



コンポーネントシステムとJava

早稲田大学 理工学部 情報学科

鷺崎 弘宜 深澤 良彰

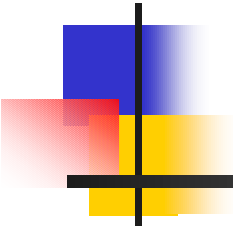
{washi,fukazawa}@fuka.info.waseda.ac.jp

<http://www.fuka.info.waseda.ac.jp/>



目次

- ✎ 1. コンポーネントウェア
 - ✎ コンポーネントとコンポーネントウェア
 - ✎ Javaにおけるコンポーネント
- ✎ 2. JavaBeans
 - ✎ 定義, 機能, 基盤, 例
- ✎ 3. EnterpriseJavaBeans (EJB)
 - ✎ 定義, 機能, 基盤, 例



1. コンポーネントウェア

コンポーネントとは

✎ 最大公約数的定義 [Stevens99]

✎ 「再利用あるいは代替可能なもの」

✎ 広義の対象

✎ アーキテクチャ、フレームワーク、コード

✎ 全てのUML文書  unified modeling language

✎ 狭義の対象

✎ ソフトウェアシステムのカプセル化された部品



[Stevens99] P.Stevens and R.Pooley,

“Using UML: Software Engineering with Objects and Components,”
Pearson Education Limited, 1999

(日本語) 児玉公信訳、

『オブジェクト指向とコンポーネントによるソフトウェア工学』
ピアソン・エデュケーション, 2000

コンポーネントウェア (Componentware)

定義 [青山98]

- ✖ アーキテクチャ上で交換可能なソフトウェア部品体系
- ✖ Plug&Play型の組み合わせ可能なソフトウェア部品体
- ✖ (実装はオブジェクト指向言語が自然)

アーキテクチャに依存

- ✖ 標準アーキテクチャがあって初めて交換・再利用可能
- ✖ (アーキテクチャは以降、“コンポーネントシステム”)

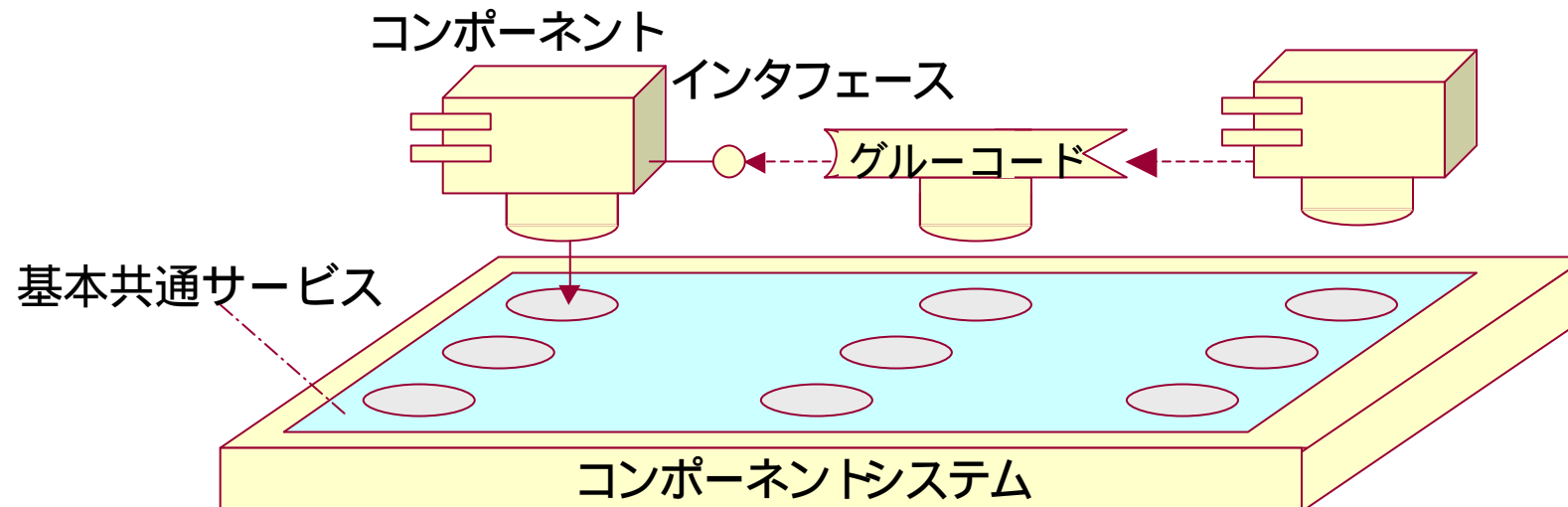


[青山98]

青山幹雄, 中所武司, 向山博 編
「コンポーネントウェア」
共立出版, 1998

コンポーネント指向開発

- ✎ コンポーネント群の組立て +
 - ✎ コンポーネントシステム
 - ✎ コンポーネントを受入れる基盤環境
 - ✎ 基本共通サービスの提供 (制御 保存)
 - ✎ コンポーネント
 - ✎ 内部を隠蔽(Black-Box)、インタフェースを公開
 - ✎ グルーコード
 - ✎ コンポーネント間を補完する“糊 (のり)”



コンポーネントウェアの出現背景

オブジェクト指向ソフトウェア開発の問題

従来のWhite-Box的な再利用

- Customizabilityが高すぎる
- 再利用可能な単位の判断: クラス? クラス群? パッケージ?

開発者と部品利用者のあいまいな境界

- 従来は、利用者が開発者でもある
- エンドユーザ自身による開発要求

ソフトウェア開発目標の変化

- 開発期間の短縮、開発コストの削減要請
- 信頼性・安全性の重視

エンドユーザ自身による
開発要求

開発コストの短縮要求

Needs



コンポーネントウェア
の出現

Sheeds



オブジェクト指向技術に基づく
再利用技術の成長

VBXコントロールの成功・
インターネット部品市場の出芽



コンポーネントシステム

クライアントコンポーネントシステム

- GUI部品, 汎用ロジック部品
- エンドユーザコンピューティング(EUC)
- ビジュアルプログラミング, スクリプト処理
- 実装モデル
 - Microsoft社: ActiveX/COM
 - Sun Microsystems社: JavaBeans

サーバコンポーネントシステム

- データベース・ネットワーク処理の再利用
- 分散オブジェクト
- 実装モデル
 - OMG: CORBA
 - Sun Microsystems社: EnterpriseJavaBeans

Java2プラットフォームとコンポーネント

3種の開発プラットフォーム



- ✂ Java2SE (Standard Edition) → **JavaBeans**
 - ✂ 標準パッケージ構成、Java2EE, Java2MEの基礎
 - ✂ デスクトップ(クライアント)用途中心
- ✂ Java2EE (Enterprise Edition) → **EnterpriseJavaBeans**
 - ✂ = Java2SE + 拡張パッケージ群 (Servlet, EJB, JNDI, ...)
 - ✂ エンタープライズ(サーバ)用途中心
- ✂ Java2ME (Micro Edition)
 - ✂ = Java2SE - (JFC 及び高機能なクラス群)
 - ✂ 組み込み用途中心 (携帯電話, PDA端末, カーナビ, ...)



2. JavaBeans

名称・機能

名称: JavaBeans (ジャワ産コーヒー豆)

JavaBeansコンポーネントを“Bean”と呼ぶ



機能:

Javaによるクライアントコンポーネントシステム

GUI部品 (Widgets), 汎用ロジック部品

EUC, ビジュアルプログラミング

開発ツールとの連携

RAD (Rapid Application Development) ツール

JBuilder (Borland社), VisualAge(IBM社), Forte(Sun社)

JavaBeans Development Kit 1.1(BDK1.1)

イントロスペクション機構

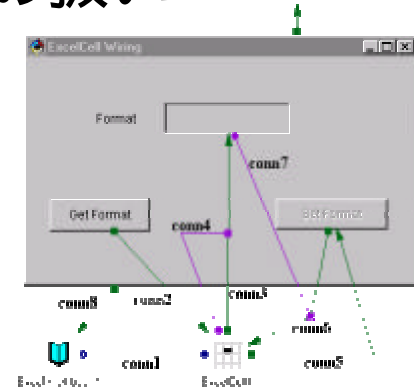
パーシステンス機構

パッケージング (Jar)

