Universität
Osnabrück

VL 18
1

# Informatik B - Objektorientierte Programmierung in Java

## Vorlesung 18:
## Applets





## Inhalt

- Applets (Teil1)
  - Einführung
  - Steuerung und Einbindung in den Browser
  - Attribute
  - Lebenszyklus
  - Die wichtigsten Methoden
  - Applets vs. Applications
  - Beispiele und Demo





## Einführung: Applets

- Wünsche:
  - Dynamische Web-Seiten mit Java-Funktionalität
  - Software-Wartung und -Verteilung
- Lösungsansatz: Java-Applets
  - Einbettung von Java-Programmen in HTML-Seiten
  - Laden dieser Programme beim Betrachten der Seiten von der externen Quelle
    - Programme können zentral gewartet werden, einschließlich der zugehörigen Bibliotheken: z.B. Umstellung auf neue Version kein Problem
    - Browser ist eine bekannte, vertraute Umgebung





## Probleme

- Probleme, die durch die Einbettung entstehen:
  - Sicherheit vor Viren?
  - Vermeidung möglicher Schäden
  - Ladezeit bei umfangreichen Ressourcen
  - "Wer" steuert das Applet?
  - Worauf muss das Applet reagieren?
  - Wie erfolgt die Einbettung in HTML?
  - Wie unterscheidet sich Applet von Application?

# Unterschiede Applet - Applikation

- Das „Hauptprogramm“ eines Applets wird aus der Klasse **`Applet`** bzw. **`JApplet`** abgeleitet.
- Browser erzeugt eine Instanz der Applet-Klasse und ruft die Methoden **`init`** und **`start`** **auf**.
- Aus Sicherheitsgründen: Applet darf – ohne ausdrückliche Erlaubnis - nicht auf Dateien und Programme des lokalen Rechners zugreifen
- Applet arbeitet immer grafik- und ereignisorientiert, also mit AWT bzw. Swing, Ausgaben auf **`System.out`** sind aber möglich.





# Steuerung eines Applets

- Der Browser stellt die Laufzeitumgebung für das Applet bereit. =>
  - Das Applet braucht deshalb keine **`main()`**-Methode besitzen.
  - Die Kontrolle bzgl. Sicherheit muss vom Browser gewährleistet werden.
  - Das Applet muss auf die Darstellung der Browserseite reagieren.
  - Das Applet sollte nicht durch lange Berechnungen die Reaktionsfähigkeit des Browsers beeinflussen: zeitaufwendige Rechnungen gehören in einen besonderen Thread (kommende Vorlesung).





# Beispiel: Applet als Popup-Fenster

- Beispiel: ButtonApplet1
- Vergleiche: JButtonMitListenerAnonym.java

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class JButtonApplet1 extends JApplet
{
  public void start()
  {
      // Anlegen des Fensters
      JFrame einFenster = new JFrame("Schaltflaeche
            mit Nachrichtenverarbeitung")
...
```





# Einbinden des Applets in HTML-Seite

Beispiel: ButtonApplet1.html:

```
<HTML>
<HEAD>
<TITLE>JButtonApplet1</TITLE>
</HEAD>
<BODY>
Applet als Popup.
<CENTER>
<APPLET CODE="JButtonApplet1.class" WIDTH="300"
  HEIGHT="100">
</APPLET>
</CENTER>
</BODY>
</HTML>
```

# Beispiel: eingebettetes Applet

- Beispiel: ButtonApplet2
- Vergleiche: JButtonMitListenerAnonym.java

```
...

