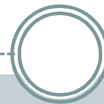
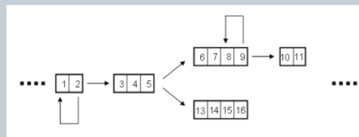
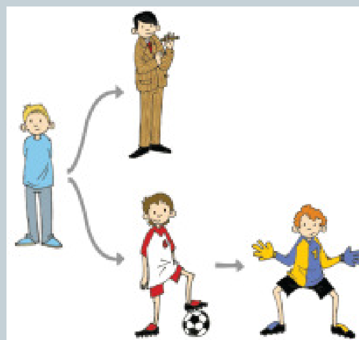
# Kurs: Programmieren in Java

## Tag 5



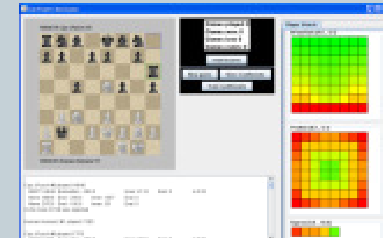


GRUNDLAGEN

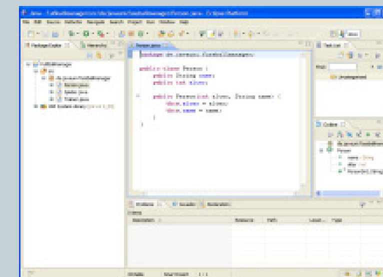


OBJEKTORIENTIERTE PROGRAMMIERUNG

GRAFIKKONZEPTE
BILDVERARBEITUNG
MUSTERERKENNUNG

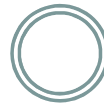


KI UND SPIELE-PROGRAMMIERUNG



ENTWICKLUNGS-UMGEBUNGEN

# Grafische Benutzeroberflächen



Inhalt:

- Fenstermanagement unter AWT
- Zeichenfunktionen
- Die Klasse Color
- Fensterereignisse

Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*" , Springer-Verlag 2007

## Ein Fenster erzeugen I

Wir können schon mit wenigen Zeilen ein Fenster anzeigen, indem wir eine Instanz der Klasse **Frame** erzeugen und sie sichtbar machen. Bevor wir die Eigenschaft , „ist sichtbar" mit *setVisible*(true) setzen, legen wir mit der Funktion *setSize*(breite, hoehe) die Fenstergröße fest.

```
import java.awt.Frame;
public class MeinErstesFenster {
  public static void main(String[] args) {
     // öffnet ein AWT-Fenster
     Frame f = new Frame("So einfach geht das?");
     f.setSize(300, 200);
     f.setVisible(true);
  }
}
```

Nach dem Start öffnet sich folgendes Fenster:

## Ein Fenster erzeugen II

Nach der Erzeugung des Fensters ist die Ausgabeposition die linke obere Ecke des Bildschirms.

Das Fenster lässt sich momentan nicht ohne Weiteres schließen. Wie wir diese Sache in den Griff bekommen und das Programm nicht jedes mal mit Tastenkombination STRG+C (innerhalb der Konsole) beenden müssen, sehen wir später. Da das dort vorgestellte Konzept doch etwas mehr Zeit in Anspruch nimmt, experimentieren wir mit den neuen Fenstern noch ein wenig herum.

Wir wollen das Fenster zentrieren:

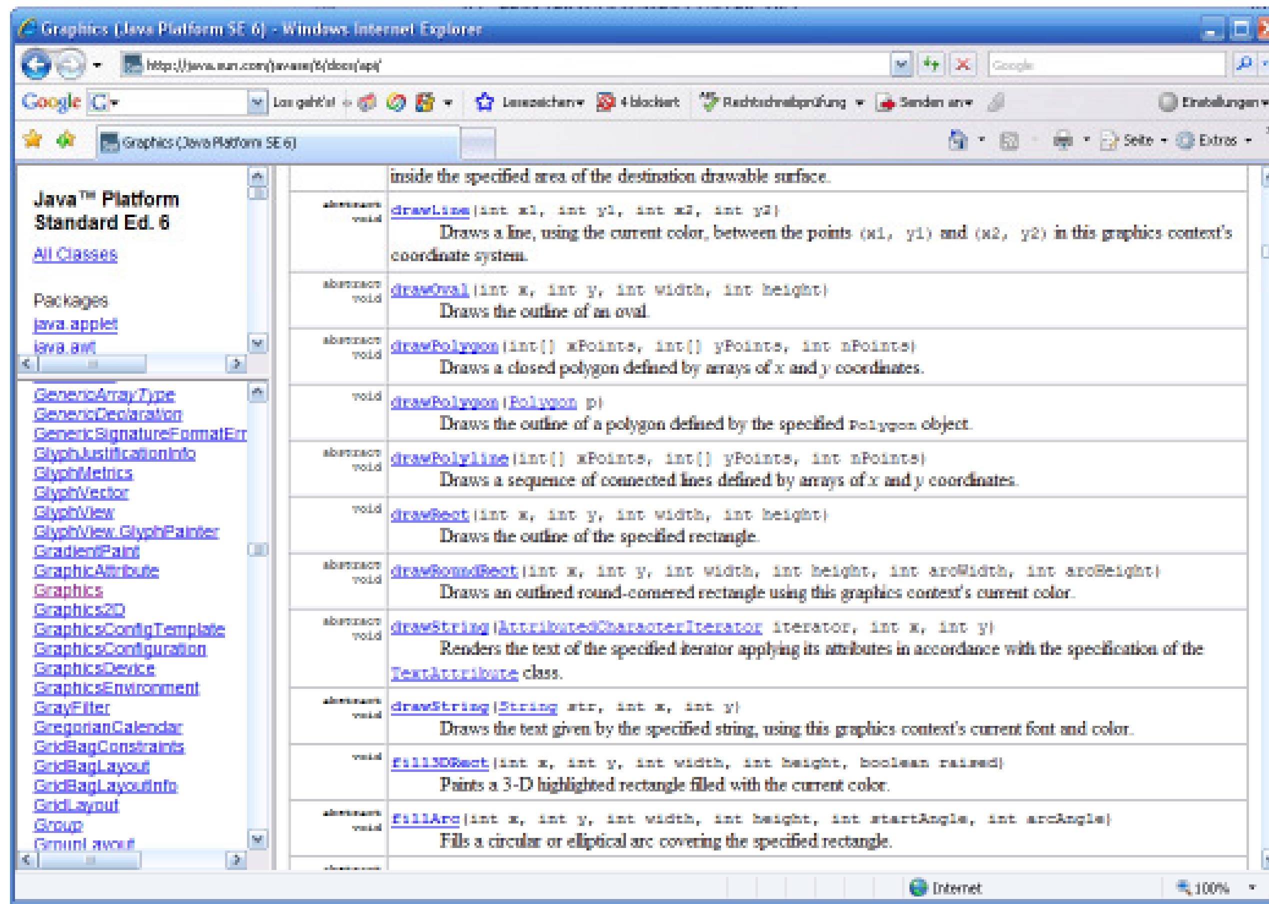
```
import java.awt.*;
public class FensterPositionieren extends Frame {
   public FensterPositionieren(int x, int y){
      setTitle("Ab in die Mitte!");
      setSize(x, y);
      Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
      setLocation((d.width-getSize().width)/2,  (d.height-getSize().height)/2);
      setVisible(true);
   }

   public static void main( String[] args ) {
      FensterPositionieren f = new FensterPositionieren(200, 100);
   }
}
```

Das Fenster wird jetzt mittig auf dem Monitor platziert.

