Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- **Threads**
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

Threads

Oft müssen oder können von der Programmlogik her mehrere Aufgaben (quasi) gleichzeitig bearbeitet werden. Beispiele:

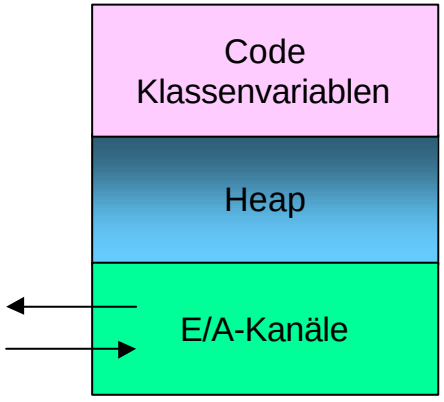
- Reagieren auf Mausbewegungen und Verarbeitung der ausgelösten Aktionen
- Bearbeiten von mehreren unabhängigen Anfragen (z.B. Web-Server)
- Pipeline-Verarbeitung auf vielen Daten (z.B. Stufe 1: Einlesen von Daten, Stufe 2: Filtern der Daten, Stufe 3: Weiterreichen der Daten)

Auf einem Rechner mit nur einem Prozessor kann das Betriebssystem dafür sorgen, dass die Aufgaben quasi-parallel ablaufen (verzahnt). Auf Mehrprozessorrechnern können mehrere CPUs gleichzeitig solche Aufgaben bearbeiten.

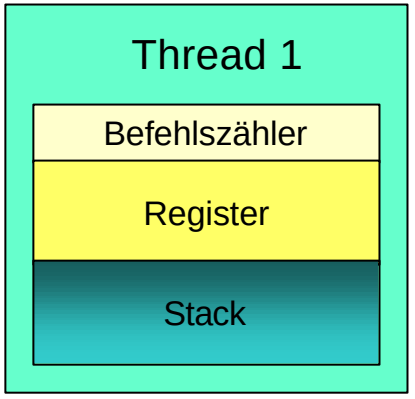
Die organisatorische Einheit für die Ausführung einer Aufgabe ist ein Thread, der eine Zustandsbeschreibung der Ausführung einer Aufgabe enthält.

Ressourcenaufteilung

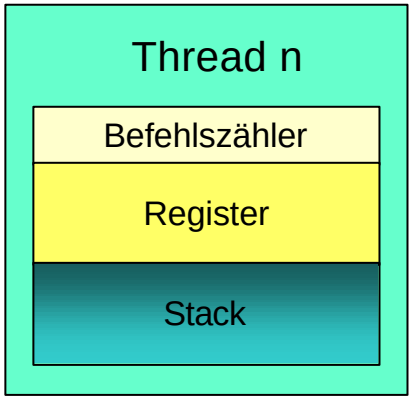
gemeinsam



jeder Thread



...



synchronized, volatile

Parallel ablaufende Threads müssen sich evtl. gegenseitig synchronisieren, um einen koordinierten Zugriff auf gemeinsame Datenbestände zu gewährleisten oder bestimmte Ereignisse abwarten, bevor sie in der Programmausführung fortfahren können.

In Java sind Sprachmittel und Methoden der Klasse Thread vorhanden dies zu gewährleisten, z.B.:

- `synchronized`, zum gegenseitigen Ausschluss
- `volatile`, zur Kennzeichnung von Datenfeldern, die von mehreren Threads modifiziert werden können
- Methoden, die einen Thread in einen anderen Zustand überführen (z.B. von runnable in blocked)

Threads

Es gibt zwei Möglichkeiten zur Programmierung von Threads:

- Ableitung einer Klasse von der Klasse `Thread`
- Übergabe eines Objektes, dessen Klasse die Schnittstelle `Runnable` implementiert, an ein Objekt der Klasse `Thread`

Beispiel zur Ableitung von Thread

```
import java.util.*;

class ThreadZeit extends Thread {
    // wir müssen nur die Methode run() zur Verfügung stellen
    public void run() {
        while(true) {
            // im 2 Sekunden Takt die Sekunden der Uhrzeit ausgeben
            GregorianCalendar d = new GregorianCalendar();
            System.out.println("Thread " + this.getName() + ":"
                               + d.get(Calendar.SECOND));
            try { this.sleep(2000);         // 2 Sekunden schlafen
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }}

public class Uhr {
    public static void main(String[] args) {
        // zwei Threads erzeugen und zeitversetzt starten
        ThreadZeit tz1 = new ThreadZeit();
        ThreadZeit tz2 = new ThreadZeit();
        tz1.start();
        try { Thread.currentThread().sleep(1000); } catch (Exception e) {}
        tz2.start();
    }}
```

Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- **Applets, Sicherheit**
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

WWW

WWW-Seiten werden in der Seitenbeschreibungssprache HTML (Hypertext Markup Language) beschrieben.