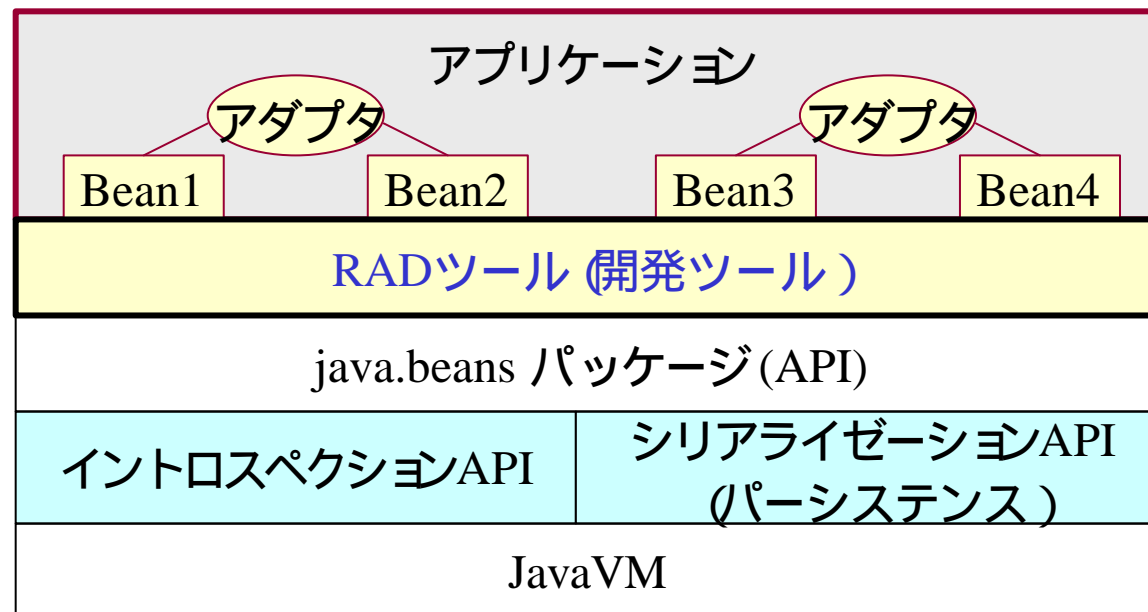
基盤

ビジュアルに再利用可能なソフトウェア部品

- 同一Java仮想マシン (JavaVM) 内での扱い
- RADツールによる扱い



JavaBeansベースなアプリケーション概念図





対比: BeanとJavaクラス

☞ Beanと通常クラスの違い

- ☞ BeanはJava言語におけるクラスでもある
- ☞ 対象ユーザの相違

比較項目	クラス	Bean
ソフトウェア部品か？		
オブジェクト指向か？		
対象ユーザ	プログラマ	エンドユーザ
プログラミング方法	コーディング	ビジュアルプログラミング
継承を使うか？		×
委譲を使うか？		
部品自体の開発工数	小さい	大きい
再利用性	低い	高い



Beanの定義

☞ 表現上の定義: [Hamilton97]

“A Java Bean is a reusable software component that can be manipulated visually in a builder tool”

☞ 実装上の定義:

- ☞ (1) デフォルトコンストラクタを持つクラス
- ☞ (2) `java.io.Serializable` インタフェースを実装するクラス、または `Serializable` インタフェースを実装したクラスを継承したクラス

☞ よくある誤解

- ☞ `java.beans.Bean` を継承したクラス? ➡ 任意のクラス
- ☞ 開発にBDKが必要? ➡ Java2SE標準セットのみで開発可能

[Hamilton97] G.Hamilton, “**JavaBeans 1.01 Specification,**”
Sun Microsystems, 1997

例: 最も単純なBean

- ☞ 定義に従う最も単純なBean
 - ☞ 開発ツールでBean名から生成可能
 - ☞ 保存・復元可能

Java言語上のクラスである

Serializable インタフェースを実装

```
// MyBean.java
public class MyBean extends Object implements java.io.Serializable {

    public MyBean() {
        super();
    }

}
```

引数の無いコンストラクタ
(デフォルトコンストラクタ)



構成: パッケージ

✎ 関連パッケージ

```
java.beans  
└─ java.beans.beancontext  
java.io  
java.util
```

Bean開発用クラス群
Beanの論理グループ設定用
入出力用 (パーシステント)
代理人イベント用

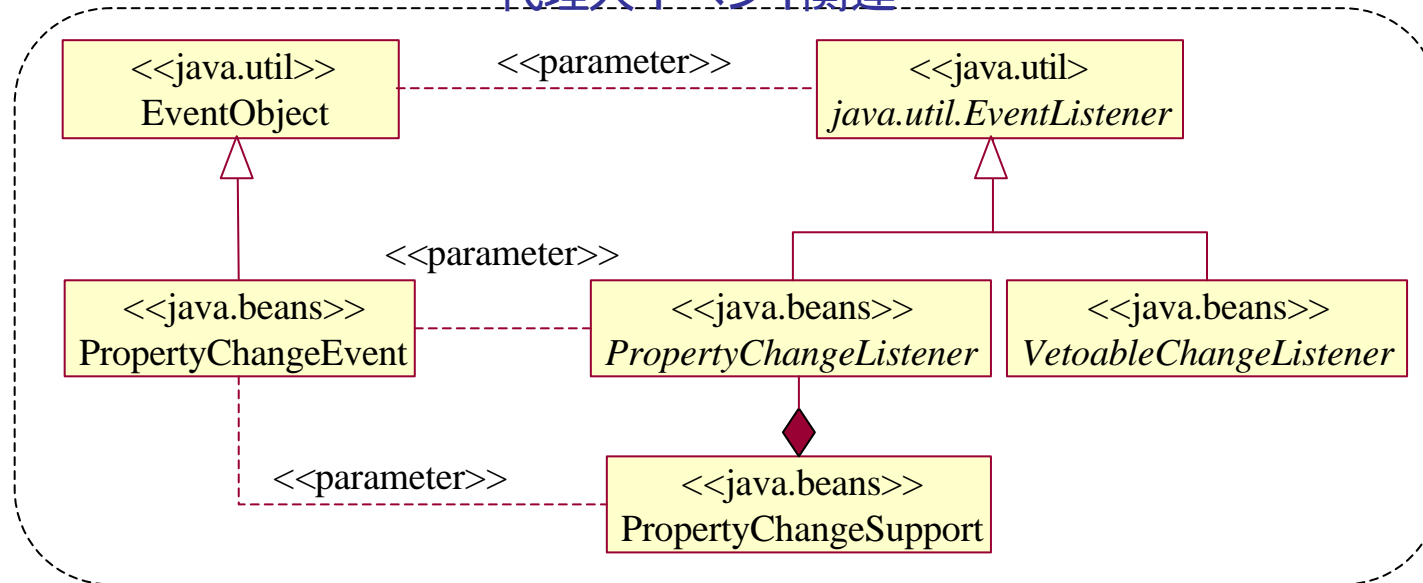
構成: 主要クラス

java.beans.*	
Introspector	対象Beanの調査 (イントロスペクション)
<i>BeanInfo</i>	対象Beanのメタ情報
PropertyDescriptor	対象Beanのプロパティ情報
MethodDescriptor	対象Beanのメソッド情報
EventDescriptor	対象Beanのイベント情報
<i>PropertyChangeListener</i>	バウンドプロパティ用イベントリスナーインタフェース
<i>VetoableChangeListener</i>	コンストレインドプロパティ用イベントリスナーインタフェース
PropertyChangeEvent	バウンド・コンストレインドプロパティ用イベント情報
PropertyChangeSupport	バウンドプロパティ用サポートクラス
java.io.*	
<i>Serializable</i>	保存・復元可能 (パーシステント)なことの宣言
java.util.*	
EventObject	イベント情報
<i>EventListener</i>	イベントリスナーインタフェース

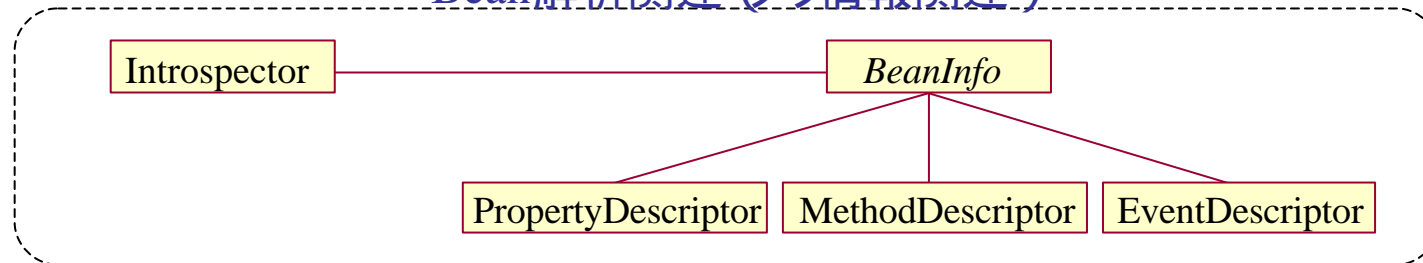
構成: 関連

■ 主要クラス関連図

代理人イベント関連



Bean解析関連 (メタ情報関連)



