Vorlesungsreihe
Entwicklung webbasierter Anwendungen

EWA - JAVA - Teil 2 - Klassenbibliotheken und GUI

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Anwendungsentwicklung mit Java
  - Objektorientierte Programmierung mit Java
  - Java-Klassenbibliotheken im Überblick
  - GUI-Entwicklung mit Java
  - Java – Sicherheit

Quelle(n) :

[1] Guido Krüger: Go To Java 2, von , 5. Auflage, Handbuch der Java-Programmierung (Online verfügbar !!) 2007

Objektorientierte Programmierung mit Java

- Java ist von Grund auf objektorientiert
- selbst Basisdatentyp wie Strings oder Arrays werden als Objekte behandelt

Alle OO-Konzepte werden unterstützt:

- Abstraktion – Bereitstellung von Basisobjekten für typische Anwendungsaufgaben
- Kapselung – Verbergen von internen Datenstrukturen und Abläufen
- Vererbung - Wiederverwendung von bereits bestehenden Objekten in neuen Klassen
- Polymorphismus – gleichartige Verwaltung verschiedenartiger Objekte
- definierte Schnittstellen – einfache, meist ähnliche Verwendung komplexer Objekte

Klassen und Objekte in Java

- Deklaration von Klassen mit dem Schlüsselwort **class** und **Klassenname**
- innerhalb der Klasse Definition der Klassenvariablen und –methoden
- Speicherklassen definieren Sichtbarkeit (public / private / protected)

```
public class Auto {
  public String name;
  public int erstzulassung;
  public int leistung;

  public int alter()
  { return 2003 - erstzulassung;
} }
```

- bei Einsatz der Klasse Definition einer Variablen vom Klassentyp (=Referenztyp) und Anlage des Objektes mit **new Klasse**
- Verwendung durch Variable und Attributangabe
- Klassenmethoden können auch ohne .-Angabe auf Klassenvariablen zugreifen
- sonst explizit über versteckten Zeiger this.

```
Auto meinKombi;
meinKombi = new Auto();//Anlage
meinKombi.name = "Mercedes";
meinKombi.erstzulassung =1972; meinKombi.leistung = 250;
```

Methoden in Java

- Übergabe und Rückgabe von Parametern erfolgt per generell "by Value"
- bei Objekten und Referenztypen enstpricht die "by value" –Übergabe der Referenz jedoch natürlich einer "by reference"-Übergabe mit allen bekannten Eigenschaften

```
public boolean leistungsvergleich( Auto  auto2)
{  boolean flag=false ;
   if (this.leistung >= auto2.leistung)   flag = true;
   return flag ;  }
```

Überladen von Methoden durch abweichende Parameterlisten :

- unterscheidbare Funktion durch andere Parameterliste
- Compiler generiert **Signatur** der Funktion aus Funktionsname und Parameterliste
- bei Aufruf wird passende Funktion anhand der Signatur gesucht und eingesetzt
  public boolean leistungsvergleich(int minleistung)

Achtung : bei Fehlern wird durch dieses Prinzip i.d.R. kein Type Mismatch o.ä. gemeldet, sondern, daß die Funktion nicht existiert !

Konstruktoren und Destruktoren

- **Definition von Konstruktoren mit Namen = Klassenname**
- zur Initialisierung neuer Objektinstanzen
- in Java als überladbare Funktion mit verschiedenen Parameterlisten ohne Rückgabewert
- Verkettung von Konstruktoren möglich (siehe Beispiele in [1])
  1. Aufruf des Konstruktor der Basisklasse
  2. Initialisierung der Objektinstanzvariablen
  3. Ausführung der eigenen Konstruktorbefehle

Destruktoren in Java

- Aufruf erst bei Freigabe des Speicherfreigabe durch Garbage Collector
- Aufruf nicht sicher und auch in unbekannter Reihenfolge
- daher deutlich geringere Bedeutung als in C++

Vererbung in Java

- Ableitung einer Kindsklasse durch Schlüsselwort **extends** und Hinzufügen von Variablen und Methoden
- Anlage und Verwendung analog
- gleicher Zugriff auf alte und neue Variablen und Methoden
- Attribut final verbietet das Ableiten einer Klasse (dient zum Schutz)

```
class Cabrio extends Auto
{ int verdeckdauer; }
```

```
Cabrio meinAuto;
meinAuto = new Cabrio ();
```

```
meinAuto.name = "Audi";
meinAuto.erstzulassung =2001;
meinAuto.verdeckdauer= 120;
```

