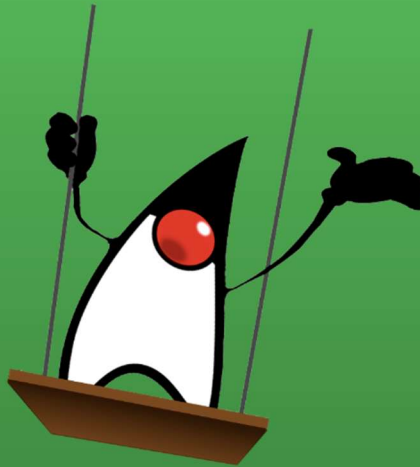


Grafiska användargränssnitt i Java

En genomgång av de viktigaste begreppen

Från början fanns AWT,
Abstract Window Toolkit

Stora delar har ersatts av Swing:
Mer omfattande, mer flexibelt



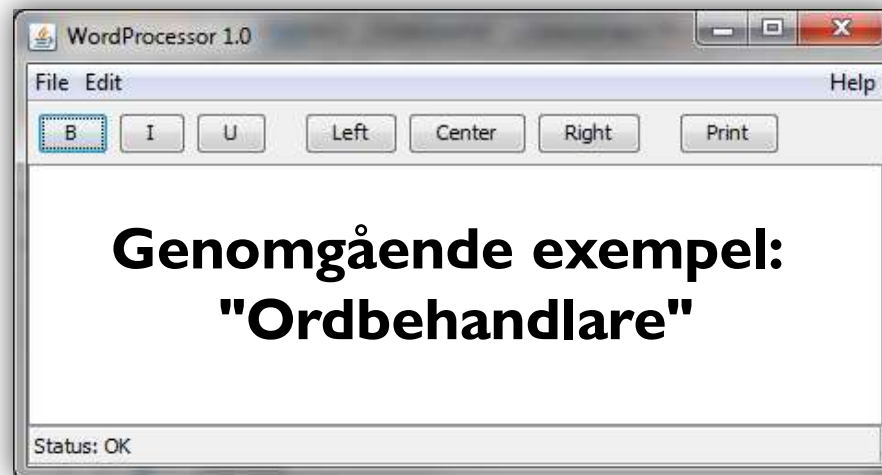
Passar bra för våra
behov

Använder begrepp
som är vanliga
i GUI-programmering

Ett alternativ: JavaFX
Helt annan struktur (scenes, stages, nodes)
Använd på eget ansvar – ingen hjälp

Del 1:

**Att använda Swing: Fönster, knappar, menyer, ...
Layout (placering)
Händelsehantering – ”någon tryckte en knapp”**

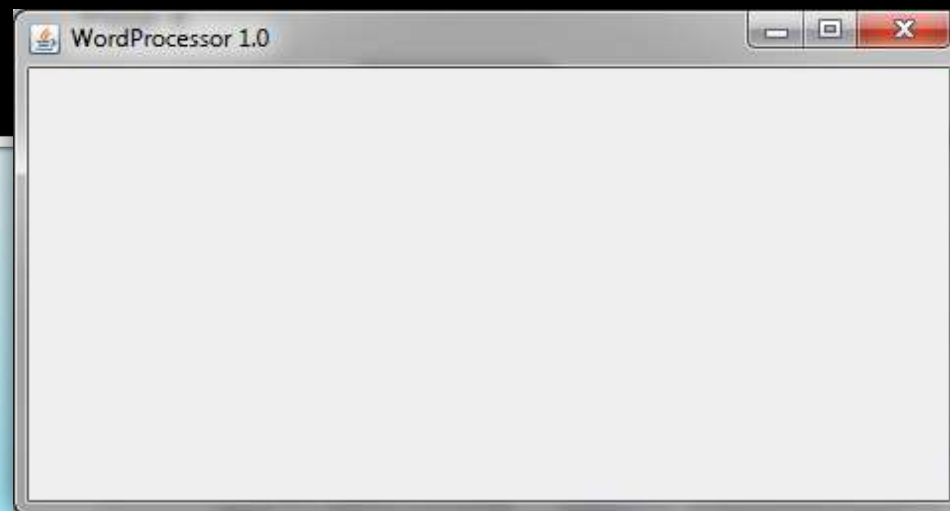


Del 2:

Att *rita* på skärmen, i en egen komponent

Steg 1

Öppna ett tomt fönster – en behållare (container)



■ Fönster: JFrame

```
public class WordProcessor01
{
    public WordProcessor01() {}

    public void show() {
        JFrame frame = new JFrame("Word Processor 1.0");

        frame.setSize(640, 300);
        // ... lägg till komponenter i fönstret ...
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        final WordProcessor01 wp = new WordProcessor01();
        wp.show();
    }
}
```

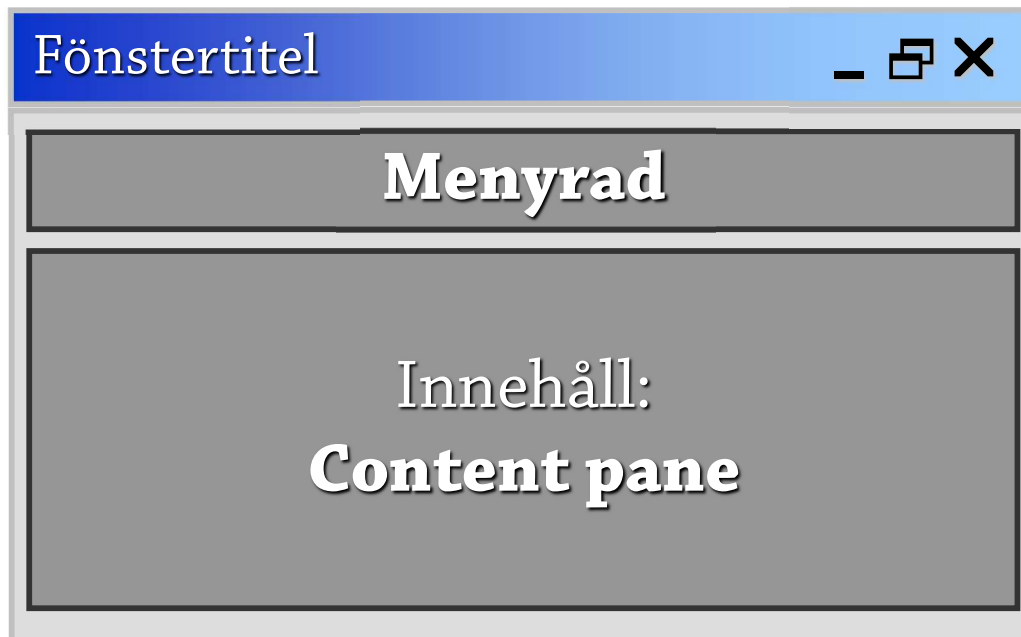
2. Metoden skapar ett fönster, som själv registrerar sig i **GUI-systemet**

3. Sätt fönsterstorleken, se till att fönstret **visas** på skärmen

1. Huvudmetoden skapar ett objekt... och ber det skapa ett grafiskt gränssnitt

- En **JFrame** är:
 - En **komponent**
 - Något grafiskt som visas på skärmen
 - En **behållare** (container)
 - Något som kan innehålla *andra* komponenter

