

Java und Enterprise Java Beans

- Arnold, Gosling. Java Language Manual. Version 1.1. www.javasoft.com. Privat einmal ausdrückbar.
- JavaSoft. Java Beans Specification. www.javasoft.com. Gut lesbar, schöne Einführung zu den Konzepten
- JavaSoft. Enterprise Java Beans Specification 1.1. www.javasoft.com. Umfassende Erklärung.

Geschichte von Java

- **Ursprung: Eingebettete Systeme für Hausgeräte**
 - Neue Sprache an C++ angelehnt
 - Reduktion der Komplexität
 - Exakt und sauber spezifiziert
- **Verbreitung durch Java-Applets in Netscape (1994)**
 - Java benutzt das Web als Zugpferd, so wie C UNIX
 - Write once, run anywhere (WORA) durch Virtual Machine
 - Frei nutzbar, doch feste Schnittstellen (Lizenzvertrag)
- **Klassenbibliotheken:**
 - AWT, Swing (GUI)
 - Java Beans (1997)
 - Enterprise Java Beans (1998/99)

Spracheigenschaften

- Objektorientiert
- Mehrfachvererbung von Schnittstellen
- Einfachvererbung von Code
- Automatische Speicherbereinigung
- Keine Zeiger, nur Referenzen
 - Keine Funktionsparameter
 - Callbacks durch Hilfsklassen möglich
- Synchronisation mit kritischen Abschnitten auf Methoden
- Einfaches, hierarchisches Paketkonzept
 - Entspricht Namensräumen
- Keine Generizität
- Extraktion von Dokumentation (javadoc)

Laufzeitumgebung

- Java Virtual Machine

- Abstraktion von konkreter Maschine und Plattform
- prinzipiell sprachunabhängig
- Übersetzer für andere Sprachen, z.B. ECMAScript

- Java Bytecode

- Stapelorientiert (für Interpretation durch Java Virtual Machine)
- Plattformunabhängig
- Volle Typinformation
- Sicherheit durch Bytecode Verifier prüfbar

Klassen und Dateien

- Einbettung in Dateisystem
 - Eine öffentliche Klasse pro Datei
 - Pakete entsprechen Verzeichnissen
 - Auffindung: Suchpfad CLASSPATH
 - Definiert Abbildung von Namen auf Klassen
 - Primitive Registrierdatenbank
 - Pendant zu CORBA interface/implementation repositories
 - Eintragen von Klassen durch Kopieren im Dateisystem
- Java Archive Format (JAR)
 - Im Prinzip ZIP
 - Klassendateien
 - Serialisierte Objekte
 - Dokumentation

Dienste der Bibliothek

- Reflektion (Java Reflection) mit Klassenobjekten `java.lang.Class`
 - Zugriff auf Felder mit `java.lang.Field`
 - Zugriff auf Methoden mit `java.lang.Method`
- Parallelität durch Threads mit `java.lang.Thread`
- Ereignisse mit `EventListener`
- Persistenz durch Serialisierung mit `java.io.ObjectOutputStream`
- Enterprise APIs: Frontend für Dienste aller Anbieter
 - Verteilung mit Java RMI
 - Namen über Java Naming Directory Interface (JNDI)
 - Datenbankzugriff über Java Database Connection (JDBC)
 - Transaktionen mit Java Transaction Server (JTS)
- Makeln mit Java Intelligent Network Infrastructure (JINI)
- Anbindung fremder Sprachen über Java Native Interface (JNI)
- Java Spaces (Tuple space à la LINDA, typisierte Whiteboards)

Kardinalproblem Leistung

- Bytecode ist Kellerarchitektur
 - gut interpretierbar
 - schlecht auf reale Maschinen abzubilden
 - Registerzuteilung
 - Superskalarität
 - Codeauswahl
- Nur primitive Skalare als Wertetypen
 - Objektidentität ist Overhead (Objekte min. 32 Byte)
 - Objekterzeugung ist teuerste Operation
- Sprache teilweise überdefiniert
 - Reihenfolge der Parameterauswertung
 - Reihenfolge von Ausnahmen

Standardisierung

- Java gehört Sun Microsystems
 - Definition von Schnittstellen in einem öffentlichen Prozess
 - Jeder kann Vorschläge einbringen
 - Java Community Process, Java Developer Connection
 - Ziel: Abwenden von Zersplitterung durch Konkurrenten
- Ob Java ein ISO-Standard wird, ist fraglich
- Was passiert, wenn Java den Markt dominiert, aber proprietär bleibt?

