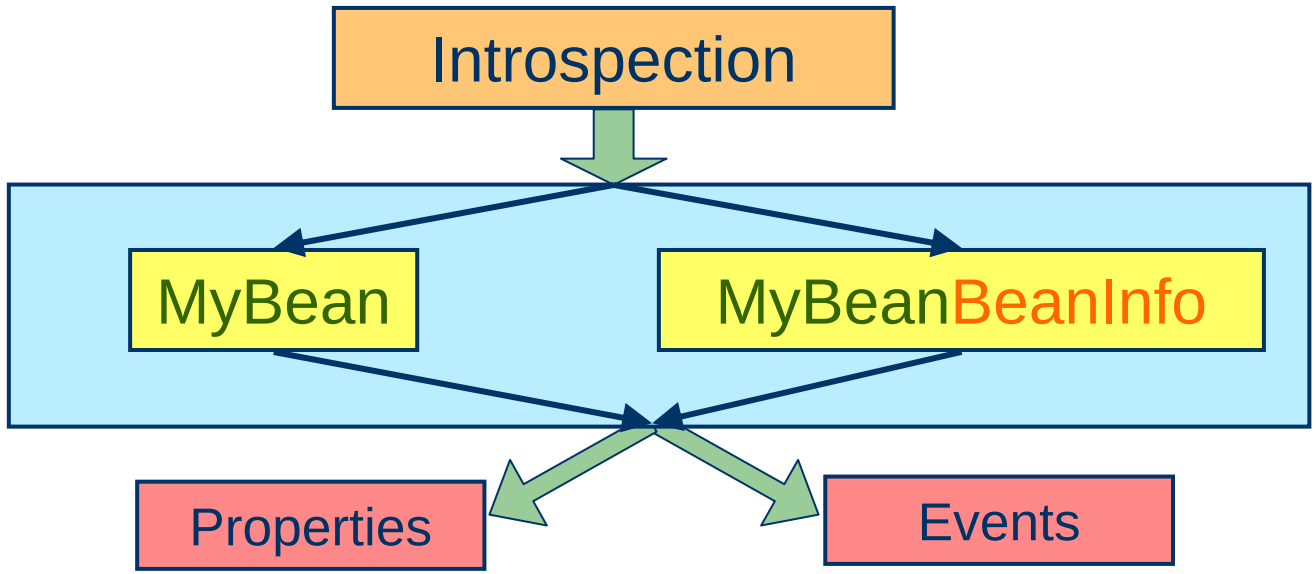
JavaBeans
BeanInfo

© J. Heinzelreiter

WS 2004/05

Introspection

- Das Builder-Tool benötigt *Metainformation* über das Bean (Properties und Events).
- *Introspection* ist der Mechanismus, der diese Metainformation aus dem Bean extrahiert.



Introspection

- Builder-Tool kennt nur Namen des Beans.
- Mithilfe des *Introspectors* wird Bean analysiert.

```
class Introspector {
    ...
    public static BeanInfo getBeanInfo(
      Class beanClass, Class stopClass);
    ...
}
```

- Introspector extrahiert zuerst Bean-Features aus MyBeanBeanInfo.class.
- Restliche Features werden mit *Reflection*-Mechanismus ermittelt (Nameskonventionen).

Bean Customizing

- Interface **`BeanInfo`**
  - Definition der exportierten *Properties*.
  - Definition der exportierten *Events.*
  - Definition der exportierten *Methoden.*
  - Hinzufügen von *Icons.*
  - Hinzufügen von spezifischen *Property-Editoren*.
  - Hinzufügen eines *Customizers*.
- Implementierung
  - Interface **`BeanInfo`** implementieren.
  - Von **`SimpleBeanInfo`** ableiten.
  - Namenskonvention: **MyBeanBeanInfo**