<http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html>



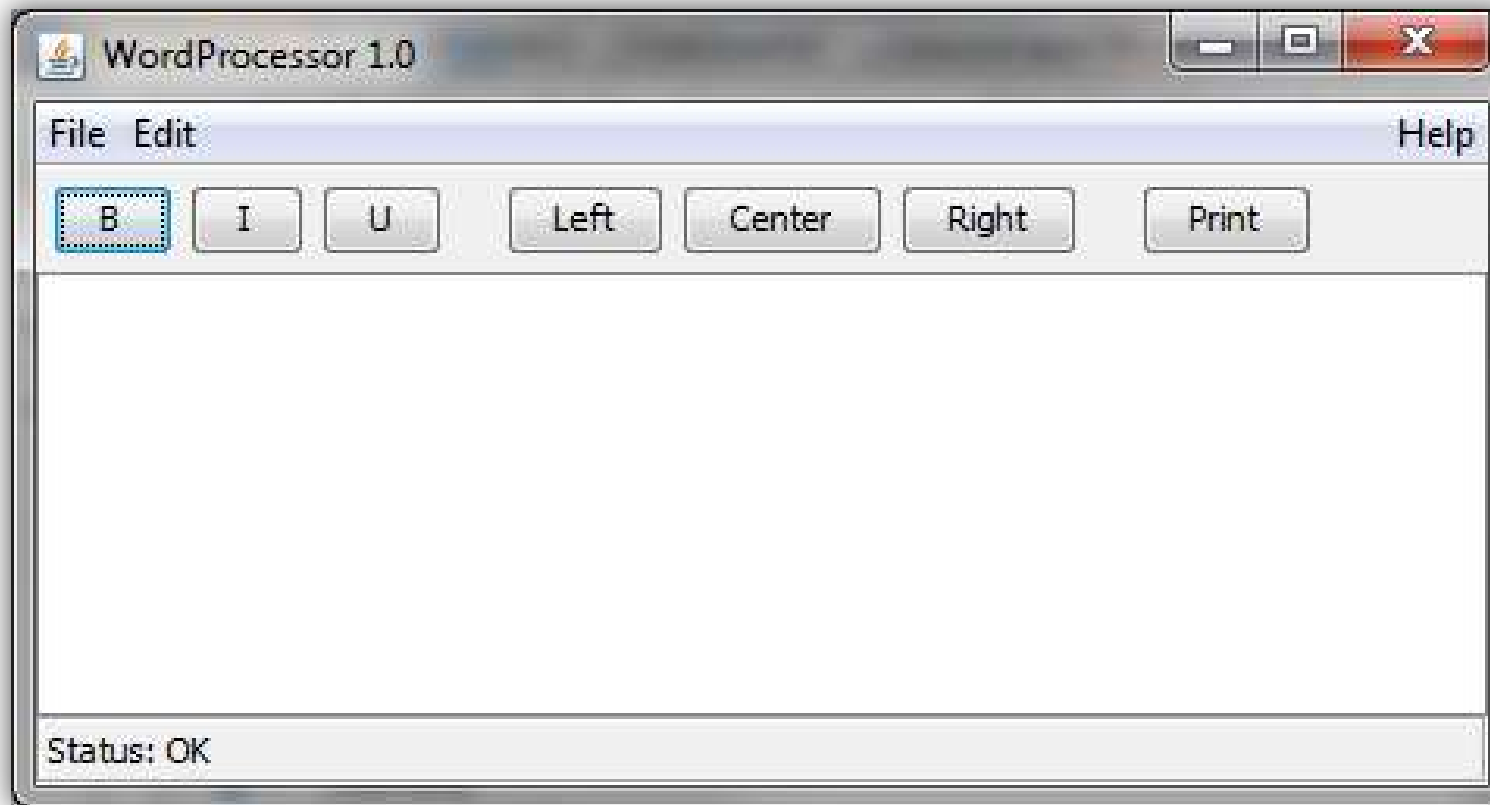
Kan läggas till vid behov

Hela fönstrets innehåll.
utom titelrad / menyrad

Steg 2

Lägg till komponenter i fönstret

- Istället för att "rita": Säg till *vilka komponenter* som ska finnas
 - Meny, knappar, textfält, statusrad
 - *Swing* sköter om utseende (med mera)



Alla komponenter:

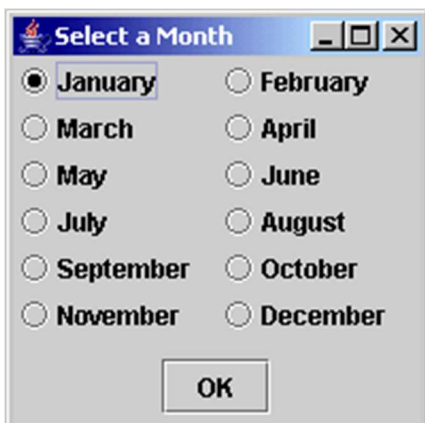
<https://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html>

Knappar i Java

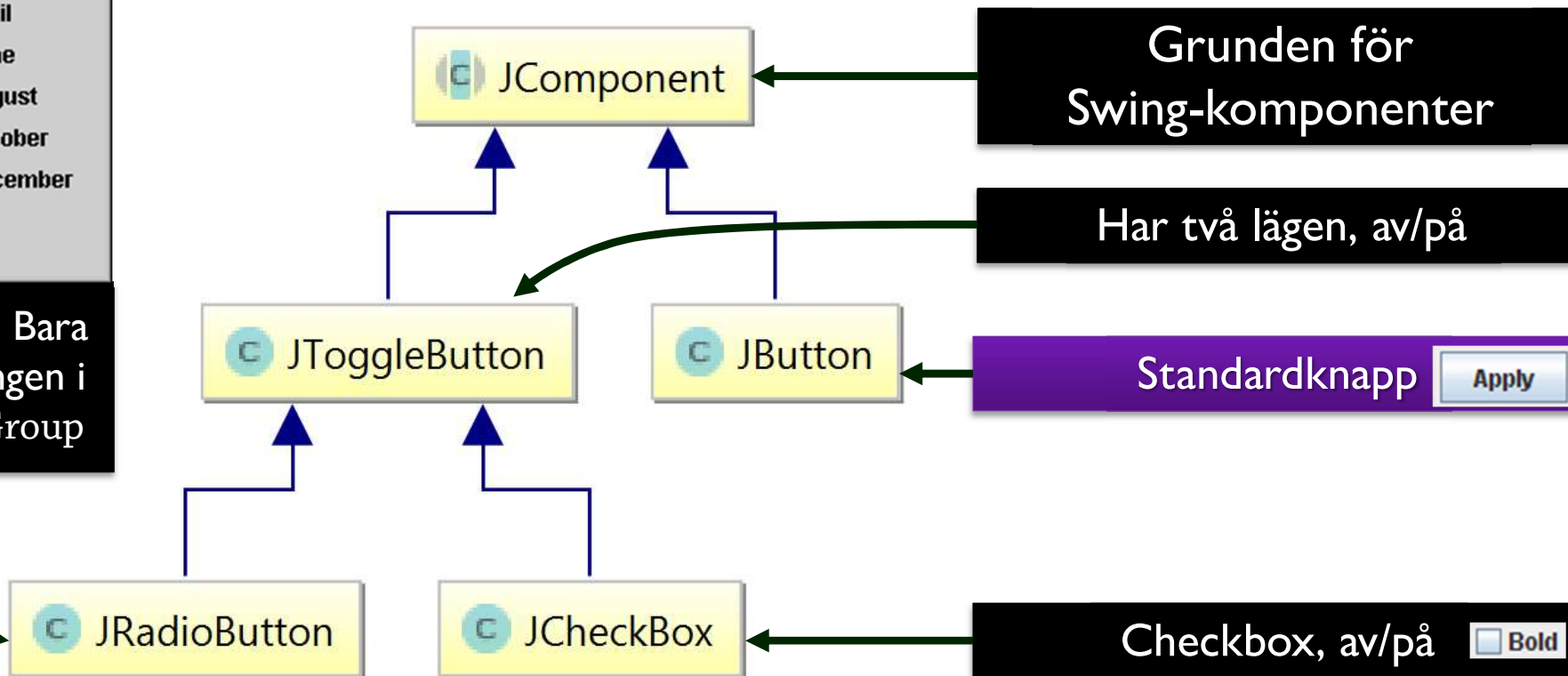


<http://docs.oracle.com/javase/tutorial/uiswing/components/button.html>

<http://docs.oracle.com/javase/tutorial/uiswing/components/buttongroup.html>

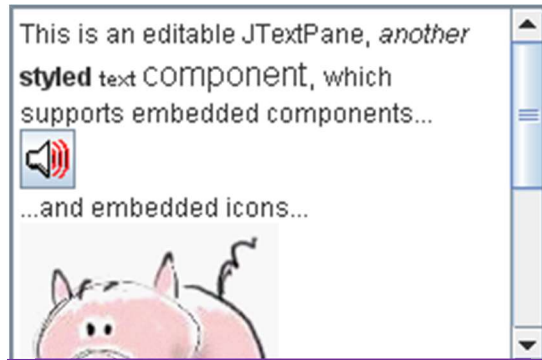


Radioknappar: Bara en aktiv åt gången i varje ButtonGroup



Textkomponenter

<http://docs.oracle.com/javase/tutorial/uiswing/components/text.html>



Editera text med flera stilar: HTML, ...


 Scrollable

 JComponent

 JTextComponent

Abstrakt, används indirekt för ärvning av gemensam funktionalitet

 JEditorPane


 JTextArea


Textarea med flera rader

 JTextField


En enda rad text


 JTextPane

 JFormattedTextField

Speciell formatering för datum, valuta, ...


 JPasswordField

Lösenord syns inte när de skrivs in


Ordbehandlare med komponenter



<http://docs.oracle.com/javase/tutorial/uiswing/components/label.html>

```
public class WordProcessor02 {
    public void show() {
        JFrame frame = new JFrame("Word Processor 1.0");

        frame.add(new JButton("B"));
        frame.add(new JButton("I"));
        frame.add(new JButton("U"));
        frame.add(new JButton("Left"));
        frame.add(new JButton("Center"));
        frame.add(new JButton("Right"));
        frame.add(new JTextPane());
        frame.add(new JLabel("Status: OK")); // JLabel visar text, kan inte editeras
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        final WordProcessor02 wp = new WordProcessor02();
        wp.show();
    }
}
```

Lägger till flera komponenter
till behållaren – fönstret

Inget syns på skärmen än!