Was fehlt von CORBA?

- Plattformunabhängig, doch de facto sprachabhängig
 - Keine IDL
- Fehlende Dienste
 - Relationen
 - strukturierter Speicher
 - Lizenzierung nicht allgemein gelöst
 - Container (keine typisierten generischen Behälter bis auf Array)
- Keine vertikalen facilities
 - Zeitfrage?

- Anschluß an CORBA in EJB vorhanden

JavaBeans

- Erste Komponentenarchitektur für Java (1997)
- Ziel: Visuelle Komposition in Entwicklungsumgebungen
 - insbesondere GUI
 - motiviert durch Borland/Inprise Delphi
- Nicht abgedeckt
 - Verteilung
 - Persistenz
 - Schnittstellen zu Fremdsystemen
- Ansatz:
 - Standardisierung von Schnittstellen
 - Speicherung von Metainformationen

Bean und BeanInfo

■ Bean

- Enthält Funktionalität
- Standardisierte Namensgebung
 - set/getXXX für Attribute
 - add/RemoveXXXListener für Ereignisse
- Unterstützung für Ereignisadapter

■ BeanInfo

- Enthält Metainformation über Attribute, Ereignisse
- Erlaubt Reflektion über Beans, z. B. durch Entwicklungsumgebung

■ Semantikannotation insgesamt schwach

- Fast ausschließlich über Namensgebung
- keine Erweiterung à la Corba Properties
- dafür erneutes Übersetzen nötig

XML Beans

- Idee: Konvergenz von JavaBeans und XML
- IBM Bean Markup Language kodiert BeanInfo

```
<bean>  
  <event> </event>  
  <property> </property>  
</bean>
```
- Auch Komposition, Referenzen auf Beans
- Serialisierung von Java-Objekten nach XML
- BML Compiler komponiert statisch
- Siehe www.alphaworks.ibm.com

Enterprise JavaBeans (EJBs)

- Ziel: Plattform für verteilte (Geschäfts)-Anwendungen
- Transparenz von
 - Verteilung und Parallelität
 - Persistenz und Transaktionen
 - Ressourcenmanagement
- Komposition ohne Rekompilation
 - Erzeugen von Anpassungen
- Offene Anbindung von Komponenten und Werkzeugen
 - durch herstellerunabhängige Schnittstellen (z.B. JNDI, JDBC)
 - durch CORBA Anbindung
- Gemeinsamkeiten mit einfachen JavaBeans
 - Name
 - Namensgebung der Schnittstellen