Freie Universität Berlin

## Zeichenfunktionen innerhalb eines Fensters

AWT bietet eine Reihe von Zeichenfunktionen an. Um etwas in einem Fenster anzeigen zu können, müssen wir die Funktion *paint* der Klasse **Frame** überschreiben. Als Parameter sehen wir den Typ **Graphics**. Die Klasse **Graphics** beinhaltet unter anderem alle Zeichenfunktionen. Schauen wir dazu mal in die API:



## Textausgaben

Ein Text wird innerhalb des Fensterbereichs ausgegeben. Zusätzlich zeigt dieses Beispiel, wie sich die Vorder- und Hintergrundfarben über die Funktionen *setBackground* und *setForground* manipulieren lassen.

Die Klasse **Color** bietet bereits vordefinierte Farben und es lassen sich neue Farben, bestehend aus den drei Farbkomponenten rot, blau und grün erzeugen.

```
import java.awt.*;
public class TextFenster extends Frame {
   public TextFenster(String titel) {
      setTitle(titel);
      setSize(500, 100);
      setBackground(Color.lightGray);
      setForeground(Color.blue);
      setVisible(true);
   }

   public void paint(Graphics g){
      g.drawString("Hier steht ein kreativer Text.", 120, 60);
   }

   public static void main(String[] args) {
      TextFenster t = new TextFenster("Text im Fenster");
   }
}
```

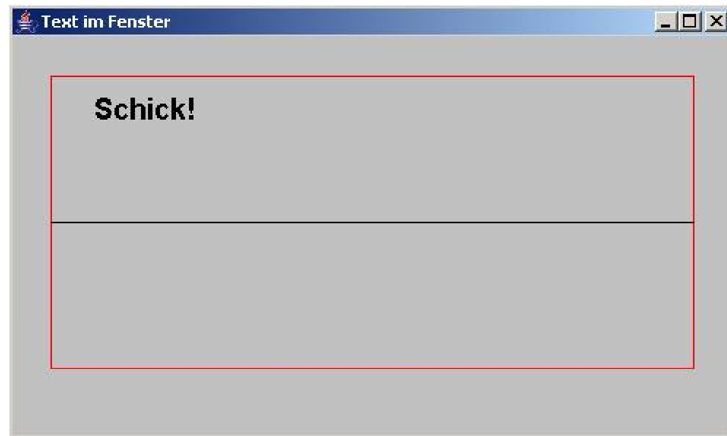
## Zeichenelemente I

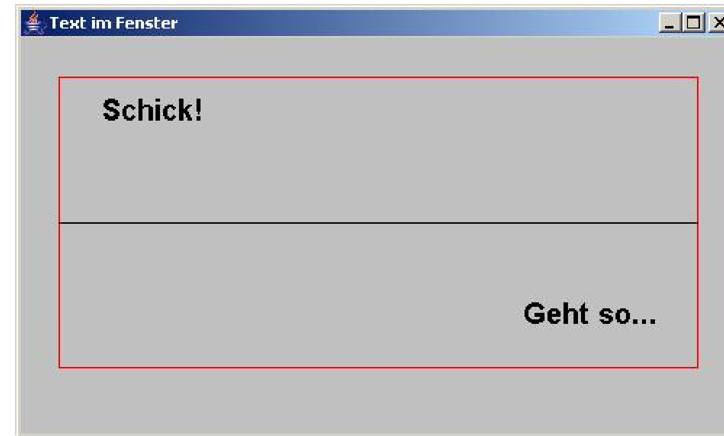
Exemplarisch zeigt dieses Beispiel die Verwendung der Zeichenfunktionen *drawRect* und *drawLine*. Auch hier haben wir eine zusätzliche Funktionalität eingebaut, die Wartefunktion:

```
import java.awt.*;
public class ZeichenElemente extends Frame {
  public ZeichenElemente(String titel) {
      setTitle(titel);
      setSize(500, 300);
      setBackground(Color.lightGray);
      setForeground(Color.red);
      setVisible(true);
  }

  public static void wartemal(long millis){
      try {
        Thread.sleep(millis);
      } catch (InterruptedException e){}
  }

  public void paint( Graphics g ){
      g.drawRect(30,50,440,200);
      g.setColor(Color.black);
      g.drawLine(30,150,470,150);
      g.setFont(new Font("SansSerif", Font.BOLD, 20));
      g.drawString("Schick!", 60, 80);
      wartemal(3000);
      g.drawString("Geht so...", 350, 220);
  }

  public static void main( String[] args ) {
      ZeichenElemente t = new ZeichenElemente("Linien im Fenster");
  }
}
```

## Zeichenelemente II

Liefert folgende Ausgabe:



Freie Universität Berlin

## Die Klasse Color I

Für grafische Benutzeroberflächen sind Farben sehr wichtig. Um das RGB-Farbmodell zu verwenden, erzeugen wir viele farbige Rechtecke, deren 3 Farbkomponenten rot, grün und blau zufällig gesetzt werden. Dazu erzeugen wir ein **Color**-Objekt und setzen dessen Farbwerte.

```
import java.awt.*;
import java.util.Random;

public class FarbBeispiel extends Frame {
   public FarbBeispiel(String titel) {
      setTitle(titel);
      setSize(500, 300);
      setBackground(Color.lightGray);
      setForeground(Color.red);
      setVisible(true);
   }

   public void paint(Graphics g){
      Random r = new Random();
      for (int y=30; y<getHeight()-10; y += 15)
         for (int x=12; x<getWidth()-10; x += 15) {
            g.setColor(new Color(r.nextInt(256),
                                 r.nextInt(256),
                                 r.nextInt(256)));
            g.fillRect(x, y, 10, 10);
            g.setColor(Color.BLACK);
            g.drawRect(x - 1, y - 1, 10, 10);
         }
   }

   public static void main(String[] args) {
      FarbBeispiel t = new FarbBeispiel("Viel Farbe im Fenster");
   }
}
```

## Die Klasse Color II

Wir erhalten kleine farbige Rechtecke und freuen uns auf mehr.

