Litt om AWT og hendelser

I. NOEN KOMPONENTER FOR GUI

Container -> Window -> Frame
kontroll-komponenter: knapper, textfelter, ...

II. HENDELSE-DELEGASJONSMODELL

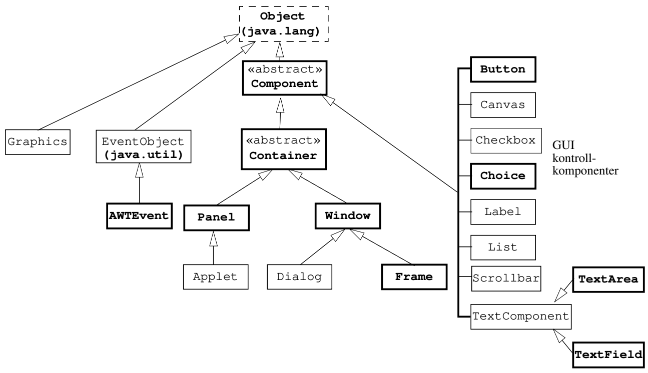
hva er en hendelse
kilder: generering av hendelser i Java
lyttere: håndtering av hendelser i Java

III. NOEN LYTTERE-GRENSESNITT

IV. LITT OM HENDELSESATTRIBUTTER

V. PROGRAMMERING

Objekter i **`java.awt.*`**



`Component`

- Alle ikke-meny-relaterte elementer som utgjør et grafisk brukergrensesnitt er utledet fra den abstrakte klassen `Component`.
- `Component`-klassen tilbyr støtte for håndtering av hendelser, endring av komponent-størrelse, kontroll av fonter og farger, og tegning av komponenter og deres innhold.
- Komponent gjøres synlig og usynlig på skjermen ved å bruke
  – `setVisible(boolean)` metoden.
- Komponent-størrelse settes ved å kalle :
  – `setSize(int,int)` metode, evt.
  – `setSize(Dimension)` metode

`Container`

```
java.lang.Object
   |
   +----java.awt.Component (abstract)
           |
           +----java.awt.Container (abstract)
```

- Den abstrakte `Container`-klassen definerer metoder for nøsting av andre `Component`-objekter i et `Container`-objekt, mao. *en kontainer er en komponent som kan inneholde andre komponenter* (og dermed andre kontainere siden en kontainer er en komponent pga arv).
  – En slik oppbygging definerer et *komponent-hierarki* av komponenter som inneholder hverandre (i motsetning til *arv-hierarkiet*).
  – `add(Component comp)` brukes av konkrete subklasser til `Container`-klassen for å tilføye en komponent til en kontainer.
- En kontainer bruker en *layout-manager* til å posisjonere komponenter i den:
  – `setLayout(LayoutManager mgr)`

Panel

```
java.lang.Object
   |
   +----java.awt.Component (abstract)
           |
           +----java.awt.Container (abstract)
                   |
                   +----java.awt.Panel
```

- `Panel`-klassen er en konkret implementasjon av `Container`-klassen.
- Et panel er en rekursiv, nøstet komponent: et vindu som *ikke* har tittel, menyer eller kanter, men inneholder andre komponenter.
  - Komponenter kan tilføyes til et `Panel`-objekt vha. de arvede `add()`-metodene i følge en layout-manager.
  - Panel bruker `FlowLayout`-manager som standard layout manager.
  - `setLayout(LayoutManager mgr)` kan brukes for å endre på det

Window

```
java.lang.Object
   |
   +----java.awt.Component (abstract)
           |
           +----java.awt.Container (abstract)
                   |
                   +----java.awt.Window
```

- `Window`-klassen oppretter et *topp-nivå* vindu som er *uten* tittel, menyer, eller kanter.
  - Et topp-nivå vindu kan ikke inkorporeres/nøstes i andre komponenter.
  - Et topp-nivå vindu kan gjøres synlig ved å kalle den overkjørte metoden `show()`, og usynlig ved å kalle den arvede metoden `setVisible(false)`.
- brukes sjelden ....

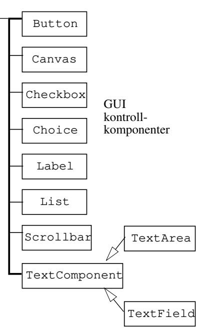
Frame

```
java.lang.Object
   |
   +----java.awt.Component (abstract)
           |
           +----java.awt.Container (abstract)
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
```

- `Frame`-klassen brukes til å opprette det som vanligvis menes med et "vindu" som har *tittel, menyer, kant, markør og et ikon*.
- Et `Frame`-objekt danner utgangspunktet for *en ramme-basert applikasjon*, og er vanligvis det som danner roten til et *komponent-hierarki*.
  - et `Frame`-objekt kan inneholde flere paneler som i sin tur kan inneholde GUI *kontroll-komponenter* og andre paneler.
  - default er `BorderLayout` (med "Center", "South", "North", "West", "East")
  - `setTitle("Hi frame")`
  - `setMenuBar(MenuBar mb)`
    `setSize(int, int)`
    `setLayout(LayoutManager mgr)`
    `add("North", Component)`
    `setVisible(boolean vis),    dispose()`