public class JButtonApplet2 extends JApplet
{
  public void init()
  {
      setLayout(new GridLayout(2,1));
...
      add(eineSchaltflaeche);
      add(scrollPane);
}
```





# Applet-Attribute in HTML

| | | |
|---|---|---|
| XOR | CODE | Name der zu startenden Applet-Klasse. |
| XOR | OBJECT | Name des zu startenden serialisierten Applets |
| OPTIONAL | WIDTH | Breite (in Pixeln) des Applet-Anzeigebereichs |
| OPTIONAL | HEIGHT | Höhe (in Pixeln) des Applet-Anzeigebereichs |
| OPTIONAL | CODEBASE | Verzeichnis (URL), aus dem das Applet geladen wird |
| OPTIONAL | ARCHIVE | Liste der JAR-Archive, in denen gesucht wird |
| OPTIONAL | ALT | Alternativer Text, falls Applet nicht geladen wird |
| OPTIONAL | NAME | Name dieser Applet-Instanz |
| OPTIONAL | ALIGN | Horizontale Ausrichtung |
| OPTIONAL | VSPACE | Abstand (in Pixeln) über und unter dem Applet |
| OPTIONAL | HSPACE | Abstand (in Pixeln) links und rechts neben dem Applet |





# Einbetten eines Applets in HTML

(fett gedruckt die notwendigen Angaben)

```
<APPLET
   CODEBASE = Applet-URL
   CODE = Dateiname_des_Applet
   WIDTH = Breite_des_Applet
   HEIGHT = Höhe_des_Applet
   NAME = Name_für_Applet
   ALT = alternativer_HTML-Text
   ALIGN = Ausrichtung
   VSPACE = Einrückung_vertikal
   HSPACE = Einrückung_horizontal
   ARCHIVES = JAR-Archiv-Liste
   OBJECT = serializedApplet
>
<PARAM NAME=Parameter VALUE=Wert>
   ...
   alternativer_HTML-Text_für_nicht_JAVA-fähige_Browser
</APPLET>
```





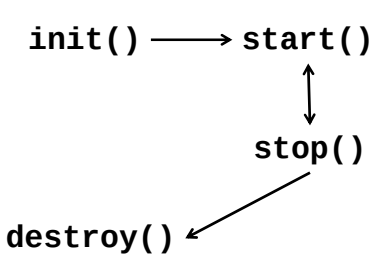
# Lebenszyklus eines Applets

Jedes Applet besitzt fünf Methoden, die seinen Lebenszyklus bestimmen. Der Aufruf dieser Methoden erfolgt bei bestimmten Benutzeraktionen automatisch vom Browser (standardisierte Schnittstelle zwischen Browser und Applet):

| | |
|---|---|
| **init()** | Initialisiere das Applet (anstatt eines Konstruktors). Dies geschieht sofort nach dem Laden. |
| **start()** | Die Seite ist geladen bzw. das Applet wurde wieder auf den Bildschirm gescrollt; wir können jetzt anzeigen. |
| **paint(Graphics g)** | Malt den Inhalt eines Applets neu, wenn das Applet verdeckt wurde. Wird explizit angestoßen von **repaint()**. |
| **stop()** | Der Benutzer verlässt die Seite oder scrollt das Applet vom sichtbaren Bildschirm weg. |
| **destroy()** | Wird aufgerufen, wenn das Applet zerstört wird, z.B., wenn es aus dem Browser entfernt wurde. |

# Lebenszyklus eines Applets (Prinzip)

- Wird eine HTML-Seite mit Applet erstmalig sichtbar, aktiviert der Browser ggf. die Java-Laufzeitumgebung, lädt das Applet und prüft den Bytecode, danach wird die Methode **`init()`** und dann **`start()`** aufgerufen.




Ereignis: Seite wird sichtbar/ neu gezeichnet

Ereignis: Seite wird unsichtbar

Ereignis: Browser wird beendet





# Die wichtigsten Methoden…

- … für Applets sind demnach **`init()`**, **`start()`** und **`stop()`**.
- Klasse **`javax.swing.JApplet`**
- Klasse **`java.applet.Applet`**
- letztere ist abgeleitet von **`java.awt.Panel`**
- Grafik-Eigenschaften von AWT bzw. Swing enthalten.





# Zu `destroy()`

- Die Laufzeitumgebung des Browsers behält die geladene Klasse, bis der Browser beendet wird.
- **Merke:** Um ein geändertes Applet zu testen, muss zunächst der Browser beendet und dann wieder neu gestartet werden!





# Beispiel: EinfachesApplet