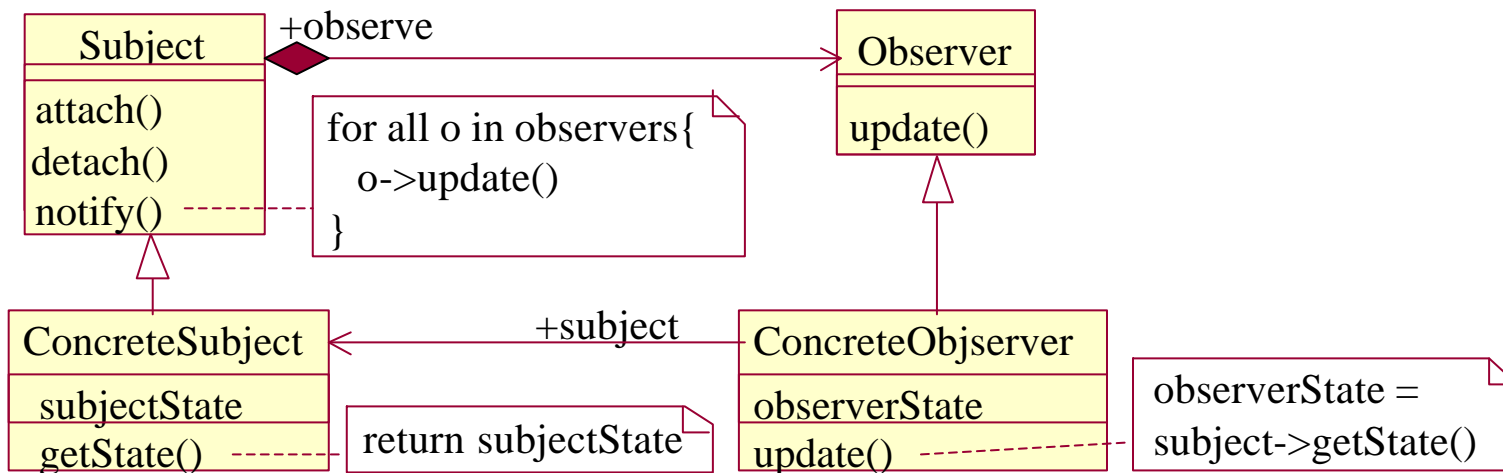


Beanの3大要素

- ✎ プロパティ (Property)
 - ✎ 外部から観測 設定可能な名前付き属性
 - ✎ 観測操作: getter メソッド
 - ✎ 設定操作: setter メソッド
- ✎ メソッド (Method)
 - ✎ 外部から実行可能なメソッド
- ✎ イベント (Event)
 - ✎ コンポーネント間通信の起点
 - ✎ イベント発火操作
 - ✎ イベントオブジェクトクラス
 - ✎ イベントリスナーインタフェース

イベント(Event)

- イベントとは: コンポーネントから外部への通知
- 設計指針: Observerデザインパターン [GoF95]
 - オブジェクトの変化を依存オブジェクトに自動通知

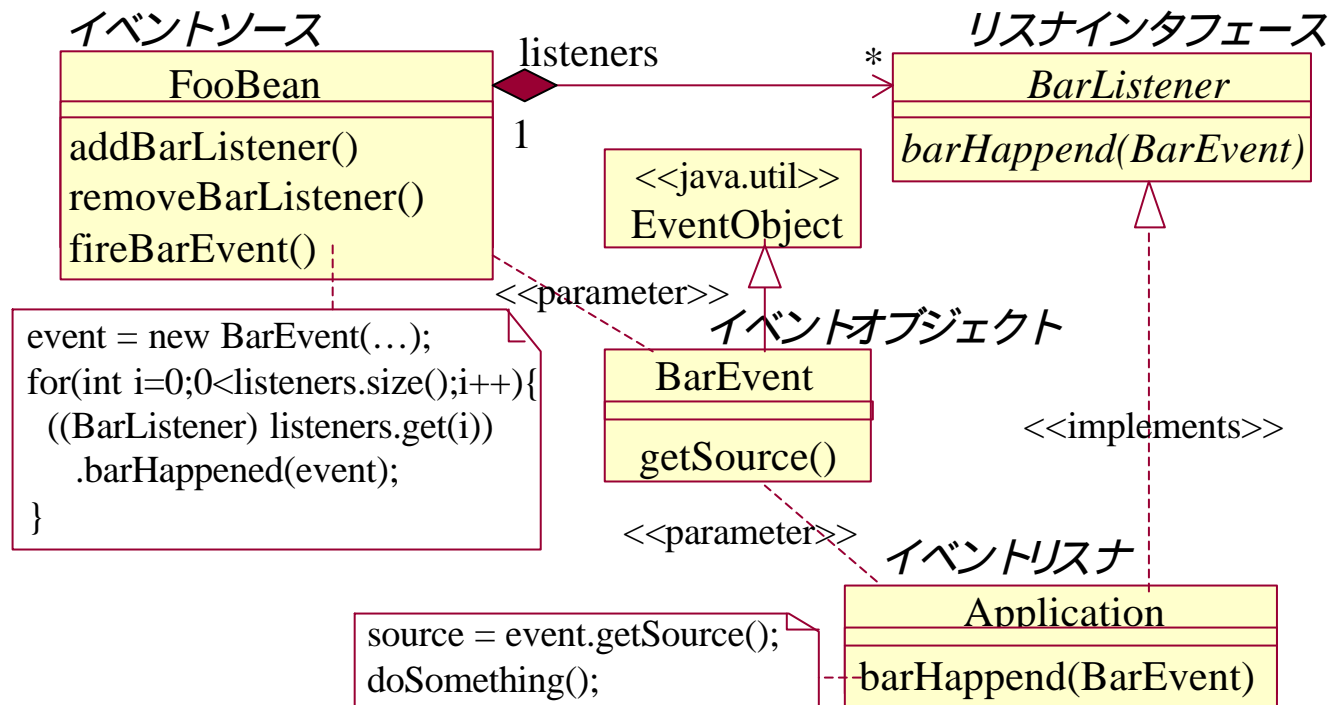


[GoF95] E. Gamma, R. Helm, R. Johnson, J. Vlissides,
“**Design Patterns: Elements of Reusable Object-Oriented Software,**”
Addison-Wesley, 1995
(日本語) 本位田真一、吉田和樹監訳、
「**オブジェクト指向における再利用のためのデザインパターン**」,
ソフトバンク, 1995 (改訂版 1999)

代理人イベントモデル

代理人イベントモデル (Delegation Event Model)

- イベントソース: イベントの発火元クラス
- イベントリスナ: イベントの受取り先クラス、リスナインタフェースの実装
- イベントオブジェクトクラス: イベント情報を持つクラス

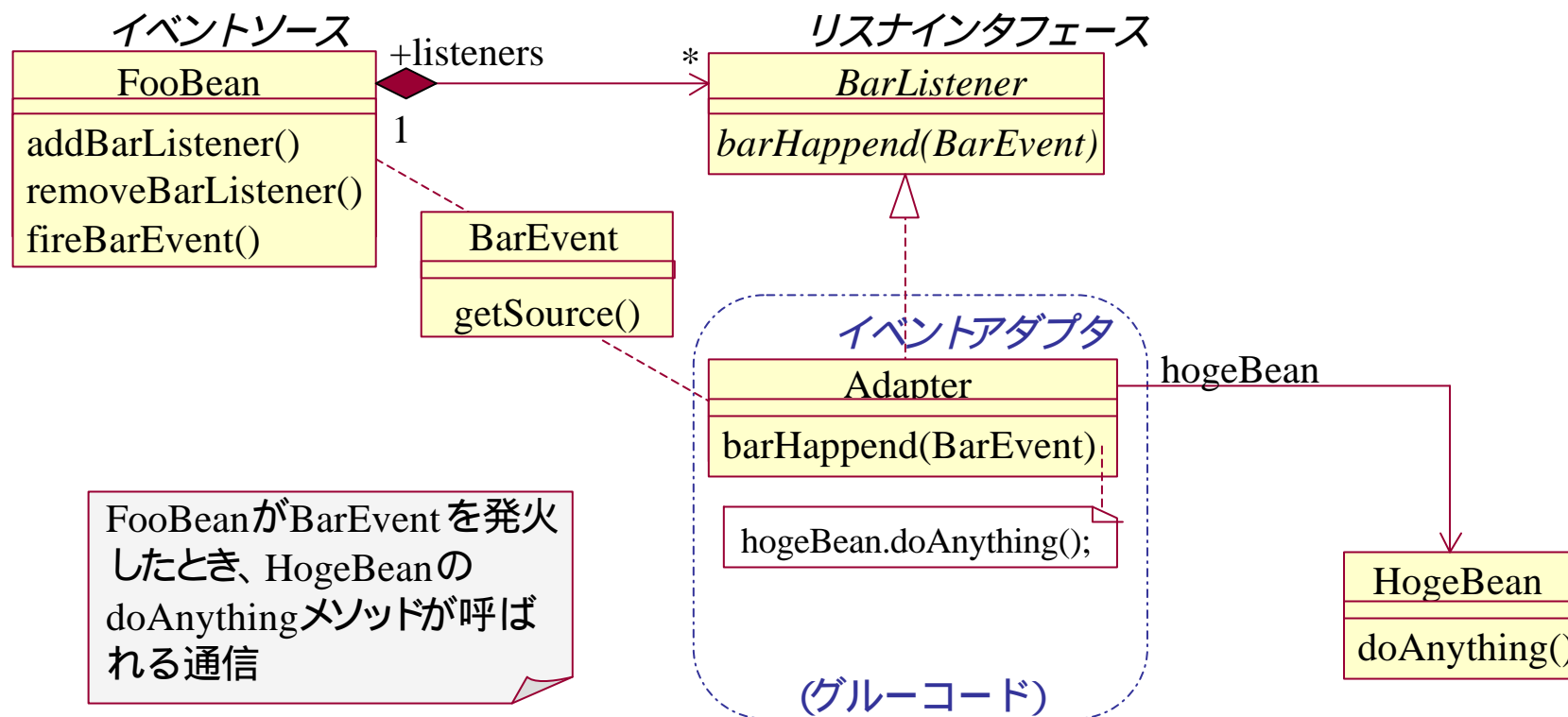


- イベントソースにイベントリスナを事前に登録
イベントソースに対してユーザ入力があると…
- 1 イベントオブジェクトの生成
 - 2 全てのリスナに通知
 - 3 リスナはイベントオブジェクトを得て、何か処理