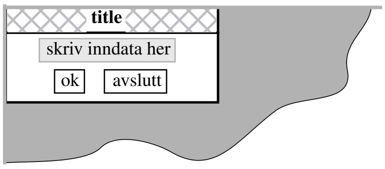
GUI kontroll-komponenter

- GUI kontroll-komponenter gjør interaksjon mellom brukeren og programmet mulig.
- *Inn-komponenter*: mottar informasjon fra brukeren, f.eks. `Button`, `TextField`, `Checkbox`.
- *Ut-komponenter*: presenterer informasjon for brukeren, f.eks. `TextField`.
- En layout-manager brukes til å posisjonere komponenter i en kontainer, f.eks. i et `Panel`-objekt.
- Komponenter tilføyes vanligvis til en kontainer vha. `add()`-metoden.
- "Look and feel" til brukergrensesnitt bestemt av plattform applikasjon kjøres på.



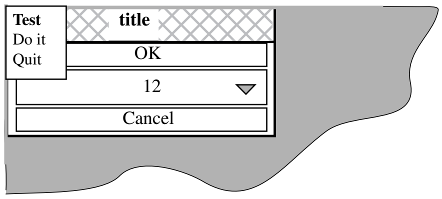
Et enkelt vindu

```
import java.awt.*;
class ioFrame extends Frame {
   protected Button Quit, OK;
   protected TextField txt;

   public ioFrame(String title, int x, int y) {
     setTitle(title);
     setSize(x,y);
     setLayout(new FlowLayout());
     txt = new TextField(“skriv inndata her”);
     txt.setEditable(true);
     add(txt);
     Quit = new Button(“avslutt”);
     OK = new Button(“ok”);
     add(OK);  add(Quit);
     setVisible(true);  }

   String les() {  return txt.getText(); }

   void skriv(String s) { txt.setText(s); }

   int lesInt() throws NumberFormatException {
     return Integer.parseInt(txt.getText()); }

   void skriv(int i) {
     txt.setText(Integer.toString(i)); }
}
```



Et annet enkelt vindu

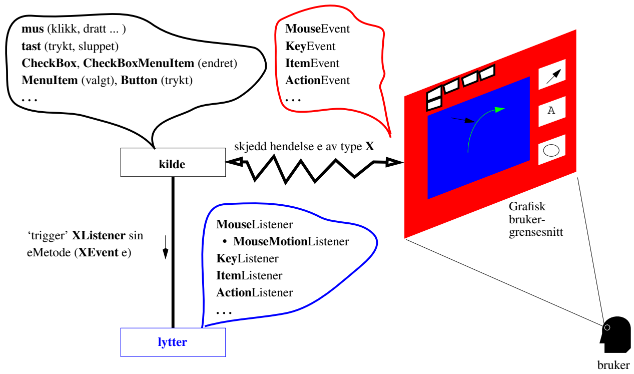
```
import java.awt.*;
public class mFrame extends Frame { ....
Button ok= new Button(“OK”);
Button ca= new Button(“Cancel”);
Menu m= new Menu(“Test”);
MenuBar bar= new MenuBar();
Choice c= new Choice();

public mFrame(String title, int x, int y) { ...
   MenuItem mi= new MenuItem(“Do it”);
   m.add(mi);
   mi= new MenuItem(“Quit”);
   m.add(mi);
   bar.add(m);
   c.addItem(“12”);
   c.addItem(“14”);
// alt dette må plasseres i vinduet ...
   setMenuBar(bar);
   add(“North”, ok); add(“South”, ca);
   add(“Center”,c);
   setVisible(true);
}
```



Informasjon inn og ut fra GUI kontroll-komponenter

- `TextArea`, `TextField`:
  - `t =` new `TextField(“Hei”,10);` new `TextArea(“Hei”,5,10);`.
  - bruk `t.getText()` for å hente et `String`-objekt.
  - bruk `t.setText(String str)` til å skrive et `String`-objekt.
  - bruk `t.setEditable(boolean)` for å tillate/forby editering av teksten.
  - Tekst må konverteres til og fra riktig data type.
- `Checkbox`:
  - `cb =` new `CheckBox(“tick me”)`.
  - bruk `cb.getState()` for å finne om den er valgt eller ikke.
  - bruk `cb.setState(boolean flagg)` til å sette tilstand til `Checkbox`-objekter lik verdien gitt i `flagg`-parameteren.
- `Choice`:
  - `ch =` new `Choice()`.
  - bruk `ch.addItem(“valg1”)` til å lage en liste av alternativer
  - bruk `ch.getSelectedItem()` for å hente navnet på valgt alternativ
- `Button`:
  - new `Button(“label”)`
- `Menu`:
  - new `Menu(“navn”)`

Hendelse-delegasjonsmodell



En kilde kan generere forskjellige hendelser og ha flere lyttere.
Samme lytter kan lytte til forskjellige hendelser fra forskjellige kilder.

