Vorlesung
Component Ware und Web-Services

- Komponentenkonzepte -

7. Suns webzentrierter Ansatz –
Java und JavaBeans

Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

Inhalt

- Java: Geschichte und Konzepte
- Java und JavaBeans
- Modell
  - Eigenschaften
  - Ereignisse
  - Introspection
  - Anpassbarkeit
  - Auslieferung und Benutzung
- Weiterführende Entwicklungen
  - Enthaltungs- und Diensteprotokoll
  - Langzeit-Speicherung von JavaBeans
  - InfoBus
  - Bean MarkUp Language
  - Marktplätze für JavaBeans

Java: Geschichte und Konzepte

Java: Geschichte und Konzepte

- große Erfolgsstory
  - 1995/96 erste Anfänge (Sun Microsystems)
  - heute (2003) eines der am häufigsten gebrauchten Schlagworte
  - objektorientierte Programmiersprache, aber Magnet war Konzept von Applets, Mini-Applikationen und Funktionalität innerhalb von Webseiten

- 2 Ansätze, womit Java wirklich zur Killerapplikation wurde
  - **Sicherheitskonzept**
    - Applet wird doppelt geprüft (Übersetzungszeit und Ladezeit)
    - strenge Sicherheitsregeln (security policies), die mit keiner anderen Programmiersprache erreicht werden
    - Sicherheit auch im compilierten Code eines JIT-Compilers
    - Sicherheitsaussagen durch Erzeugerzertifikate möglich
      - „signed applets“ (unter Nutzerkontrolle)

Java: Geschichte und Konzepte

Java virtual machine

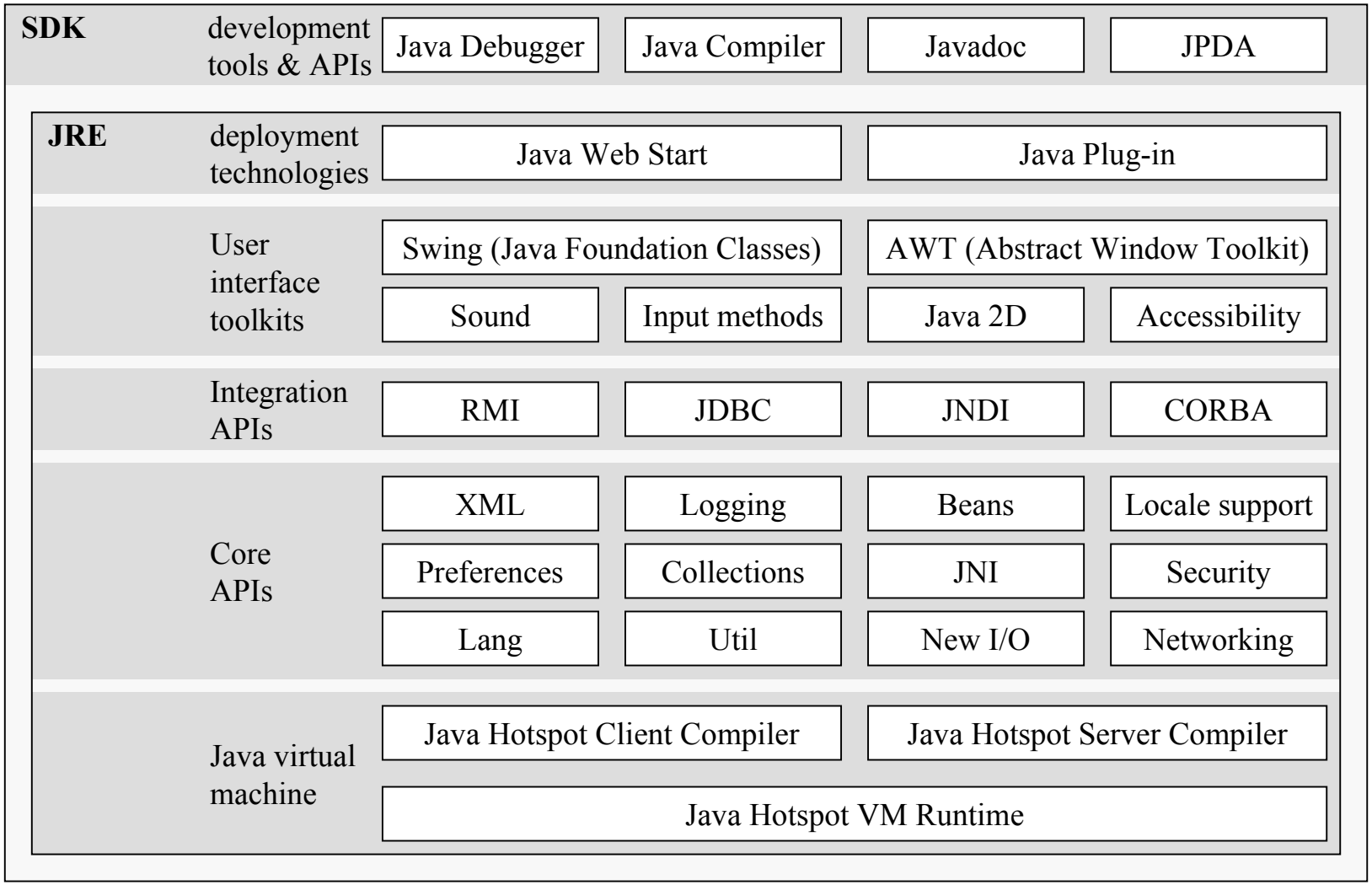
- ° Plattformunabhängigkeit
- ° großer Vorteil für Applikationen, die übers Web verteilt werden
- ° Vorteil vor allem im Standard
  - – Java class-File Format und Java JAR-Archiv-Format
- ▪ beides nicht neu, jedoch in der Kombination und zu diesem Zeitpunkt durchschlagend