コンポーネント間イベント通信

コンポーネント間の通信

- 実行時に互いを知らない同士の間通信 (FooBean と HogeBean)
- イベントをイベントアダプタで仲介



プロパティ (Property)

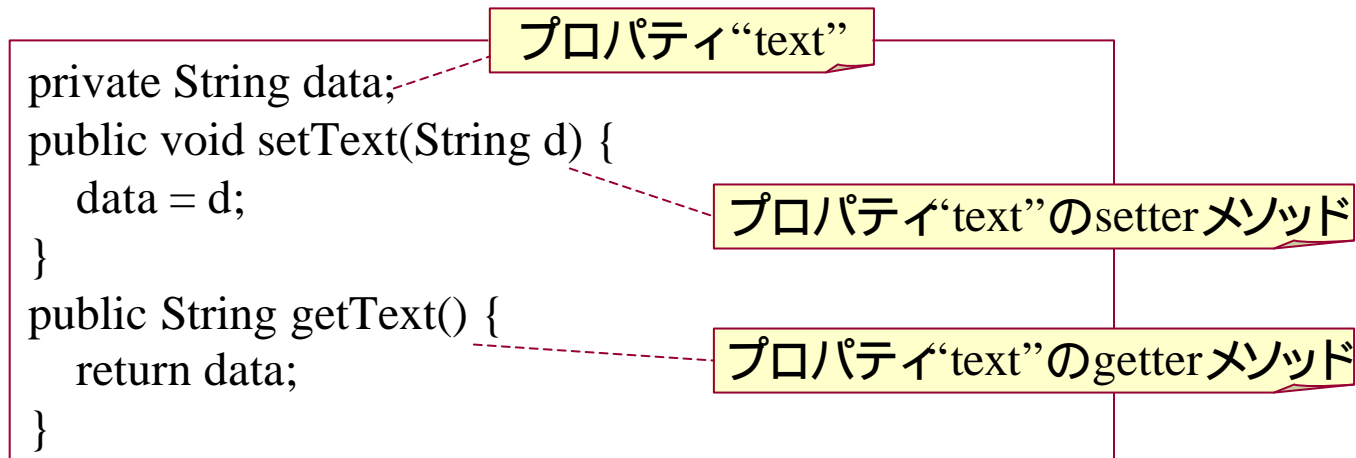
プロパティとは

- 外部から観測 設定可能な名前付き属性
 - 設定操作: setter メソッド
 - 観測操作: getter メソッド
- プロパティアクセサ

仕組み

命名規則 (Naming Conventions)

- (setter): setXXX
 - (getter): getXXX
- プロパティ“XXX”に対応

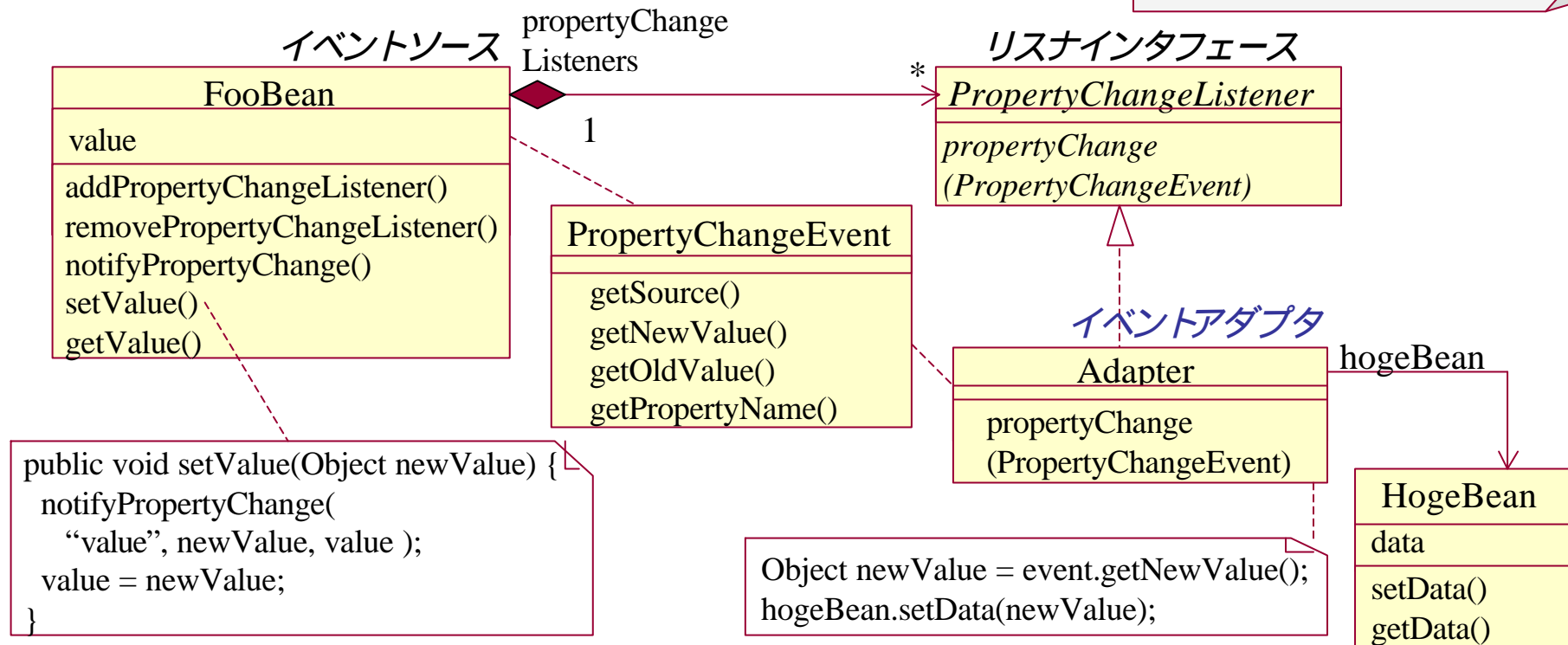


バウンドプロパティ (Bound Property)

プロパティの値の変更通知・共有

- プロパティ変更イベントリスナ (PropertyChangeListener)
- プロパティ変更イベント (PropertyChangeEvent)
 - プロパティの変更前と変更後の値を格納

FooBeanのプロパティ“value”の値変更をHogeBeanのプロパティ“data”に反映



メソッド (Method)

メソッドとは

- 外部から実行可能なメソッド

仕組み

- 基本的にはpublicなメソッド
- カスタムなBeanInfoの指定により隠されることもある

外部から実行可能

```
public double doSomething(int value) {  
    .....  
}  
private String getData() {  
    .....  
}
```

外部から実行不可能!
(一見するとプロパティ“data”のgetterメソッド)



Beanと開発ツール

- ✎ 再利用可能な部品として扱う仕組み
 - ✎ インtrospeクシヨシメカニスムとBeanInfo
 - ✎ 開発時・実行時のメタ情報取得
 - ✎ パーシステンスムメカニスム
 - ✎ 開発時・実行時の保存/復元
 - ✎ パッケージング
 - ✎ Jarコマンド



ビジュアルプログラミング・コーディングの両サポート

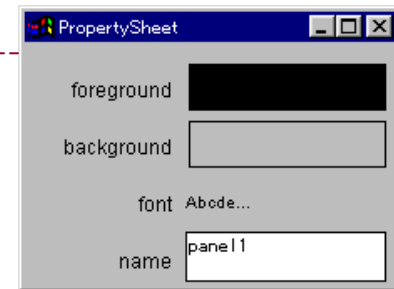
イントロスペクション機構(Introspection)

イントロスペクションとは

- Beanからメタ情報を得る仕組み, ソースコード不要
- 開発ツールによるBeanの統一的な扱い

Beanのメタ情報: BeanInfoオブジェクト

- PropertyDescriptor: プロパティ情報
- MethodDescriptor: メソッド情報
- EventDescriptor: イベント情報



BeanInfoオブジェクト取得方法

- (a) リフレクションAPIを用いた自動解析
 - 命名規則 (プロパティ用), publicなメソッド (メソッド用), イベントリスナ追加・削除メソッド (イベント用)
- (b) カスタムBeanInfoクラスをBeanと共に配布
 - FooBeanInfoクラスをFooBeanクラスと同時提供

パーシステンス (Persistent)

- Beanの保存/復元: 持つデータ(特にプロパティ値)が不揮発
 - 開発ツールによるプロパティの値変更結果を保存し、完成したアプリケーションの初期化時に復元
- 仕組み: java.io.Serializable インタフェースの実装
 - static または transient でない属性の自動保存/復元
 - writeObject, readObject メソッドによるカスタム実装

そのまま自動保存

```
public class MyBean implements Serializable {
```

```
    int a = 0;
```

```
    transient String b = "data";
```

自動保存/復元対象外の宣言

```
    static double c = 0.1;
```

```
    private void writeObject(ObjectOutputStream oos) throws
```

```
IOException {
```

```
        oos.defaultWriteObject();
```

```
        oos.writeDouble(c);
```