```
import java.applet.Applet;
import java.awt.Graphics;
public class EinfachesApplet extends Applet {
  // es soll nur eine Zeichenkette ausgegeben werden
  public void paint( Graphics g ) {
    g.drawString( "Dies ist ein Applet", 25, 50 );
  }
}
```

- Wo sind die Methoden **`init()`**, **`start()`**, **`stop()`**?

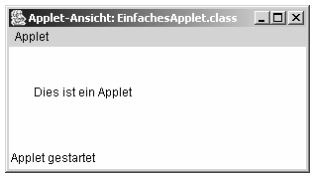
# HTML für EinfachesApplet


```
<HTML>
<HEAD>
<TITLE>EnfachesApplet</TITLE>
</HEAD>
<BODY>
<CENTER>
<APPLET CODE="EinfachesApplet.class" WIDTH="300“
  HEIGHT="100">
</APPLET>
</CENTER>
</BODY>
</HTML>
```





# appletviewer



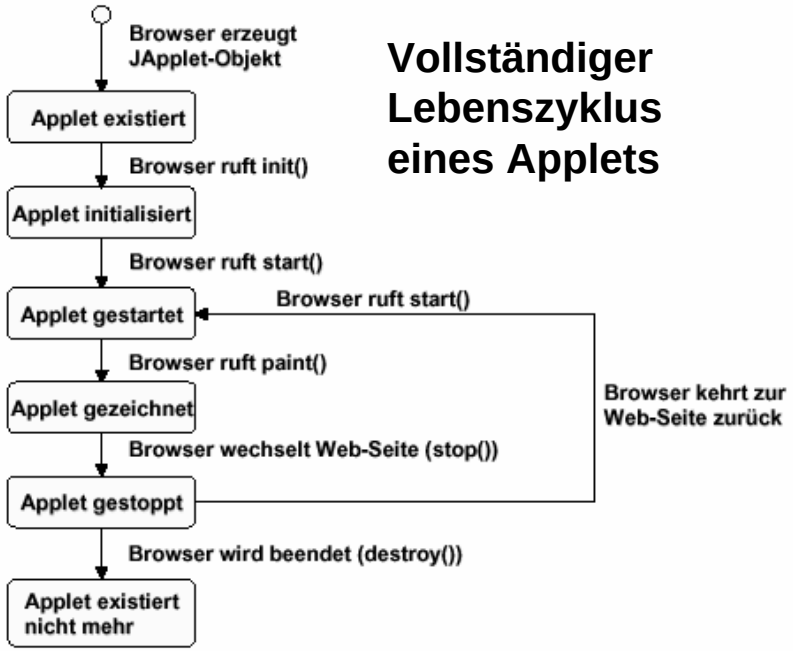



# Weitere Applet-Methoden

- **`paint( Graphics g )`**
  - wird vom Browser jedesmal aufgerufen, wenn die Seite mit dem Applet neu gezeichnet werden muss.



# Vollständiger Lebenszyklus eines Applets



# Demo des Applet-Zyklus

```
import java.applet.Applet;
import java.awt.Graphics;
public class ZyklusDemo
       extends Applet {
  public void init() {
    System.out.println("init");
  }
  public void start() {
    System.out.println("start");
  }
  public void stop() {
    System.out.println("stop");
  }
  public void destroy() {
    System.out.println("destroy");
  }
  public void paint(Graphics g) {
    g.drawString("Willkommen bei Java", 50, 50);
    System.out.println("paint");
  }
}
```






# Applet vs. Application

- Lässt sich eine Anwendung entwickeln, die sowohl als Applet als auch als Application eingesetzt werden kann?
- Eine Klasse muss zumindest sowohl von **`Applet`** (bzw. **`JApplet`**) abgeleitet sein, als auch eine **`main()`**-Methode besitzen, die die Aufgaben des Browsers übernimmt.





# Applet vs. Application (2)

- **Erster Ansatz:**