## Bilder laden und anzeigen I

Mit der Methode *drawImage* können wir Bilder anzeigen lassen. In den Zeilen 14 bis 16 geschieht aber leider nicht das, was wir erwarten würden. Die Funktion *getImage* bereitet das Laden des Bildes nur vor. Der eigentliche Ladevorgang erfolgt erst beim Aufruf von *drawImage*. Das hat den Nachteil, dass bei einer Wiederverwendung der Methode *paint* jedes Mal das Bild neu geladen wird.

```
import java.awt.*;
import java.util.Random;

public class BildFenster extends Frame {
  public BildFenster(String titel) {
      setTitle(titel);
      setSize(423, 317);
      setBackground(Color.lightGray);
      setForeground(Color.red);
      setVisible(true);
  }

  public void paint(Graphics g){
      Image pic = Toolkit.getDefaultToolkit().getImage(
                                "C:\\kreidefelsen.jpg");
      g.drawImage(pic, 0, 0, this);
  }

  public static void main( String[] args ) {
      BildFenster t = new BildFenster("Bild im Fenster");
  }
}
```

Jetzt könnten wir sogar schon eine Diashow der letzten Urlaubsbilder realisieren.

Freie Universität Berlin

## Bilder laden und anzeigen II

Kleine Ausgabe:

Um zu verhindern, dass das Bild neu geladen werden soll, können mit Hilfe der Klasse **Mediatracker** die Bilder vor der eigentlichen Anzeige in den Speicher laden.

## Bilder laden und anzeigen III

Eine globale Variable img vom Typ **Image** wird angelegt und im Konstruktor mit dem **Mediatracker** verknüpft. Der **Mediatracker** liest die entsprechenden Bilder ein und speichert sie:

```
// wird dem Konstruktor hinzugefügt
img = getToolkit().getImage("c:\\kreidefelsen.jpg");
Mediatracker mt = new Mediatracker(this);
mt.addImage(img, 0);
try {
   mt.waitForAll();
} catch (InterruptedException e){}
```

Jetzt können wir in der *paint*-Methode mit dem Aufruf

```
g.drawImage(img, 0, 0, this);
```

das Bild aus dem **Mediatracker** laden und anzeigen.

## Auf Fensterereignisse reagieren und sie behandeln I

Als Standardfensterklasse werden wir in den folgenden Abschnitten immer von dieser erben:

```
import java.awt.*;

public class MeinFenster extends Frame {
  public MeinFenster(String titel, int w, int h){
    this.setTitle(titel);
    this.setSize(w, h);

    // zentriere das Fenster
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    this.setLocation((d.width-this.getSize().width)/2,
                     (d.height-this.getSize().height)/2);
  }
}
```

Die Klasse **MeinFenster** erzeugt ein auf dem Bildschirm zentriertes Fenster und kann mit einem Konstruktor und den Attributen titel, breite und hoehe erzeugt werden.

## Auf Fensterereignisse reagieren und sie behandeln II

Unser folgendes Beispiel erbt zunächst von der Klasse **MeinFenster** und implementiert anschließend das Interface **WindowListener**. In Zeile 4 verknüpfen wir unsere Anwendung mit dem Interface **WindowListener** und erreichen damit, dass bei Ereignissen, wie z.B. „schließe Fenster“, die entsprechenden implementierten Methoden aufgerufen werden.

```
import java.awt.*;
import java.awt.event.*;
public class FensterSchliesst extends MeinFenster implements WindowListener {
  public FensterSchliesst(String titel, int w, int h){
         super(titel, w, h);
         addWindowListener(this);  // wir registrieren hier den Ereignistyp für WindowEvents
         setVisible(true);
  }

  public void windowClosing( WindowEvent event ) {
    System.exit(0);
  }
  public void windowClosed( WindowEvent event )      {}
  public void windowDeiconified( WindowEvent event ) {}
  public void windowIconified( WindowEvent event )   {}
  public void windowActivated( WindowEvent event )   {}
  public void windowDeactivated( WindowEvent event ) {}
  public void windowOpened( WindowEvent event )      {}
  // **************************************************************

  public static void main( String[] args ) {
    FensterSchliesst f = new FensterSchliesst("Schliesse mich!", 200, 100);
  }
}
```

## Auf Fenserereignisse reagieren und sie behandeln III

Leider haben wir mit der Implementierung des Interfaces **WindowListener** den Nachteil, dass wir alle Methoden implementieren müssen. Das Programm wird schnell unübersichtlich, wenn wir verschiedene Eventtypen abfangen wollen und für jedes Interface alle Methoden implementieren müssen.

Hilfe verspricht die Klasse **WindowAdapter**, die das Interface **WindowListener** bereits mit leeren Funktionskörpern implementiert hat. Wir können einfach von dieser Klasse erben und eine der Methoden überschreiben. Um die restlichen brauchen wir uns nicht zu kümmern.

```
import java.awt.*;
import java.awt.event.*;
public class FensterSchliesstSchick extends MeinFenster{
  public FensterSchliesstSchick(String titel, int w, int h){
      super(titel, w, h);
      // Wir verwenden eine Klasse, die nur die gewünschten Methoden
      // der Klasse WindowAdapter überschreibt.
      addWindowListener(new WindowClosingAdapter());
      setVisible(true);
  }

  public static void main( String[] args ) {
    FensterSchliesstSchick f = new FensterSchliesstSchick("Schliesse mich!", 200, 100);
  }
}

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

## Auf Fensterereignisse reagieren und sie behandeln IV

Wir können die Klasse **WindowClosingAdapter** auch als innere Klasse deklarieren.

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

public class FensterSchliesstSchick2 extends MeinFenster{
  public FensterSchliesstSchick2(String titel, int w, int h){
         super(titel, w, h);
         addWindowListener(new WindowClosingAdapter());
         setVisible(true);
  }

  //*****************************************************************
  // innere Klasse
  private class WindowClosingAdapter extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
           System.exit(0);
    }
  }
  //*****************************************************************

  public static void main( String[] args ) {
    FensterSchliesstSchick2 f = new FensterSchliesstSchick2("Schliesse mich!", 200, 100);
  }
}
```

## Auf Fensterereignisse reagieren und sie behandeln V

Eine noch kürzere Schreibweise könnten wir erreichen, indem wir die Klasse **WindowAdapter** nur lokal erzeugen und die Funktion überschreiben [Ullenboom 2006, siehe Java-Intensivkurs].