NU är det dags att visa
fönstret och dess innehåll

Hålla reda på komponenter

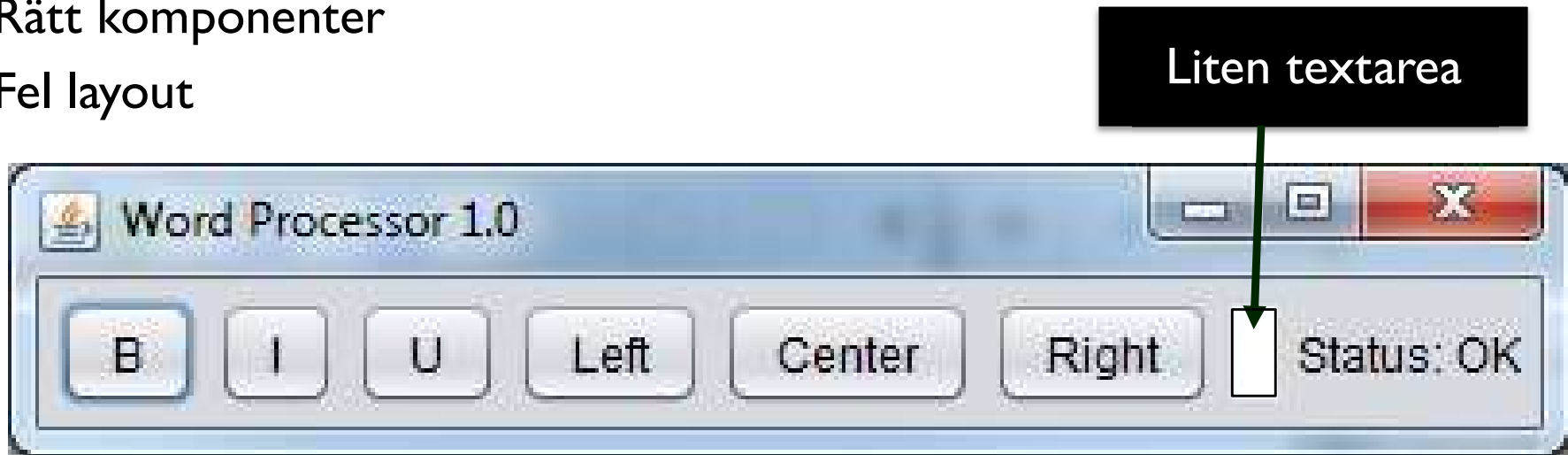


<http://docs.oracle.com/javase/tutorial/uiswing/components/label.html>

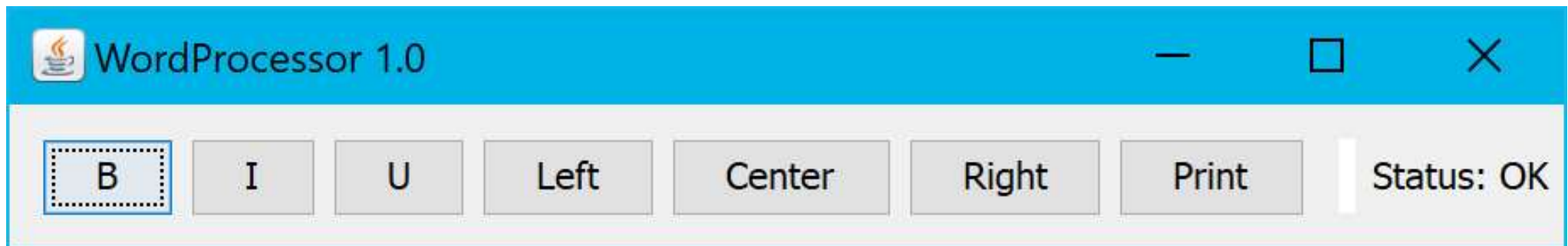
```
public class WordProcessor02b {  
    private JFrame frame;  
    private JTextPane text;  
    private JLabel status;  
    public void show() {  
        this.frame = new JFrame("Word Processor 1.0");  
        frame.setLayout(new FlowLayout());  
        frame.add(new JButton("B"));  
        frame.add(new JButton("I"));  
        frame.add(new JButton("U"));  
        frame.add(new JButton("Left"));  
        frame.add(new JButton("Center"));  
        frame.add(new JButton("Right"));  
        this.text = new JTextPane();  
        frame.add(text);  
        this.status = new JLabel("Status: OK");  
        frame.add(status);  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

Behåll pekare
till det du vill använda senare
(plocka ut texten
för att spara den, ...)

- Resultat:
 - Rätt komponenter
 - Fel layout

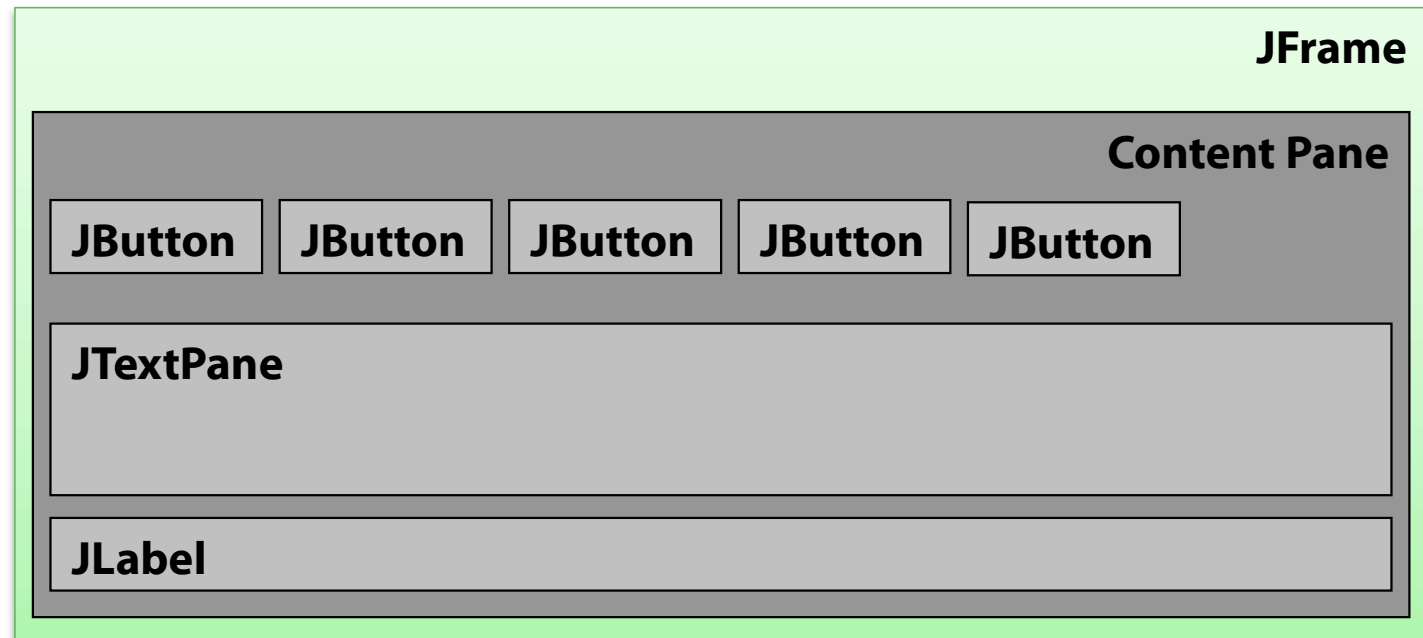


- (Java har *pluggable look and feel*, möjlighet till olika utseenden...)



Step 3 Layout

- Hur får vi önskad layout (positioner och storlek)?



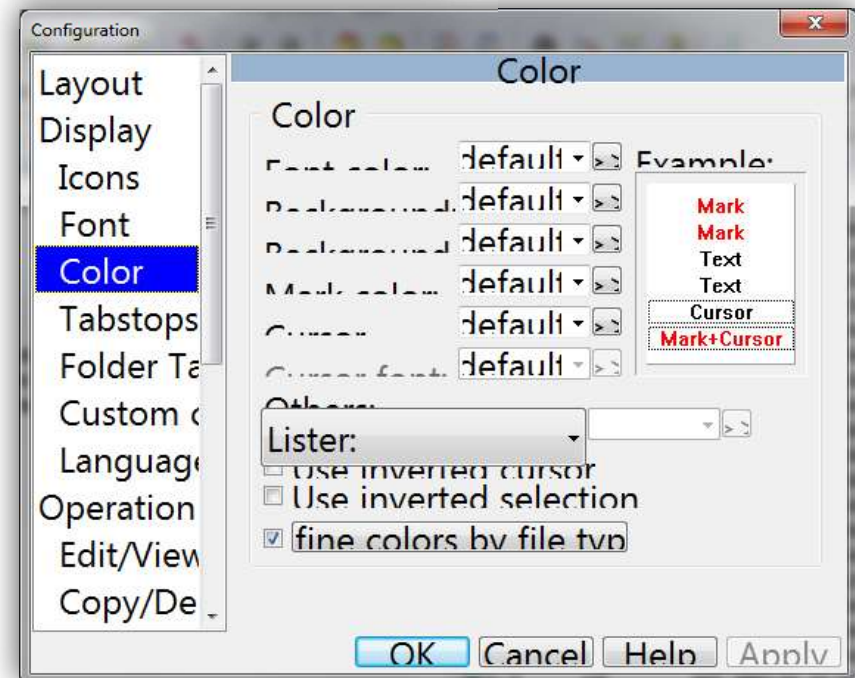
Layout 2: Absoluta koordinater?