```
import java.awt.Frame;
public class AppDemo extends java.applet.Applet {
    public void init() {System.out.println("init");}
    public void start(){System.out.println("start");}
    public void stop(){System.out.println("stop");}

    public static void main( String args[] ) {
          Frame f = new Frame();
          AppDemo ad = new AppDemo();
          ad.init();
          f.add( ad );
          f.pack();
          f.show();
          ad.start();
    }
}
```





# Inhalt

- Applets (Teil2)
  - Kommunikation zwischen Applet und Browser
  - Der Applet-Kontext
  - Komprimierte Speicherung: JAR
  - Sicherheit von Applets
  - Zusammenfassung und Defizite

## Kommunikation zwischen Applet und Browser

- Einlesen von Parametern
- Verändern der Statutszeile im Browser
- Anzeigen einer URL
- Kommunikation zwischen Applets





## Kommunikation zwischen Applet und Browser (1)

- Einlesen von Parametern





## Applet-Methode `getParameter(…)`

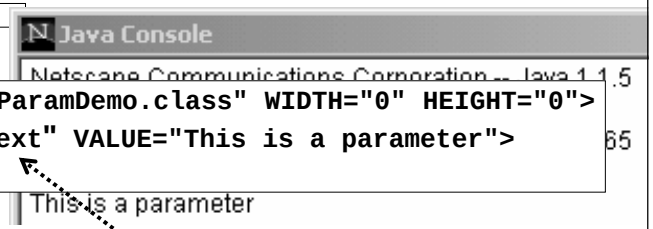
**`String getParameter(String name)`**

ist eine Methode zum Nachrichtentransfer und liefert zum Parameter **`name`**, der im Parameter-Tag der HTML-Seite angegeben ist, den zugehörigen Wert als String. Wird der angegebene Parameter nicht gefunden, wird **`null`** zurückgegeben.





## Einlesen von Parametern ins Applet




```
<APPLET CODE="ParamDemo.class" WIDTH="0" HEIGHT="0">
<PARAM NAME="text" VALUE="This is a parameter">
</APPLET>
```

```
import java.applet.Applet;
public class ParamDemo extends Applet {
  String input;
  public void init() {
    if ((input = getParameter("text")) == null)
      // Festlegen von Alternativ-Text, falls der
      // Parameter nicht angegeben ist
      input = "no suitable parameter specified";
    System.out.println(input);
} }
```

# Kommunikation zwischen Applet und Browser (2)

- Die Statuszeile des Browsers verändern





# Applet-Methode `showStatus(…)`

```
void showStatus(String msg)
```

Mit dieser Methode zum Nachrichtentransfer kann das Applet einen Text in die Statuszeile des Browserfensters schreiben, der das Applet ausführt.






# Verändern der Statuszeile im Browser