Wir nennen solche Klassen innere, anonyme Klassen.

```
import java.awt.event.*;
public class FensterSchliesstSchickKurz extends MeinFenster{
  public FensterSchliesstSchickKurz(String titel, int w, int h){
    super(titel, w, h);
    // Wir verwenden eine innere anonyme Klasse. Kurz und knapp.
    addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) {
           System.exit(0);
       }
    });
    setVisible(true);
  }

  public static void main( String[] args ) {
    FensterSchliesstSchickKurz f = new FensterSchliesstSchickKurz("Schliesse mich!", 200,
   100);
  }
}
```

## Layoutmanager

Java bietet viele vordefinierte Layoutmanager an. Der einfachste Layoutmanager **FlowLayout** fügt die Elemente, je nach Größe, ähnlich einem Textfluss, links oben beginnend, in das Fenster ein.

Diese lassen sich z.B. innerhalb eines Frames mit der folgenden Zeile aktivieren:

```
setLayout(new FlowLayout());
```

## Die Komponenten Label und Button I

Zwei Button und ein Label werden in die GUI eingebettet. Die Anordnung der Elemente innerhalb des Fensters kann von einem Layoutmanager übernommen werden. Für dieses Beispiel haben wir den Layoutmanager **FlowLayout** verwendet.

```
import java.awt.*;
import java.awt.event.*;
public class GUI_Button extends MeinFenster{
   private Button button1, button2;
   private Label label1;

   // Im Konstruktor erzeugen wir die GUI-Elemente
   public GUI_Button(String titel, int w, int h){
      super(titel, w, h);
      setSize(w, h);
      // Wir registrieren den WindowListener, um auf
      // WindowEvents reagieren zu können
      addWindowListener(new MeinWindowListener());
      // wir bauen einen ActionListener, der nur auf Knopfdruck
      // reagiert
      ActionListener aktion = new Knopfdruck();
      ...
```

## Die Komponenten Label und Button II

weiter geht’s:

```
      setLayout(new FlowLayout());

      button1 = new Button("Linker Knopf");
      add(button1);
      button1.addActionListener(aktion);
      button1.setActionCommand("b1");
      button2 = new Button("Rechter Knopf");
      add(button2);
      button2.addActionListener (aktion);
      button2.setActionCommand("b2");
      label1 = new Label("Ein Label");
      add(label1);
      setVisible(true);
   }

   // Innere Klassen für das Eventmanagement
   class MeinWindowListener extends WindowAdapter{
      public void windowClosing(WindowEvent event){
         System.exit(0);
      }
   }

   class Knopfdruck implements ActionListener{
      public void actionPerformed (ActionEvent e){
         label1.setText(e.getActionCommand());
      }
   }

   public static void main( String[] args ) {
      GUI_Button f = new GUI_Button("Schliesse mich!", 500, 500);
   }
}
```

## Die Komponenten Label und Button III

Als Ausgabe erhalten wir:

Oder wir reagieren individuell:

```
class Knopfdruck implements ActionListener{
   public void actionPerformed (ActionEvent e){
      // wir behandeln die Ereignisse
      String cmd = e.getActionCommand();
      if (cmd.equals("b1"))
         Button1Clicked();
      if (cmd.equals("b2"))
         Button2Clicked();
   }
}
```

Freie Universität Berlin

## Die Komponenten TextField I

Wie wir Eingaben über ein TextField erhalten, zeigt das folgende Beispiel:

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

public class GUI_Button_TextField extends MeinFenster{
  private Button button1;
  private Label label1;
  private TextField textfield1;

  // Im Konstruktor erzeugen wir die GUI-Elemente
  public GUI_Button_TextField(String titel, int w, int h){
    super(titel, w, h);
    setSize(w, h);

    // Wir registrieren den WindowListener, um auf
    // WindowEvents reagieren zu können
    addWindowListener(new MeinWindowListener());

    // wir bauen einen ActionListener, der nur auf Knopfdruck
    // reagiert
    ActionListener aktion = new Knopfdruck();

    setLayout(new FlowLayout());

    textfield1 = new TextField("hier steht schon was", 25);
    add(textfield1);

    button1 = new Button("Knopf");
    add(button1);
    button1.addActionListener(aktion);
    button1.setActionCommand("b1");
    ...
```

Freie Universität Berlin

## Die Komponenten TextField II

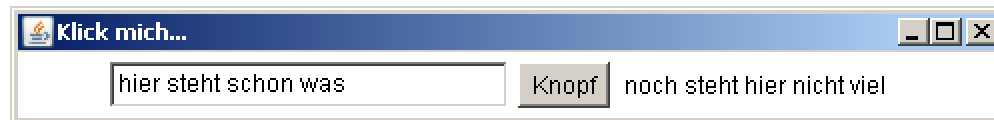
weiter geht's:

```
    ...
    label1 = new Label("noch steht hier nicht viel");
    add(label1);
    setVisible(true);
  }

  private void Button1Clicked(){
          String txt = textfield1.getText();
          label1.setText(txt);
  }

  // Innere Klassen für das Eventmanagement
  class MeinWindowListener extends WindowAdapter{
    public void windowClosing(WindowEvent event){
      System.exit(0);
    }
  }

  class Knopfdruck implements ActionListener{
    public void actionPerformed (ActionEvent e){
        // wir behandeln die Ereignisse
        String cmd = e.getActionCommand();
        if (cmd.equals("b1"))
           Button1Clicked();
    }
  }