■ Absoluta koordinater?

- `component.setSize(int width, int height);`
- `component.setLocation(int x, int y);` // Från behållarens övre vänstra hörn

■ Fördelar / nackdelar:

- (+) Enkelt att förstå
- (-) Inget stöd för att *ändra fönsterstorlek*
- (-) Hur hanterar man *större text?*
- (-) *Översättningar* till andra språk, där ord kan ta mer plats?
 - Tools / Инструменти
 - URL / nettsadress



Layout 3: Layouthanterare

- Swing: Använd layouthanterare med olika layout-regler!
 - Om ingen annan hanterare är angiven:
 - Layout **vänster till höger**

<http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html>



Ingen text
inskriven →
preferred size
är minimal!

Komponenter kan tala om sin önskade storlek (*preferred size*)

JButton: Storlek på texten i inställd fontstorlek + lite marginaler
Returneras av **getPreferredSize()**
Respekteras om det finns *tillräckligt med utrymme*

Layout 4: Fönsterstorlek

- Så varje **komponent** vet önskad storlek...



- ...men hur fick **hela fönstret** precis "lagom" storlek?

frame.pack() → anropar **layouthanterare** (*layout manager*)

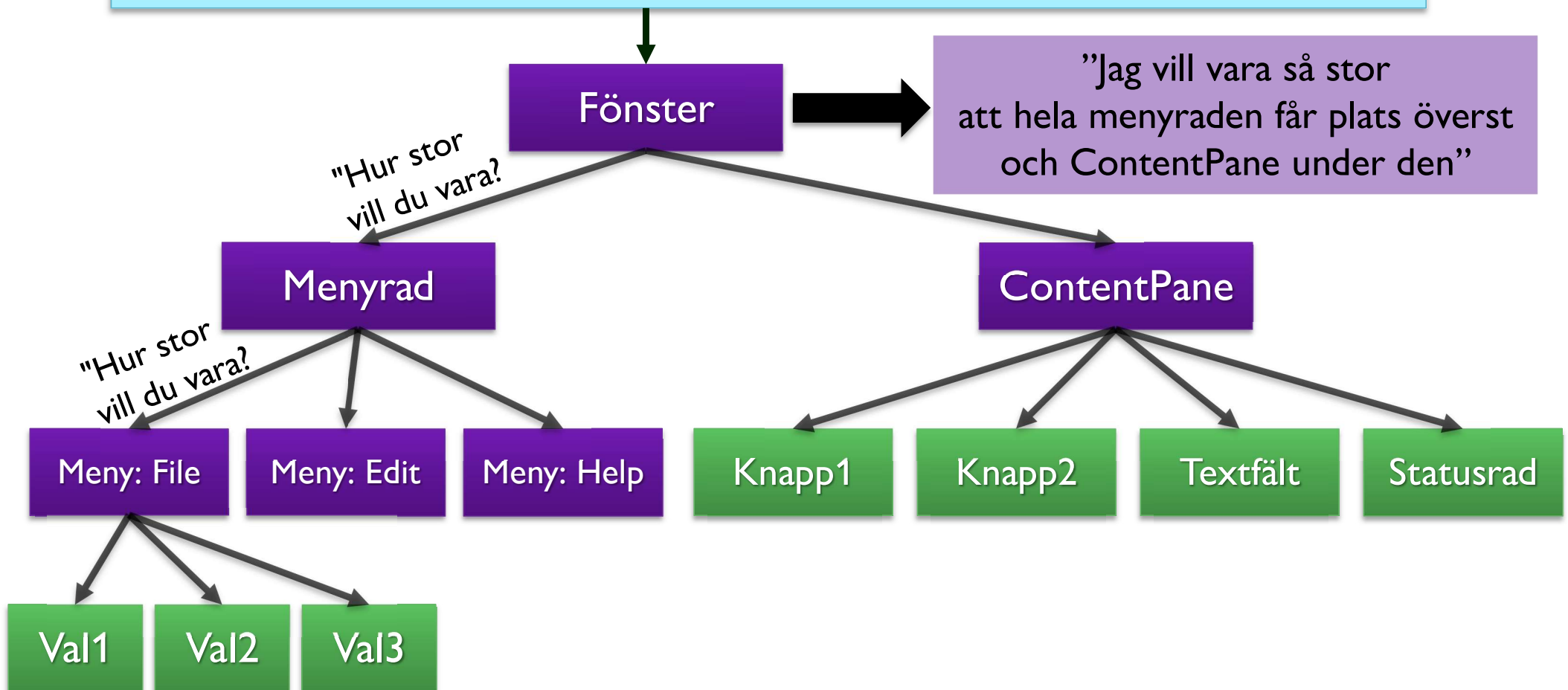
Layout: Frågar subkomponenter hur stora de vill vara
Beräknar **behållarens** önskade storlek
enligt sina **layoutregler** (här: "Allt i en rad")

frame.pack() → **frame.setSize(önskad storlek)**