Beispiel:

```
<HTML>
<HEAD>
Meine erste WWW-Seite
</HEAD>

<BODY>
Es ist auch ein Link auf eine andere WWW-Seite möglich. Das geschieht
<A HREF="http://java.sun.com"> hier </A> .
</BODY>
</HTML>
```

Siehe WWW-Seiten zur Veranstaltung bzgl. HTML-Dokumentation.

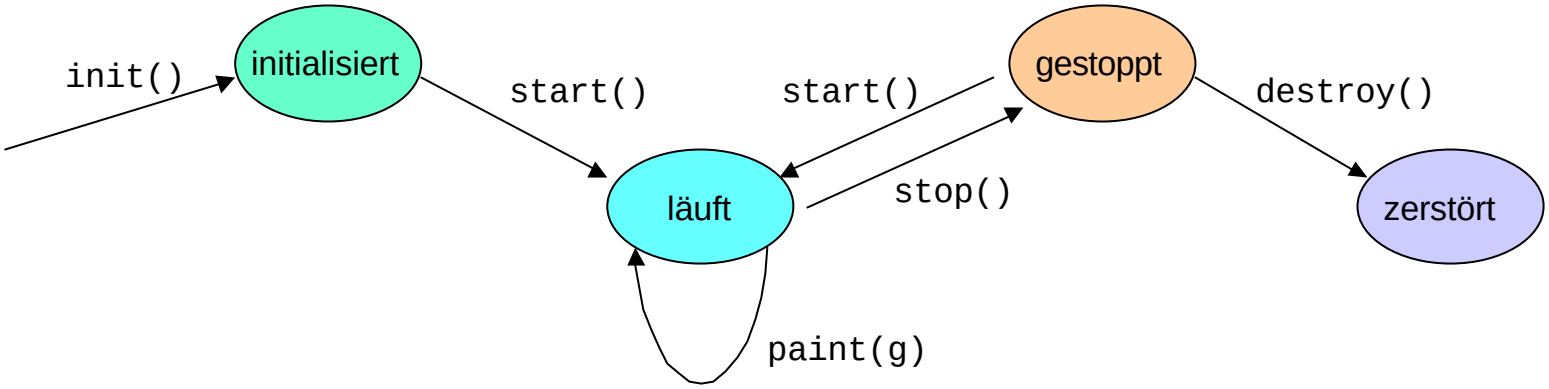
Applets

Applets (`java.applet`) sind keine eigenständigen Programme (Anwendungen), sondern nur in Kombination mit einem Applet-Browser (z.B. Netscape, Internet Explorer) lauffähig. Applets werden in HTML-Seiten eingebunden über das Applet-Tag:

```
<APPLET
        CODE= Klassenname
        WIDTH= Ausgabeweite in Pixeln
        HEIGHT= Ausgabehöhe in Pixeln
        CODEBASE= Klassenpfad relativ zur HTML-Datei oder absolut
        ALIGN= Ausrichtung (TEXTTOP,TOP,ABSMIDDLE,MIDDLE,BASELINE,ABSBOTTOM)
        ALT= alternativer Text für Browser ohne Applet-Fähigkeit
        NAME= Bezeichner für Applet
        HSPACE= rechter und linker Rand in Pixeln
        VSPACE= oberer und unterer Rand in Pixeln
>
<PARAM NAME="Parametername" VALUE="Parameterwert">
</APPLET>
```

Applet-Tag mit CODE,WIDTH,HEIGHT sind obligatorisch ebenso wie /Applet-Tag, der Rest ist optional.