  public static void main( String[] args ) {
    GUI_Button_TextField f = new GUI_Button_TextField("Klick mich...", 500, 500);
  }
}
```

Freie Universität Berlin

## Die Komponenten TextField III

Sollte der Button gedrückt werden, so wird der Textinhalt von *textfield1* in *label1* geschrieben.



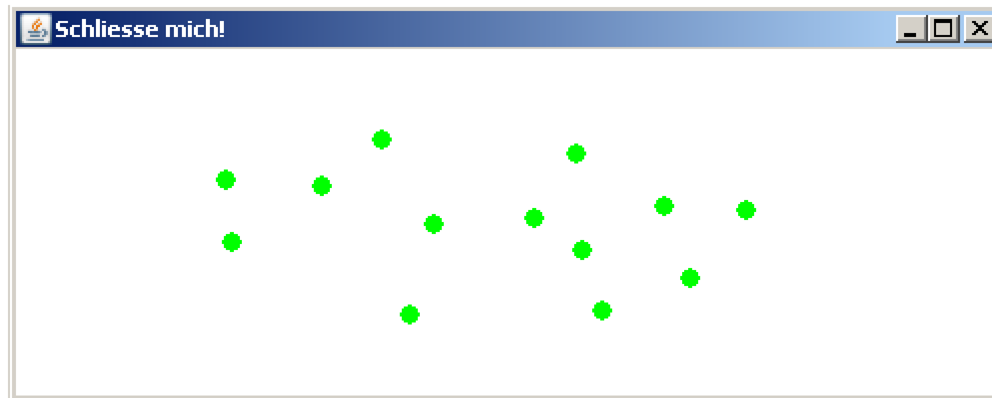
## Auf Mausereignisse reagieren I

Wir haben mit **WindowListener** und **ActionListener** bereit zwei Interaktionsmöglichkeiten kennen gelernt. Es fehlt noch eine wichtige, die Reaktion auf Mausereignisse. Im folgenden Beispiel wollen wir für einen Mausklick innerhalb des Fensters einen grünen Punkt an die entsprechende Stelle zeichnen. Hier wird einfach das Interface **MouseListener** implementiert oder die leeren Methoden der Klasse **MouseAdapter** überschrieben.

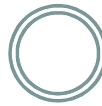
```
import java.awt.*;
import java.awt.event.*;
public class MausKlick extends MeinFenster {
   public MausKlick(String titel, int w, int h){
      super(titel, w, h);
      setSize(w, h);
      // Wir verwenden eine innere anonyme Klasse. Kurz und knapp.
      addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
             System.exit(0);
          }
      });
      addMouseListener(new MouseAdapter() {
          public void mousePressed(MouseEvent e) {
             Graphics g = getGraphics();
             g.setColor(Color.green);
             g.fillOval(e.getX(),e.getY(),10,10);
          }
      });
      setVisible(true);
   }

   public static void main( String[] args ) {
      MausKlick f = new MausKlick("Schliesse mich!", 500, 200);
   }
}
```

## Auf Mausereignisse reagieren II

Liefert nach mehrfachem Klicken mit der Maus, innerhalb des Fensters, folgende Ausgabe:



# Appletprogrammierung



Inhalt:

- Einführung in HTML
- Konstruktion eines Applets
- AppletViewer
- Applikation zu Applet umbauen
- Flackernde Applets vermeiden

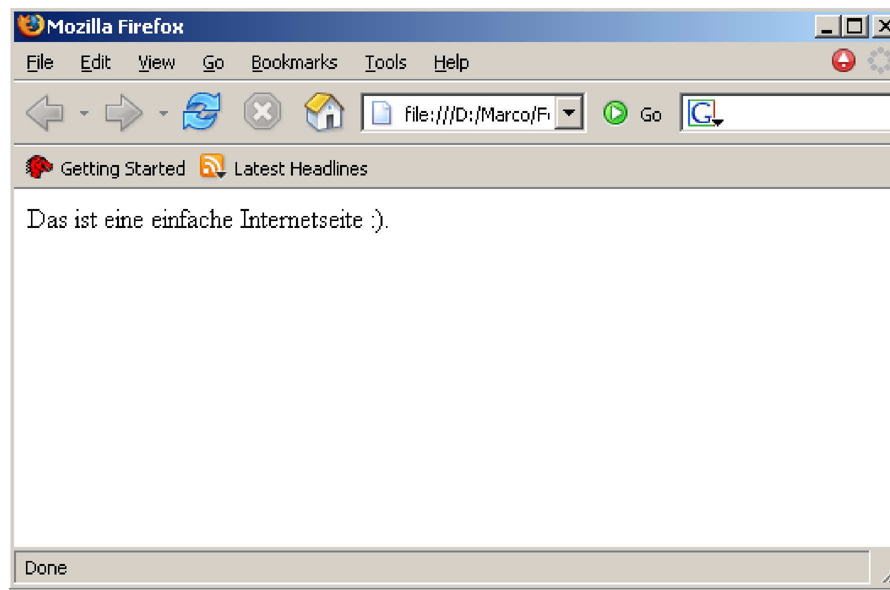
Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*" , Springer-Verlag 2007
Gries D., Gries P.: "*Multimedia Introduction to Programming Using Java*“, Springer-Verlag 2005
Abts D.: „*Grundkurs JAVA: Von den Grundlagen bis zu Datenbank- und Netzanwendungen*“, Vieweg-Verlag 2007

## Kurzeinführung in HTML

HTML (= HyperText Markup Language) ist eine Sprache, mit der sich Internetseiten erstellen lassen. Jeder Internetbrowser kann diese Sprache lesen und die gewünschten Inhalte darstellen. Mit folgenden Zeilen lässt sich eine sehr einfache HTML-Seite erzeugen.

Dazu gibt man die Codezeilen in einem beliebigen Editor ein und speichert sie mit der Endung „.html“.

```
<HTML>
   <BODY>
      Das ist eine einfache Internetseite :).
   </BODY>
</HTML>
```



## Bauen eines kleinen Applets

Wir benötigen für ein Applet keinen Konstruktor. Wir verwenden lediglich die Basisklasse **Applet** und können einige Methoden überschreiben.

init() — Bei der Initialisierung übernimmt die *init*-Methode die gleiche Funktion wie es ein Konstruktor tun würde. Die Methode wird beim Start des Applets als erstes genau einmal aufgerufen. Es können Variablen initialisiert, Parameter von der HTML-Seite übernommen oder Bilder geladen werden.

start() — Die *start*-Methode wird aufgerufen, nachdem die Initialisierung abgeschlossen wurde. Sie kann sogar mehrmals aufgerufen werden.

stop() — Mit *stop* kann das Applet eingefroren werden, bis es mit *start* wieder erweckt wird. Die *stop*-Methode wird z.B. aufgerufen, wenn das HTML-Fenster, indem das Applet gestartet wurde, verlassen wird.

destroy() — Bei Beendigung des Applets, z.B. Schließen des Fensters, wird *destroy* aufgerufen und alle Speicherressourcen wieder freigegeben.

## Verwendung des Appletviewers I

Nach der Installation von Java steht uns das Programm **AppletViewer** zur Verfügung. Wir können die HTML-Seite, in der ein oder mehrere Applets eingebettet sind, aufrufen. Es wird separat für jedes Applet ein Fenster geöffnet und das Applet darin gestartet. Nun lassen sich alle Funktionen quasi per Knopfdruck ausführen und testen.

So starten wir den Appletviewer:

```
appletviewer <html-seite.html>
```

## Verwendung des Appletviewers II

Verwenden wir für das erste Applet folgenden Programmcode und testen das Applet mit dem Appletviewer:

```
import java.awt.*;
import java.applet.*;

public class AppletZyklus extends Applet {
   private int zaehler;
   private String txt;

   public void init(){
      zaehler=0;
      System.out.println("init");
   }

   public void start(){
      txt = "start: "+zaehler;
      System.out.println("start");
   }

   public void stop(){
      zaehler++;
      System.out.println("stop");
   }

   public void destroy(){
      System.out.println("destroy");
   }

   public void paint(Graphics g){
      g.drawString(txt, 20, 20);
   }
}
```

## Verwendung des Appletviewers II

Folgende HTML-Seite erzeugt das Applet. Die beiden Parameter *width* und *height* legen die Größe des Darstellungsbereichs fest.

