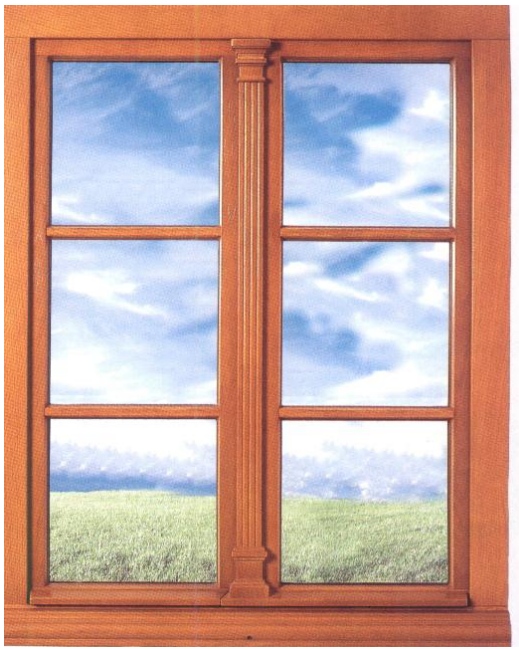
Benutzeroberflächen

Abstract Windowing Toolkit, Rahmen, Wächter, Ereignis-behandlung, graphische Ausgabe, Menüs.

Der Abstract Windowing Toolkit (awt)

- Jedes moderne Betriebssystem stellt bereit
  - Fenster
  - Menüs
    - *Popup, Pulldown, ContextMenü*
  - Schriftarten,
    - verschieden Fonts, Größen, Stile
  - Dialoge
    - Meldungen, Textinput, Listboxen, Radioboxen

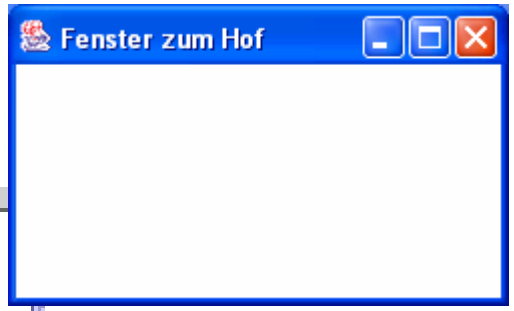


- Benutzeroberfläche simuliert Schreibtisch
  - neue Programme sind intuitiv benutzbar
  - manchmal genauso unaufgeräumt
- Moderne Programme nutzen diese Features
  - Terminal-Eingabe nur für Testzwecke
- Plattformunabhängig programmieren bedeutet
  - abstrakte Fensteroberfläche bereitstellen
    - nur Funktionen, die auf jeder Plattform unterstützt werden
  - Programmierer benutzt nur die abstrakten Funktionen
  - auf jeder Plattform werden diese geeignet implementiert

Ein Fensterrahmen

- Ein Fenster besteht aus einem Rahmen
  - engl.: *Frame*
  - Die Klasse **`Frame`** ist in dem Paket **`java.awt`**

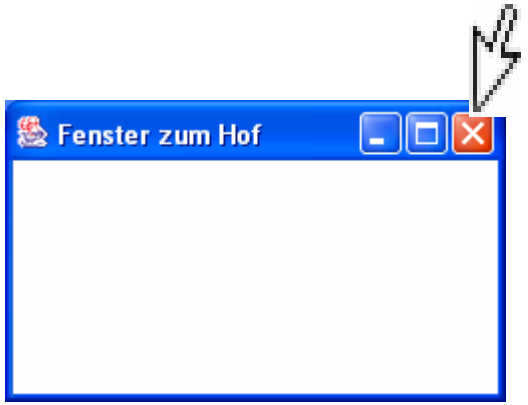
```
import java.awt.*;
public class RahmenTest{

    static void test(){
        Frame f = new Frame();
        f.setTitle("Fenster zum Hof");
        f.setSize(250,150);
        f.setVisible(true);
    }
```



- Das Ergebnis ist ein Fenster mit
  - Titel „Fenster zum Hof“
  - Breite 250
  - Höhe 150
- Das Fenster können wir
  - maximieren
  - minimieren
  - vergrößern
  - verkleinern
- aber nicht schließen

Das Fenster schließt nicht

- Wir benötigen einen Wächter (WindowListener)
  - dieser wartet auf eins der Ereignisse:
    - Maus klickt auf 
    - Menü-Punkt **Close**
    - Tastenkombination **Alt-F4**
  - und schließt dann das Fenster mit `System.exit(0);`



- In der Klasse `WindowAdapter` müssen wir nur die Methode

  `void windowClosing(WindowEvent e)`

  überschreiben. Bisher tut sie nichts, jetzt soll sie den Fenster-Prozess löschen:

  `System.exit(0)`.