- Java 2 (seit 1998)
  - ▪ Fokus auf Applets aufgegeben
    - ° Applets heute nur noch marginal
  - ▪ Plattform-Editionen
    - ° Funktionalitätsbündel für verschiedene Klassen von Nutzern
      - – J2SE mit JavaBeans als Plattform für Einzelanwendungen
      - – J2EE mit Enterprise JavaBeans (EJB) als Serverplattform (seit Ende 1999)
      - – J2ME für mobile und eingebettete Anwendungen
  - ▪ Formalisierung der Bezeichnungen Laufzeitumgebung (JRE), Entwicklungsumgebung (JDK) und Referenzimplementierung

Java: Geschichte und Konzepte

- Java 2 (Fortsetzung)
  - Referenzimplementierung der J2SE von Sun auf der Basis der HotSpot-JVM
  - J2EE als Standard mit Implementierungen von verschiedenen (unabhängigen) Anbietern
    - Referenzimplementierung von Sun als Beispiel-Implementierung im Quellcode verfügbar
    - nicht laufzeitoptimiert
  - Prüfung der Unterstützung von Standards durch Kompatibilitätstest-Reihen
  - Java BluePrints als Sammlung von Design-Richtlinien und Mustern, um spezielle Technologien zu unterstützen

Java: Geschichte und Konzepte



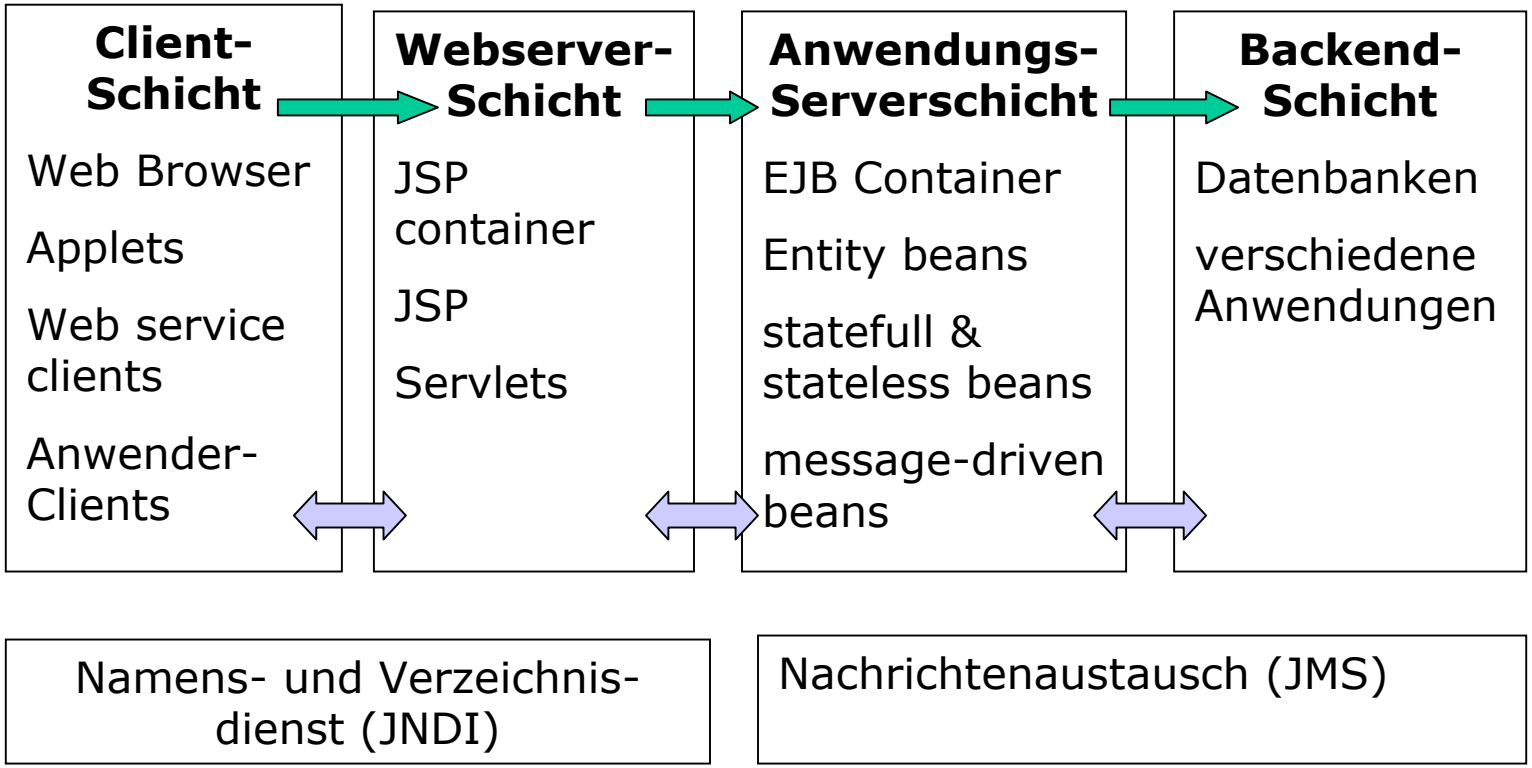
Aufbau der Java 2 Plattform, Standard Edition v1.4 (Quelle: java.sun.com)

