grid Layout 2

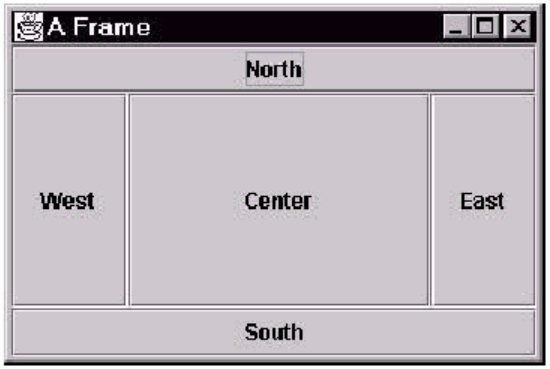


3x4 grid

- Se `Grid.java`

Border Layout

- Storlek av områdena beror på komponenterna som finns där (de får även vara tomma)
- Default för `Frame` och `Window`



- Se `Border.java`

Grid Bag Layout

- Liknar grid layout, men cellerna får ha varierande storlek och egenskaper
- Komponenter kan går över flera rader och/eller kolumner
- Till varje komponent finns en mängd restriktioner som definieras m h a `GridBagConstraints` klassen, t. ex.
  - Hur storleken förändras med containerns storlek
  - När en ny rad påbörjas
  - ...
- Grid bag layout är den mest avancerade av de fördefinierade layout manager
- Se `GridBag.java`

Labels

- En ***label*** (etikett) är en statisk textrad
- En label kan inte selekteras eller modifieras av användaren
- Hanteringen av labels stöds av klassen `Label`

```
new Label (aString, alignment)
```

CENTER, LEFT eller RIGHT

- Se `Mimic.java` och `Fahrenheit.java`

Textfält och Textytor

- Ett ***textfält*** visar en modifierbar textrad
- Textfält är lämpliga för indata hantering
- En ***textyta*** kan visa flera textrader och har scrollbars längst ned och till höger
- Hanteringen av textfält och -ytor sker m.h.a. klasserna `TextField` och `TextArea`

- Se `Fahrenheit.java`

Listor

- En AWT lista används för att visa en lista av strängar
- Antalet strängar i listan är obegränsade

```
new List (rows, multipleSelections)
```

antalet synliga element

om flera element kan vara selekterade samtidigt

- Scrollbars görs automatiskt om inte alla element kan ses
- Kolla `List`-klassen för detaljer

Knappar

- Det finns fyra olika knapptyper:
  - Push buttons (tryckknapp)
  - Choice buttons (urvalsknappar)
  - Checkbox buttons
  - Radio buttons
- En ***tryckknapp*** är en "vanlig" knapp som kan tryckas ned (och sen händer nåt)
- Kolla `Button`-klassen för detaljer
- En ***urvalsknapp*** visar en lista med valmöjligheter när den trycks ned
- Kolla `Choice`-klassen för detaljer
- En ***checkbox knapp*** kann vara från eller till
- Kolla `Checkbox`-klassen för detaljer

Radio Buttons

- Radio buttons är en grupp av checkbox knappar (till/från knappar) där bara en får vara till
- Om en knapp selekteras byter de andra automatiskt till från läget
- Kolla `Checkbox` och `CheckboxGroup` klasserna för detaljer

- Se `Quotes2.java` för ett exempel med alla knapptyper

Scrollbars

- En scrollbar visar en skala med ett aktuellt värde som motsvarar "slädets" relativa position

```
new Scrollbar (
    orientation,   <- HORIZONTAL eller VERTICAL
    value,         <- initialt värde
    visible,       <- storleken på "slädet"
    minimum,       <- minimum värdet
    maximum)       <- maximum värdet
```

- Kolla `Scrollbar`-klassen för detaljer
- Se `Zoom.java`

Paint Metoden

- Central ”call back” metod i `Component` klassen som ritar (om) en komponent
- Anropas av fönstersystemet vid lämpligt tillfälle
- Koden för att rita ut en komponent ska finnas i `paint` (AWT)/ `paintComponent` (Swing)
- I programmet ska `paint`/ `paintComponent` inte anropas direkt
- Omritning begärs genom `repaint` (resulterar så småningom i att `paint`/ `paintComponent` anropas)