- Bei Klassendef. Ohne extends erfolgte implizite Ableitung von Object-Klasse
- Alle Klasse haben daher Object als gemeinsame Superklasse mit den Basismethoden

boolean equals(Object obj)	- zum inhaltsbezogenen Vergleich (statt == !!)
protected Object clone()	- zum Erzeugen einer Kopie
String toString()	- zur Umwandlung des Objekts in einen String
int hashCode()	- Generierung eines Hashschlüssels

Überlagerung von Methoden und Modifier

Methoden in vererbten Klassen können überlagert werden :

- bei gleicher Signatur der Methode verwendet der Compiler die am nächsten gelegene Methode in der Vererbungshierarchie (i.d.R. statisch)
- in speziellen Fällen (z.B. bei Arbeit mit einer universellen Referenzvariable mit wechselnden Zielobjekt) kann auch eine dynamische Entscheidung notwendig sein
  - Relativ aufwendig
  - Kann verboten werden durch Schlüsselworte private / final / static

Allgemeine Bedeutung der Modifier

- **private** - nur in der aktuellen Klasse sichtbar (sonst unsichtbar)
- **protected** - in der aktuellen Klasse und in abgeleiteten Klassen sichtbar (auch in Klassen des gleichen Pakets sichtbar)
- **public** - im Rahmen ihrer Lebensdauer überall sichtbar (In jeder Quelldatei darf nur eine Klasse mit dem Attribut public angelegt werden.)
- **Standard (package scoped)** - nur innerhalb des eigenen Pakets sichtbar
- **static** - existieren vom Laden der Klasse bis zum Beenden des Programms
- **final** – definiert Klasse als unveränderlich (Konstant) – keine Ableitungen und Modifikationen mehr erlaubt (-> bessere Performance und Schutz -> siehe String-Klasse)

Überblick zu weiteren OO-Eigenschaften von JAVA

Interfaces :

- stellen abstrakte Schnittstellen bereit (Details in [1])
- dienen als Ersatz für nicht vorhandene Mehrfachvererbung

Lokale und anonyme Klassen

- dienen zur schnellen Implementierung von relativ kleinen Klassen innerhalb von Klassen (Ausrichtung auf Eventprogrammierung von GUI´s mit kleinen, den Ereignissen zugeordneten Methoden)

Wrapperklassen

- kapseln primitive Datentypen als Klasse (byte -> Klasse Byte)
- sinnvoll zur Verwendung von Objektbezogenen Methoden (z.B. Speicherung oder Netztransport) bei primitiven Datentypen

Design-Patterns

- vorgefertigte, komplexe Entwurfsmuster für typische Aufgabenstellungen, z.B.
  - Iterator: durchwandert Objektsammlungen (->Collections)
  - Observer : beobachtet Daten und meldet Veränderungen z.B. an GUI zwecks Update

Strukturierung von Java-Programmen

Java-Anwendungen setzen sich in aufsteigender Reihenfolge zusammen aus :

- **Anweisungen** y = 2 * x ;
- **Blöcken** { int z; y = y * z; }
- **Methoden** (wie Block nur mit eigenem Namen; Parameter und Rückgabewert)
- **Klassen** (auch mit Subklassen)
- **Pakete** als Sammlung von Klassen für einen bestimmten Zweck (als eine Datei)
  - die Java-Bibliotheken unterteilen sich in verschiedene Pakete (siehe → ...)
  - Gekennzeichnet durch package am Beginn der Quelltextdatei
- **Anwendungen** (= Klasse mit einer Methode main())
  - können eigenständig ausgeführt werden
- **Applets** - sind Anwendungen , welche in einem Browser ausgeführt werden
  - abgeleitet von der Klasse Applets
  - nur lauffähig im Browser oder im Java-Applet-Viewer !