Java: Geschichte und Konzepte

- J2EE Architektur
  - Im Zentrum steht **Familie von Komponentenmodellen**
    - Client-Schicht: Anwenderkomponenten, JavaBeans, Applets
    - Webserver-Schicht: Servlets, JSP
    - Anwendungsserver-Schicht: EJB in vier Varianten (stateless session, statefull session, entity, message-driven session)
  - JavaBeans kommt in allen Schichten zum Einsatz
  - **Integrations-Ebenen**:
    - Namens- und Verzeichnis-Infrastruktur (naming and directory interface, JNDI) sowie Nachrichten-Infrastruktur (Java messaging service, JMS) bilden die Klammern zwischen den verschiedenen Schichten
    - weitere I.-E.: Transaktionskoordinierung, Sicherheitsdienste

Java: Geschichte und Konzepte

- J2EE Architektur
  - Kontrollfluss: 
  - Datenfluss: 



Java: Geschichte und Konzepte

Wichtige, von Java unterstützte Grundkonzepte

- **Methoden** (Verhalten, behavior) und **Attribute** (Status, state)
- **Schnittstellen:** Es können Interfaces und abstrakte Klassen definiert werden, die später (mittels ‚*implements*') implementiert werden sollen
  - Mehrfachvererbung von Schnittstellen (ohne Status und Verhalten)
- **Klassen:** Implementierungen von Schnittstellen. Es können von bestehenden Klassen spezielle Unterklassen (mittels ‚*extends*') abgeleitet werden.
  - Einfachvererbung von Implementierungen
  - vermeidet das Diamant-Problem
  - unveränderbare Implementierungen (final class, method, attribute)
- **Pakete** und **Pakethierarchien** als Modularisierungskonzept jenseits von Klassen
  - Namensgebung und Namensräume auf dieser Basis
  - keine Unterstützung von Mehrfachversionen
  - company-name.productname Präfix als Standard
  - Namensraum-Importe
- **Sichtbarkeitsklassen** von Attributen und Methoden (default, public, protected, private)

Wichtige, von Java unterstützte Grundkonzepte

- **Ausnahmebehandlung** (exception handling): Es besteht die Möglichkeit, über Ausnahmen (mittels ‚try-catch'-Blöcken) vom Standard-Kontrollfluss abzuweichen
- **Threads und Synchronisation:** Nebenläufige Programmabläufe lassen sich mit Threads erzeugen und synchronisieren
- **Garbage Collection:** Nicht mehr referenzierte Objekte werden automatisch auf kontrollierbare Weise (‚*finalize*') zerstört
  - Anwender hat darauf aus Sicherheitserwägungen keinen Einfluss
- **Objektserialisierung:** Objekte, welche die Schnittstelle *Serializable* implementieren, können in einen Datenstrom geschrieben oder aus einem solchen aufgebaut werden (z.B. Speichern in eine Datei)
- **Ereignisse (events):** werden einige Folien später genauer besprochen