Layout 5: Hierarkisk pack()

Fönster har en *hierarki* av komponenter

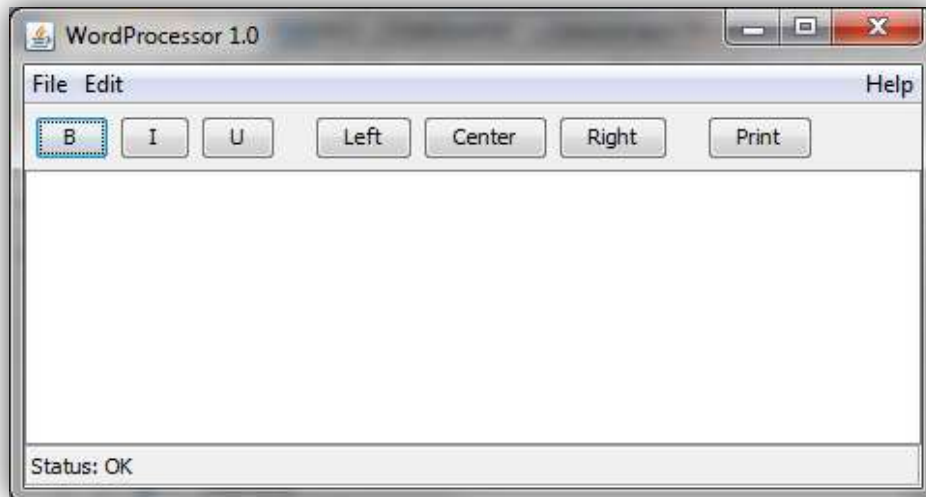
pack() → Varje **behållare** frågar sina **direkta subkomponenter**



Layout 6: Annan storlek än önskad?

En komponent får inte alltid begärd storlek!

Kan få **mer** (fönstermaximering) eller **mindre** (ont om plats)
Layouthanteraren har **regler** för hur utrymmet ska användas



Extra utrymme på höjden:

Ska användas till textarean

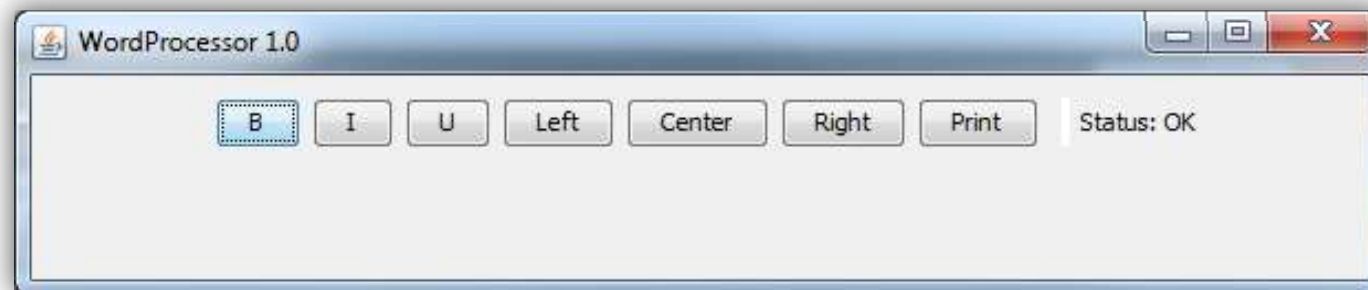
Extra utrymme på bredden:

Menyrad: Extra plats mellan Edit och Help

Knapprad: Allt vänsterjusterat

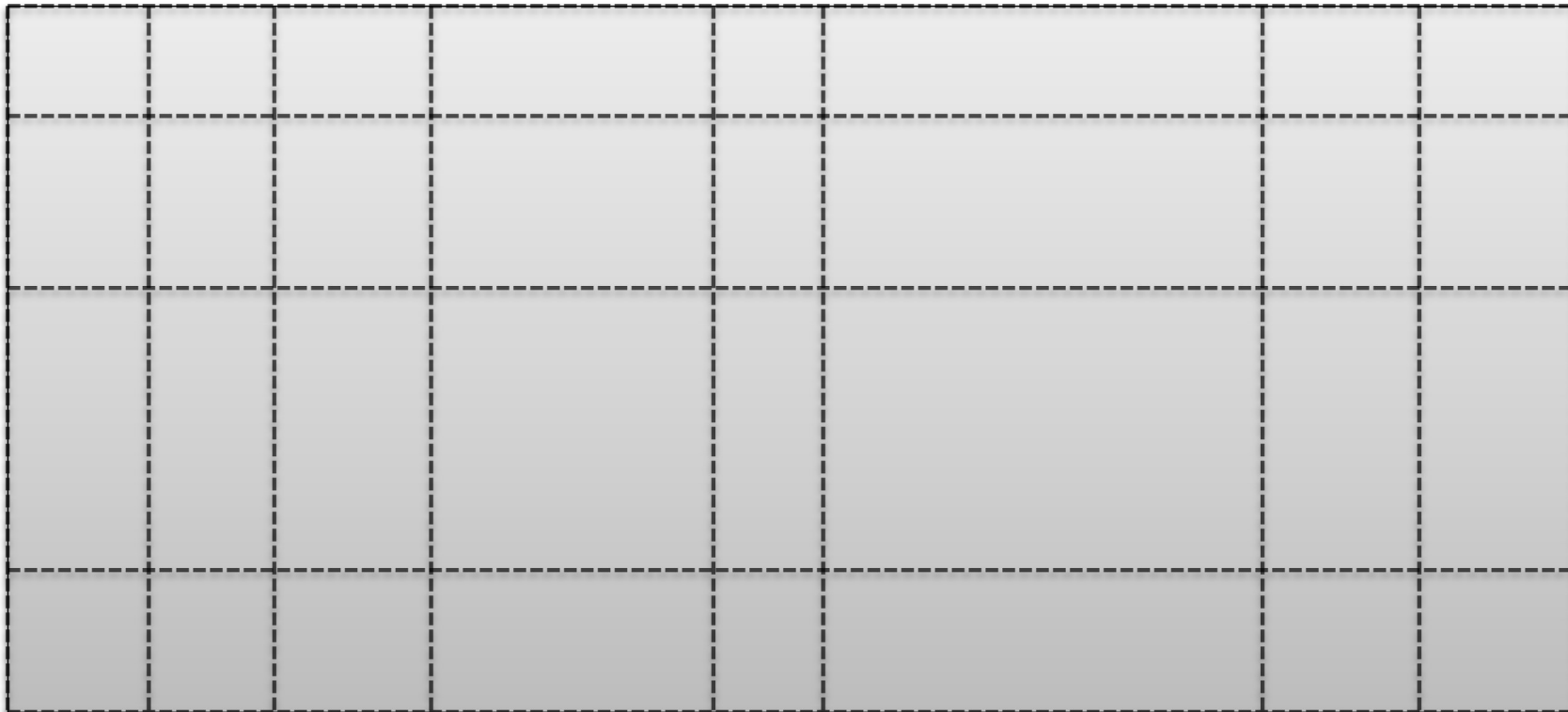
Textfält, statusrad: Använder hela bredden

Just nu:

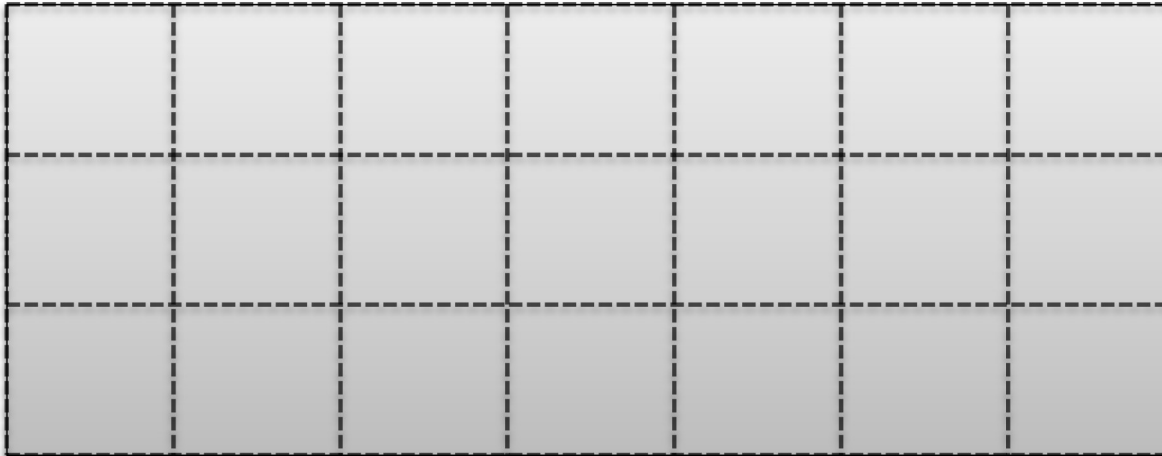


Exempel: MigLayout

- **MigLayout** – rekommenderas starkt!
 - Nu: De enklaste funktionerna
 - Se <http://www.miglayout.com/QuickStart.pdf>,
<http://www.migcalendar.com/miglayout/mavensite/docs/cheatsheet.html>
- Baserad på **oregelbundet rutnät**



- Vi anger **antal rader och kolumner** – (normalt) ingen storlek

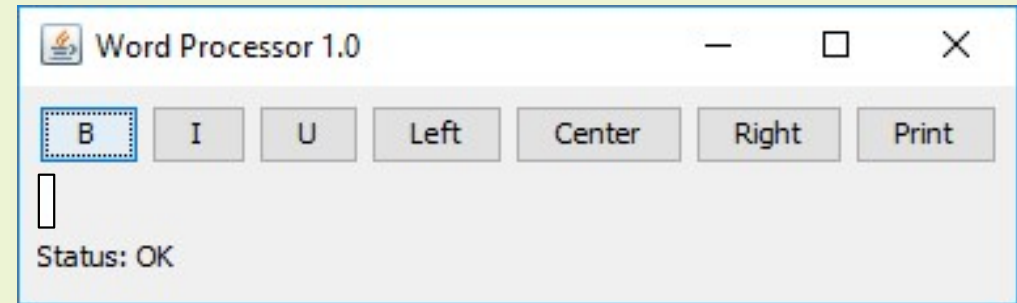


- Vi anger **vilka celler** en komponent måste **hålla sig inom**
 - **Ungefärlig design** – kanske inte kommer att använda hela utrymmet!

B	I	U	Left	Center	Right	Print
Textarea						
Statusrad						

MigLayout 3: En första layout