```
public class StatusDemo extends
Applet implements MouseListener {

  public void init() {
    // Der Hintergrund des Applets
    // wird 'rot' gesetzt
  }
  public void mouseEntered(MouseEvent e) {
    // Begrüßungsmeldung, wenn die Maus
    // in das Applet hinein bewegt wird
    showStatus("Welcome to this Applet");
  }
  public void mouseExited(MouseEvent e) {
    // Abschiedsmeldung, wenn Maus aus
    // dem Applet heraus bewegt wird
    showStatus("Good bye");
  }
```





# Der Applet-Kontext

- Die folgende Kommunikation zwischen Applet und Browser wird über das Interface **`AppletContext`** abgewickelt, das der Browser bzw. der AppletViewer implementiert
- Unter dem Applet-Kontext versteht Java das Programm, das das aktuelle Applet ausführt. Dies ist in der Regel der Browser, der das Applet geladen hat (oder der AppletViewer).

# Der Applet-Kontext (2)

- In der Klasse **`AppletContext`** gibt es drei Methoden, die dem Applet Funktionalität des Browsers zur Verfügung stellen:
  - **`getApplets`**
  - **`getApplet`**
  - **`showDocument`**





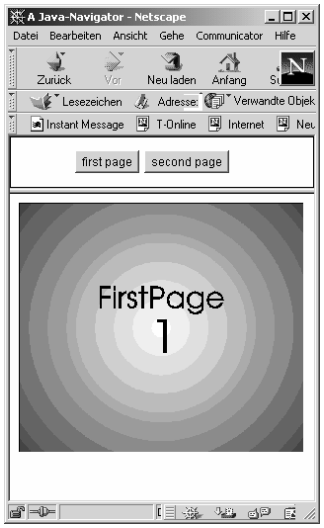
# Interaktion zwischen Applet und Browser (3)

- Eine neue URL im Browser anzeigen





# Anzeigen einer URL




Inhalt der Datei **`Navigator.html`**=

```
<HTML>
<HEAD>
<TITLE>
A Java-Navigator</TITLE>
</HEAD>
<FRAMESET ROWS="15%,80%">
  <FRAME SRC="ShowDocumentDemo.html" NAME="navigator">
  <FRAME SRC="FirstPage.html" NAME="data">
</FRAMESET>
</HTML>
```





# Applet-Methode `getCodeBase()`

**`URL getCodeBase()`**

liefert einen URL des Verzeichnisses, aus dem das Applet gestartet wurde.

**URL-Beispiel:**

**`http://www.inf.fh-dortmund.de/personen/professoren/thiesing/`**

## Applet-Methode showDocument(…)

```
void showDocument(URL url)
void showDocument(URL url, String target)
```

erwartet als Parameter url die Adresse der Seite. Hier muss ein gültiges URL-Objekt angegeben werden, also eine komplette Adresse mit Protokoll, Host-Name und Pfad.

target gibt den Namen des Frames an, in dem die Ausgabe erfolgen soll.





## Applet zum Anzeigen einer URL

```
public class ShowDocumentDemo extends Applet
                            implements ActionListener {
  public void init() { // Buttons erzeugen und hinzufügen
    Button first = new Button("first page"); // usw.
  }
  public void actionPerformed(ActionEvent e) {
    URL page;
    // Ist der Button 'first page' gedrückt?
    if ("first page".equals(e.getActionCommand())) {
      try {
        // Erzeugen der URL und Anzeigen des Dokumentes
        page = new URL(getCodeBase(), "FirstPage.html");
        getAppletContext().showDocument(page, "data");
      }
      catch (MalformedURLException ex) {
        ex.printStackTrace();
      }
    }
    // Ist der Button 'second page' gedrückt? (analog)
  }
}
```





## Interaktion zwischen Applet und Browser (4)

- Nachrichten mit Applets austauschen, die sich auf derselben HTML-Seite befinden.





## Kommunikation der Applets untereinander

Applets, die sich auf derselben HTML-Seite befinden, können direkt untereinander kommunizieren. Diese Kommunikation innerhalb von Java wird durch zwei Methoden des **`AppletContext`** ermöglicht:

- **`Applet getApplet(String name)`**
  Liefert des Applet-Objekt des durch **`name`** bezeichneten Applets. **`name`** ist z.B. der Name, der dem **`<APPLET>`**-Tag als Wert des **`NAME`**-Attributs übergeben wird.
- **`Enumeration getApplets()`**
  Liefert die Aufzählung aller Applet-Objekte, die auf der Seite erreichbar sind. Dies schließt das Applet, das diese Methode aufruft, mit ein.

# Kommunikation zwischen Applets

Zwei Applets: Immer wenn man auf eines klickt, wird im anderen Applet die Zahl der Klicks ausgegeben.