Verwendung von Java-Packets

- Verwendung einer Funktion in externen packets durch vollständige Angabe :

  java.util.Date d = new java.util.Date();

  -> ist relativ aufwendig und nur bei einmaliger Anwendung sinnvoll !
- effizientere Einbindung von packets durch Schlüsselwort import

  import java.util.* ; // lädt alle packets im/unter dem Pfad java/util/

  import java.util.Date; // lädt nur die Klasse Date
- Die Verwendung von * ist in der Regel nicht langsamer, da erst bei Bedarf die Bibliotheken im Pfad gesucht und eingebunden werden
- automatisch wird immer import java.lang.*; eingebunden
- aus Effizienzgründen sind die Basisklassen in einer ZIP-Datei (ohne Komprimierung) zusammengefasst : rt.jar (früher classes.zip)
- Domainnamen-ähnliche (Rückwärts-) Spezifikation von Funktionen :
  - java.* - alle Java-Standardbibliotheken aus dem JDK
  - javax.* - Java-Standarderweiterungen ab JDK 1.2 (meist aber nicht immer verfügb.)
  - com.sun.* org.xml.* - Hersteller- oder Technologiespezifische Zusatzpakete
- Achtung: JDK- und BS-abhängige Probleme bei der Pfaddefinition !!!

Besonders relevante Pakete für Webapplikationen

Sehr nützliche, allgemeine Pakete

- java.util - allgemeine Utilities, Collection-Klassen und Datenstrukturen
- java.io Bildschirm- und Datei-I/O

Netzwerkkommunikation

- java.net -Netzwerkunterstützung
- java.nio - das seit dem JDK 1.4 vorhandene New I/O Package
- java.rmi - Remote Method Invocation (RMI)
- java.security, javax.security.auth - Security-Dienste und Authentifizierung
- javax.crypto - Kryptographische Erweiterungen
- javax.imageio Lesen und Schreiben von Bilddateien

Direkte Anwendungsentwicklung

- java.sql - Datenbankzugriff (JDBC)
- java.applet - Applets
- java.awt - GUI - Abstract Windowing Toolkit inkl. diverser Unterpakete
- javax.swing - das SWING-Toolkit (besser und komplexer als AWT)
- java.beans - Java Beans
- javax.xml Zugriff auf XML-Dateien

Entwicklung grafischer Oberflächen mit Java

- **in Analogie zu anderen Entwicklungsumgebungen können mit Java unterschiedliche Applikationen entwickelt werden :**

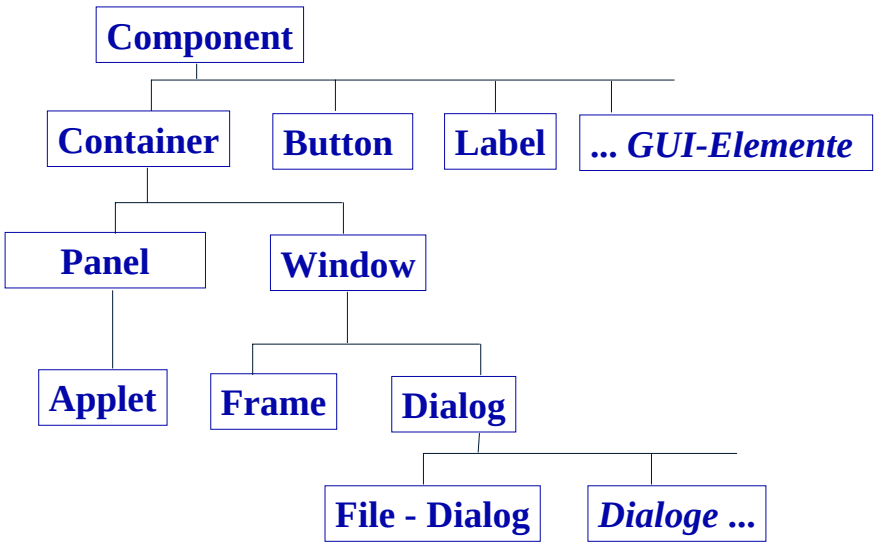
Stand-alone-Anwendungen

- als Ersatz für traditionelle Desktopprogrammierung, viele Entwicklungssysteme (z.B. auch der Jdeveloper von Oracle) sind mit Java programmiert
  - Vorteile: portabel auf Bytecodeebene
  - Nachteile: kein direkter Zugriff auf Hardwareressourcen

Java-Applikationen in Webbrowser eingebunden – Applets

- Entwicklung wie Standalone-Anwendungen, jedoch Ableitung von der Basisklasse Applets
- Einbindung in HTL-Seite
- nach erster Euphorie jetzt Anwendung abgeklungen,
  Hauptgrund : starke Sicherheitseinschränkungen in Browsers behindern breite Anwendung im Web, im Intranet oder als Fat-Client noch sinnvoll

Java – GUI - Fensterhierarchie



Entwicklung grafischer Oberflächen mit Java – Framesbsp.