- **Dazu schreiben wir eine Klasse Wächter, die `WindowAdapter` beerbt und `windowClosing` redefiniert:**

```
import java.awt.event.*;
class Wächter extends WindowAdapter{ ... }
```

Ein Fensterwächter

- Die Klasse **`Wächter`** brauchen wir nur hier, wir definieren sie also lokal:

```
class Wächter extends WindowAdapter{
   public void
     windowClosing(WindowEvent e){
             System.exit(0);
   }
 }
```

- Wir erzeugen einen Wächter
  - `Wächter wächter = new Wächter();`
- und fügen ihn zum Fenster hinzu:
  - `f.addWindowListener(wächter);`

Das Fenster schließt.

```
import java.awt.*;
import java.awt.event.*;

public class RahmenTest{
    void test(){
        Frame f = new Frame();
        f.setTitle("Fenster zum Hof");
        f.setSize(250,150);
        f.setVisible(true);
        f.addWindowListener(wächter);
    }
    class Wächter extends WindowAdapter{
          public void windowClosing(WindowEvent e){
              System.exit(0);
           };
    }
    Wächter wächter = new Wächter();
}
```

- der grau unterlegte Code wurde hinzugefügt:
- **`wächter`** dem Rahmen hinzufügen
- Ein **`Wächter`** ist ein **`WindowAdapter`**, der das Fenster schließen kann
- **`WindowAdapter`** implementiert das *interface* **`WindowListener`**
- Einen **`Wächter`** erzeugen

Unsere eigene Klasse: Fenster

```
/** Eine Klasse gut schließender Fenster
  * @author H. P. Gumm
  * @version 3.10.2002  */
import java.awt.*;
import java.awt.event.*;
public class Fenster extends Frame{
    // Ein Fenster ist ein Frame mit einem Wächter
    Wächter wächter = new Wächter();

    public Fenster(){};  // default Konstruktor

    /** richtiger Konstruktor */
    public Fenster(String titel,int breite,int höhe){
        super(titel);
        setSize(breite,höhe);
        setVisible(true);
        addWindowListener(wächter);
    }
    private class Wächter extends WindowAdapter{
         public void windowClosing(WindowEvent e){
                dispose();
         }
    }
}
```