OBS! Mekanismen fungerar lite olika i AWT och Swing

Paint--Graphics Context

- Man kan inte direkt rita i/på en GUI komponent
- Varje GUI komponent har en s.k. ***graphics context*** där all ritning äger rum
- Graphics context fås som parameter i `paint` (AWT) metoden eller `paintComponent` (Swing)
- Graphics context är ett objekt av klassen `Graphics`
- `Graphics` hanterar själva ytan där man ritar på
- `Graphics` innehåller ritmetoder och hanterar färg och typsnitt

Paint Exempel

```
public void paint (Graphics page)
{
    page.drawRect (50, 50, 40, 40);      // a square
    page.drawRect (60, 80, 225, 30);     // a rectangle
    page.drawOval (75, 65, 20, 20);      // a circle
    page.drawLine (35, 60, 100, 120);    // a line

    // texts
    page.drawString ("Out of clutter, find simplicity.", 110, 70);
    page.drawString ("-- Albert Einstein", 130, 100);
}
```

- Se `Einstein.java`

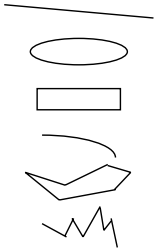
Färg

- `Color` klassen används för färghantering
- Färger definieras genom sina RGB värden, som anger andelen ***r***öd, ***g***rön och ***b***lå i färgen
  - Svart: `0,0,0`
  - Vit: `255,255,255`
- Varje rityta har en förgrunds- och en bakgrunds färg
- `SetColor` metoden i `Graphics` klassen sätter förgrunds färgen och `setBackground` metoden i GUI komponenten sätter bakgrunds färgen
- `Color` klassen tillhandahåller vissa fördefinierade färger som publika konstanter, t.ex. `Color.red`

- Se `Snowman.java`

Att Rita Enkla Figurer

- `Graphics` klassen har ritmetoder för:
  - Linier
  - Ovaler
  - Rektanglar
  - Bågar
  - Polygoner
  - Polylinier

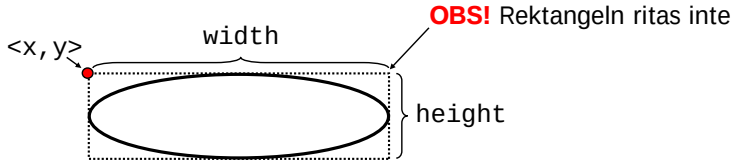


- De flesta figurer kan ritas ifylld (filled), rektangeln t.o.m. i 3D
- Linier är alltid en bildpunkt tjock

```
drawLine (fromX, fromY, toX, toY)
```


Ovaler

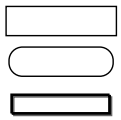
- En oval definieras genom sin omgivande rektangel (***bounding rectangle***):




```
drawOval (x, y, width, height)
fillOval (x, y, width, height)
```


Rektanglar

- Det finns tre typer av rektanglar
  - Vanliga
  - Med rundade hörn
  - Tredimensionella

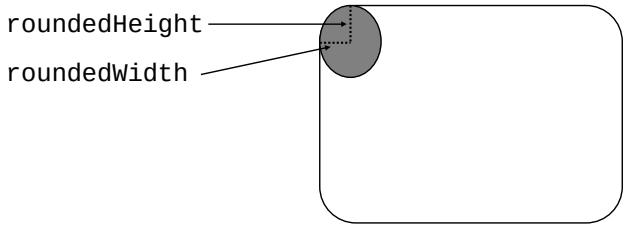


- Varje rektangeltyp kan ritas ifylld eller “tom”

```
drawRect (x, y, width, height)
draw3DRect (x, y, width, height, upOrDown)

fillRect (x, y, width, height)
fill3DRect (x, y, width, height, upOrDown)

clearRect (x, y, width, height)
```


Rundade Rektanglar

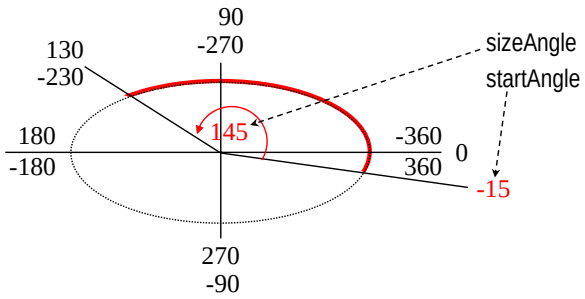
- Det rundade hörnet specifieras m h a en oval




```
drawRoundRect (x, y, width, height,
   roundedWidth, roundedHeight)

fillRoundRect (x, y, width, height,
   roundedWidth, roundedHeight)
```

Bågar

- En båge är en segment av en oval




```
drawArc (x, y, width, height, startAngle, sizeAngle)
fillArc (x, y, width, height, startAngle, sizeAngle)
```