```
<HTML>
   <BODY>
       <APPLET width=200 height=50 code="AppletZyklus.class"></APPLET>
   </BODY>
</HTML>
```

Wir erhalten nach mehrmaligem Stoppen und wieder Starten folgende Ausgabe auf der Konsole:

```
C:\Applets>appletviewer AppletZyklus.html
init
start
stop
start
stop
start
stop
start
stop
destroy
```

Freie Universität Berlin

## Eine Applikation zum Applet umbauen I

In den meisten Fällen ist es sehr einfach, eine Applikation zu einem Applet umzufunktionieren. Wir erinnern uns an die Klasse **Farbbeispiel** :

```
import java.awt.*;
import java.util.Random;

public class FarbBeispiel extends Frame {
   public FarbBeispiel(String titel) {
      setTitle(titel);
      setSize(500, 300);
      setBackground(Color.lightGray);
      setForeground(Color.red);
      setVisible(true);
   }

   public void paint(Graphics g){
      Random r = new Random();
      for (int y=30; y<getHeight()-10; y += 15)
         for (int x=12; x<getWidth()-10; x += 15) {
            g.setColor(new Color(r.nextInt(256), r.nextInt(256), r.nextInt(256)));
            g.fillRect(x, y, 10, 10);
            g.setColor(Color.BLACK);
            g.drawRect(x - 1, y - 1, 10, 10);
         }
   }

   public static void main(String[] args) {
      FarbBeispiel t = new FarbBeispiel("Viel Farbe im Fenster");
   }
}
```

## Eine Applikation zum Applet umbauen II

### Schritt 1: **Konstruktor zu init**

Da ein Applet keinen Konstruktor benötigt, wird die Klasse innerhalb einer HTML-Seite erzeugt. Wir können an dieser Stelle die Parameter für *width*, *height* und *titel* übergeben und in *init* setzen.

```
<HTML>
   <BODY>
       <APPLET CODE = "FarbBeispielApplet.class" WIDTH=500 HEIGHT=300>
          <PARAM NAME="width"  VALUE=500>
          <PARAM NAME="height" VALUE=300>
          <PARAM NAME="titel"  VALUE="Farbe im Applet">
       </APPLET>
   </BODY>
</HTML>
```

```
import java.awt.*;
import java.applet.*;
import java.util.Random;

public class FarbBeispielApplet extends Applet {
   private int width;
   private int height;
   private String titel;

   public void init() {
      // Parameterübernahme
      width  = Integer.parseInt(getParameter("width"));
      height = Integer.parseInt(getParameter("height"));
      titel  = getParameter("titel");
      setSize(width, height);
      setBackground(Color.lightGray);
      setForeground(Color.red);
   }
   ...
```

## Eine Applikation zum Applet umbauen III

**Schritt 2: paint-Methode anpassen**

Für unser Beispiel können wir den Inhalt der *paint*-Methode komplett übernehmen.

```
   ...
   public void paint(Graphics g){
      Random r = new Random();
      for (int y=30; y<getHeight()-10; y += 15)
         for (int x=12; x<getWidth()-10; x += 15) {
            g.setColor(new Color(r.nextInt(256),
                                 r.nextInt(256),
                                 r.nextInt(256)));
            g.fillRect(x, y, 10, 10);
            g.setColor(Color.BLACK);
            g.drawRect(x - 1, y - 1, 10, 10);
         }
   }
}
```

Das Applet ist fertig, wir können es mit dem Appletviewer starten und erhalten die gleiche Ausgabe.

## TextField-Beispiel zum Applet umbauen I

Im ersten Applet gab es keine Ereignissbehandlung. Aber auch dafür wollen wir uns ein einfaches Beispiel anschauen. Das TextField-Beispiel aus Abschnitt . Nach der Konvertierung zu einem Applet sieht es wie folgt aus:

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

public class GUI_Button_TextField_Applet extends Applet{
  private int width;
  private int height;
  private String titel;

  Button button1;
  Label label1;
  TextField textfield1;

  public void init() {
      // Parameterübergabe
      width  = Integer.parseInt(getParameter("width"));
      height = Integer.parseInt(getParameter("height"));
      titel  = getParameter("titel");
      setSize(width, height);

      // wir bauen einen ActionListener, der nur auf Knopfdruck
      // reagiert
      ActionListener aktion = new Knopfdruck();

      setLayout(new FlowLayout());

      textfield1 = new TextField("hier steht schon was", 25);
      add(textfield1);
      ...
```

## TextField-Beispiel zum Applet umbauen II

weiter geht's:

```
      ...
      button1 = new Button("Knopf");
      add(button1);
      button1.addActionListener(aktion);
      button1.setActionCommand("b1");

      label1 = new Label("noch steht hier nicht viel");
      add(label1);
  }

  private void Button1Clicked(){
      String txt = textfield1.getText();
      label1.setText(txt);
  }

  // ***********************************************************
  // Innere Klassen für das Eventmanagement
  class Knopfdruck implements ActionListener{
    public void actionPerformed (ActionEvent e){
        // wir behandeln die Ereignisse
        String cmd = e.getActionCommand();
        if (cmd.equals("b1"))
           Button1Clicked();
    }
  }
  // ***********************************************************
}
```

Die Ereignissbehandlung verhält sich analog zu den bisher bekannten Applikationen.