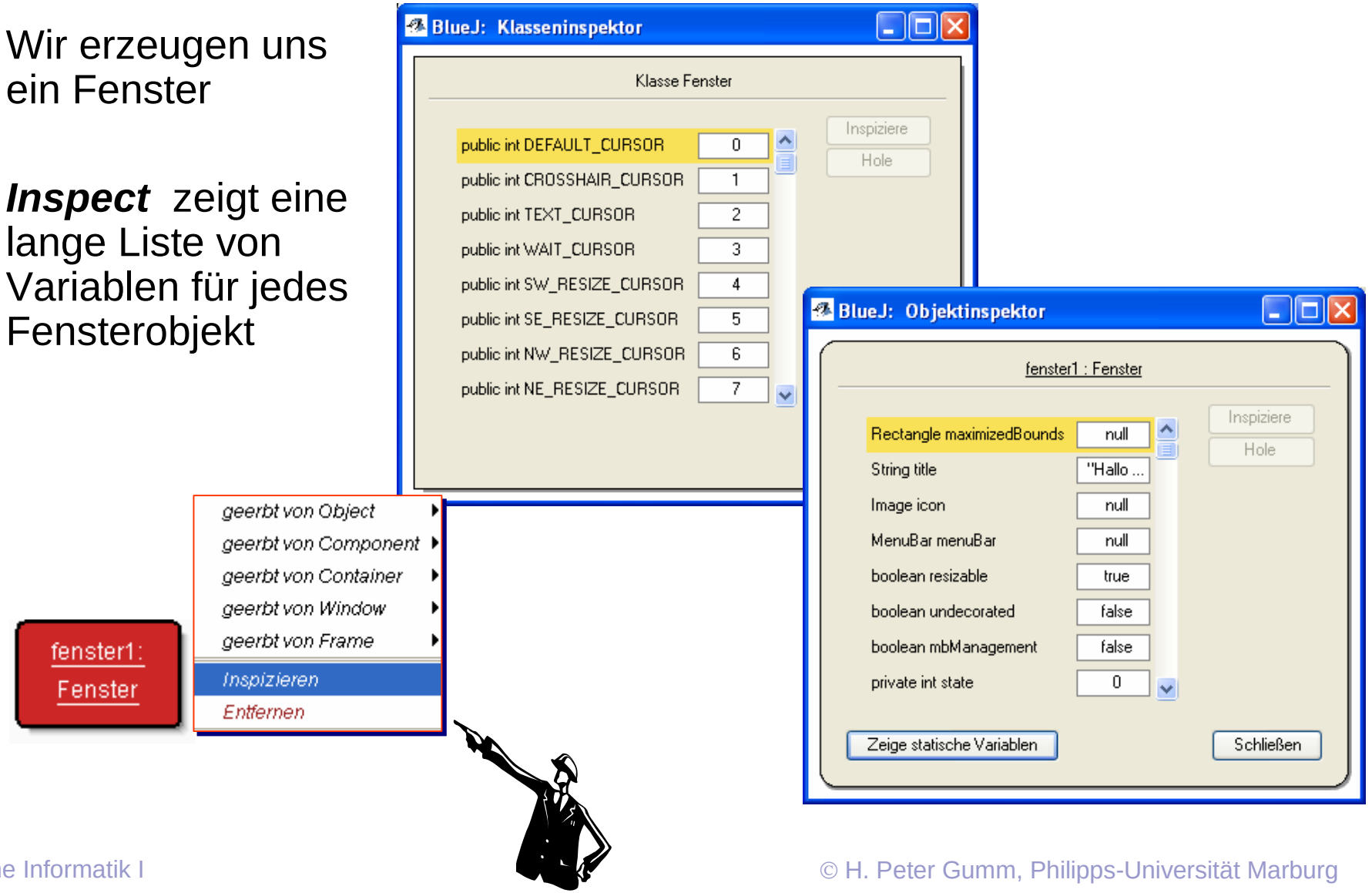
- In Zukunft leiten wir Fenster aus dieser Klasse ab. Dann brauchen wir uns um das korrekte Schließen nicht mehr zu kümmern
- **Wächter** ist eine *innere Klasse* von **Fenster**. Alle Felder und Methoden von **Fenster** sind in **Wächter** sichtbar
- Statt **System.exit(0)** können wir auch **dispose()** aufrufen.
- **dispose()** ist eine Methode von **Frame**
- to dispose : vornehm für „auf den Müll werfen“

Auf den Schultern von Riesen ...

- *Javadoc* zeigt uns für unsere neue Klasse Fenster unter anderem die Klassenhierarchie
- Fenster ererbt alle Fähigkeiten der Vorgänger
  - das sind nicht wenige ...