Grundlagen zu JavaBeans

- Komponentengedanke wird von Java nicht direkt unterstützt.
- JavaBeans ist die Spezifikation *eines* Komponentenmodells für Java
  - 1996 von Sun Microsystems eingeführt in Java Version 1.1
  - fasst mehrere Klassen und Ressourcen zusammen
  - keine Unterscheidung zwischen Beans und Beans-Instanzen
- Ziel: Das Zusammenfügen von Komponenten mit Hilfe von **visueller Programmierung** zu ermöglichen
  - Komponenten können einfach mit der Maus »zusammengeklickt« und in anderen Java-Applikationen oder in Java-Applets innerhalb von Web-Browsern eingesetzt werden
  - Design-Zeit und Laufzeit
- jede Java-Klasse ist im Prinzip eine JavaBean
  - um jedoch die **Kompositions- und Anpassungsfähigkeit** einer JavaBean optimal zu unterstützen, müssen gewisse Konventionen beachtet werden
  - JavaBeans sind damit plattformunabhängig, es muss lediglich eine Java Virtual Machine auf dem Zielsystem existieren

Grundlagen zu JavaBeans

Charakterisierung von JavaBeans

- JavaBeans sind Mengen gewöhnlicher Java-Klassen, die zusätzlichen Standards folgen, um sie mit Werkzeugen - etwa in einem GUI-Builder - zu manipulieren:
  - ***Eigenschaften (properties)*** = Attribute, auf die über get- und set-Methoden nach einem festgelegten Muster zugegriffen werden kann.
  - ***Ereignisse (events)***, die von der Bean erzeugt werden und über einen Ereignisbeobachter (event listener) an andere Bean weiter gegeben werden können.
  - ***Introspektion***, ein Mechanismus, über welchen Methoden, Eigenschaften und registrierte Ereignisse der Bean abgefragt werden können.
  - ***Anpassung***, ein Mechanismus, mit dem Eigenschaften einer Bean mit einem Werkzeug konfiguriert werden können.
  - ***Persistenz***, ein Mechanismus, mit dem die Einstellungen einer Bean dauerhaft gespeichert und wieder geladen werden können.

Eigenschaften

- einfache Eigenschaften
  - realisiert als Attribut der Bean-Klasse
- indizierte Eigenschaften
  - Feld von Werten, die über einen Index angesprochen werden
- gebundene Eigenschaften
  - Änderungen von Eigenschaften werden registrierten externen Beobachtern mitgeteilt
  - andere Objekte müssen sich dazu bei der Bean-Instanz als Eigenschaftenbeobachter (property listener) registrieren
- Eigenschaften mit Nebenbedingungen
  - erlauben es externen Beobachtern (vetoable change listener), vor der Änderung von Eigenschaften ihr Veto einzulegen und damit eine Änderung des Eigenschaftswertes zu unterbinden

Konventionen von Eigenschaften

- Entwurf
  - die Eigenschaften sollten möglichst beim ersten Entwurf spezifiziert werden, um Änderungen an der Schnittstelle zu vermeiden
  - zusätzlich muss für jede Eigenschaft entschieden werden
    - ob sie nur lesbar, nur schreibbar, lesbar und schreibbar ist
    - ob sie gebunden, indiziert oder mit Nebenbedingungen realisiert wird

- Zugriffsoperationen
  - für jede Eigenschaft wird eine *get*- und eine *set*-Operation implementiert
  - soll die Eigenschaft nur lesbar oder nur schreibbar sein, dann ist nur eine der beiden Operationen zu implementieren
  - alle zusätzlich zu den *get*- und *set*-Operationen implementierten öffentlichen Operationen werden nach außen exportiert
  - sie können bei Werkzeugen durch visuelle Programmierung zum Beispiel mit Ereignisbeobachtern verknüpft werden