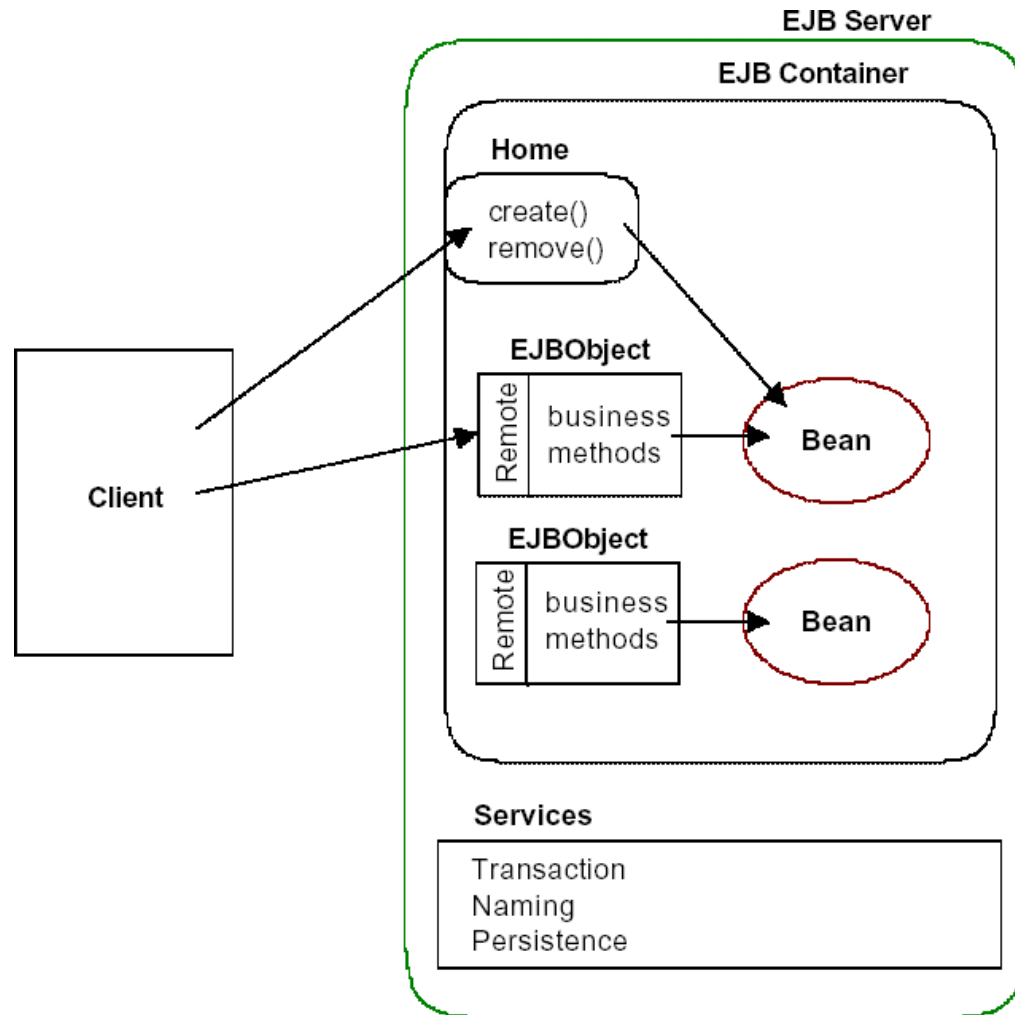
EJBs, Behälter und Deskriptoren

- Komponenten: Klassen, die EJB Interface implementieren
 - Lebenszyklus
 - Geschäftsfunktionalität
- Verwaltung in Behältern (BeanContainer)
 - Auffindung
 - Stellt Ausführungskontext bereit, z.B. Thread
 - Verwaltet Lebenszyklus, Persistenz von Zuständen
 - Sicherheits- und Transaktionsverwaltung
 - Kontrolle aller Außenkontakte
- Beschreibung durch XML Bean Descriptor
 - z.B. Attribute, Geschäftsmethoden
 - Anpassung an Behälter in Deployment Phase

Drei Sorten EJBs

- Zustandslose Session Beans
 - Träger von Geschäftslogik
 - gebunden an eine Benutzersitzung
 - Instanzen einer Klasse ununterscheidbar
- Zustandsbehaftete Session Beans
 - wie oben, nur mit Zustand und damit unterscheidbar
- Entity Beans
 - Träger von Geschäftsdaten
 - persistent
 - entsprechen meist einer Datenbankzeile
 - selbst oder durch Container verwaltet

Beans aus Kundensicht



Die Kundenschnittstelle (client view)