Kilder og Lyttere

- En *kilde* (source) er en komponent (Component) som kan generere hendelser.
- En *lytter* (listener) er et objekt som er interessert å bli informert om hendelser når disse skjer.
- En kilde informerer lyttere om hendelser når de skjer, og sender nødvendig informasjon om hendelser til lyttere.
- En lytter, som er interessert i å motta informasjon om bestemte hendelser, må *på forhånd registrere seg* med kilde(r) som kan generere disse hendelsene.
- En kilde kaller en *bestemt metode* i alle lyttere for en gitt hendelse, og lyttere garanterer at denne metoden eksisterer ved å implementere et *lytter-grensesnitt* (Listener interface) for denne hendelsen.
- Hvilket som helst objekt kan være en lytter så lenge det implementerer riktig lytter-grensesnitt (`XListener`) for spesifikke hendelser (`XEvent`), og *registrerer seg* (`addXListener()`) *hos en kilde* som genererer disse hendelsene.
- Subklasser til en komponent kan generere samme hendelser som denne komponenten pga. arv.

Noen lytter-interface

```
public interface MouseListener {
    void mouseClicked(MouseEvent evt) ;
    void mouseReleased(MouseEvent evt) ;
    void mousePressed(MouseEvent evt) ;
    void mouseEntered(MouseEvent evt) ;
    void mouseExited(MouseEvent evt) ;
}
```

MouseEvent e

```
public interface MouseMotionListener {
    void mouseMoved(MouseEvent evt) ;
    void mouseDragged(MouseEvent evt) ;
}
```

```
public interface KeyListener {
    void keyPressed(KeyEvent evt) ;
    void keyReleased(KeyEvent evt) ;
    void keyTyped(keyEvent evt) ;
}
```

KeyEvent e

```
public interface ActionListener {
    void actionPerformed(ActionEvent evt) ;
}
```

ActionEvent e

```
public interface ItemListener {
    void itemStateChanged(ItemEvent evt) ;
}
```

ItemEvent e

Hendelsesatributter

Event **e**	fra	når	metode **e. ...**	returnerer
EventObject	–	–	Object **getSource**()	kilde til hendelsen
AWTEvent	–	–		
ActionEvent	Button MenuItem List TextField	klikk valg dobbel klikk [ENTER]	String **getActionCommand**()	knapp-tekst meny-tekst valgt-tekst tekst i feltet
ItemEvent	CheckBox / Choice ...	seleksjon / deseleksjon	Object **getItem**() **...**	valgt 'Item'
InputEvent	– Component	–	int **getModifiers**() boolean **isControlDown**() boolean **isShiftDown**() **...**	!= 0 hvis Crl/Shft/Esc/Alt
MouseEvent	mus- knapp mus- bevegelse	klikk trykt sluppet gikk inn/ut flyttet/dratt	int **getX**() int **getY**() Point **getPoint**() int **getClickCount**() **...**	x / y koordinat / punktet (relativt til kilde) der hendelse skjedde # klikk
KeyEvent	en tast	trykt sluppet tastet	char **getKeyChar**() int **getKeyCode**() **...**	tegnet for tasten tast-kode for tasten

Programmering av en lytter

```
import java.awt.*; import java.awt.event.*;
public class ioFrame  extends Frame  implements ActionListener , MouseListener
    , ItemListener   { ....
Button ok= new Button("OK");  Button ca= new Button("Cancel");
Menu m= new Menu("Test"); MenuBar bar= new MenuBar();
Choice c= new Choice()

public ioFrame(String title, int x, int y) { ...
    MenuItem mi= new MenuItem("Do it");    m.add(mi);
    mi= new MenuItem("Quit");  m.add(mi);
    bar.add(m);                         mi . addActionListener((ActionListener)this);
    c.addItem("12"); c.addItem("14");
// alt dette må plasseres i et vindu ...   ok/ca . addActionListener((ActionListener)this);
    setMenuBar(bar);
    add("North", ok); add("South", ca);
    add("Center",c);          c.addItemListener((ItemListener)this);
    setVisible(true);         addMouseListener((MouseListener)this);
}
```

```
public void actionPerformed(ActionEvent e) {
    Object s = e.getSource();
    if (s == ok) ...
    else if (s == ca) ...
    else if (s instanceof MenuItem) {
        MenuItem mi = (MenuItem)s;
        String arg = mi.getLabel();
        if (arg.equals("Do it")) ...
        else if (arg.equals("Quit")) { setVisible(false); dispose(); System.exit(0); } }  }
```

```
public void itemStatesChanged(ItemEvent e) {
    if (e.getSource() instanceof Choice) {
        String s = c.getSelectedItem();
        if (s.equals("12")) ... }
... }
```

```
public void mouseClicked(MouseEvent e) {
    int c= e.getClickCount(); int x= e.getX(); int y= e.getY(); .... }
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
```