Konventionen von Eigenschaften

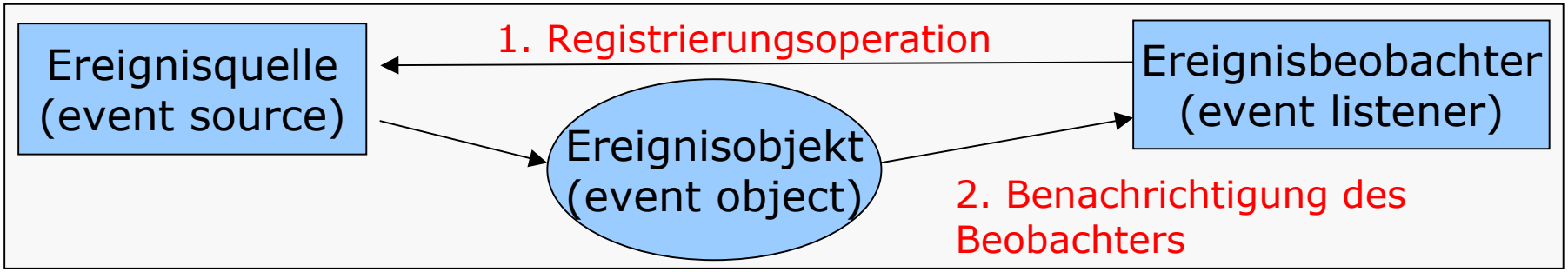
- einfache Eigenschaften (simple properties)
  - *public <Eigenschaftstyp> get <Eigenschaftsname>()*
  - *public void set <Eigenschaftsname> (<Eigenschaftstyp> a)*
    - für Eigenschaften vom Typ *boolean* auch:
      - public boolean is<Eigenschaftsname>
- indizierte Eigenschaften (indexed properties)
  - *public <Eigenschaftstyp> get<Eigenschaftsname>(int index)*
  - *public void set<Eigenschaftsname>(int index, <Eigenschaftstyp> a)*
- gebundene Eigenschaften
  - zusätzlich zu den normalen Zugriffsoperationen müssen Operationen zum Verwalten von PropertyChangeListener-Objekten implementiert werden:
    - *public void addPropertyChangeListener (PropertyChangeListener x)*
    - *public void removePropertyChangeListener (PropertyChangeListener x)*

Konventionen von Eigenschaften

- Eigenschaften mit Nebenbedingungen
  - bei Eigenschaften mit Nebenbedingungen muss die set-Operation folgende Signatur besitzen:
    - *public void set<Eigenschaftsname> (<Eigenschaftstyp> wert) throws PropertyVetoException*
  - zusätzlich müssen Operationen zum Verwalten von VetoableChangeListener-Objekten implementiert werden:
    - *public void addVetoableChangeListener (VetoableChangeListener x)*
    - *public void removeVetoableChangeListener (VetoableChangeListener x)*