Lebenszyklus eines Applets



- `init()`: Wird ein mal nach dem Erzeugen des Applets (vom Browser) aufgerufen und dient zur Initialisierung.
- `start()`: Nach der Initialisierung und bei jedem erneuten Anzeigen
- `paint(Graphics g)`: Ererbt aus `java.awt.Container`. Wird jedes mal aufgerufen, wenn ein Neuzeichnen erforderlich ist.
- `stop()`: Wenn das Applet in der Anzeige nicht mehr sichtbar ist (z.B. durch Scrollen), wird stop aufgerufen. Man sollte dann z.B. Berechnungen im Applet stoppen.
- `destroy()`: Wird beim Löschen der HTML-Seite aufgerufen.

Sinnvollerweise sollte man mindestens die paint-Methode überschreiben.

Beispiel

wwwseite.html:

```
<HTML>
<BODY>
Hier kommt das Applet:
<APPLET CODE=MeinApplet HEIGHT=200 WIDTH=200>
</APPLET>
</BODY>
</HTML>
```

MeinApplet.java:

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

public class MeinApplet extends Applet {

    // init, start, stop, destroy werden nicht überschrieben

    public void paint(Graphics g) {
        // gibt einen String an der relativen Position (30,40) aus
        g.drawString("Mein tolles Applet", 30, 40);
    }
}
```

Sicherheit

Es gibt eine weiterführende Pflichtveranstaltung zu dieser Thematik. Aus diesem Grund wird dieses Gebiet hier nur kurz angesprochen.

Da über das Netz Klassen geladen werden können, besteht ohne weitere Schutzmechanismen die Möglichkeit, dass diese Klassen z.B.:

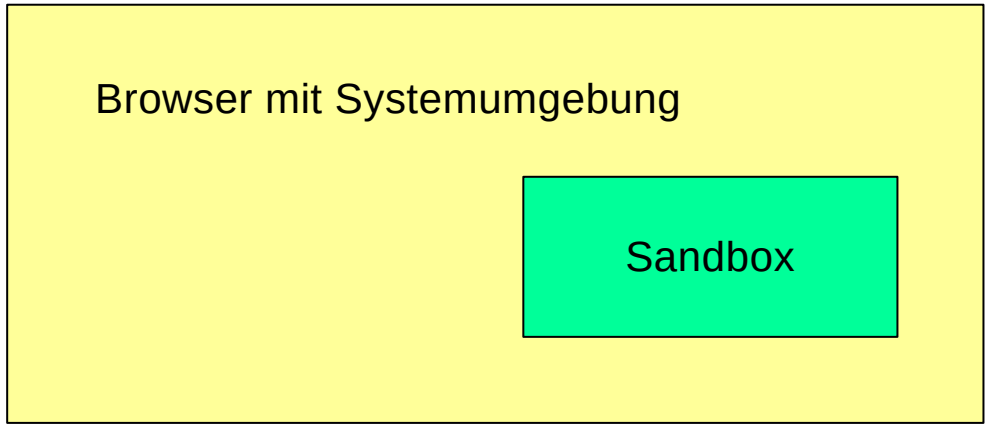
- illegalen Code enthalten
- nicht vom ursprünglichen Autor sind (z.B. bei Übertragung "untergeschoben")
- auf nichterlaubte Ressourcen des Rechners zugreifen (Dateien anlegen oder modifizieren)

Aus diesem Grund gibt es in Java Mechanismen auf verschiedenen Ebenen und Abstufungen, die die Sicherheit gewährleisten sollen.

Sandkasten

Ursprünglich wurde in Browsern das Sandkastenprinzip (sandbox) verwandt: Unsignierte Applets dürfen sich im Sandkasten austoben, aber nicht auf den Rasen. Applets, die in einer Sandbox ablaufen, dürfen u.a. nicht:

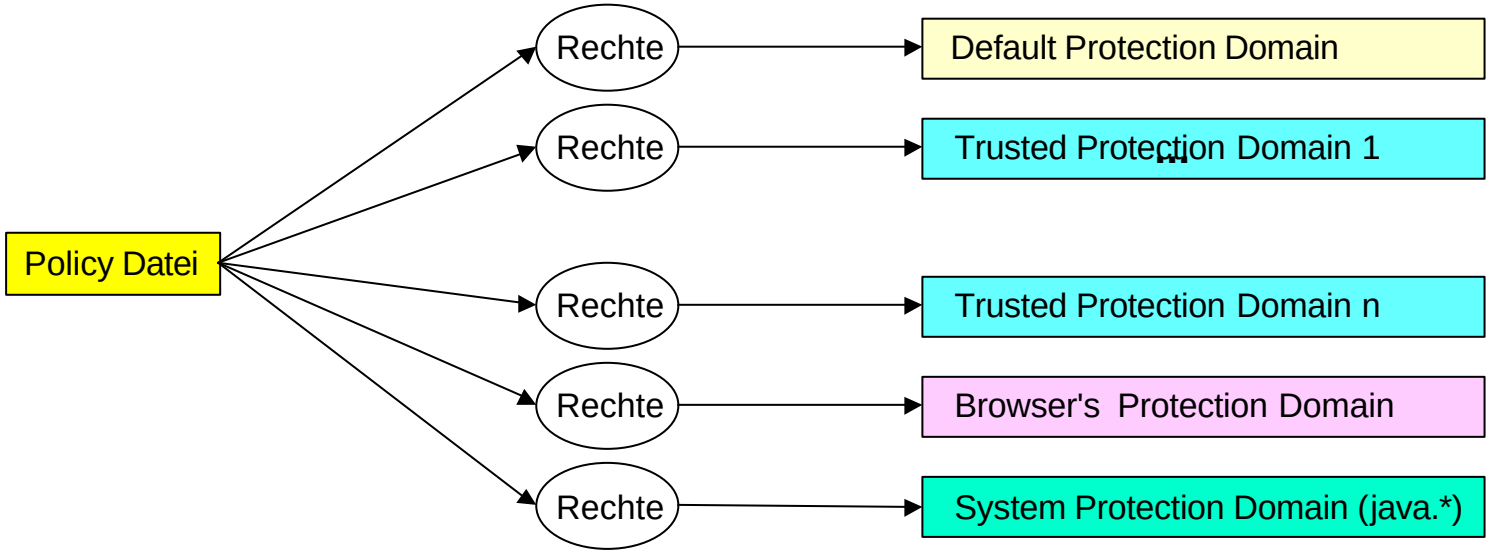
- Operationen auf dem lokalen Dateisystem (z.B. Anlegen neuer Dateien)
- Operationen auf Netzebene (z.B. neue Netzverbindungen öffnen)
- Native Methoden aufrufen
- Systemeigenschaften abfragen
- Programme starten



Policy Domains

Seit JDK 1.2 gibt es ein in das JDK integrierte Sicherheitsmodell basierend auf Protection Domains.

In einer Protection Domain werden Klassen gruppiert, die alle die gleichen Rechte haben. In einer Policy-Datei wird festgelegt, welche Protection Domain welche Rechte hat. Beim Start der JVM wird die Policy Datei (falls vorhanden) gelesen.



Beispiel

java.policy:

```
grant codeBase "http://www.deinserver.de/" {
    permission java.util.PropertyPermission "java.home", "read";
};
grant {
    permission java.net.SocketPermission "*:1024-65535", connect,accept";
    permission java.net.SocketPermission "*:80", "connect";
}
```

In dieser (nicht sehr restriktiven!) Policy-Datei wird erlaubt, dass von Klassen mit der Startadresse `http://www.deinserver.de` die Property "`java.home`" gelesen werden darf.
Weiterhin wird erlaubt, dass alle Anfrager über die Socket-Ports 1024 bis 65535 eine Verbindung zum Programm aufbauen können. Weiterhin ist allen erlaubt, über den Socket-Port 80 (WWW) eine Verbindung aufzubauen.

Starten eines Programmes unter des Obhut eines Java Security Managers:

```
java -Djava.security.manager -Djava.security.policy=java.policy MeinProgramm
```

Über das im JDK enthaltene Programm `policytool` lassen sich Policy-Dateien über eine grafische Benutzeroberfläche erstellen und modifizieren.

jar-Dateien

Zum Laden einer Klasse über ein Netz muss eine Netzwerkverbindung aufgebaut werden. Besteht ein Programm aus vielen Klassen, so kann das Laden relativ lange dauern. U.a. aus diesem Grund lassen sich Klassendateien mit dem JDK-Programm `jar` in einer jar-Dateien archivieren (ähnlich den zip-Dateien in Windows oder den tar.gz-Dateien in Unix).

Packen von Dateien zu einer jar-Datei: Packt alle Dateien mit der Endung .java in eine jar-Datei mit Namen MeineJarDatei.jar