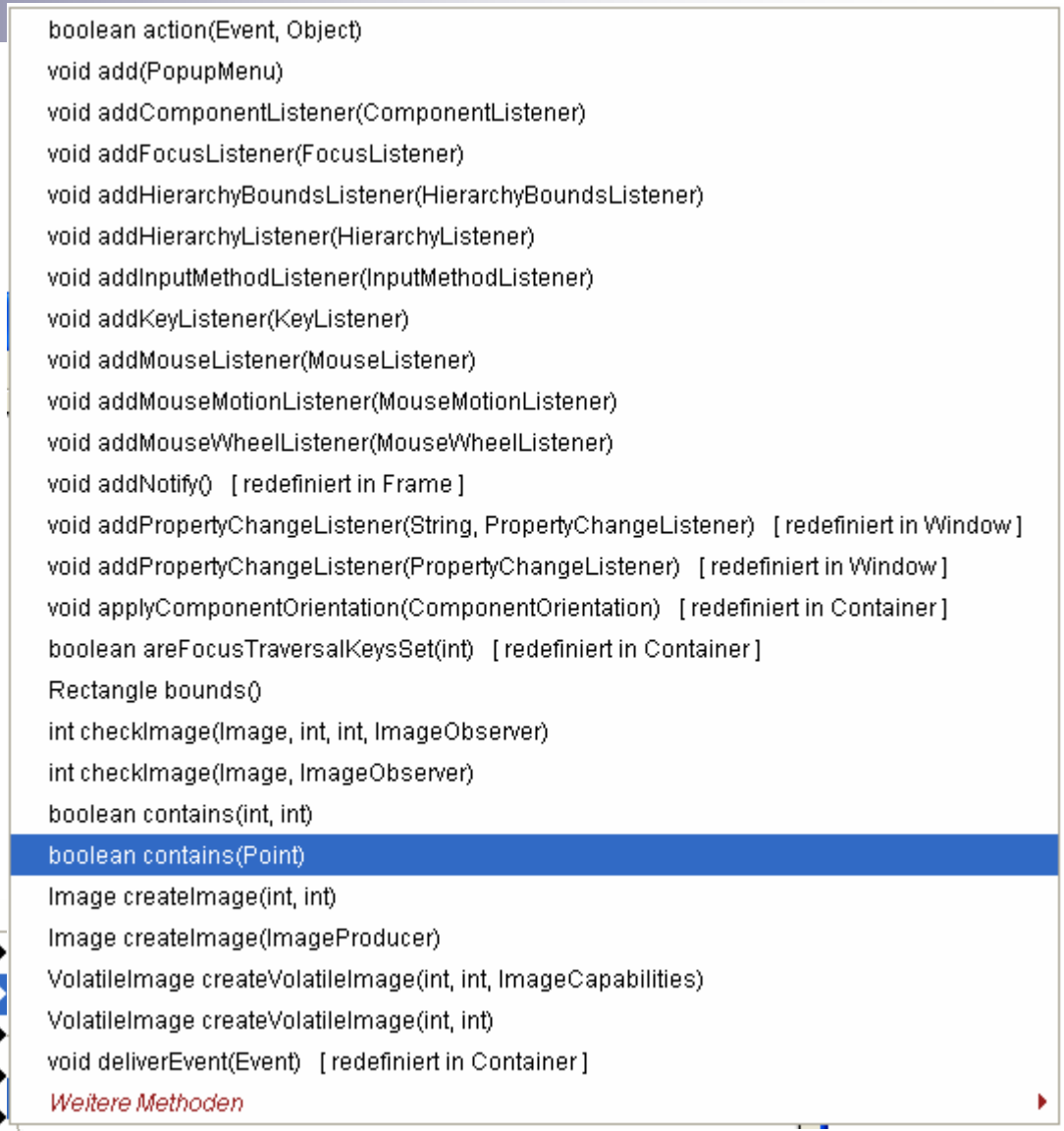
Was das Fenster so alles hat

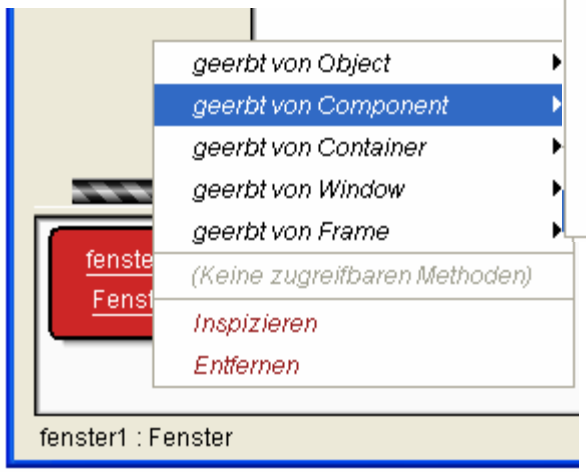
- Wir erzeugen uns ein Fenster
- ***Inspect*** zeigt eine lange Liste von Variablen für jedes Fensterobjekt



Was das Fenster schon alles kann

- Das Fenster erbt alle Methoden seiner Oberklassen ...
- insbesondere erbt es von **`Component`** die Methode:
  `getGraphics()`
- damit können wir graphische Ausgaben erzeugen ...





Fenstermalerei


```
import java.awt.*;

public class Tafel extends Fenster{

    Tafel(String titel,int höhe, int breite){
        super(titel,höhe,breite);
    }

public static void main(String[] args)
    {
    Tafel t = new Tafel("Meine Tafel",400,200);
    Graphics g = t.getGraphics();
    g.setFont(new Font("SansSerif",Font.PLAIN,28));
    g.drawString("Hallo Welt",20,70);
    g.drawOval(100,100,30,30);
    g.setColor(Color.BLUE);
    g.fillRect(150,100,30,30);
    }
```

getGraphics() wird von Component einer Oberklasse von Frame vererbt.

liefert das zum Fenster gehörige Objekt g der Klasse Graphic mit Feldern wie
- akt. Farbe,
- Font,

mit Methoden wie
- drawString,
- drawLine,
- drawOval, ...