Ereignisse

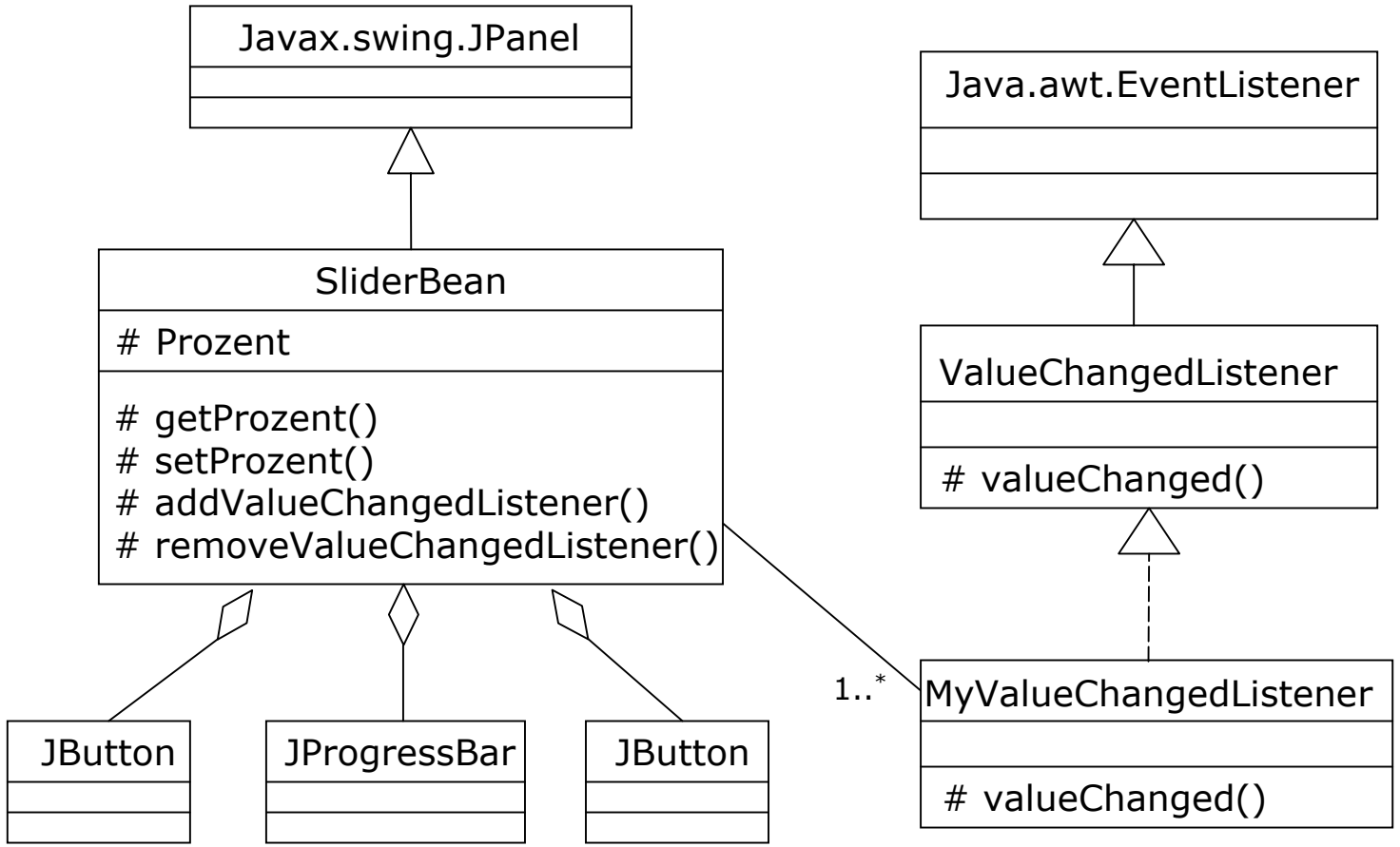
- JavaBeans benutzen das Ereignismodell von Java
- Ereignisse sind Objekte, die von einer Ereignisquelle erzeugt werden und an alle im Moment angemeldeten Ereignisbeobachter propagiert werden (ähnlich wie bei COM-Komponenten)
- Ereignisobjekte sollten keine Änderung ihrer Attribute zulassen
- eine Komponente kann sich zur Laufzeit bei einer anderen Komponente als Beobachter an- und abmelden
- sie muss allerdings auch das zugehörige Ereignis-Interface mit einer Rückrufmethode implementiert haben
- es existieren zwei Ereignisquellenarten
  - mehrere Beobachter anmeldbar (multicast event source, Standard)
  - nur ein Beobachter anmeldbar (unicast event source)