```
jar cvf MeineJarDatei.jar *.java
```

Manuelles Entpacken einer jar-Datei:

```
jar xvf MeineJarDatei.jar
```

Mit dem JDK-Programm `jarsigner` lassen sich weiterhin jar-dateien signieren.

Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- **Grafik**
- Beans
- Integrierte Entwicklungsumgebungen

Foundation Classes

Zur Programmierung grafischer Oberflächen stehen in Java mehrere aufeinander aufbauende, zusammenarbeitende und zum Teil konkurrierende Pakete zur Verfügung, die Java Foundation Classes:

- AWT (Abstract Windowing Toolkit; `java.awt.*`): Einfache Klassenbibliothek zur Programmierung grafischer Oberflächen über Zugriff auf System-Komponenten (z.B. Fenster von MS-Windows). Dadurch uneinheitlich Aussehen auf unterschiedlichen Systemen.

- Swing (`javax.swing.*`): Systemunabhängige von SUN propagierte Klassenbibliothek für grafische Oberflächen

- Java 2D: 2D-Grafik (Linien, Bilder etc.)

- Drag and Drop: Markieren, Kopieren und Einfügen von Informationen über Mausfunktionen

- Accessibility: Einbindung zusätzlicher Eingabegeräte

Alles Weitere in der Veranstaltung von Prof. Heiden!

Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- **Beans**
- Integrierte Entwicklungsumgebungen

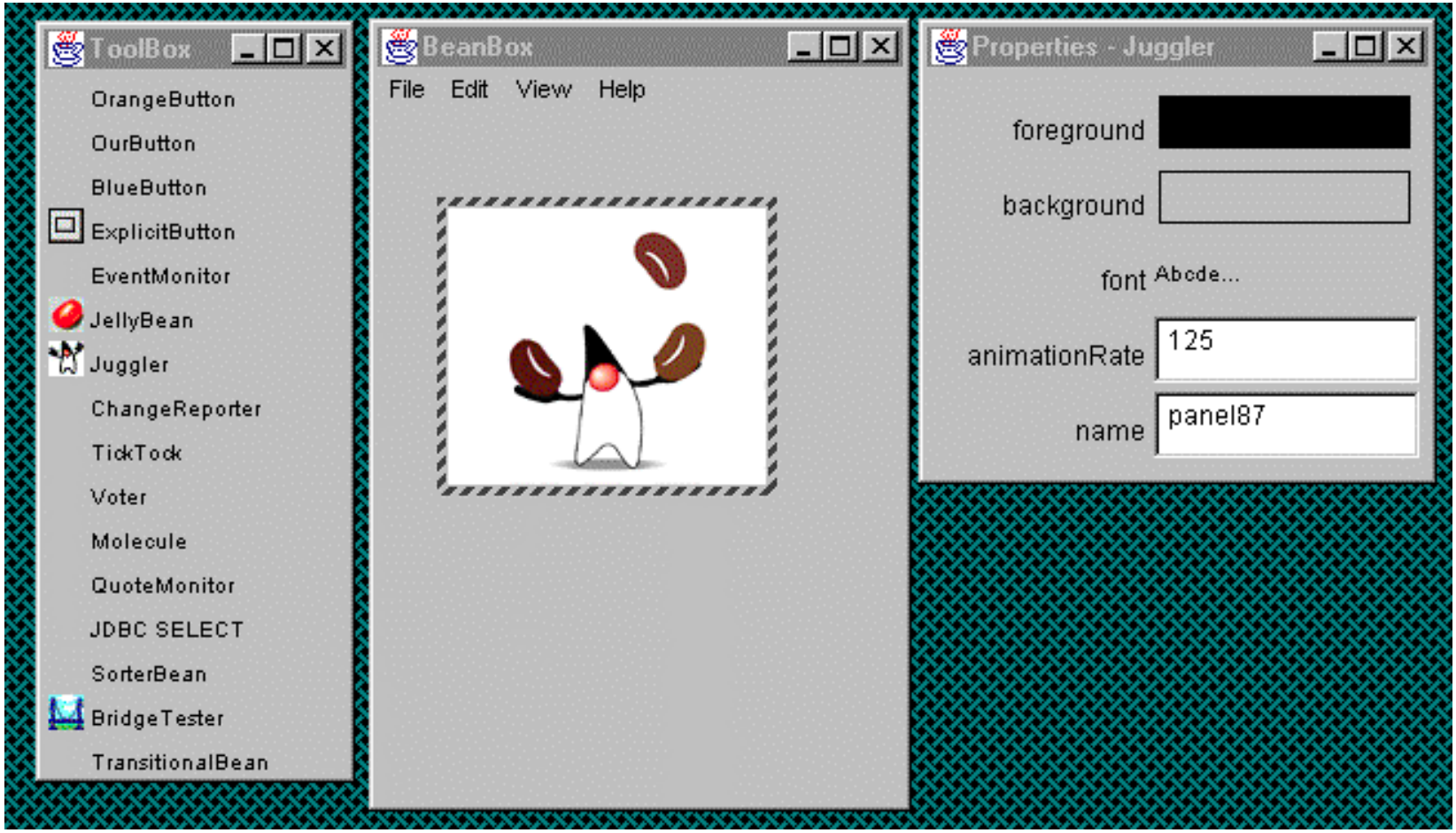
Java Beans

Java Beans sind in Java geschriebene Software-Komponenten:

- Gleiches Entwicklungsmuster für alle Beans
- Eigenschaften erfragbar (u.a. aufgrund standardisierter Namensgebung: `getXYZ(), setXYZ()`)
- Zusammensetzbar
- Nutzung in der Software-Erstellung über grafische Werkzeuge
- Manipulierbarkeit des Erscheinungsbildes und Verhaltens über Properties