```
public void show() {  
    this.frame = new JFrame("Word Processor 1.0");  
    frame.setLayout(new MigLayout());  
  
    frame.add(new JButton("B"));  
    frame.add(new JButton("I"));  
    frame.add(new JButton("U"));  
    frame.add(new JButton("Left"));  
    frame.add(new JButton("Center"));  
    frame.add(new JButton("Right"));  
    frame.add(new JButton("Print"),  
              "wrap");  
  
    frame.add(text,  
              "span 7, wrap");  
  
    frame.add(status,  
              "span 7, wrap");  
  
    frame.pack();  
    frame.setVisible(true);  
}
```



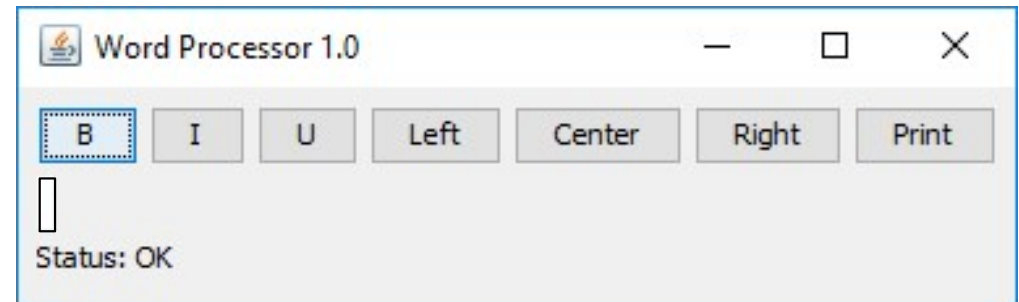
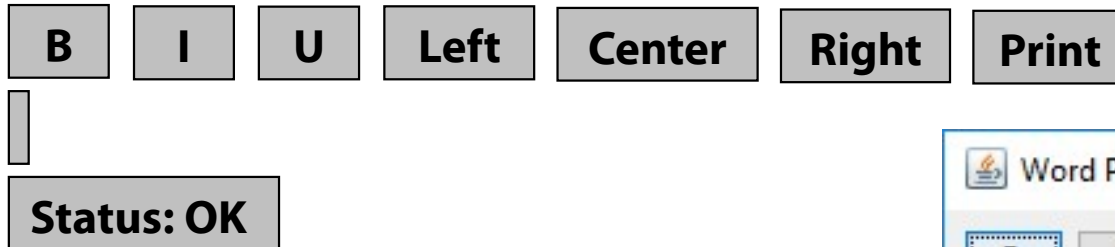
Wrap: Radbyte

Span: Antal celler

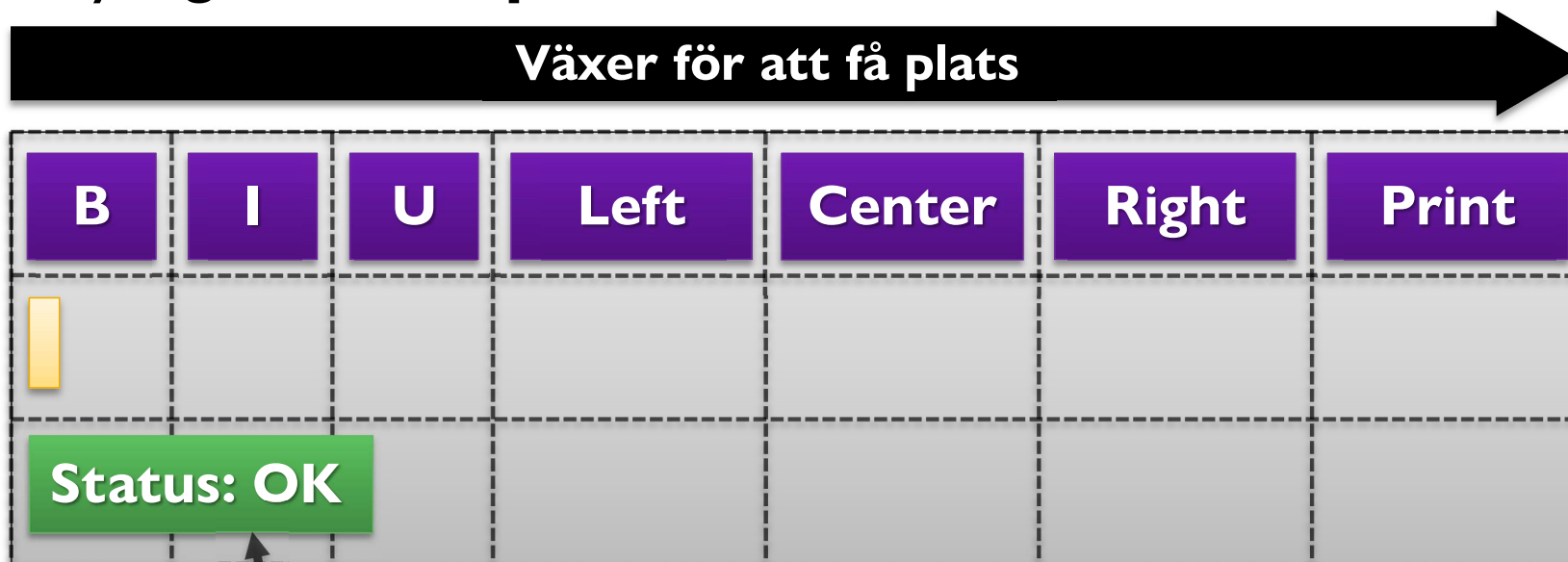
Reservera 7 celler
horisontellt

MigLayout 4: Rutanpassning

- **MigLayout** frågar komponenter efter **önskad** storlek



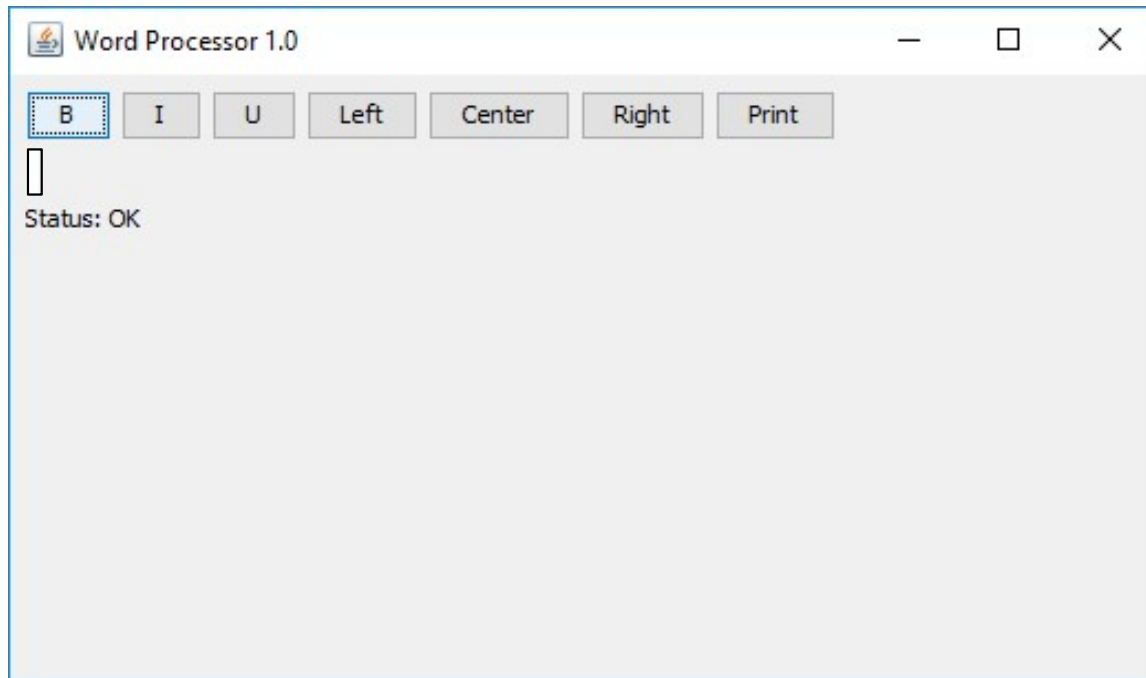
- Osynliga rutor **anpassas**



OK om komponenter inte täcker alla "sina" rutor!

MigLayout 5: Outnyttjat utrymme

- Om vi drar ut fönstret:
 - Ingen cell är bredare eller högre än absolut nödvändigt!

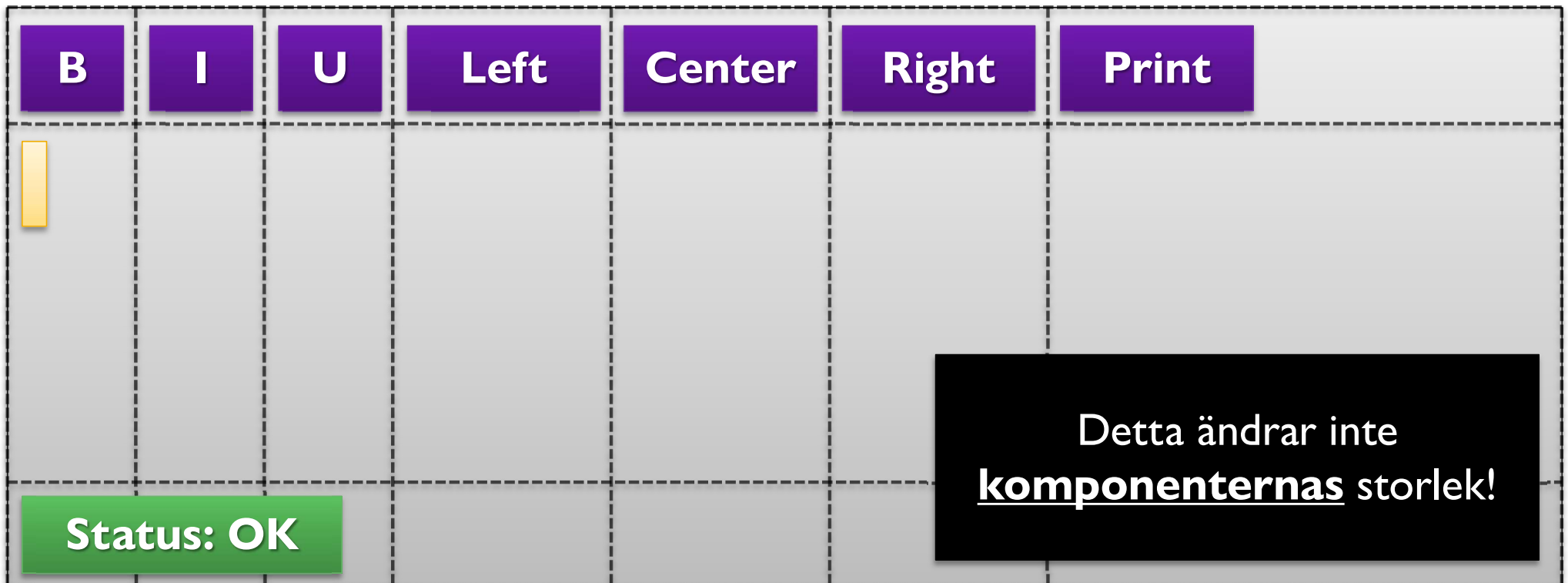


MigLayout 6: Hur får cellerna växa?

```
public void show3() {  
    this.frame = new JFrame("Word Processor 1.0");  
    frame.setLayout(new MigLayout(  
        "", // Globala inställningar  
        "[][][][][]][grow]", // Varje kolumn (7 st)  
        "[][grow][]" // Varje rad (3 st)  
    )); ...  
}
```

Vi förstorar fönstret →
Extra utrymme fördelas
mellan cellerna

- sista kolumnen får växa
- mittersta får växa



Detta ändrar inte
komponenternas storlek!

MigLayout 7: Avstånd, komponentstorlek

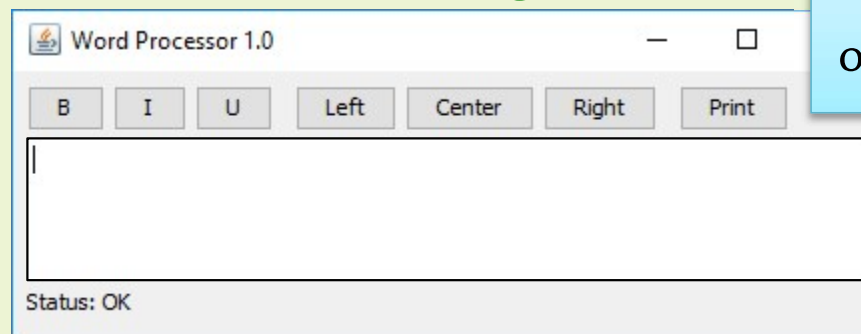
```
public void show4() {  
    this.frame = new JFrame("Word Processor 1.0");  
    frame.setLayout(new MigLayout("", "[][][][][] [grow]", "[] [grow] []"));  
  
    frame.add(new JButton("B"));  
    frame.add(new JButton("I"));  
    frame.add(new JButton("U"), "gapright unrelated");  
    frame.add(new JButton("Left"));  
    frame.add(new JButton("Center"));  
    frame.add(new JButton("Right"), "gapright unrelated");  
    frame.add(new JButton("Print"), "wrap");  
  
    frame.add(text, "span 7, grow, wrap");  
  
    frame.add(status, "span 7, grow, wrap");  
  
    frame.pack();  
    frame.setVisible(true);  
}
```

Separation

Avsluta "gruppen":
Mellanrum till höger,
lagom för "orelaterad
komponent"

Grow: Komponenter

Textrutan+statusraden
ska täcka sina celler,
oavsett "preferred size"



B	I	U	Left	Center	Right	Print
TextArea						
Status: OK						

"Print"
saknar
"grow"

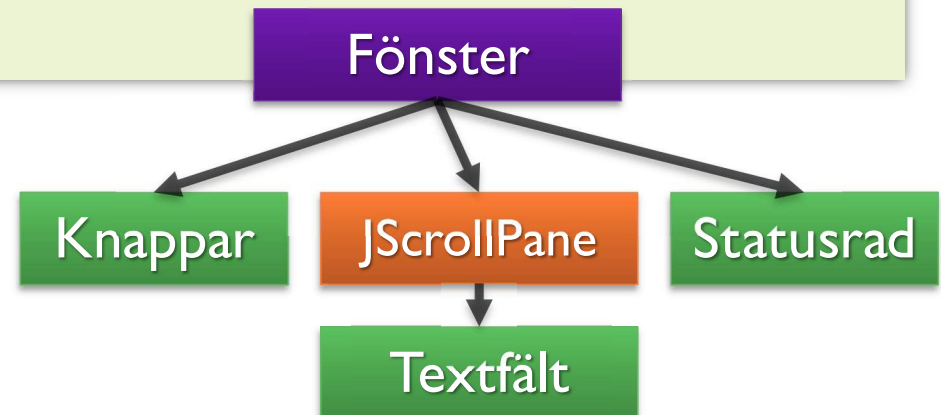
B	I	U	Left	Center	Right	Print
TextArea						
Status: OK						

- Textfältet måste kunna scrollas
 - Inte "inbyggt" i JTextPane – lägg den i en JScrollPane