Interface BeanInfo

```
public interface BeanInfo {
  public final static int ICON_COLOR_16x16;
  public final static int ICON_COLOR_32x32;

  public BeanDescriptor getBeanDescriptor();

  public PropertyDescriptor[] getPropertyDescriptors();
  public EventSetDescriptor[] getEventSetDescriptors();
  public MethodDescriptor[]   getMethodDescriptors();

  public int getDefaultPropertyIndex();
  public int getDefaultEventIndex();

  public BeanInfo[] getAdditionalBeanInfo();

  public Image getIcon(int iconKind);
}
```

Methoden von `BeanInfo`

- `getBeanDescriptor`
  - Liefert Referenz auf Bean-*Customizer* (Wizard).
- `getDefaultPropertyIndex`/ `getDefaultEventIndex`
  - Information kann von Builder-Tool genutzt werden.
- `getAdditionalBeanInfo`
  - Zusätzliche Infos von anderen Beans.
  - Z.B.: Verweis auf `BeanInfo` von Basisklasse.
- `getIcon`
  - Liefert Icon zur Darstellung in Komponenten-Palette.

SimpleBeanInfo

- Standardimplementierung aller Methoden von `BeanInfo`.
  - Abgeleitete Klassen müssen nicht alle Methoden implementieren.
- Zusätzliche Hilfsfunktion `loadImage()`

```
class MyBeanBeanInfo extends SimpleBeanInfo {
  public Image getIcon(int iconKind) {
    if (iconKind == BeanInfo.ICON_COLOR_16x16) {
      return loadImage("MyBeanIcon16.gif");
    }
    else if (iconKind == BeanInfo.ICON_COLOR_32x32) {
      ...
    }
    else
       return null;
  }
}
```

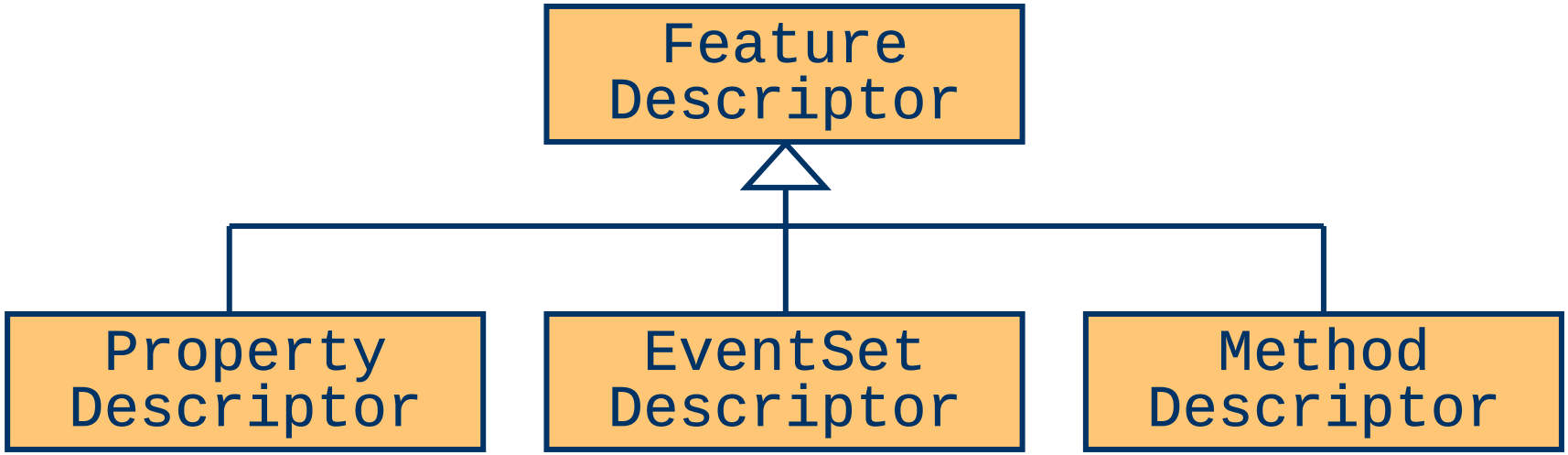
Feature Descriptor (1)

```
public class FeatureDescriptor extends Object {
  public FeatureDescriptor();

  public String      getName();
  public String      getDisplayName();
  public Object      getValue();
  public Enumeration attibuteNames();
  public String      getShortDescription();
  public boolean     isExpert();
  public boolean     isHidden();

  public void setName(String name);
  public void setDisplayName(String name);
  public void setValue(String attrName, Object value);
  public void setShortDescription(String text);
  public void setExpert(boolean expert);
  public void setHidden(boolean hidden);
}
```