Freie Universität Berlin

## Flackernde Applets vermeiden I

Bei den ersten eigenen Applets wird eine unangenehme Eigenschaft auftauchen. Sie flackern. Schauen wir uns dazu einmal folgendes Programm an:

```
<HTML>
   <BODY>
      <APPLET width=472 height=482 code="BildUndStrukturFlackern.class">
      </APPLET>
   </BODY>
</HTML>
```

Flackerndes Applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import java.lang.Math;
import java.util.Random;

/* Bild als Hintergrund und mausempfindliche Linienstruktur im
   Vordergrund erzeugt unangenehmes flackern */
public class BildUndStrukturFlackern extends Applet {
   int width, height, mx, my, counter=0;
   final int N=100;
   Point[] points;
   Image img;
   Random r;

   ...
```

Freie Universität Berlin

## Flackernde Applets vermeiden II

Weiter geht's:

```
...
public void init() {
   width  = getSize().width;
   height = getSize().height;
   setBackground(Color.black);

   // Mauskoordinaten und MouseMotionListener
   addMouseMotionListener(new MouseMotionHelper());

   img = getImage(getDocumentBase(), "apfelmaennchen.jpg");
   r = new Random();
}

// *************************************************************
// Klassenmethoden
private void zeichneLinien(Graphics g){
   for (int i=1; i<N; i++) {
      // wähle zufällige Farbe
      g.setColor(new Color(r.nextInt(256),
                           r.nextInt(256),
                           r.nextInt(256)));

       // verbinde die Punkte
       g.drawLine(mx+(int)((r.nextFloat()-0.2)*(width /2)),
                  my+(int)((r.nextFloat()-0.2)*(height/2)),
                  mx+(int)((r.nextFloat()-0.2)*(width /2)),
                  my+(int)((r.nextFloat()-0.2)*(height/2)));
   }
}
...
```

## Flackernde Applets vermeiden III

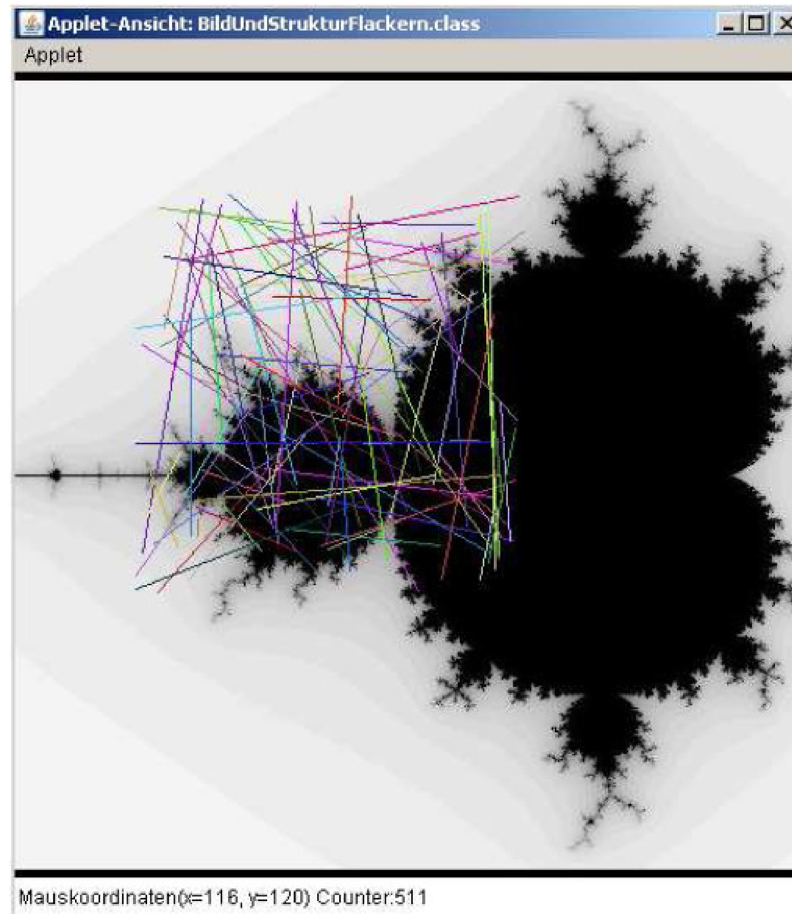
Weiter geht's:

```
   ...
   private void zeichneBild(Graphics g){
      g.drawImage(img, 0, 5, this);
   }
   // *************************************************************

   // *************************************************************
   private class MouseMotionHelper extends MouseMotionAdapter{
      // Maus wird innerhalb der Appletflaeche bewegt
      public void mouseMoved(MouseEvent e) {
         mx = e.getX();
         my = e.getY();
         showStatus("Mauskoordinaten(x="+mx+", y="+my+")
                    Counter:"+counter);
         // ruft die paint-Methode auf
         repaint();
         e.consume();
      }
   }
   // *************************************************************

   public void paint(Graphics g) {
      zeichneBild(g);
      zeichneLinien(g);
      counter++;
   }
}
```

Freie Universität Berlin

## Flackernde Applets vermeiden IV

Bei der Bewegung der Maus über das Apfelmännchen werden zufällige Linien mit zufälligen Farben erzeugt. Die Darstellung ist relativ rechenintensiv und das Applet beginnt zu Flackern. Nebenbei erfahren wir in diesem Beispiel, wie wir dem Browser mit der Methode *showStatus* eine Ausgabe geben können.



# Appletprogrammierung

Freie Universität Berlin

## Ghosttechnik I

Unsere erste Idee an dieser Stelle könnte sein, die Darstellung in der Art zu beschleunigen, dass wir zunächst auf einem unsichtbaren Bild arbeiten und dieses anschließend neu zeichnen.

Diese Technik könnten wir als Ghosttechnik bezeichnen, da wir auf einem unsichtbaren Bild arbeiten und es anschließend wie von Geisterhand anzeigen lassen.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import java.lang.Math;
import java.util.Random;

/* Bild als Hintergrund und mausempfindliche Linienstruktur im
   Vordergrund erzeugt unangenehmes Flackern, trotz Ghostimage!
*/
public class BildUndStrukturFlackernGhost extends Applet {
   int width, height, mx, my, counter=0;
   final int N=100;
   Point[] points;
   Image img;

   // +++++++++++++++++++
   Image img_ghost;
   Graphics g_ghost;
   // +++++++++++++++++++

   Random r;
   ...
```

## Ghosttechnik II

weiter geht's:

```
...
public void init() {
   width  = getSize().width;
   height = getSize().height;
   setBackground(Color.black);
   // Mauskoordinaten und MouseMotionListener
   addMouseMotionListener(new MouseMotionHelper());
   img = getImage(getDocumentBase(), "fractalkohl.jpg");

   // +++++++++++++++++++
   img_ghost = createImage(width, height);
   g_ghost   = img_ghost.getGraphics();
   // +++++++++++++++++++

   r = new Random();
}

// *************************************************************
// Klassenmethoden
private void zeichneLinien(Graphics g){
   for (int i=1; i<N; i++) {
     // wähle zufällige Farbe
     g.setColor(new Color(r.nextInt(256), r.nextInt(256), r.nextInt(256)));

     // verbinde N zufällig erzeugte Punkte
     g.drawLine(mx+(int)((r.nextFloat()-0.2)*(width /2)),
                my+(int)((r.nextFloat()-0.2)*(height/2)),
                mx+(int)((r.nextFloat()-0.2)*(width /2)),
                my+(int)((r.nextFloat()-0.2)*(height/2)));
   }
}
...
```