Die Methode paint()

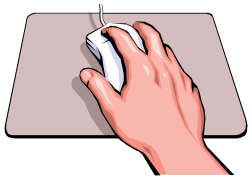
- Fenster wird nur einmal bemalt
- Verändert man die Größe, so wird es nicht wieder neu bemalt.

- Bei jeder Veränderung des Fensterhintergrundes wird automatisch die Methode **`paint(Graphics g)`** aufgerufen
- Verlegt man die graphische Ausgabe in paint(), so wird das Fenster immer wieder korrekt nachgezeichnet.

```
import java.awt.*;

public class Tafel extends Fenster{
    Tafel(String titel,int höhe, int breite){
        super(titel,höhe,breite);
    }

public static void main(String[] args){
    Tafel t = new Tafel("Meine Tafel",400,200);
    }

    public void paint(Graphics g){
      g.setFont(new Font("SansSerif",Font.PLAIN,28));
      g.drawString("Hallo Welt",20,70);
      g.drawOval(100,100,30,30);
      g.setColor(Color.BLUE);
      g.fillRect(150,100,30,30);
    }
```

Events

- Ereignisse, die zu unvorhersehbaren Zeitpunkten auftreten kann
  - Maus-Ereignisse
    - klick
    - drücken
    - loslassen
    - Maus betritt Fenster
    - Maus verlässt das Fenster

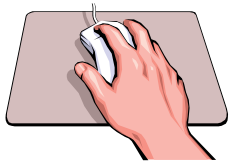


  - Maus-Bewegungen
    - Maus wurde bewegt (to move)
    - Maus wurde geschleppt (to drag)
  - Fenster-Ereignisse
    - schließen
    - maximieren, minimieren
    - vergrößern
    - verschieben

  - Aktionen
    - Menüpunkt ausgewählt
  - TastaturEreignisse
    - Taste gedrückt,
    - Taste unten
    - Taste losgelassen

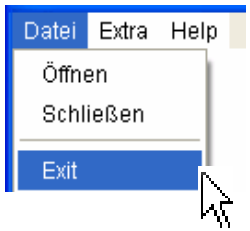
EventKlassen

- Jedes Ereignis erzeugt ein Objekt einer entsprechenden Eventklasse
- diese Klassen findet man in dem Paket java.awt.event
  - **ActionEvent**
  - **KeyEvent**
  - **MouseEvent**
  - **WindowEvent**
- Jedes Event kodiert nähere Informationen über das Ereignis,
- diese kann man mit entsprechenden Methoden abfragen



MouseEvent :

- int getX( ) x-Position des Ereignisses
- int getY( ) y-Positzion des Ereignisses



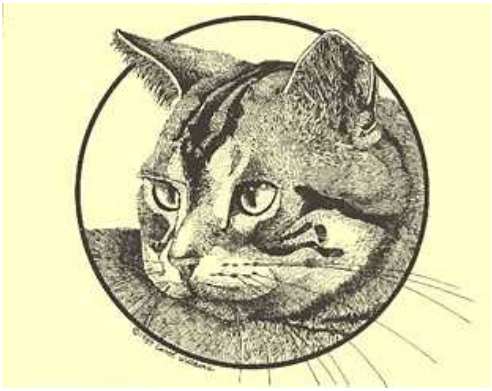
ActionEvent :

- String getActionCommand( ) Text des Menüeintrags
- getModifiers( ) war zusätzlich Shift, Alt etc. gedrückt ?