- Invariant bezüglich Server und Container
- Heimatschnittstelle (erbt von `javax.ejb.EJBHome`)
 - Standardisiert
 - Auffinden von Beans mit JNDI
 - Erzeugen, Löschen
 - evtl. globale Identifikatoren mit `getPrimaryKey()`
- Entfernte Schnittstelle (erbt von `javax.ejb.EJBObject`)
 - Definiert durch Anbieter der EJB
 - Bietet Geschäftsfunktionalität
- Metadaten (erbt von `javax.ejb.Metadata`)
 - zur Reflektion über die Bean

Die Container-Komponenten-Schnittstelle

- Entspricht CORBA BOA
- Schnittstellen für Container und Komponente
 - SessionBean-Schnittstellen javax.ejb.SessionBean
 - EntityBean-Schnittstelle javax.ejb.EntityBean
 - Container-verwaltete Persistenz
 - Session-Kontext-Schnittstelle
 - JNDI-Namenskontext
 - Transaktionsverwaltung
- Distribution als EJB JAR
 - Klassenbytecode
 - Deployment-Deskriptor

Deployment-Deskriptor

- EJB Eigenschaften
 - Namen: Name, Klasse, Heimatschnittstellename, Remote-Schnittstellename, Klasse des primären Schlüssels
 - Typ (Session, entity)
 - Zustand
 - Persistenzverwaltung
 - Referenzen auf andere EJB
- Zusatzinformation für Anwendungsersteller
 - Name, Umgebungswerte, Beschreibungsfelder
 - Einbettung in Transaktionen
- Deskriptor ist XML-Dokument nach DD-DTD