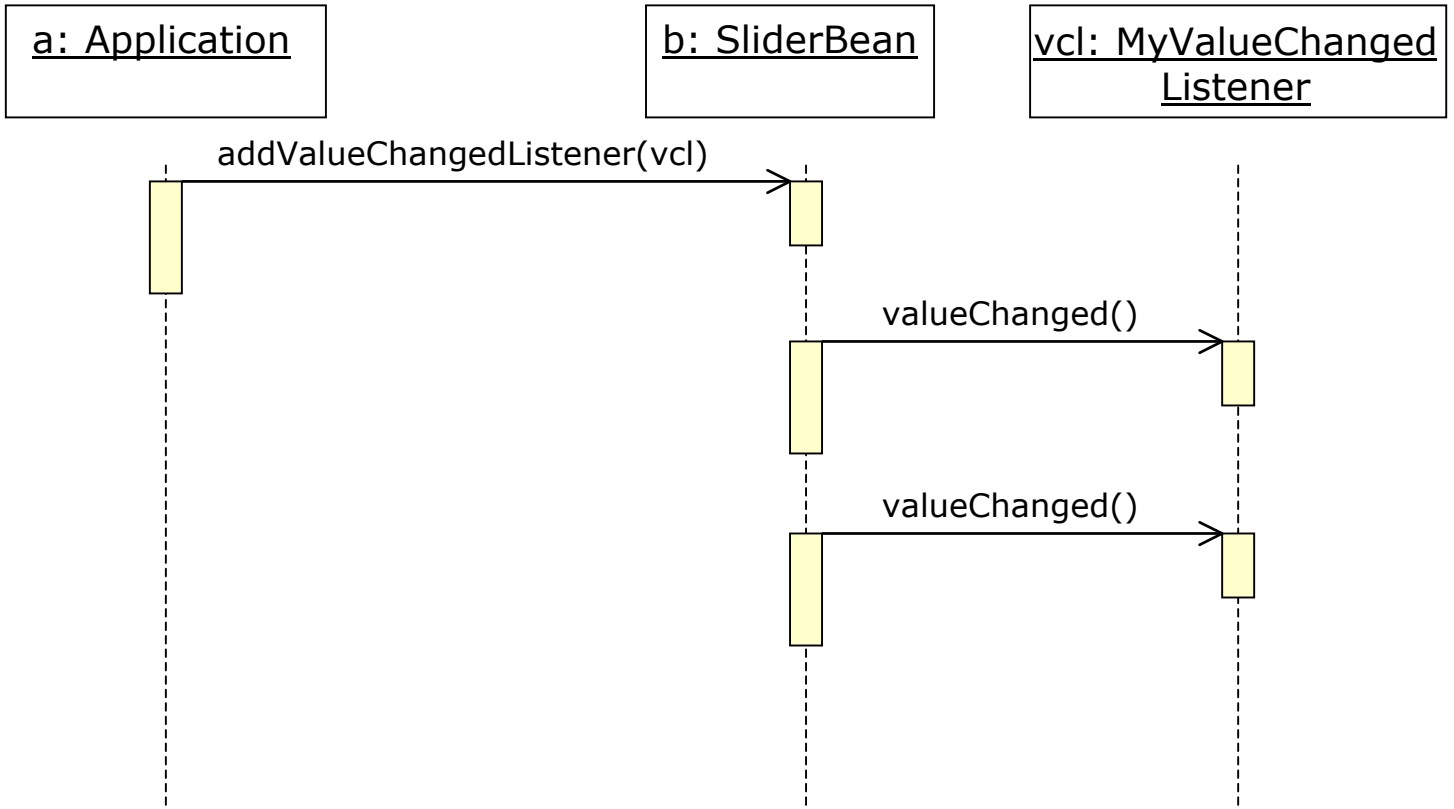
Konventionen für Ereignisse

- Ereignisse
  - die Operationen zur Verwaltung von Ereignisbeobachtern müssen konform zu den folgenden Namenskonventionen implementiert werden
- Unicast-Ereignisquelle
  - public void **add***Beobachtertyp*(*Beobachtertyp* l) throws **TooManyListenersException**
  - public void **remove***Beobachtertyp*(*Beobachtertyp* l)
- Multicast-Ereignisquelle
  - public void **add***Beobachtertyp*(*Beobachtertyp* l)
  - public void **remove***Beobachtertyp*(*Beobachtertyp* l)

Ereignismodell (UML-Klassendiagramm)



Ereignismodell (UML-Sequenzdiagramm)



Introspection

- Introspection
  - Komponente besitzt die Fähigkeit, relevante Informationen nach außen offen zu legen
  - durch Beachtung von Namenskonventionen bei Operations- und Klassennamen können Entwicklungswerkzeuge die gewünschten Informationen über Eigenschaften, Operationen und Ereignisse erhalten
  - zusätzliche Informationen können über die BeanInfo-Schnittstelle nach außen mitgeteilt werden
    - Beispiel: Piktogramm, das die JavaBean in der Werkzeugleiste einer Entwicklungsumgebung repräsentieren soll

Anpassbarkeit

- Entwurfsüberlegungen
  - Welches Problem soll durch die JavaBean gelöst werden?
  - Wo und wie wird die JavaBean eingesetzt (Kontext)?
  - Welche Veränderungen und Erweiterungen sind in Zukunft denkbar?

- **Antwort:** Detaillierte Beschreibung der 3 Hauptteile einer JavaBean (Eigenschaften, Ereignisse und Operationen)

Nachträgliche Anpassung (customization)

- ein Entwickler, der die Komponente nach ihrer Übersetzung und Auslieferung benutzt, kann sie an seine Bedürfnisse in gewissem Maße anpassen
- er kann Eigenschaften setzen, die das Erscheinungsbild und Verhalten beeinflussen
- hierfür gibt es 2 Möglichkeiten:
  - es wird ein Eigenschaften-Editor (property editor) des Entwicklungswerkzeuges benutzt
  - die JavaBean stellt eigene Klasse (customizer) zur Verfügung, die eine Anpassung komplexer Eigenschaften ermöglicht
    - oftmals wird hierfür ein Assistent (wizard) benutzt, der Schritt für Schritt durch die nötigen Schritte führt
    - Customizer können durch den GUI-Builder aufgerufen werden

Auslieferung und Benutzung

- die JavaBean mit allen benötigten Ressourcen wird in einer Archiv-Datei mit der Endung .jar verpackt, um sie als Einheit verschicken zu können
- eine .jar-Datei ist eine nach dem bekannten ZIP-Algorithmus komprimierte Archiv-Datei
- will man eine JavaBean in einem Entwicklungswerkzeug verwenden, so muss man in der Regel zunächst mit dem Entwicklungswerkzeug die .jar Datei der Komponente einlesen
- das Entwicklungswerkzeug merkt sich dann den Pfad, unter dem das Archiv zu finden ist und welche Komponenten in dem Archiv enthalten sind.