Polygoner

- En polygon är en sluten figur som definieras genom en rad punkter som förbindas med linier
- Punkterna definieras antingen genom
  - Två fält (array) för x- och y-koordinaterna

```
drawPolygon (xPoints, yPoints, polySize)
fillPolygon (xPoints, yPoints, polySize)
```

antal punkter
fält med y-koordinater
fält med x-koordinater

  - M h a `Polygon` klassen

```
drawPolygon (aPolygon)
fillPolygon (aPolygon)
```

en polygon

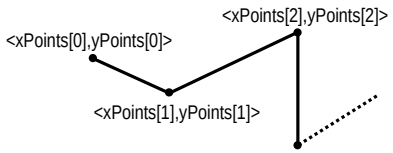
Polylinier

- Liknar polygon men är inte sluten och kan inte ifyllas

```
drawPolyline (xPoints, yPoints, numberOfPoints)
```

fält med x-koordinater — fält med y-koordinater — antal punkter

- Exempel:



Typsnitt

- Typsnittet (font) definierar hur ett tecken ser ut
- `Font` klassen stödjer manipulering av typsnitt
- **OBS!** Det brukar finnas olika typsnitt på olika datorsystem
- I `Graphics` klassen:

```
setFont (new Font (
   fontName, fontStyle, fontStorlek))
```

t.ex. “Monospaced” — t.ex. Font.BOLD — i antal punkter, t.ex. 12

- Se `Fonts.java`

Animering

- Enkla animeringar kan göras m.h.a. XOR mode
- XOR mode byter emellan kontrasterande färger
- ➔ Om man ritar samma objekt två gånger så försvinner det
- Med `translate` kan koordinatrummet förflyttas (i `Graphics`, `Point`, `Polygon`, `Rectangle`)
- **OBS!** Hastigheten kan bli ett problem
- Se `Rotating_Disk.java`

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Att organisera Javakod
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

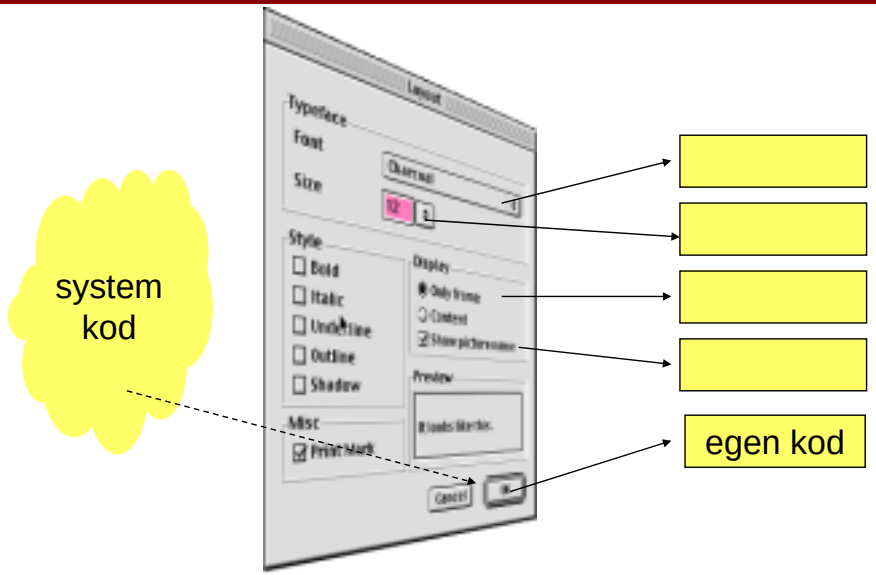
GUI Program

- I traditionella program
  - Bestämmer programmet kontrollflödet
  - Kontrollerar programtexten hur och när input görs
  - ➔ Anropar användarprogrammet systemkod
- GUI program är ***händelsebaserat***
- I händelsebaserade program
  - Bestämmer användaren hur och när input görs
  - Styr inputkontrollerna kontrollflödet
  - ➔ Anropas användarens kod av systemet

Händelsebaserat Programmering 1

- Händelsehanteringen kan inte programmeras sekventiellt
- Interaktionen i ett GUI är inte uppstyrd
- Användaren kan interagera med valfri komponent i valfri ordning
- Ordningen på händelserna kan inte förutses
- Se `DistanceGUIFrame.java` [~Koffman/Wolz 7.4] och `Quotes2.java`