Ein Deployment-Deskriptor

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Micro..//EN">
<ejb-jar>
  <description> This is an EJB</description>
  <enterprise-beans>
    <session> <description> Session bean 1 here </description>
      <ejb-name>EmployeeService</ejb-name>
      <home>com-wombat.empl.EmployeeServiceHome</home>
      <remote>com.wombat.empl.EmployeeService</remote>
      <ejb-class>com.wombat.empl.EmployeeServiceBean</ejb-class>
      <ejb-ref><ejb-ref-name>ejb-EmplRecords</ejb-ref-name> ...
      </ejb-ref>
    </session> ...
    <entity>... </entity> ...
  </enterprise-beans>
  <assembly-descriptor>
    <security-role> ....
    <method-permission>...
    <container-transaction>...
  </assembly-descriptor>
</ejb-jar>
```

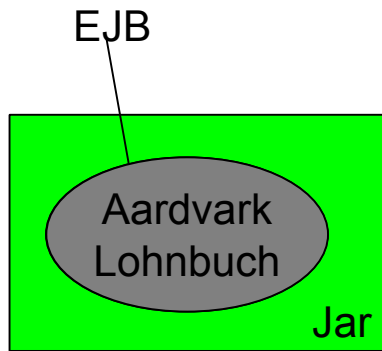
Anpassungen in der Deployment Phase

- Erzeugung von Klassen
 - entfernte Schnittstelle
 - Metadaten
 - Lokale Handles und Klebecode
- Generierte Anpassungen
 - Anpassung Bean / Container
 - analog zu CORBA BOA
 - Einweben von Aspekte aus Bean Descriptor
 - Persistenz
 - Transaktionsverwaltung
 - Komposition

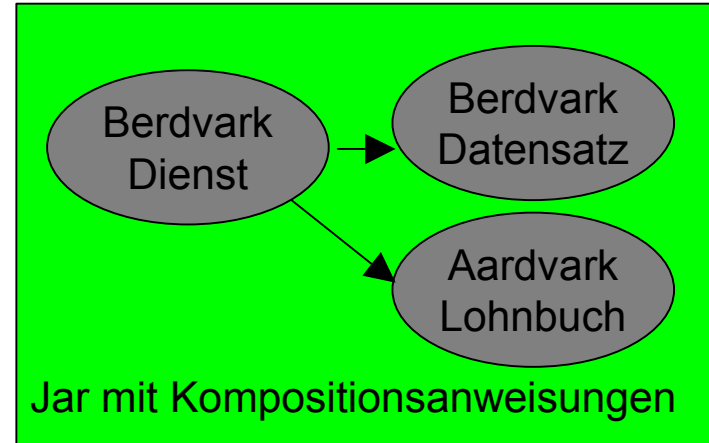
Entwicklerrollen bei EJB

- *Bohnenbauer* (bean provider) ist ein Anwendungsexperte und baut ein EJB-jar mit fachspezifischen Methoden, Deployment-Deskriptor, remote, home interface
- *Anwendungsersteller* (application assembler, Anwendungsexperte) komponiert EJB zu grösseren EJB, d.h. Anwendungseinheiten. Er erweitert dazu die Deployment-Deskriptoren.