```
public void show4() {  
    ...  
    frame.add(text, "span 7, grow, wrap");  
    ...  
}
```



```
public void show5() {  
    ...  
    frame.add(new JScrollPane(text), "span 7, grow, wrap");  
    ...  
}
```



Steg 4

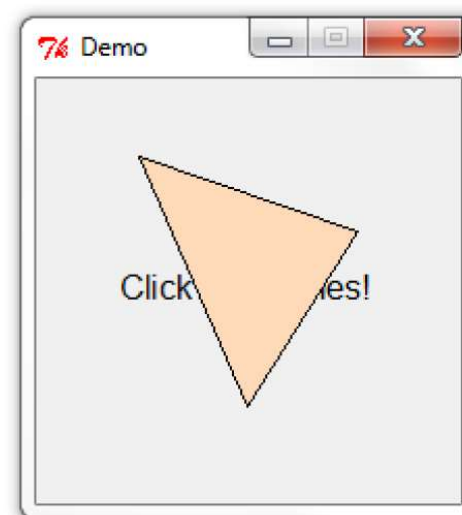
Händelsehantering

**Händelsehantering och callbacks:
högre ordningens funktioner → lyssnarobjekt**

- Hur vet vi när någon tryckte på en knapp?
 - En möjlighet: **Vänta** på ett klick
 - Måste veta *när* det kan hända
 - Måste hantera allt själva – koordinater för knappen, ...

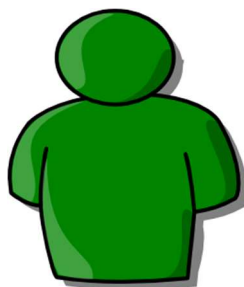
Interaktivitet

```
from graphics import *  
  
def main():  
    win = GraphWin('Demo')  
    msg = Text(Point(100,100), \  
                'Click three times!')  
    msg.draw(win)  
    p1 = win.getMouse()  
    p2 = win.getMouse()  
    p3 = win.getMouse()  
    triangle = Polygon(p1,p2,p3)  
    triangle.setFill('peachpuff')  
    triangle.draw(win)  
  
main()
```



Vänta på ett musklick och returnera ett Point-objekt.

- I många grafiska gränssnitt:
 - **Säg till** systemet **vad det ska göra** vid en **händelse** (ex: musklick)
 - Fortsätt själv med något annat
 - När händelsen inträffar hanteras detta **asynkront** ("i bakgrunden"), i en annan **tråd**
- Tråd: Som om du har en *assistent*



Du säger:
"Om någon klickar, gör *detta*"



Assistenten håller koll
på inmatning
Ser att ett klick kommer,
gör vad du bad om

- Hur vet assistenten vad som ska göras?
 - Låt knappen lagra en "uppgift" – kod att utföra



- I vissa språk: Ge komponenten en funktion – en callbackfunktion

- Liknar högre ordningens funktioner

- **def** task():
 print("Someone pushed the button")

Vanlig funktion

frame.callback = task

Lagra en pekare till
själva funktionen!

- Assistenten:

- Varje gång knappen trycks:
frame.callback()

Använd pekaren
till funktionen

- I Java: Ge komponenten ett objekt som har funktioner (metoder)
 - I Swing: Lyssnare (objekt som "lyssnar" efter händelser)

**Lyssnare:
Gränssnitt och konkret implementation**

- Typsäkerhet: Vi definierar ett **gränssnitt**
 - Vad måste ett **callback-objekt**, en **lyssnare**, kunna göra?

```
public interface MouseListener extends EventListener {  
    public void mouseClicked(MouseEvent e);  
    public void mousePressed(MouseEvent e);  
    ...  
}
```

Ta hand om ett klick!

- Komponenten har en **lista** på **lyssnare av denna typ**

```
// Existerande klass i Swing (något förenklat...)  
public class Component {  
    private List<MouseListener> mouseListeners;  
    public void addMouseListener(MouseListener ml) {  
        mouseListeners.add(ml);  
    }  
}
```

- Vi implementerar konkreta lyssnare

```
public class ClickPrinter implements MouseListener {  
    public void mouseClicked(MouseEvent e) {  
        System.out.println("Hey! Someone clicked at x=" + e.x + ", y=" + e.y);  
    }  
    ...  
}
```

- Vi adderar lyssnare till komponenter

```
public class MyGUI {  
    private JFrame frame;  
    public void show() {  
        MouseListener printer = new ClickPrinter();  
        frame.addMouseListener(printer);  
        // Vårt jobb är klart – assistenten kan be fönstret om alla lyssnare,  
        // lyssnaren vet vad man ska göra  
    }  
}
```

Detta är ett objekt
Objekt kan göra saker

Listan **frame.listeners** innehåller nu
(en pekare till) vårt ClickPrinter-objekt!

- När någon klickar med musen:

Assistenten (händelsehanteringstråden) får info från operativsystemet

- Skapar ett objekt som *beskriver* händelsen:
MouseEvent, **KeyEvent**, ...

```
public class MouseEvent extends InputEvent {  
    int x;           // Koordinater  
    int y;  
    int clickCount; // Hur många gånger klickade de?  
    int button;     // Med vilken musknapp?  
    ...  
}
```

Samtidigt kan vår
"huvudtråd" vara
upptagen med
annat...

Eller kan ha *avslutats*
så att allt nu drivs av
händelsehanteringen

Lyssnare 4: När något händer



- När någon klickar med musen:

Assistenten (händelsehanteringstråden) får info från operativsystemet

- Skapar ett objekt som *beskriver* händelsen: **MouseEvent**, **KeyEvent**, ...
- Tar reda på vilken *komponent* det gäller: **comp**

Assistenten *informerar* varje lyssnare hos den relevanta komponenten

- **for** (**MouseListener ml** : **comp.mouseListeners**) {
 ml.mouseClicked(**event**); // Vanligt metodanrop – men av “assistenten”
}
- Listan inkluderar vår registrerade **ClickPrinter**

```
public class ClickPrinter implements MouseListener {  
    public void mouseClicked(MouseEvent e) {  
        System.out.println("Hey! Someone clicked at x=" + e.x + ", y=" + e.y);  
    }  
}
```

**Lyssnare:
Händelser på olika abstraktionsnivå**

Exempel på lyssnartyper

Lågnivå, kopplade till "rå" input

MouseListener:

Tryck på musknappar, koordinater

MouseMotionListener:

Förflyttning av muspekare

KeyListener:

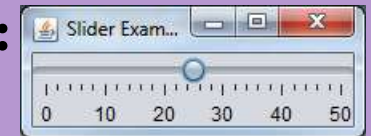
Tangent tryckt / släppt, tecken skrivet

Högnivå, "meningsfulla" händelser

ActionListener: En handling utförs
(korrekt klick på knapp, menyval, ...)

AdjustmentListener:

Slider/scrollbar flyttad



ComponentListener:

Komponent flyttas, ändrar storlek

Komplicerat, använd Key Bindings istället
Mer info på "vanliga lösningar"

Jag vill...

- Göra något med strängar
- Göra något med listor
- Skapa en mappning, som *dict*
- Skapa menyer
- Fråga användaren något
- Reagera på tangenttryckningar
- Stänga ett fönster
- Rita ut en bitmap-bild
- Måla med texturer
- Måla genomskinligt
- Måla med kantutjämning
- Hitte typsnitt
- Upptäcka kollisioner på skärmen
- Spela upp ett ljud
- Spara eller skicka hela objekt

Reagera på tangenttryckningar

I Java finns det två huvudsakliga sätt att få något att hända när man trycker (eller släpper) en tangent i ett GUI-program.

- Använda en `KeyListener` som blir informerad när något händer på tangentbordet, och sedan själv ta reda på vad det var som hände.
- Sätta upp en `InputMap` och en `ActionMap` som kopplar tangenthändelser till handlingar.

Det första sättet kan vara ganska knepigt, bland annat för att det alltid är en specifik GUI-komponent som har *tangentbordsfokus* och man måste se till att just den komponenten har en lyssnare. `InputMap` plus `ActionMap` ser mer komplicerade ut, men brukar leda till färre problem i praktiken. Därför går vi genom den metoden, och använder föreläsningens "ordbehandlarexempel" som bas.

I ordbehandlaren kan vi definiera en handling som kan utföras. Detta är en speciell typ av `ActionListener`. Vi kan få hjälp att skapa den genom att ärva från hjälpklassen `AbstractAction`, så behöver vi bara implementera själva `actionPerformed`-metoden precis som "vanligt".

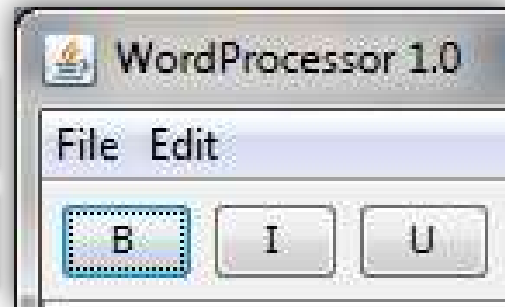
```
public class WordProcessor20 {
    ...