- **Anlegen, Anzeigen und Schliessen eines einfachen Fensters**

```
import java.awt.*;
public class demo1
{  public static void main(String[] args)
{  Frame frame = new Frame("Frame – Demo ");
     frame.setSize(300,200);
     frame.setVisible(true);
     try {  Thread.sleep(3000);
                  } catch (InterruptedException e) {   //nichts
        }  frame.setVisible(false);
    frame.dispose();  System.exit(0);
} }
```


Das Abstract Windowing Toolkit (AWT)

Ab JDK 1.0 existiert Grafikbibliothek "*Abstract Windowing Toolkit* (*AWT*)" mit:

- grafischen Primitivoperationen zum Zeichnen von Linien oder Füllen von Flächen und zur Ausgabe von Text
- Methoden zur Steuerung des Programmablaufs auf der Basis von Nachrichten für Tastatur-, Maus- und Fensterereignisse
- Dialogelemente zur Kommunikation mit dem Anwender und Funktionen zum portablen Design von Dialogboxen
- Fortgeschrittenere Grafikfunktionen zur Darstellung und Manipulation von Bitmaps und zur Ausgabe von Sound (nach [1])

Bewertung:

- relativ einfach zur Erzeugung portabler Programme mit grafischer Oberfläche
- einige Fehler und Restriktionen, dadurch relativ eingeschränkt
- sehr langsam -> schlechter Ruf von JAVA bzgl. Performance rührt von AWT !
- heute kaum noch verwendet

Die Grafikbibliothek Swing

Seit dem JDK 1.1 ist eine zweite GUI-Bibliothek Swing als Add-on verfübar:

- seit der Version 1.2 fester Bestandteil des JDK
- wesentlich komplexer und schneller als AWT
- die meisten Java-Programme werden heute mit Swing geschrieben
- nachfolgend wird daher nur Swing vorgestellt

Wesentliche Eigenschaften von Swing:

- im Gegensatz zum AWT (welches noch stark auf Betriebssystemroutinen für die GUI setzt (=wenig portabel)) ist Swing weitgehend autonom und realisiert auch das Zeichnen von Buttons und anderen Dialogelementen selbst
- Realisierung durch Lightweight Components ("leichtgewichtige« Komponenten")
- von dieser Basiseigenschaft abgeleitet sind :
  - zur Laufzeit (!!) frei wählbares GUI-Design (Windows / Motif / Metal/ ...)
  - spezieller, künstlich verlangsamter Debug-Grafikmodus zum Testen
  - das Model-View-Controller-Prinzip mit einer Dreiteilung der GUI-Elemente in jeweils ein *Modell* mit den Daten, ein oder mehrere Views (!!) des Dialogelements und ein *Controller* als Verbindungsglied zwischen beiden
- Swing nutzt immer noch viele Konzepte und Funktionen von AWT

GUI-Layoutmanager (für AWT und Swing)

- im Gegensatz zur rasterorientierten Desktopentwicklung müssen Webanwendungen in relativ großen Grenzen an die Browser und Geräte adaptiert werden

In Java-AWT und Swing werden dazu spezielle Layoutmanager verwendet [1]:

- Das FlowLayout ordnet Dialogelemente nebeneinander in einer Zeile an.Wenn keine weiteren Elemente in die Zeile passen, wird nächste Zeile verwendet.
- Das GridLayout ordnet die Dialogelemente in einem rechteckigen Gitter an, dessen Zeilen- und Spaltenzahl beim Erstellen des Layoutmanagers angegeben wird.
- Das BorderLayout verteilt die Dialogelemente nach Vorgabe des Programms auf die vier Randbereiche und den Mittelbereich des Fensters.
- Das CardLayout ist in der Lage, mehrere Unterdialoge in einem Fenster unterzubringen und jeweils einen davon auf Anforderung des Programms anzuzeigen.
- Das GridBagLayout ist ein komplexer Layoutmanager, der die Fähigkeiten von GridLayout erweitert und es ermöglicht, mit Hilfe von Bedingungsobjekten sehr komplexe Layouts zu erzeugen.
- NULL-Layout zur völlig frei definierten Anordnung (ohne Manager)

Die konkrete Reihenfolge der Anordnung hängt von der add-Aufrufsequenz ab.

Über Dialogelement Panel können die Layoutmanager auch geschachtelt werden.