- *Anwender* (deployer) setzt die EJB in eine Umgebung ein, die aus einem EJB Server und Container (Adapter) besteht. Damit wird die EJB mit einem EJB-Container verbunden, konfiguriert und einsatzfähig
- *EJB-Server-Hersteller* (server provider) ist ein Spezialist in Transaktionsmanagement und verteilten Systemen und stellt für die Bohnen diese Basisfunktionalität zur Verfügung.
- *EJB-Container-Hersteller* (container provider) liefert mit dem Container Werkzeuge zur Konfiguration und zur Laufzeitinspektion von EJB. Der Container sorgt für die Persistenz von Dauerbohlen, Generierung von Kommunikationscode (Klebecode) zu unterliegenden Datenbanken,

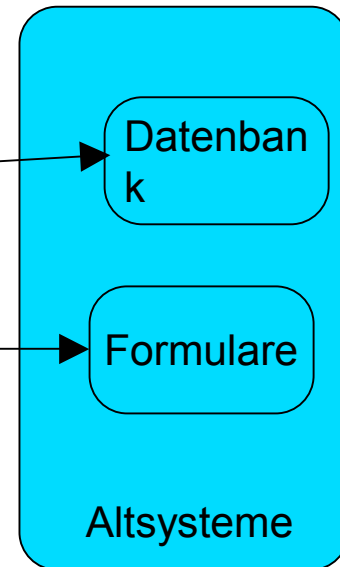
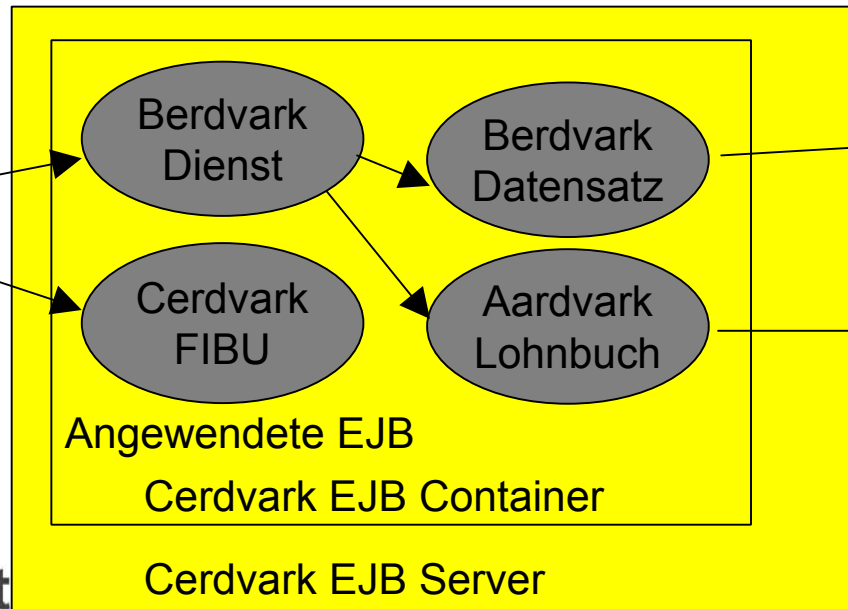
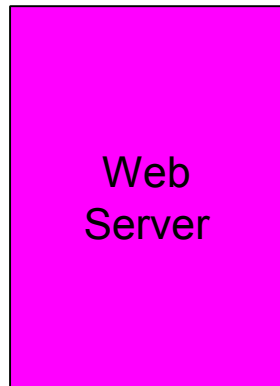
Szenario



Assembling
➔



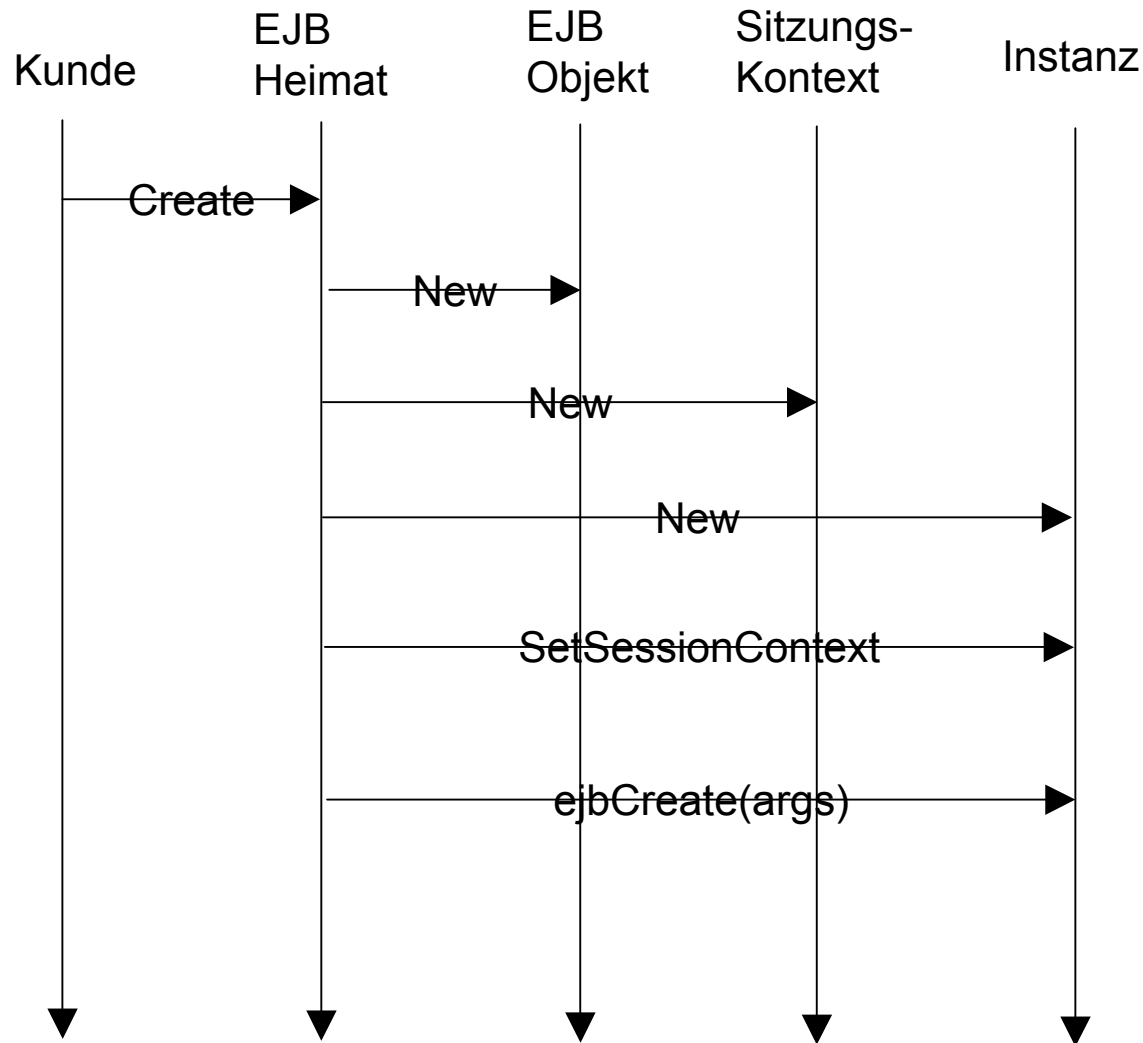
Deploying
➔



Sitzungsbohnen (session beans)

- Transient oder halb-transient, ausgeführt auf Anfrage eines einzelnen Kunden hin
- möglicherweise in Transaktion ausgeführt
- stirbt mit dem Bohnen-Container
- zustandsbehaftet oder auch nicht
- Passivierung bedeutet
 - Serialisierung aller Objekte, die transitiv von nicht-transienten Attributen aus erreichbar sind
- Aktivierung bedeutet Deserialisierung.
- Session-Kontext hält den jeweiligen Zustand fest

Erzeugen von Sitzungsbohnen



Das ist nur eine von sehr vielen Protokollspezifikationen der EJB Standardkomponenten. Die Protokolle definieren zusätzlich zum Typcheck verschärfte Zusammensetzbarkeitskriterien.

Dauerbohnen (entity beans)

- persistent
- mehrbenutzerfähig, transaktionsorientiert
- globaler Identifikator (analog GUID, IOR)
- ähnlich zu Monikers und Corba-Objekten

Szenario