KeyEvent :

- getKeyChar()
- getKeyText()

Listener



- Listener sind *Wächter*, die auf bestimmte Ereignisse lauschen
  - Damit ein Fenster auf Mausereignisse lauschen kann, muss ein MausWächter (MouseListener) installiert werden
  - Damit ein Menü auf eine Auswahl reagiert, muss ein MenüWächter (ActionListener) installiert werden
- Der Wächter bestimmt die Reaktion auf ein Ereignis
- Wächter sind *interfaces*, die man selber *implementiert*

- Beispiel: interface MouseListener spezifiziert die Methoden
  - `public void mouseClicked(MouseEvent e)`
  - `public void mouseEntered(MouseEvent e)`
  - `public void mouseExited(MouseEvent e)`
  - `public void mousePressed(MouseEvent e)`
  - `public void mouseRelesed(MouseEvent e)`

Ein Kater als MouseListener

- interface **MouseListener** spezifiziert die Methoden
- **public void mouseEntered(MouseEvent e)**
- **public void mouseExited(MouseEvent e)**
- **public void mouseClicked(MouseEvent e)**
- **public void mousePressed(MouseEvent e)**
- **public void mouseReleased(MouseEvent e)**
- wir müssen sie alle implementieren, notfalls tun sie nichts.

```
class Kater implements MouseListener{
   public void mouseEntered(MouseEvent e){
        g.drawString("Hallo Maus ... ",50,50);
   }
   public void mouseExited(MouseEvent e){
        g.drawString("... tschüss, Maus.",150,200);
    }
   public void mouseClicked(MouseEvent e){
       int xPos=e.getX();
       int yPos=e.getY();
       g.drawString("("+xPos+","+xPos+")",xPos,yPos);
   }
   public void mousePressed(MouseEvent e){}
   public void mouseReleased(MouseEvent e){}
}
```

Kater im Fenster

Wir erzeugen ein Objekt kater vom Typ Kater.

Wir setzen den Kater ins Fenster

Jeder Kater ist ein MouseListener

MausFenster

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Close | Implementation

```
import java.awt.Graphics;
import java.awt.event.*;

public class MausFenster extends Fenster{

    Graphics g = getGraphics();
    Kater kater = new Kater();

    MausFenster(String titel, int breite, int höhe){
        super(titel,breite,höhe);
        addMouseListener(kater);
    }

    class Kater implements MouseListener{
        public void mouseEntered(MouseEvent e){
            g.drawString("Hallo Maus ... ",50,50);
        }
        public void mouseExited(MouseEvent e){
```

saved

Fenster mit Menü

```
Menu datei = new Menu("Datei");
datei.add("Öffnen");
datei.add("Schließen");
datei.addSeparator();
datei.add("Exit");

Menu extra = new Menu("Extra");
extra.add("Info");
extra.add("Version");

Menu help = new Menu("Help");

MenuBar mb = new MenuBar();
mb.add(datei);
mb.add(extra);
mb.add(help);

setMenuBar(mb);
```

Zusammenfassung zu einer Menü-Zeile

Zum Fenster hinzufügen

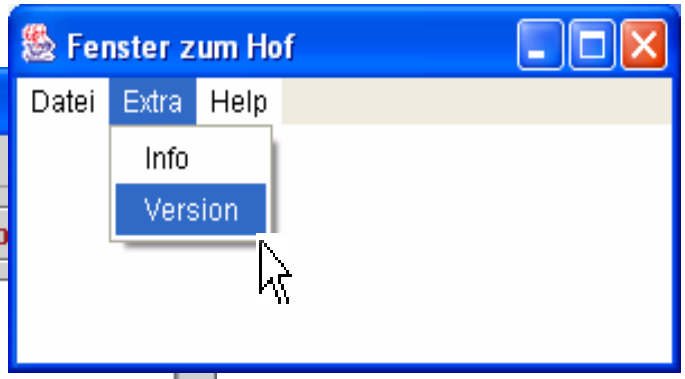
ActionListener überwachen Menüs

- ActionListener ist ein interface, das nur eine einzige Methode spezifiziert:
  - `public void actionPerformed(ActionEvent e)`
- Aus dem ActionEvent e extrahiert man mit
  - `public String getActionCommand()`

  das MenüKommando.

```
MenüWächter wächter = new MenüWächter();


class MenüWächter implements ActionListener{
    public void actionPerformed(ActionEvent e){
        String kommando = e.getActionCommand();
        if (kommando.equals("Exit"))
            System.exit(0);
        else if (kommando.equals("Info"))
            System.out.println("Nix besonderes");
        else if (kommando.equals("Help"))
            System.out.println("No help available");}
}
```

Ein Wächter für jedes Menü



FensterMitMenü

Class Edit Tools Options

Compile Undo Cut Copy Paste Close Implementatio

```
        MenuBar mb = new MenuBar();
        mb.add(datei);
        mb.add(extra);
        mb.add(help);

        setMenuBar(mb);

        datei.addActionListener(wächter);
        extra.addActionListener(wächter);
        help.addActionListener(wächter);
    }

    MenüWächter wächter = new MenüWächter();

    class MenüWächter implements ActionListener{
```

Jedes Menü erhält mit addActionListener seinen Wächter.