Swing – Dialogelemente I

Die wichtigsten Basis-Swing-Klassen im Überblick (Details in [1]) :

- JFrame , JDialog, JWindow und Japplet zur Erzeugung von Hauptfenstern
- JPanel - zur Definition abgegrenzter Fensterbereiche
- JLabel - Beschriftung , auch mit integriertem Icon
- JTextField - einzeiliges Eingabefeld für Textdaten
- JButton - zur Realisierung von Tasten (Buttons)
- JList und JScrollPane - Listenelement und zusätzliche Komponente zum Scrollen von Listeneinträgen
- JCheckBox und JRadioButton für entsprechende Checkboxen und Buttons
- JComboBox für Pulldownlisten (spart Platz gegenüber Jlist)
- JScrollBar , JSlider und JProgressBar für Schieberegler und Fortschrittsanzeigen
- JMenuBar, Jmenu, JMenuItem für Menüs

Alle diese Dialogelemente haben als gemeinsame Vaterklasse JComponent mit :

- Basismethoden zur Tastaturevents, Fokuseinst., Umrandung, Tooltips usw.
- Doppelpufferung der Bildschirmausgabe (besonders wichtig für Animationen)

Swing – Dialogelemente II

Für komplexe Datenstrukturen stehen entsprechende Dialogelemente bereit:

- JTable für mehrzeilige, mehrspaltige Darstellung von Daten (kein Analog in AWT) mit sehr flexiblem Verhalten und Methoden
- JTree für die Darstellung, Navigation und Bearbeitung baumartiger, hierarchischer Datenstrukturen

Bei beiden Komponenten kommt das **Model-View-Controller-Prinzip** zum Einsatz :

- Datenmodelle liegen im Hintergrund und enthalten die eigentlichen Daten und Basiseinstellungen (Tablemodel / Spalten-Modell bei Tabellen)
- über externe Tabellen oder Methoden können die Daten gesetzt/verwaltet werden
- bei der Anzeige kommen **Renderer** zum Einsatz, welche auch überladen und damit an spezielle Aufgaben angepasst werden können (setDefaultRenderer)
- der Controller wertet Benutzerinteraktionen aus und gibt diese über eine Vielzahl von Ereignissen (events) and die Komponenten weiter (ListSelectionListener , valueChanged , MouseListener , rowAtPoint)
- Durch Ereignisse werden Selektionen aktiviert oder Knoten des Baumes in der Anzeige aktiviert/deaktiviert und können im Datenmodell abgefragt werden.

Anwendungsentwicklung mit Java

Gegenüberstellung Java-Applet

- **Generell Ableitung von Klasse Applet**
- **Start durch Instanzbildung im Web-Browser und Aufruf der Methoden init und start**
- **Kein Zugriff auf Dateien des lokalen Rechners, kein Start externer Programme erlaubt (Ausnahme: signierte Applets)**
- **Applet arbeitet immer grafik- und ereignisorientiert**
- **verfügt durch Vererbung bereits über viele Basis - methoden**

und Java-Applikation

- **Applikation kann von beliebiger Klasse abgeleitet werden**
- **Start durch Aufruf der Klassenmethode main( )**
- **Keine Einschränkungen beim Zugriff und beim Starten externer Programme**
- **Applikation können sowohl grafikorientiert (AWT / SWING) wie auch komplett textorientiert (Ausgaben im DOS-Fenster) arbeiten**

Entwicklung von Applets

Grundstruktur von Applets

```
public class ClickApplet extends Applet {
        Button button;  // GUI-Element anlegen
    public void init() {  // Aufbau der GUI und Enhängen von  Benachrichtigungen
        setLayout(new java.awt.GridLayout(1,0));
        button = new Button("Hier klicken!");
        button.addActionListener(new MyListener());
        add(button);   // Button hinzufügen
        validate();  }   // checke Container und Neujustierung Layout – (nur 1x )
class MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e ) {
        System.out.println(„Button-Klick !");
    } } }
```

Browser ruft applet()-Konstruktor, dann init(), dann start(); (falls vorhanden)

Demos: http://java.sun.com/applets/jdk/1.4/index.html

Einbindung von Applets in HTML

Beispiel zur Einbindung von Java-Applets in Webseiten

```
<HTML><BODY>
    <APPLET CODE=„demo.class" WIDTH=150 HEIGHT=25>
</APPLET></BODY></HTML>
```

Weitere Parameter des des Applett-Tags (width/height müssen gesetzt werden)