Beschreibung des Bean-Interfaces



- Explizite Beschreibung des Bean-Interfaces
  - Features können explizit definiert werden.
  - Feature-Attribute können explizit vergeben werden
    - Vererbte Features können z.B. versteckt werden.

Feature Descriptor (2)

- `name`
  - Interner Name des Features.
- `displayName`
  - Angezeigter Name des Features im Builder-Tool.
- `expert/hidden`
  - Soll Feature in Builder-Tool dargestellt werden?
- `shortDescription`
  - Kurzbeschreibung (z.B. für Tooltip in Builder-Tool).
- `setValue/getValue/attributeNames`
  - Verwalten von Attributen und Attributwerten

PropertyDescriptor

```
public class PropertyDescriptor
                    extends FeatureDescriptor {
  public PropertyDescriptor(String propName, Class bean);
  public PropertyDescriptor(String propName, Class bean,
                    String getterName, String setterName);
  public PropertyDescriptor(String propName,
                    Method getter, Method setter);

  public Class   getPropertyType();
  public Method  getReadMethod();
  public Method  getWriteMethod();
  public boolean isBound();
  public boolean isConstrained();
  public Class   getPropertyEditorClass(...);

  public void setBound(boolean bound);
  public void setConstrained(boolean constrained);
  public void setPropertyEditorClass(...);
}
```

PropertyDescriptor – Beispiel

```
class ClockBeanInfo extends SimpleBeanInfo {

   public PropertyDescriptor[] getPropertyDescriptors() {
     try {
       PropertyDescriptor[] props = {
         new PropertyDescriptor("noTimeOuts",
                                Timer.class);
         new PropertyDescriptor("timeOutInterval",
                                Timer.class,
                                "getTimeOut“,
                                "setTimeOut")
       }
       props[0].setBound(true);
       props[1].setExpert(true);
       return props;
     }
     catch (IntrospectionException) { ... }
   }
 }
```

EventSetDescriptor

```
public class EventSetDescriptor
                    extends FeatureDescriptor {

  public EventDescriptor(Class bean,
                         String eventSetName,
                         Class listener,
                         String[] listenerMethodNames,
                         String addListenerMethodName,
                         String removeListenerMethodName);
  ...
  public Class    getListenerType();
  public Method[] getListenerMethods();
  public Method   getAddListenerMethod();
  public Method   getRemoveListenerMethod();

  public boolean  isUnicast();
  public void setUnicast(boolean unicast);
}
```

EventSetDescriptor – Beispiel

```
class ClockBeanInfo extends SimpleBeanInfo {

  public EventSetDescriptor[] getEventSetDescriptors () {
    try {
       String[] listenerMethods {"clockTicked",
                                 "timerExpired");
       EventSetDescriptor[] events = {
         new EventSetDescriptor(Timer.class,
                                "ClockEvents",
                                TimerListener.class,
                                listenerMethods,
                                "addTimerListener",
                                "removeTimerListener");

       return events;
    }
    catch (IntrospectionException) { ... }
  }
 }
```

Java Archives (JAR)

- Paket aller Files, die zur Laufzeit benötigt werden.
- Anwendung
  - Applets
  - JavaBeans
- JAR-File beinhaltet
  - die Bean-Klasse,
  - die `BeanInfo`-Klasse,
  - Hillfklassen,
  - Customizer-Klasse,
  - PropertyEditor-Klasse,
  - Icons (gif-Files),
  - Ressourcen.