クラス変数 c のカスタム保存実装

```
    }
```

```
    private void readObject(ObjectInputStream ois) throws IOException {
```

```
        ois.defaultReadObject();
```

```
        MyBean.c = ois.readDouble();
```

クラス変数 c のカスタム復元実装

```
    }
```

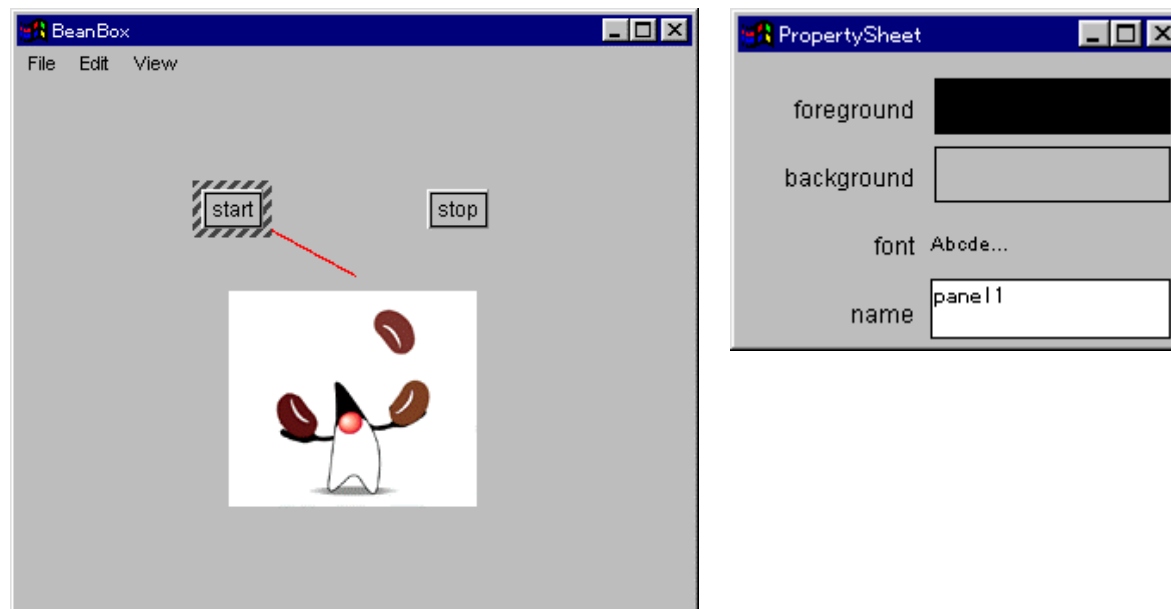
```
}
```

BeanBox (BDK)

BeanBox とは

- BDK 付属のサンプル開発ツール
- Bean 間のビジュアルな接続
 - イベント → メソッド接続 (イベントアダプタ自動生成)
 - プロパティ ↔ プロパティ接続 (バウンドプロパティ, イベントアダプタ自動生成)

ButtonBean の ActionEvent と JugglerBean の何らかのメソッド接続



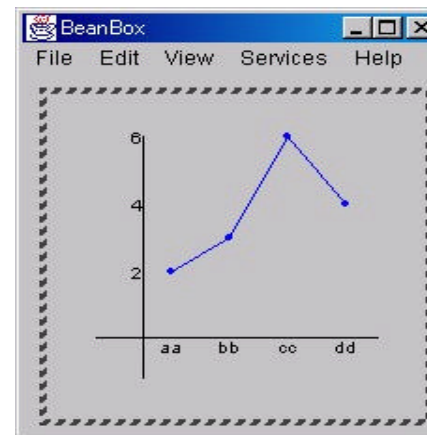
Bean利用例

目的

- 数値列を入力してグラフを得て、その画像を保存し印刷したい

利用Bean

- JButton: 押下操作
- JTextField: テキスト入力操作
- FukaGraphBean: 折線グラフ表示・画像保存 [FukaBeans]



FukaBeans

[FukaBeans] <http://www.fuka.info.waseda.ac.jp/Project/CBSE/>

ビジュアルプログラミング例

結線方式 (Wiring method)

- 値の入力: プロパティ- プロパティ接続
- 処理の開始: イベントメソッド接続

結線例 (抜粋)

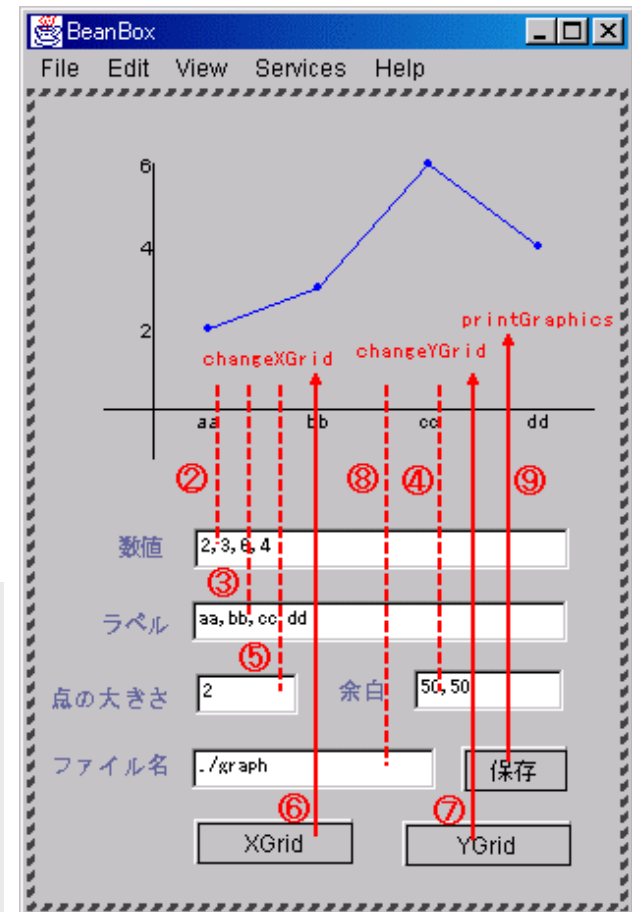
折れ線グラフを表示したいところへFukaGraphBeanを設置
JTextFieldを置いて、JTextFieldの“text”プロパティと
FukaGraphBeanの“docData”プロパティをバインド接続

.....

JTextFieldを置いて、JTextFieldの“text”プロパティと
FukaGraphBeanの“fileName”プロパティをバインド接続
JButtonを置いて、JButtonのactionPerformedイベントから、
FukaGraphBeanのprintGraphicsメソッドの呼出し接続

コーディング例 (抜粋)

```
graphBean = new FukaGraphBean();  
textData = new JTextField();  
textData.addPropertyChangeListener(new PropertyChangeListener(){  
    public void propertyChange(PropertyChangeEvent e){  
        graphBean.setDocData(textData.getText());  
    }  
});
```



業界動向

Beanの流通・販売

高機能なGUI部品へのニーズが確かにある

Bean販売ベンダ:

- Sitraka Software社 (JClass)
- Infragistics社 (JSuite) など

マーケットプレイス

- 各社のBeanの代理販売、流通仲介
- ComponentSource.com, Flashline.com など



ビジュアルプログラミングの是非

結線方式の限界、コーディングとの併用必要性

JavaStudioの販売中止 (SunMicrosystems)



まとめ (JavaBeans)

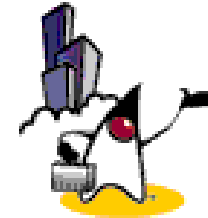
- ✎ Javaにおけるクライアントコンポーネントシステム
 - ✎ GUI部品を中心として普及
 - ✎ 再利用性を高めるための要素技術
 - ✎ 代理人イベント機構, パーシステント, 開発ツール
- ✎ 今後の可能性
 - ✎ ビジュアルプログラミングによるEUCは不完全
 - ✎ 多様なBean群をライブラリとしてコーディング再利用
 - ✎ アプリケーション設計・開発時に、各クラスの将来の再利用性と、内部における依存性の排除を目的として、プロパティ・イベント等の要素利用
 - ✎ JSP (JavaServerPages)との連携
 - ✎ <jsp:useBean>タグの利用
 - ✎ 例 :<jsp:getProperty name="clock" property="year"/>



3. Enterprise JavaBeans (EJB)

名称・機能

名称: Enterprise JavaBeans
(企業向きジャワ産コーヒー豆)



機能:

Javaによるサーバコンポーネントシステム

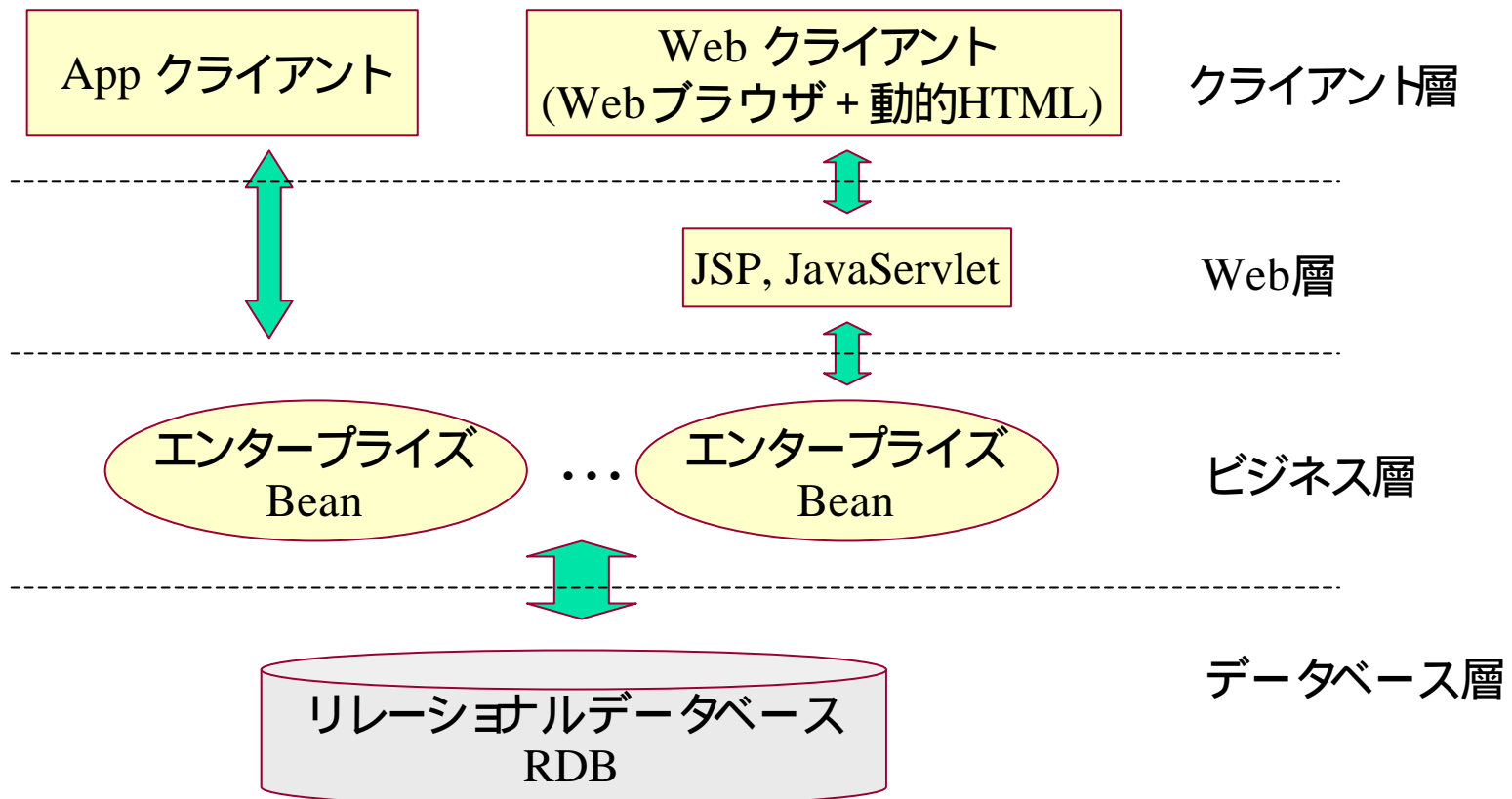
- ビジネスロジックをカプセル化したサーバ側コンポーネント
- Webを用いたe-commerce
- データベース処理の容易化

J2EEの他の技術との連携

- JavaRMI (RemoteMethodInvocation):
 - 分散オブジェクト技術
- JNDI (Java Naming and Directory Interface):
 - 名前解決
- JDBC:
 - データベース接続

基盤

- ネットワーク上の分散コンポーネント
 - 異なるJavaVM上のオブジェクト間の通信
 - J2EEサーバ上での稼働



エンタープライズBeanの利用例

エンタープライズBeanのメソッド呼出し

- ✂ (1) EJBサーバ内で生成されているEJBHomeオブジェクトをネームサービス (JNDI) を用いて獲得
- ✂ (2) EJBHomeオブジェクトへのcreateメソッド呼出し
- ✂ (3) EJBObjectオブジェクトを得る
- ✂ (4) EJBObjectへ処理を行う

Sun Microsystems,
(Java Pet Store Demo)



EJBHomeオブジェクト取得例

```
public static CatalogHome getCatalogHome() {
    try {
        InitialContext context = new InitialContext();
        Object objref = context.lookup("java:comp/env/ejb/catalog");
        return (CatalogHome)PortableRemoteObject.narrow(objref);
    } catch (NamingException e) {
        throw new GeneralFailureException(e);
    }
}
```

リファレンスの型を狭める



エンタープライズBeanの種類

- ✎ セッションBean (SessionBean)
 - ✎ クライアントが呼出す処理の集合
 - ✎ ライフサイクル
 - ✎ 生成: クライアントの接続によるセッション開始
 - ✎ 消滅: セッション終了
- ✎ エンティティBean (EntityBean)
 - ✎ リレーショナルデータベース上のTableに対応
 - ✎ クライアントのセッションと無関係
 - ✎ EJBサーバが停止しても“永続 (persist)”
- ✎ メッセージ駆動型Bean (Message-Driven Bean)
 - ✎ クライアントとの対話状態を保持しない
 - ✎ Java Message Service (JMS)を基盤とした非同期呼出し
 - ✎ 参考: JMS Tutorial <http://java.sun.com/products/jms/tutorial/>



エンタープライズBeanの利用効果と使い分け

✎ 利点

- ✎ EJBサーバ内の「EJBコンテナ」がトランザクション管理・セキュリティ認証等の基本サービスを提供
- ✎ クライアント側とビジネスロジック実装の分離
- ✎ 移植性がある

✎ エンタープライズBeanの使い分け

- ✎ 単にサーバ上のオブジェクトの呼び出し
 - ➡ セッションBeanのみ
- ✎ リレーショナルデータベース上の永続的データの利用
 - ➡ エンティティBeanの併用

セッションBeanの種類

- ☞ ステートレスセッションBean (stateless: 状態無)
 - ☞ メソッドが呼出しの間だけの状態 (インスタンス変数の値)保持
 - ☞ 同じステートレスセッションBeanインスタンスの利用
 - ☞ 例: カタログ情報取得ユーティリティ
- ☞ ステートフルセッションBean (stateful: 状態有)
 - ☞ 同一セッション中で変化する状態の保持
 - ☞ セッション毎のステートフルセッションBeanインスタンス
 - ☞ 例: ショッピングカート

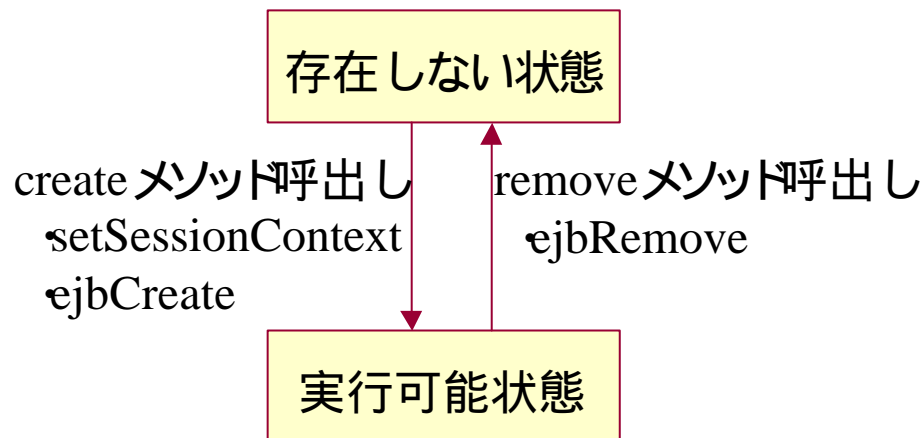
```
public interface ShoppingCart extends EJBObject {  
    public void addItem(String item) throws RemoteException;  
    public void removeItem(String item) throws RemoteException;  
    .....  
}
```

Itemという状態管理の必要性

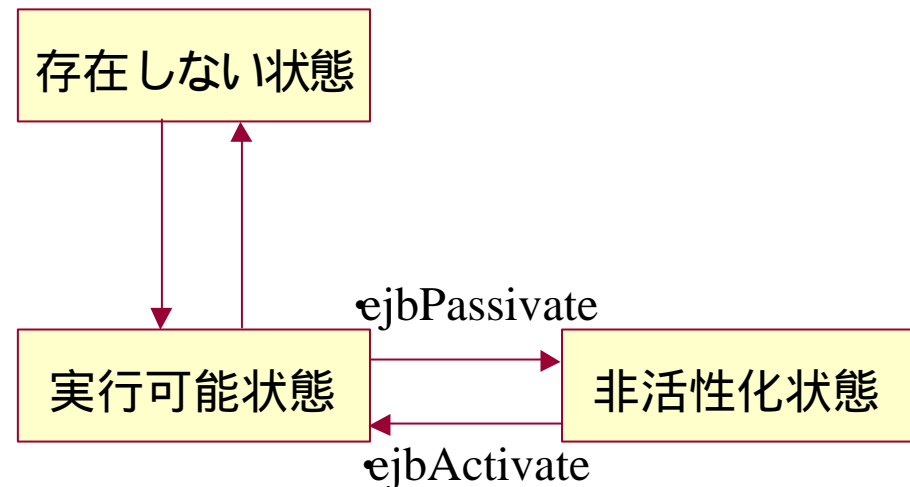
セッションBeanのライフサイクル

- ステートレスセッションBean
 - 非活性化 (passivate) されない
- ステートフルセッションBean
 - EJBコンテナによって非活性化される
 - メモリから二次記憶領域への退避
 - 一般に LRU (Least Recently Used) アルゴリズムに従う