## Ghosttechnik III

weiter geht's:

```
    ...
    private void zeichneBild(Graphics g){
       g.drawImage(img, 0, 0, this);
    }

    private class MouseMotionHelper extends MouseMotionAdapter{
       // Maus wird innerhalb der Appletflaeche bewegt
       public void mouseMoved(MouseEvent e) {
          mx = e.getX();
          my = e.getY();
          showStatus("Mauskoordinaten(x="+mx+", y="+my+")
                      Counter:"+counter);
          // ruft die paint-Methode auf
          repaint();
          e.consume();
       }
    }

    public void paint(Graphics g) {
       zeichneBild(g_ghost);
       zeichneLinien(g_ghost);
       // +++++++++++++++++++
       g.drawImage(img_ghost, 0, 0, this);
       // +++++++++++++++++++
       counter++;
    }
}
```

Wenn wir dieses Applet starten, müssen wir ebenfalls ein Flackern feststellen, es ist sogar ein wenig schlimmer geworden. Im Prinzip stellt diese Technik eine Verbesserung dar. In Kombination mit dem folgenden Tipp, lassen sich rechenintensive Grafikausgaben realisieren.

## Überschreiben der paint-Methode

Der Grund für das permanente Flackern ist der Aufruf der *update*-Funktion. Die *update*-Funktion sorgt dafür, dass zunächst der Hintergrund gelöscht und anschließend die *paint*-Methode aufgerufen wird.

Da wir aber von der Appletklasse erben, können wir diese Methode einfach überschreiben!

```
...
public void update( Graphics g ) {
   zeichneBild(g_ghost);
   zeichneLinien(g_ghost);

   g.drawImage(img_ghost, 0, 0, this);
}

public void paint( Graphics g ) {
   update(g);
   counter++;
}
...
```

Das gestartete Applet zeigt nun kein Flackern mehr.

Freie Universität Berlin

## Beispiel mit mouseDragged I

Abschließend wollen wir uns noch ein Beispiel für die Verwendung der *mouseDragged()*-Methode anschauen. Mit der Maus kann ein kleines Objekt mit der gedrückten linken Maustaste verschoben werden. In diesem Fall wird der kleine Elch im schwedischen Sonnenuntergang platziert.

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

public class MausBild extends Applet {
   int width, height, x, y, mx, my;
   Image img, img2, img_ghost;
   Graphics g_ghost;
   boolean isInBox = false;

   public void init() {
      width  = getSize().width;
      height = getSize().height;
      setBackground(Color.black);

      addMouseListener(new MouseHelper());
      addMouseMotionListener(new MouseMotionHelper());

      img  = getImage(getDocumentBase(), "schweden.jpg");
      img2 = getImage(getDocumentBase(), "elch.jpg");

      img_ghost = createImage(width, height);
      g_ghost   = img_ghost.getGraphics();

      // mittige Position für den Elch zu Beginn
      x = width/2  - 62;
      y = height/2 - 55;
   }
   ...
```

# Appletprogrammierung

Freie Universität Berlin

## Beispiel mit mouseDragged II

weiter geht's:

```
...
private class MouseHelper extends MouseAdapter{
  public void mousePressed(MouseEvent e) {
     mx = e.getX();
     my = e.getY();
     // ist die Maus innerhalb des Elch-Bildes?
     if (x<mx && mx<x+124 && y<my && my<y+111)
        isInBox = true;
     e.consume();
  }

  public void mouseReleased(MouseEvent e) {
     isInBox = false;
     e.consume();
  }
}

private class MouseMotionHelper extends MouseMotionAdapter{
   public void mouseDragged(MouseEvent e) {
      if (isInBox) {
         int new_mx = e.getX();
         int new_my = e.getY();
         // Offset ermitteln
         x += new_mx - mx;             y += new_my - my;
         mx = new_mx;                  my = new_my;
         repaint();
         e.consume();
      }
   }
}
...
```

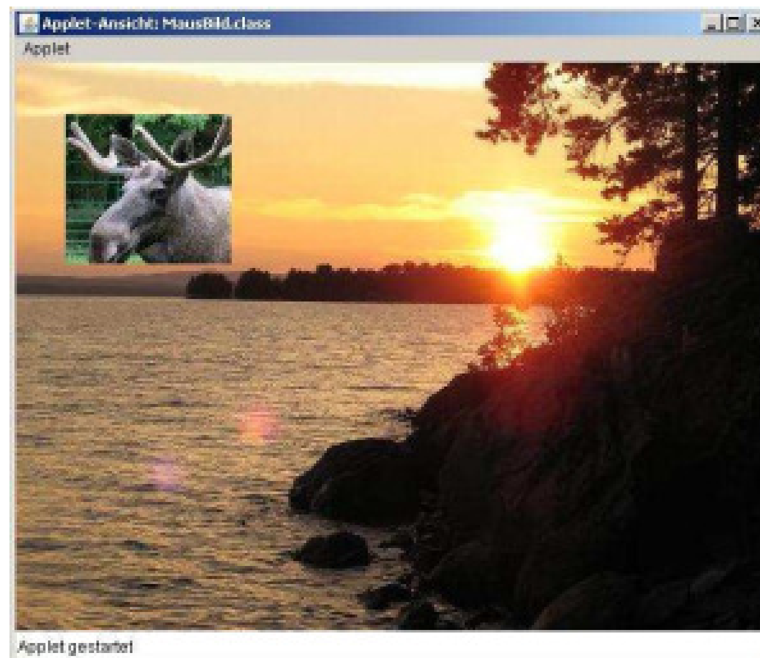
# Appletprogrammierung



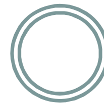
## Beispiel mit mouseDragged III

und fertig:

```
   ...
   public void update( Graphics g ) {
      g_ghost.drawImage(img, 0, 0,  this);
      g_ghost.drawImage(img2, x, y, this);
      g.drawImage(img_ghost, 0, 0,  this);
   }

   public void paint( Graphics g ) {
      update(g);
   }
}
```

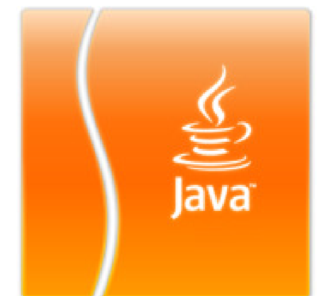


# Appletprogrammierung



Vielen Dank fürs Zuhören!

Das genügt erstmal für heute ...




Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*“, Springer-Verlag 2007