- Kommunikation zwischen Beans über Ereignismitteilungen (Events)
- Insbesondere zur Nutzung in grafischen Benutzeroberflächen interessant

SUN stellt ein BDK (Bean Development Kit) zur Verfügung, das u.a. grafische Werkzeuge zum Zusammensetzen von Beans (und einfache Beispiele) enthält.

BeanBox



(aus dem Java Tutorial über Beans)

Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- **Integrierte Entwicklungsumgebungen**

Integrierte Entwicklungsumgebungen

Häufige Aufgaben bei der Entwicklung von Programmen (Codierung) sind z.B.:

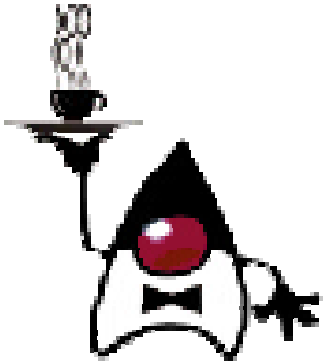
- Editieren
- Übersetzen
- Fehler suchen mit einem Debugger
- Projektverwaltung

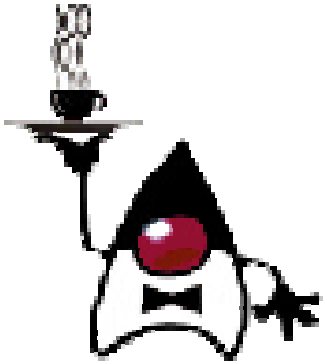
In einer integrierten Entwicklungsumgebung sind mehrere dieser Funktionalitäten (und andere) integriert und unter einer grafischen Oberfläche zur Verfügung gestellt. So kann der Debugger z.B. direkt auf dem durch die Projektverwaltung verwalteten Source-Code zugreifen und im Editor-Fenster darstellen.

Der Startaufwand zur sinnvollen Nutzung dieser Umgebungen ist relativ groß, jedoch bieten diese Umgebungen nach einer Einarbeitungszeit sehr viel Komfort und können die Entwicklungszeit reduzieren.

Beispiele:
Forte (SUN), JBuilder (Borland), VisualAge (IBM), ...
(siehe WWW-Seite zur Veranstaltung)

The End





Typische Studierende, die

- in der Vorlesung immer aufgepasst und mitgedacht haben,
- alle Übungen eigenständig und sorgfältig gelöst haben,
- über den Tellerrand geschaut haben (z.B. Java API),
- ...
- und die Fachprüfung nach dem 3. Semester erfolgreich abgeschlossen haben!

Viel Erfolg!