Es darf auch immer der gleiche sein

MouseMotionListener

- MouseMotionListener ist ein *interface*, für Mausbewegungen
- Es spezifiziert nur zwei Methoden
  - public void mouseMoved(MouseEvent e)
  - public void mouseDragged(MouseEvent e)

```
class Katze implements MouseMotionListener{
      public void mouseMoved(MouseEvent e){ }
      public void mouseDragged(MouseEvent e){
            int xNew=e.getX();
            int yNew=e.getY();
            g.drawLine(xOld,yOld,xNew,yNew);
            xOld = xNew;
            yOld = yNew;
      }
}
```

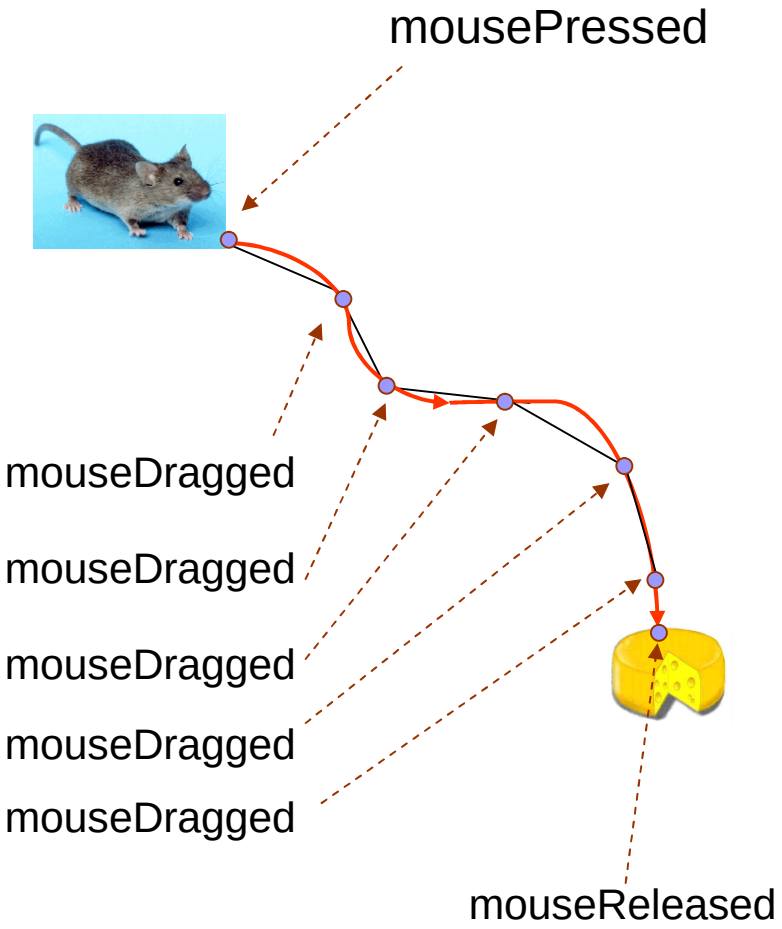
Fenstermalerei

- Wir wollen mit der Maus im Fenster malen
- Beim Drücken der Maus wird
  - **mousePressed**(MouseEvent e)
- aufgerufen. Wir halten den Anfangspunkt in den Feldern **xOld** und **yOld** fest.
- Wenn wir die Maus ziehen, wird immer und immer wieder die Methode
  - **mouseDragged**(MouseEvent **e**) aufgerufen.
- Aus e extrahieren wir die aktuelle Position:

```
xNew = e.getX();
yNew = e.getY();
```

- .Von der alten zur aktuellen Position ziehen wir eine Linie:
  - `drawLine(xOld,yOld,xNew,yNew);`
- Die aktuelle Position wird zur aktuellen Position :

```
xOld = xNew;
yOld = yNew;
```



Das ist die letzte Folie

- Machen Sie hier weiter
  - implementieren Sie
  - ein Menü für die Auswahl von Zeichenstiften
    - dick, dünn,...
  - Farbauswahl mittels Schieberegler ...
  - Knöpfe zum Löschen ...
  - einen Radiergummi ...
  - ...