Einbettungs- und Diensteprotokoll

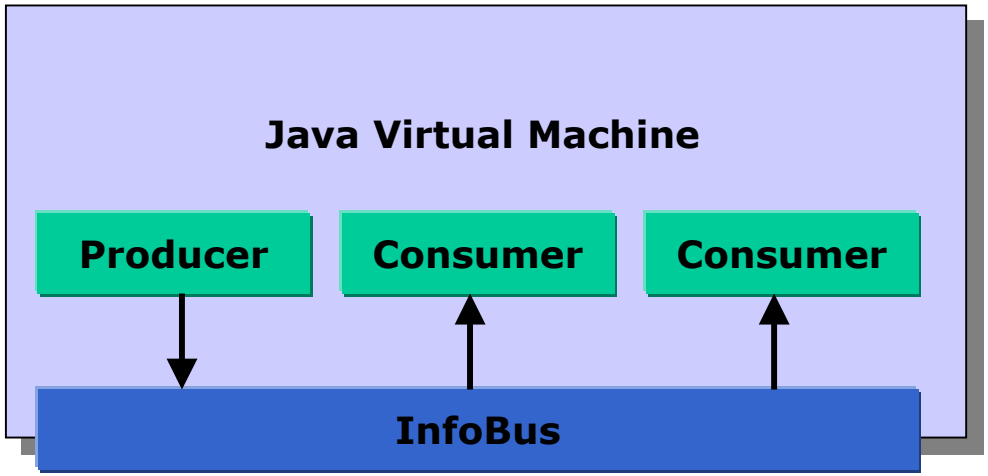
- Containment and services protocol
  - unterstützt logische Einbettung von JavaBeans-Instanzen
  - Dadurch kann die JavaBean nicht nur Dienste der JavaVM und der Core APIs in Anspruch nehmen, sondern auch zusätzliche Dienste des Containers, in dem sie sich zur Laufzeit befindet
  - genauso kann der Container die Dienste der eingebetteten JavaBean erweitern oder einschränken

Langzeit-Speicherung von JavaBeans

- Archivierung einer JavaBean
  - Alternative zum Serialisierungsprozess von Java
  - Format ist XML mit DTD oder ein anderes proprietäres Dateiformat
  - dabei wird nicht wie bei der Serialisierung alles gespeichert, sondern nur ein Teil der internen Objektdaten (public properties)
  - so kann ein neueres Objekt trotz anderer Attribute mit den „alten" Daten etwas anfangen, was bei Serialisierung nicht geht (da dort nur der komplette Zustand des Objekts wiederhergestellt werden kann)
  - dadurch wird eine Art „Versionsunabhängigkeit" erreicht

InfoBus

- gemeinsame Entwicklung von Sun und Lotus
- einfaches Verbinden von JavaBeans innerhalb einer VM über standardisierte Schnittstellen
- lose Kopplung zwischen den Komponenten
- ***Data-Consumer***: ruft Daten ab
- ***Data-Producer***: bietet Daten an und informiert über Änderungen
- ***Data-Controller***: wird zur Verarbeitung von Daten zwischen Producer und Consumer geschaltet



Bean Markup Language

- Entwicklung von IBM
- Scriptsprache um Beans zu erzeugen, zu konfigurieren und zu verbinden
- XML-basierte Syntax (wird durch XML-DTD beschrieben)
- 2 Ausführungsmodelle
  - in Java geschriebener Player, der BML-Skripte interpretativ ausführt
  - Compiler, der BML-Skripte in Java umsetzt (Geschwindigkeitserhöhung)

```
<bean class="java.awt.Frame"  id="topFrame">
<property  name="title"  value="Testfenster"/>
<property  name="background"  value="0xeeeeeee"/>
<property  name="layout">
        <bean  class"java.awt.BorderLayout"/>
</property>
```

Marktplätze für JavaBeans

- im Netz sind viele Beans als Freeware, Shareware oder kommerzielle Produkte vorhanden

- Beispiel: www.gamelan.com
  - visuelle Bean: Terminplaner
  - nicht-visuelle Bean: Socket-Schnittstelle

Quellen

- C. Szyperski: Component Software. Addison-Wesley 2002
  - Kap. 14 | S. 261 – 300
- Sun Microsystems Homepage
  - http://java.sun.com