```
<HTML>
<HEAD>
<TITLE>TouchDemo</TITLE>
</HEAD>
<BODY>
<CENTER>
<APPLET CODE="Coffee.class" NAME="Coffee“
  WIDTH="250" HEIGHT="100">
<PARAM NAME="otherApplet“
  VALUE="Espresso">
</APPLET>
<P><HR><P>
<APPLET CODE="Espresso.class“
NAME="Espresso“ WIDTH="250" HEIGHT="100">
<PARAM NAME="otherApplet" VALUE="Coffee">
</APPLET>
</CENTER>
</BODY>
</HTML>
```





Ausschnitt aus Applet **Coffee.java**

```
public void processMouseEvent(MouseEvent e) {
   if (e.getID() == MouseEvent.MOUSE_CLICKED) {
     // Prüfen, ob schon ein Applet gefunden wurde
     if (applet == null)
       // Versuch das andere Applet direkt über seinen
       // Namen anzusprechen
       applet = getAppletContext().getApplet(otherAppletName);
     if (applet == null) {
       // Abfragen aller Applets auf der Seite
       Enumeration applets = getAppletContext().getApplets();
       // Suchen des zu kontaktierenden Applets in der Liste
       System.out.println(applets);
       while(applets.hasMoreElements() && (! appletfound)) {
         applet = (Applet)applets.nextElement();
         if (applet instanceof Espresso)
           appletfound = true;
       }
     }
     else
       appletfound = true;
     // Wenn das Applet gefunden wurde,
     // wird dessen Methode TouchMe aufgerufen.
     if (appletfound)
       ((Espresso)applet).touchMe(appletName);
   }
   super.processMouseEvent(e);
 }
```





# Komprimierte Speicherung: `jar`

- Benötigt ein nicht lokales Applet mehrere Ressourcen, z.B. Bilder oder Sound, dann werden alle Dateien einzeln über das Netz geladen.
- Besser ist es, alle benötigten Dateien komprimiert in *einer* JAR-Datei zu speichern (Java-ARchive).
- **`jar cf demo.jar *.class *.gif *.au`**

  erzeugt die Datei **`demo.jar`**, die alle Klassen und Ressourcen komprimiert enthält.
- Im **`Applet`**-Tag von HTML dient **`ARCHIVE=demo.jar`** dazu, die benötigte(n) **`jar`**-Datei(en) zu laden.





# Gefahrenquellen und Sicherheitslecks mobiler Programme

Das Sandbox-System wurde so konzipiert, dass einem potentiellen Hacker von außen nur sehr wenig Angriffsmöglichkeiten bleiben. Wenn der Empfänger ein Java-Applet nicht bewusst aus seiner Sandbox befreit, kann dieses seinem System prinzipiell auch keinen Schaden zufügen. Der einzige Schwachpunkt des Sandbox-Systems liegt bei der Virtual Machine. Diese muss nämlich die von Java zur Verfügung gestellten Sicherheitsstandards auch erfüllen. Dabei kann es zu kleineren, browserabhängigen Lücken in der Sicherheitsarchitektur kommen.

**Achtung:** Gefahr durch das Übertragen von Daten: Ein Applet kann eingegebene Daten auch übermitteln, ohne Sie vorher zu fragen.

# Applets - Zusammenfassung

- Bestandteile
  - HTML-Parameter-Tags
  - Java Runtime Environment (JRE)
  - Applet Sandbox
- Defizite
  - Sicherheit?
  - Ladezeiten
  - Integration in Oberfläche
  - Unterstützung in unterschiedlichen Browsern





# Applets - Zusammenfassung

- Ziele
  - Einfaches Modell für dynamischen Inhalt von Web-Seite
  - Standardisierte Ablaufumgebung für beliebige (Java-) Programme
  - Mobile Programme
  - Sicherheitskonzept
  - Einfache Schnittstellen