Händelsebaserat Programmering 2



Händelsehantering 1

- Olika komponenter genererar olika händelser
- För varje händelse genereras ett `Event` objekt som skickas till hanteraren för denna sortens händelser

Användaraktion	Källobjekt	`Event` typ	Hanterare
Knapptryckning	`(J)Button`	`ActionEvent`	`actionPerformed`
Slå return i ett textfält	`(J)TextField`	`ActionEvent`	`actionPerformed`
Välj rad i menu	`(J)MenuItem`	`ActionEvent`	`actionPerformed`
Flytta scroll bar	`(J)ScrollBar`	`AdjustmentEvent`	`adjustmentValueChanged`
Flytta musen, trycka musknapp	`Component`	`MouseEvent`	`mouseClicked`, `mousePressed`, `mouseReleased`, `mouseEntered`, `mouseExited`
...	...	...	...

Händelsehantering 2

- För att ta hand om en händelse måste rätt hanterare implementeras

OBS! Programmet behöver inte ta hand om alla händelser

- Se `DistanceGUIFrame.java` igen

➔ Hur/ vart implementeras händelsehanterare?

➔ Hur kopplas komponenter och händelsehanterare ihop?

Lyssnare 1

- Hanterarnas signatur specificeras i ett `Listener` ***interface***
- Komponenterna måste registreras med en matchande `Listener` som implementerar rätt interface

`Event` klass	`Listener` interface	Hanterare
`ActionEvent`	`ActionListener`	`actionPerformed (ActioEvent e)`
`ItemEvent`	`ItemListener`	`itemStateChanged (ItemEvent e)`

`WindowEvent`	`WindowListener`	`windowClosed (WindowEvent e)`
		`windowOpened (WindowEvent e)`
		`windowIconified (WindowEvent e)`
		`windowDeiconified (WindowEvent e)`
		`windowActivated (WindowEvent e)`
		`windowDeactivated (WindowEvent e)`
		`windowClosing (WindowEvent e)`

...	...	...

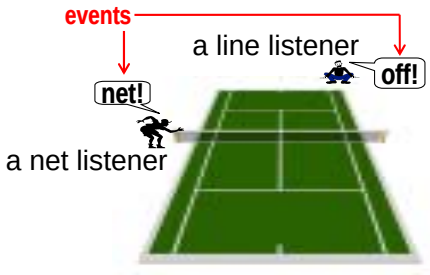
Lyssnare 2

- Programmet kan alltså ”prenumerara” på händelser genom lyssnare, t.ex.

```
enKnapp.addActionListener (enActionListener)
```

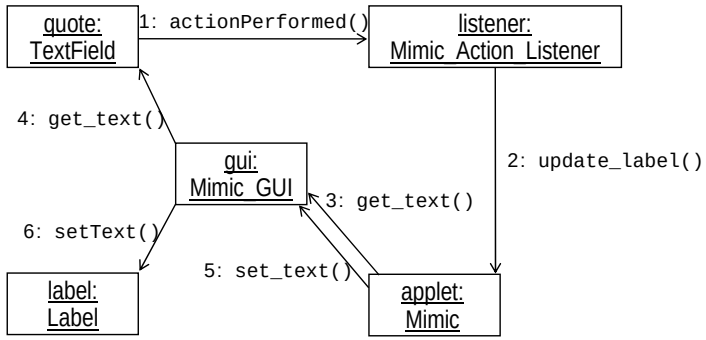
ta emot `ActionEvents` från objektet

- När en händelse inträffar skickas den vidare till de registrerade lyssnarna som ta hand om händelsen



Lyssnare 3

- Exempel 1: `Mimic.java`
- Så här går det till:



- Exempel 2: `DistanceGUIFrame.java`

GUI Programöversikt

- “Huvudprogrammet”
  - Implementerar aktioner till alla händelser
- GUIet
  - Instantierar och sätter ihop GUI komponenterna
  - Registrerar lyssnarna
- Lyssnare
  - Implementerar ett eller flera interface
  - Anropar rätt aktion i “huvudprogrammet” när händelse sker
- Kolla `Mimic.java` igen

Att skilja åt händelsekällor

- `GetSource` metoden
  - ➔Eventhanteraren och händelsekälla måste ha samma scope
- `SetActionCommand/ getActionCommand` metoderna
  - ➔Fungerar bara för `ActionEvent`
- Privata/ anonyma klasser för lyssnarna
  - ➔Mest generella ansatsen
  - ➔Lite mera invecklat

AWT.EVENT