- CODEBASE - alternatives Verzeichnis für Klassendateien (optional – default – aktuelles Dokumentenverzeichnis genommen)
- ARCHIVE - Angabe eines JAR-Archivs, aus dem die Klassendateien und sonstigen Ressourcen des Applets geladen werden sollen
- OBJECT - Name einer Datei, die den serialisierten Inhalt des Applets enthält
- ALT - Alternativer Text für solche Browser, die zwar das Applet-Tag verstehen, aber Java nicht unterstützen
- NAME - eindeutiger Name für das Applet (bei mehreren Applets auf Seite)
- ALIGN Vertikale Anordnung { left, right, top, texttop, middle, ... }
- VSPACE Rand über und unter dem Applet
- HSPACE Rand links und rechts vom Applet

Übernahme von Parametern beim Applet-Aufruf

Aufruf des Applets

```
<HTML<BODY>
<APPLET CODE="ParamApplet.class" WIDTH=300 HEIGHT=150>
<PARAM NAME="WELCOME" VALUE="Welcome Message !">
</APPLET>
</BODY></HTML>
```

Abruf der Parameter im Applet

```
  public class ParamApplet extends Applet {
    public void init() {
        String welcomeText = getParameter("WELCOME");
        System.out.println("init  ... ");
        if( welcomeText == null )   System.out.println("No welcome text!");
        else   System.out.println(welcomeText);
}   }
```

Entwicklung von komplexeren Applets

Laufende Animation erfordert neuen Thread :

```
import java.awt.*;
import java.applet.Applet;
import java.util.*;
public class Bubbles extends Applet implements Runnable
{    int x, y;          Random rand;        Color c; int red, green, blue;
     Thread myThread;
     public void init() { rand = new Random(); }
     public void start( ) { myThread = new Thread(this);     myThread.start();    }
     public void stop( ) { myThread.stop(); }
     public void run() {
       while (true) { x =  rand.nextInt() % size().width; x = Math.abs(x);
          y =  rand.nextInt() % size().height; y = Math.abs(y);
          red = rand.nextInt() % 255; red = Math.abs(red);
          Graphics g = this.getGraphics(); c = new Color(red,100,100); // Grafikcontext
                    g.setColor(c);  g.fillOval(x, y, 30, 30);   g.setColor(Color.black);
                    g.drawString("Java", x+5,y+18);
          try {Thread.sleep(5);} catch (InterruptedException e) { }
          } }          }
```


Das Sicherheitskonzept von Java (auch speziell für Java Applets)

Spezielle Sicherheitsmerkmale von Java

- keine direkten Zugriffe auf den Hauptspeicher und keine Pointerarithmetik
- vollautomatisches Memory-Management (damit keine Sicherheitslücken durch (provozierte) Speicherüberläufe
- alle Typkonvertierungen werden zur Laufzeit geprüft
- generelle Bereichsgrenzenprüfung bei Zugriffe auf Array-Elemente und Strings (bei Verletzung wird eine Ausnahme ausgelöst)
- beim Laden des übersetzten Java-Bytecodes über das Netnetz wird dieser von einem Bytecode-Verifier untersucht (Prüfung auf gültige Opcodes, korrekte Sprunganweisungen, Methoden mit korrekten Signaturen, korrekte Typisierung von Variablen, ...)
- Unterscheidung zwischen **lokal und per Netzwerk** geladenen Code :
  - lokaler Code hat Zugriff auf lokale Ressourcen
  - Code aus dem Netzwerk läuft in einer Sandbox (siehe Folgeseite)

Das Sicherheitskonzept von Java – die Java Sandbox

Die Java Sandbox kapselt Netzwerk_Code vom Client-Rechner

Im Standardmodus sind folgende Operationen NICHT ERLAUBT :

- lesender/schreibender Zugriff auf Client-Dateien,
- Öffnen von TCP/IP-Verbindungen zu anderen Servern als dem eigenen Host, von dem das Applet geladen wurde,
- Öffnen von TCP/IP-Verbindungen auf privilegierten Portnummern,
- Lesen benutzerbezogener System-Properties wie »user.name«, »user.home«, »user.dir« oder »java.home«,
- Erzeugen eines Top-Level-Fensters ohne Warnhinweis,
- Ausführen externer Programme,
- Laden von System-Libraries,
- das Beenden der virtuellen Maschine.

- Ab der JDK 1.2 können die Zugriffsbeschränkungen auch für Netzwerkcode schrittweise mit einer Policy-Datei auch ohne Programmänderungen angepasst bzw. gelockert werden.