■ ステートレスセッションBean



■ ステートフルセッションBean





ステートレスセッションBean例(1): インタフェース定義

リモートインタフェース (Remote Interface)

呼ばれるビジネスメソッドの定義

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Convert extends EJBObject {
    public double convertD2Y(double dollar) throws RemoteException;
}
```

ホームインタフェース (Home Interface)

エンタープライズBeanの生成・発見・消去用メソッドの定義

```
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;
public interface ConvertHome extends EJBHome {
    Convert create() throws RemoteException, CreateException;
}
```



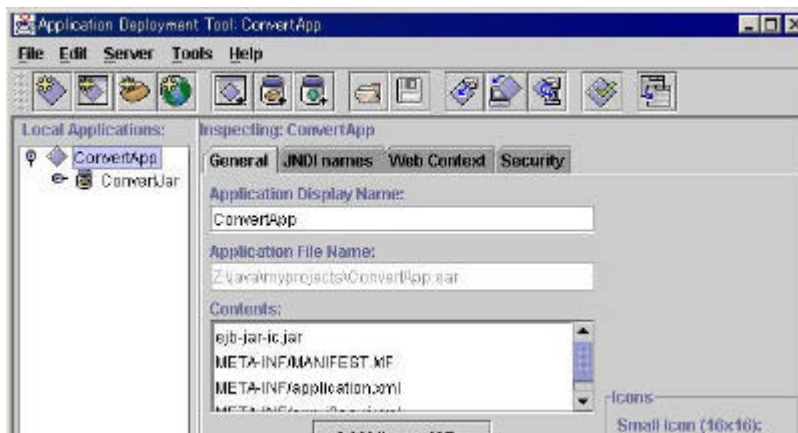
ステートレスセッションBean例(2): Beanの実装

- ☞ セッションBeanクラス (SessionBean)
 - ☞ インタフェースSessionBeanを実装
 - ☞ リモートインタフェースで定義した全メソッドを実装

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
public class ConvertEJB implements SessionBean {
    public double convertD2Y(double dollar) {
        return dollar*120.0;
    }
    public ConvertEJB() { }
    public void ejbCreate() { }
    public void ejbRemove() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void setSessionContext(SessionContext sc) { }
}
```

ステートレスセッションBean例(3): 配備

- ✎ 実行環境: Sunのリファレンス実装
 - <http://java.sun.com/j2ee/download.html#sdk>
- ✎ J2EEサーバへの配備(deploytoolの利用)
 - ✎ ConvertApp.ear ファイルの新規作成
 - ✎ ConvertJar.jar ファイルの新規作成
 - ✎ Convert.class, ConvertEJB.class, ConvertHome.class
 - ✎ deploytool descriptor ファイルの自動生成
 - ✎ JNDIへの名前登録 (“MyConvert”)
 - ✎ 配備, 完了



ステートレスセッションBean例(4): Appクライアント

利用クライアントの開発

JNDI上で“MyConvert”の指定

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import Convert; import ConvertHome;
public class ConvertClient {
    public static void main(String[] args) {
        try {
            Context initial = new InitialContext();
            Object objref = initial.lookup("MyConvert"); ----- JNDIの指定
            ConvertHome home =
                (ConvertHome) PortableRemoteObject.narrow(objref, ConvertHome.class);
            Convert currencyConvert = home.create();
            double amount = currencyConvert.convertD2Y(100.00);
            System.out.println(String.valueOf(amount));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ステートレスセッションBean例(5): 実行例

利用クライアントの実行

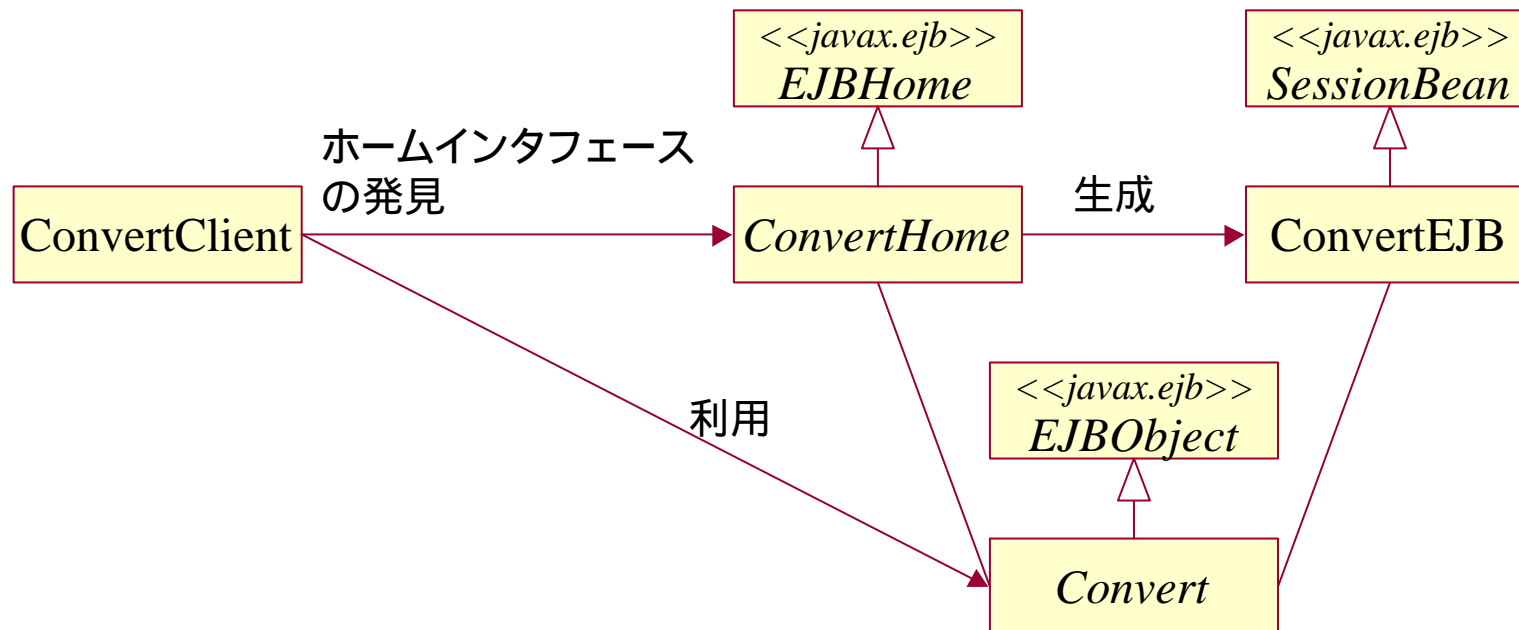
クライアント用Jarファイルの利用

```
コマンドプロンプト

Z:\java\myprojects>set CPATH=.;%J2EE_HOME%\lib\j2ee.jar;ConvertAppClient.jar

Z:\java\myprojects>java -classpath %CPATH% ConvertClient
12000.0

Z:\java\myprojects>
```





エンティティBean (EntityBean)

データの永続性管理

EJB Bean自身

- Bean-Managed Persistence (BMP)

コンテナ (EJBサーバ)

- Container-Managed Persistence (CMP)

プライマリキー (PrimaryKey)