Das JAR Command-Line-Tool

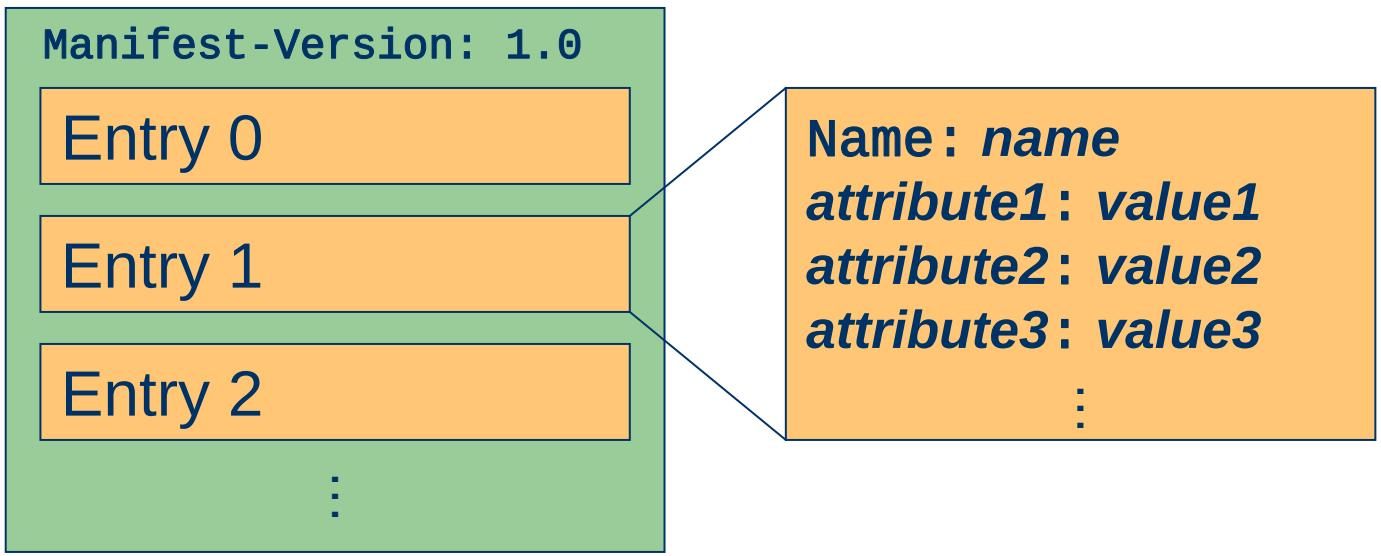
- `jar` (`c`|`t`|`x`) [`vfmOM`] [*jar-file*] [*manifest*] *files* ...
  - `c`: *create* - Archiv erzeugen,
  - `t`: *list* - Inhaltsverzeichnis ausgeben,
  - `x`: *extract* - entpacken der Files,
  - `f`: *file* - Filename des Archivs,
  - `m`: *manifest*, Manifest-File hinzfügen.
- Beispiel – Erzeugen eines JAR-Files
  - `jar cvmf MyBean.jar MyBean.mf MyBean.class Support.class MyBean.gif`
- Beispiel – „Ausführen“ eines JAR-Files
  - `java –jar JavaApplication.jar`

Erzeugung eines JAR-Files mit Ant

```
<project name="MyBeans"
  default="makejar" basedir=".">
  <target name ="makejar">
    <jar jarfile="AdvancedBeans.jar"
      basedir="bin"
      includes="beans/**/*.class beans/**/*.gif"
      manifest="src/meta-inf/manifest.mf"
    />
  </target>
</project>
```

Der Manifest-File

- JAR-File wird im ZIP-Format komprimiert
  - Kann mit Tools wie WinZip bearbeitet werden
- JAR-File enthält einen Manifest-File
  - Name: `META-INF/MANIFEST.MF`



JavaBeans – Manifest-File

- Beans müssen durch das Attribut/Wert-Paar `Java-Bean: True` markiert werden.
- Beispiel:

```
Manifest-Version: 1.0

Name: MyBean.class
Java-Bean: True

Name: MyBeanInfo.class
Name: MyBean.gif
```

- `jar`-Tool fügt zusätzliche Attribut/Wert-Paare zur Ermittlung der Prüfsumme hinzu.