    private class QuitAction extends AbstractAction {
        @Override public void actionPerformed(final ActionEvent e) {
            // Gör vad som behövs -- öppna dialogruta och fråga, ...
            System.exit(0);
        }
    }
}
```

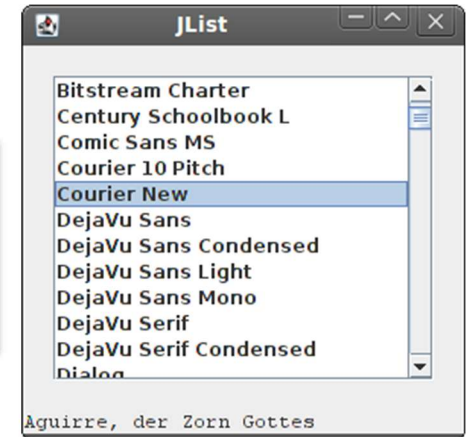
**En av de viktigaste:
ActionListener**

- Vanlig och enkel lyssnare på högre nivå: **ActionListener**
 - Anger vad som ska göras när någon vill "utföra en handling"

Menyval
Knapptryck



Dubbelklick i
lista



```
package java.awt.event;  
import java.util.EventListener;  
  
public interface ActionListener extends EventListener {  
  
    /** Invoked when an action occurs. */  
    public void actionPerformed(ActionEvent e);  
}
```

ActionListener 2: Exempel

```
public class WordProcessor07 {  
    ...  
    private JTextPane text = new JTextPane();  
    ...  
    public void show7() {  
        ...  
        final JButton bold = new JButton("B");  
        bold.addActionListener(new BoldListener(text));  
        frame.add(bold);  
        ...  
    }  
}
```

Skapa knapp
Ge den en lyssnare
Lägg till den i fönstret

Menyer hanteras på samma sätt

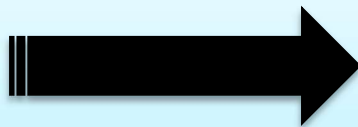
```
public class BoldListener implements ActionListener {  
    private final JTextPane textPane;  
    public BoldListener(final JTextPane text) { this.textPane = text; }  
  
    public void actionPerformed(final ActionEvent e) {  
        textPane.getStyledDocument().setCharacterAttributes(...);  
    }  
}
```

Håll reda på
textkomponenten

När vi anropas:
Manipulera
textkomponenten

Lyssnare: Alternativ och "förkortningar"

Grundläggande OO,
syns exakt vad som händer,
men mycket att skriva



Mycket "magi" och
automatiska
transformationer, men kort
och koncist

En översikt över olika möjligheter – behöver inte kunna allt i detalj!

```
package se.liu.ida.jonkv.wordpro.gui;

public class BoldListener implements ActionListener {
    private final JTextPane textPane;
    public BoldListener(final JTextPane text) {
        this.textPane = text;
    }

    public void actionPerformed(final ActionEvent e) {
        textPane.getStyledDocument().setCharacterAttributes(..
    }
}
```

1. BoldListener **synlig** i hela paketet:
“Förorenar namnrymden”
(och vi måste *hitta på* ett bra namn)

2. Mycket arbete att hålla
reda på information som
lyssnaren behöver

3. Mycket extra kod runt den
enda rad vi egentligen är
intresserade av

```
public class WordProcessor08 {  
    ...  
    private JTextPane text = new JTextPane();  
    ...  
    public void show8() {  
        ...  
        final JButton bold = new JButton("B");  
        bold.addActionListener(new BoldListener());  
        frame.add(bold);  
        ...  
    }  
}
```

Nästlade "inre" klasser

BoldListener är en **implementationsdetalj** i WordProcessor08

Kan ligga **nästlad** (**inuti** WordProcessor08), som nästlade funktioner i Python

Kan **döljas utifrån** (private)

```
private class BoldListener implements ActionListener {  
    public void actionPerformed(final(ActionEvent e) {  
        text.getStyledDocument().setCharacterAttributes(...);  
    }  
}
```

Inre klasser (2)

```
public class WordProcessor08 {  
    ...  
    private JTextPane text = new JTextPane();  
    ...  
    public void show8() {  
        ...  
        final JButton bold = new JButton("B");  
        bold.addActionListener(new BoldListener());  
        frame.add(bold);  
        ...  
    }  
}
```

Nästlade "inre" klasser

Kan direkt komma åt fält i den omgivande klassen, som Pythons nästlade funktioner

Överkurs: Magi?

Nej: Kompilatorn "gör om" BoldListener, ger den en pekare till det omgivande objektet som skapade det ("denna WP08")

```
private class BoldListener implements ActionListener {  
    public void actionPerformed(final ActionEvent e) {  
        text.getStyledDocument().setCharacterAttributes(...);  
    }  
}
```

```
public class BoldListener {  
    private WP08 $outer$...  
    ...  
    $outer$.text.get...()  
}
```

WordProcessor02:
JTextPane text;

JTextPane

```
public class WordProcessor09 {  
    ...  
    private JTextPane text = new JTextPane();  
    ...  
    public void show9() {  
        ...  
        final JButton bold = new JButton("B");  
        frame.add(bold);  
  
        ActionListener makebold = new ActionListener() {  
            public void actionPerformed(final ActionEvent e) {  
                text.getStyledDocument().setCharacterAttributes(...);  
            }  
        }  
        bold.addActionListener(makebold);  
    }  
}
```

Anonymt

Klasser kan skapas "anonymt"

Helt ekvivalent med förra sidan:

Detta deklarerar en **ny klass**, som *implementerar* `ActionListener`, där *kompilatorn* hittar på klassnamn, och ett objekt av nya klassen skapas

Även inuti en metod

Man kan till och med lägga en klass inuti en metod...

- 1) Kan ge bättre lokalitet: Kopplar lyssnarklassen till användningen
- 2) Kan komma åt lokala variabler

Hur många lyssnare?

En lyssnare, flera knappar



```
public class WordProcessor09 {  
    private JTextPane text = new JTextPane();  
    public void show9() {  
        final JButton bold = new JButton("B");  
        frame.add(bold);  
        final JButton italics = new JButton("I");  
        frame.add(italics);  
  
        class ButtonListener implements ActionListener {  
            public void actionPerformed(final ActionEvent e) {  
                if (e.getSource() == bold) {  
                    text.getStyledDocument().setCharacterAttributes(...);  
                } else if (e.getSource() == italics) {  
                    text.getStyledDocument().setCharacterAttributes(...);  
                }  
            }  
        }  
  
        bold.addActionListener(new ButtonListener());  
        italics.addActionListener(new ButtonListener());  
    }  
}
```

Lyssna på flera knappar

Måste undersöka källan till en händelse (event)

Klass i metod →
kommer åt final-variabler i metoden

Vill vi göra så?

Beror på – hitta en bra balans!

Flera liknande lyssnarklasser?

- Ibland blir det alltför upprepat...

```
public class GuiWithManyListeners
```

```
{  
    Game game = new Game();  
  
    private class UpKeyAction implements ActionListener  
    {  
        public void actionPerformed(final ActionEvent e) {  
            game.moveCharacters(Direction.UP);  
        }  
    }  
  
    private class RightKeyAction implements ActionListener  
    {  
        public void actionPerformed(final ActionEvent e) {  
            game.moveCharacters(Direction.RIGHT);  
        }  
    }  
}
```

```
private class DownKeyAction implements ActionListener  
{  
    public void actionPerformed(final ActionEvent e) {  
        game.moveCharacters(Direction.DOWN);  
    }  
}  
  
private class LeftKeyAction implements ActionListener  
{  
    public void actionPerformed(final ActionEvent e) {  
        game.moveCharacters(Direction.LEFT);  
    }  
}
```

En parametriserad lyssnarklass!



```
public class GuiWithParameterizedListener
{
    public GuiWithParameterizedListener() {
        ... new DirectionKeyAction(Direction.UP) ...
        ... new DirectionKeyAction(Direction.DOWN) ...
    }

    private class DirectionKeyAction implements ActionListener
    private Direction dir;

    public DirectionKeyAction(final Direction dir) {
        this.dir = dir;
    }

    public void actionPerformed(final ActionEvent e) {
        game.moveCharacters(this.dir);
    }
}
}
```

Parametrisera!

- Vilken Direction det gäller
- Vilken färg det gäller
- Vilken spelare det gäller
- ...

Om lyssnaren har en metod

```
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent e);  
}
```

```
public class WordProcessor10 {  
  
    public void show10() {  
        final JButton bold = new JButton("B");  
        bold.addActionListener(e -> {  
            text.getStyledDocument().setCharacterAttributes(...);  
        });  
    }  
}
```

Vi kan använda ett lambdauttryck!

Fast metoden addActionListener() kräver ett objekt av *ActionListener*-typ!

Bakgrundsmagi ser till att detta fungerar...

Python

```
lambda x: doSomething(x)  
lambda x, y: calculate(x,y)
```

Java

```
x -> { doSomething(x); }  
(x, y) -> { return calculate(x,y); }
```