- Primarykey: リレーショナルデータベース中のテーブル中の全ての行の違いを識別可能な項目の最小組

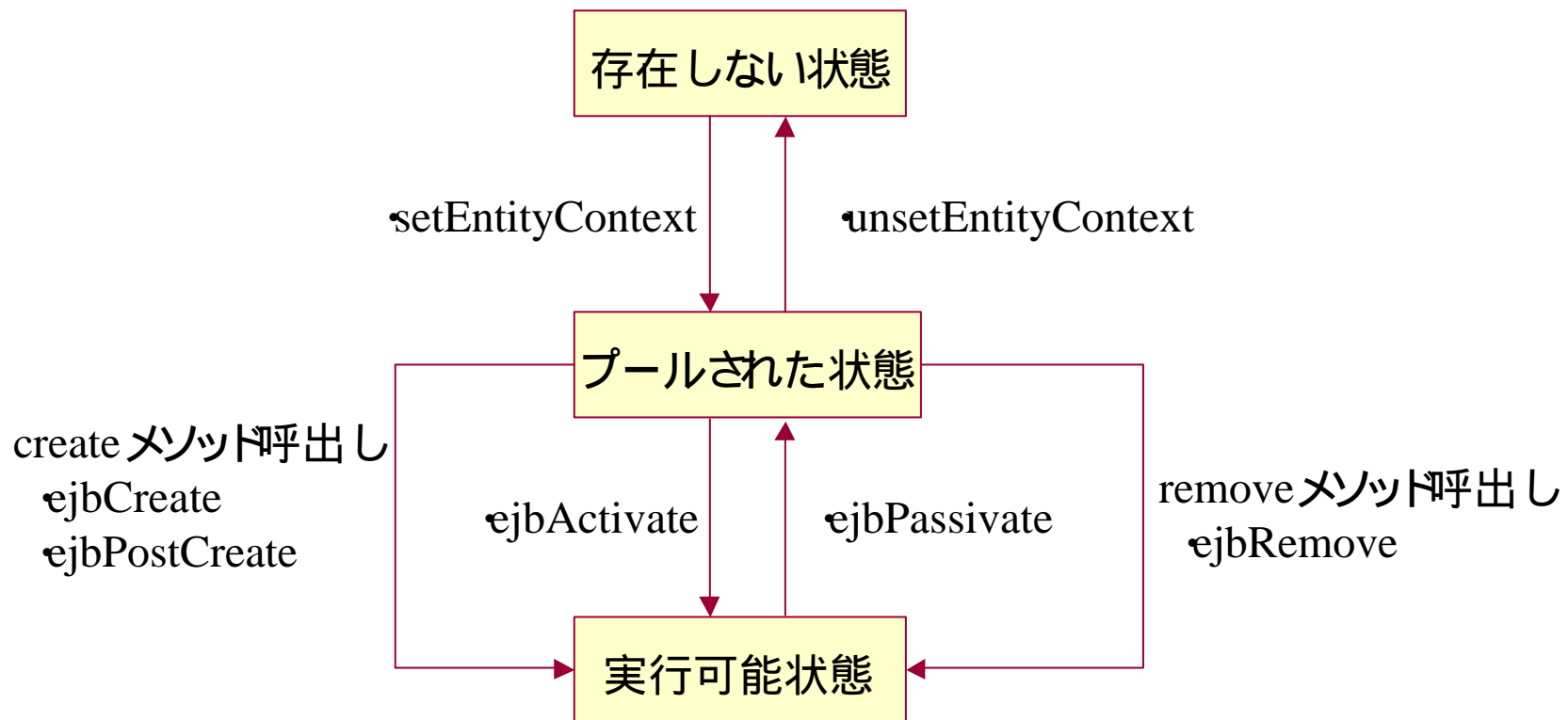
エンティティBeanのfindByPrimaryKeyメソッド

- テーブルの一行を表すエンティティBeanの取得

エンティティBeanのライフサイクル

EJB コンテナとプール

- エンティティBeanはインスタンス化後にプールへ移動
- プライマリキーを伴う呼出しによって実行可能状態へ





エンティティBeanの種類: BMPとCMP

- ✎ BMP (Bean-Managed Persistence)
 - ✎ エンティティビーン永続化処理を自身で実装
 - ✎ EntityBeanインタフェース中の各メソッド実装時にSQL文の記述と処理が必要
 - ✎ #ejbCreate : insert
 - ✎ #ejbRemove : delete
 - ✎ #ejbLoad : select
 - ✎ #ejbStore : update
 - ✎ #ejbFindByPrimaryKey : select
- ✎ CMP (Container-Managed Persistence)
 - ✎ EJBコンテナに永続化処理を一任
 - ✎ 永続化対象フィールドとデータベース中の項目名の対応付けのみ



BMPエンティティBean例(1): データベースとSQL

- ✎ J2EE Tutorial 付属例
<http://java.sun.com/j2ee/tutorial/>
- ✎ リレーショナルデータベース
 - ✎ J2EESDK付属の cloudescape
 - ✎ エンティティBeanに対応するTableの準備

```
CREATE TABLE savingsaccount
(id VARCHAR(3) CONSTRAINT pk_savings_account
PRIMARY KEY,
firstname VARCHAR(24),
lastname VARCHAR(24),
balance NUMERIC(10,2));
```



BMPエンティティBean例(2): インタフェース定義

リモートインタフェース (Remote Interface) 呼ばれるビジネスメソッドの定義

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.math.BigDecimal;
public interface SavingsAccount extends EJBObject {
    public void debit(BigDecimal amount) throws InsufficientBalanceException, RemoteException;
    public void credit(BigDecimal amount) throws RemoteException;
    public String getFirstName() throws RemoteException;
    public String getLastName() throws RemoteException;
    public BigDecimal getBalance() throws RemoteException;
}
```

ホームインタフェース (Home Interface)

```
public interface SavingsAccountHome extends EJBHome {
    public SavingsAccount create(String id, String firstName, String lastName, BigDecimal balance)
        throws RemoteException, CreateException;
    public SavingsAccount findByPrimaryKey(String id) throws FinderException, RemoteException;
    public Collection findByLastName(String lastName) throws FinderException, RemoteException;
    ...
}
```



BMPエンティティBean例(3): Beanの実装

- ✎ エンティティBeanクラス (SavingAccountBean)
- ✎ リモートインタフェースで定義した全メソッドを実装

```
public class SavingsAccountBean implements EntityBean {
    private String id; private String firstName; private String lastName; private BigDecimal balance;
    private EntityContext context; private Connection con;
    private String dbName = "java:comp/env/jdbc/SavingsAccountDB";
    public String ejbCreate(String id, String firstName, String lastName, BigDecimal balance) throws
    CreateException {
        ...
        try {
            insertRow(id, firstName, lastName, balance);
        } catch (Exception ex) { throw new EJBException("ejbCreate: " + ex.getMessage()); }
        this.id = id; this.firstName = firstName; this.lastName = lastName; this.balance = balance; return id;
    }

    private void insertRow (String id, String firstName, String lastName, BigDecimal balance) throws
    SQLException {
        String insertStatement = "insert into savingsaccount values ( ?, ?, ?, ? )";
        PreparedStatement prepStmt = con.prepareStatement(insertStatement);
        prepStmt.setString(1, id); prepStmt.setString(2, firstName); prepStmt.setString(3, lastName);
        prepStmt.setBigDecimal(4, balance);
        prepStmt.executeUpdate(); prepStmt.close();
    }
    public String getFirstName() { return firstName; }
```



BMPエンティティBean例(4): Appクライアント

利用クライアントの開発

JNDIの指定

```
public class SavingsAccountClient {
    public static void main(String[] args) {
        try {
            Context initial = new InitialContext();
            Object objref = initial.lookup("java:comp/env/ejb/SimpleSavingsAccount");
            SavingsAccountHome home =
                (SavingsAccountHome)PortableRemoteObject.narrow(objref,
                    SavingsAccountHome.class);

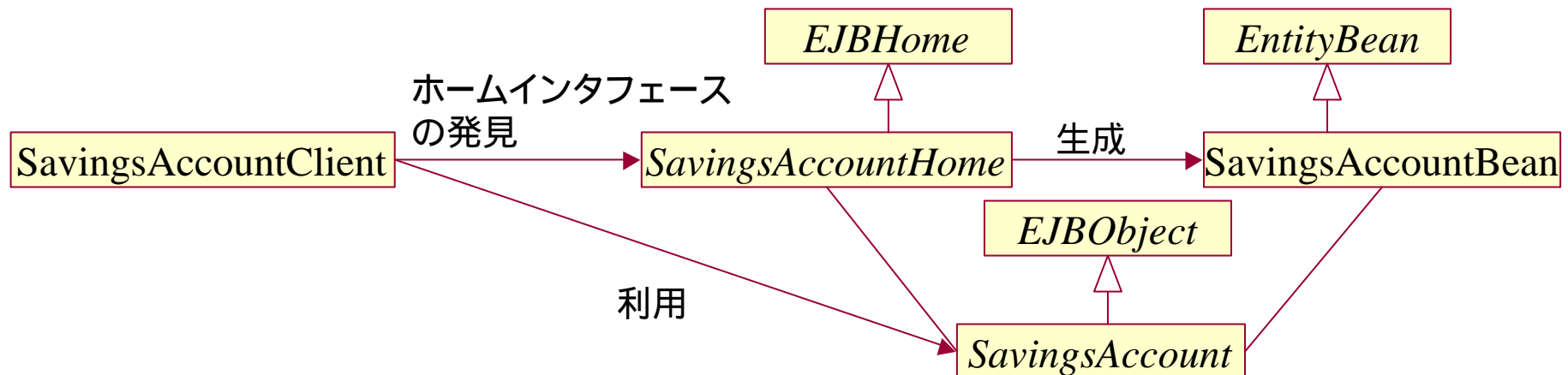
            BigDecimal zeroAmount = new BigDecimal("0.00");
            SavingsAccount duke = home.create("123", "Duke", "Earl", zeroAmount);
            duke.credit(new BigDecimal("88.50"));
            duke.debit(new BigDecimal("20.25"));
            BigDecimal balance = duke.getBalance();
            System.out.println("balance = " + balance);
            duke.remove();
            ...
        }
    }
}
```

BMPエンティティBean例(5): 実行例

利用クライアントの実行

```
CMD.EXE へのショートカット
C:\j2eetutorial\examples\years>set APPCPATH=SavingsAccountAppClient.jar

C:\j2eetutorial\examples\years>runclient -client SavingsAccountApp.ear -name SavingsAccountClient -textauth
ログインを起動しています ...
ユーザ名入力:guest
パスワード入力:guest123
バインドされた名前: `java:comp/env/ejb/SimpleSavingsAccount`
balance = 68.25
balance = 32.55
456: 44.77
730: 19.54
268: 100.07
836: 32.55
456: 44.77
4.00
7.00
```





おわりに

✍ 現状

- ✍ EJBの普及はまだこれから
- ✍ 各社EJBサーバ付属の配備ツールの違いに問題があるとの指摘

✍ 将来

- ✍ エンタープライズBeanが流通する可能性
- ✍ 「EJBコンポーネントに関するコンソーシアム」
<http://www.ejbcons.gr.jp/>