```
Context initialContext = new InitialContext();
CartHome cartHome = (CartHome)javax.rmi.PortableRemoteObject.narrow(
    initialContext.lookup("applications/mail/freds-carts"),
    CartHome.class);
Cart cart = cartHome.create(...); // EJB spezifisch, vom Bohnenbauer spezifiziert
cart.addItem(66);
cart.addItem(44);

// Speichere die Sitzungsbohne
Handle cartHandle = cart.getHandle();

.. serialisiere auf Datei

Handle cartHandle = .. deserialisiere von Datei
Cart cart = (Cart)javax.rmi.PortableRemoteObject.narrow(
    cartHandle.getEJBObject(), Cart.class);

cart.purchase();
cart.remove();
```

JINI intelligent networks

- JINI ist der Java Maklerdienst (trading service).
 - Verwirklicht dynamischen Aufruf und Dienstvermittlung
 - ähnlich wie in CORBA werden Dienste durch Schnittstellen und Eigenschaften (properties) beschrieben
- Dienste können im Netz gesucht werden durch Makler (lookup service LUS)
 - Wird ein entsprechender Dienst geladen, wird auf dem Kunden ein Stellvertreterobjekt erzeugt, das die Kommunikation mit dem Dienst regelt (analog DCOM)
 - baut auf RMI, JNDI, Migration auf. Klassen werden im Bytecode verschickt
- Leasing ist möglich durch eine Leases-Schnittstelle
- Dienste können in Gruppen organisieren.
 - Ein Drucker kann sich zur Hardware-, Drucker- oder Raum111-Gruppe zuordnen
- Es gibt verschiedene Suchprotokolle

JINI - Generelles Schema

- Anbieten
 - Suchen des Maklers (discovery)
 - Anbieten beim Makler (join)
 - Dienst wird nur eine Weile veröffentlicht (leasing)
- Finden
 - Suche nach Makler (discovery)
 - Suche nach Dienst (lookup)
 - Benutzung (use)
- Also: nichts neues unter der Sonne

Wie die Zukunft aussehen wird

- Java wird ein harter DCOM-Konkurrent werden. Prognose der Gartner Group (Information week 10/99):
 - 75% aller Großunternehmen setzen bereits heute Java ein
 - Microsoft wird es nicht gelingen, Java an den Rand zu drängen.
 - Im Jahr 2002 wird Java die wichtigste Technologie für Netzwerkanwendungen sein (bei mehr als 60% der Unternehmen)
 - 2001 wird die Geschwindigkeit von Java in 95% aller Anwendungen keine Rolle mehr spielen