JavaBeans and InfoBus: a Tutorial

Ghodrat Moghadampour

ghodrat.moghadampour@puv.fi

Department of Information Technology

Vaasa Polytechnic

Wolffintie 30

65200 Vaasa, Finland

Outline

- Introduction
- JavaBeans
  - Bean Terminology
  - JAR Files and Utility
  - Manifest Files
- Creating a New JavaBean
- InfoBus

What is a JavaBean?

- A reusable software component
- A simple piece of software for checking the spelling of a document, or a complex one for forecasting the performance of a stock portfolio.
- Visible to the end user, like a button on a graphical user interface or invisible to the user, like a software to decode a stream of multimedia information in real time.

What is a JavaBean?

- It may work autonomously on a user’s workstation or work in cooperation with a set of other distributed components.
- A Bean that provides real-time price information from a stock or commodities exchange would need to work in cooperation with other distributed software to obtain its data

What is a JavaBean?

- A simple Java object becomes a Java bean when all of the object’s data fields are **private** and are only **accessible through methods,** known as **accessor methods**.

Advantages of Java Beans

- Mainly provides standard mechanisms to deal with software building blocks.
- A Bean obtains all the benefits of Java’s “write-once, run-anywhere” paradigm.
- The **properties**, **events**, and **methods** of a Bean exposed to an application builder tool can be controlled.

Advantages of JavaBeans cont.

- The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
- A Bean may register to receive events from other objects and can generate events that are sent to other objects.

Bean Terminology

- A JavaBean is defined via its interface: its **properties**, its **events** and its **methods**.
- **Properties**: attributes of the Bean that can be modified by anything outside the Bean, like size, color, etc.
- **Events**: used to allow one component to communicate with another component
- **Methods**: public methods that can be used to directly request some service to a Bean.

Introspection

- The process of analyzing a Bean to determine its capabilities.
- It allows an application builder tool to present information about a component to a software designer.
- Without introspection, the Java Beans technology could not operate.
- There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool:

Introspection cont.

- With the first method, simple naming conversion are used.
- In the second way, an additional class, which inherits `SimpleBeanInfo`, is provided that explicitly supplies this information.

Design Patterns for Properties

- A property is a subset of a Bean’s state. The values assigned to the properties determine the behaviour and appearance of that component.
- There are three types of properties: **simple**, **Boolean**, and **indexed**.

Simple Properties

- A simple property has a single value, like:

```
public void setP(T arg);
public T getP();
```

Simple Properties cont.

- Example 1:

```
class Guest {
  private String name = new String();
  private int roomnro;
  public void setName(String n){
  name=name.concat(n);
}
```

Simple Properties cont.

- Example 1 cont.:

```
 public void getName(){
   return name;
  }
 public void setRoomnro(int n) {
  roomnro=n;
 }
 public int getRoomnro() {
       return roomnro;
} }
```

Boolean Properties

- A Boolean property has a value of true or false, like:

```
public boolean isP();
public boolean getP();
public     void     setP(boolean
 value);
```

Boolean Properties cont.

- Example 2:

```
public class Guest {
private boolean present=true;
public boolean isPresent() {
 return present;
}
public  void  setPresent(boolean
 pr) {
 present=pr;}}
```

Indexed Properties

- An indexed property consists of multiple values, like:

```
public T getP(int index);
public void setP(int index, T
  value);
public T[] getP();
public void setP(T values[]);
```

Indexed Properties cont.

- Example 3:

```
public class Individual {
   private int data [];
   public void setData(int index,
   int value){
    data[index]=value;}
   public int getData(int index) {
    return data[index];}
   public int [] getData() {
    return data;}
```

Indexed Properties cont.

- Example 3 cont.:

```
public void setData(int []
  values){

  data = new int[values.length];
  System.arraycopy(values, 0,
   data, 0,
   values.length);
}}
```

JAR Files

- Tools such as the BDK expect Beans to be packaged within JAR files.
- A JAR file allows to efficiently deploy a set of classes and their associated resources.
- JAR technology makes it much easier to deliver and install software.
- The elements in a JAR file are compressed
- Digital signatures may also be associated with the individual elements in a JAR file (keytool).

JAR Files cont.

- This allows a consumer to be sure that these elements were produced by a specific organization or individual.

Manifest Files

- A manifest file indicates which of the components in a JAR file are Java Beans, like:

```
Name: sunw/demo/BeanEx/pic1.gif
Name: sunw/demo/BeanEx/pic2.gif
Name: sunw/demo/BeanEx/pic3.gif
Name: sunw/demo/BeanEx/BExCls.class
Java-Bean: True
```

The JAR Utility

- JAR utility is used to generate a JAR file. Its syntax is:

  jar *options* files

- , where options can be one or several of the followings:

  Option Description

  c A new archive is to be created.

  C Change directories during command execution.

The JAR Utility cont.

`f` The first element in the file list is the name of the archive that is to be created or accessed.

`m` The second element in the file list is the name of the external manifest file.

`M` Manifest file not created.

`t` The archive contents should be tabulated.

`u` Update existing JAR file.

The JAR Utilities cont.

v Verbose output should be provided by the utility as it executes.

X Files are to be executed from the archive. (If there is only one file, that is the name of the archive, and all files in it are extracted.

0 Do not use compression.

Creating a JAR File

- To create a JAR file named **myjarfile.jar** that contains all of the **.class** and **.gif** files in the current directory we write:

```
jar cf myjar.jar *.class *.gif
```

- If a manifest file such as **myman.mf** is available, it can be used with the following command:

```
jar cfm myjar.jar myman.mf *.class
*.gif
```

Creating a New Bean

- Steps for creating a new Bean are:
  1. Create a directory for the new Bean.
  2. Create the Java source file(s).
  3. Compile the source file(s).
  4. Create a manifest file.
  5. Generate a JAR file.
  6. Start the BDK.
  7. Test.

Creating a New Bean cont.

- We first create a directory like: **c:\bdk\demo\sunw\demo\colors** and move to it**.**
- We also set the **CLASSPATH** to **C:\bdk\demo.**
- We then create our java source code file and compile it, for instance:

```
 javac Colors.java
```

Creating a New Bean cont.

- The **colors.mft** manifest file is created under **c:\bdk\demo** directory, where the manifest files for the **BDK** demos are located.

```
Name:sunw/demo/colors/Colors.
class
 Java-Bean: True
```

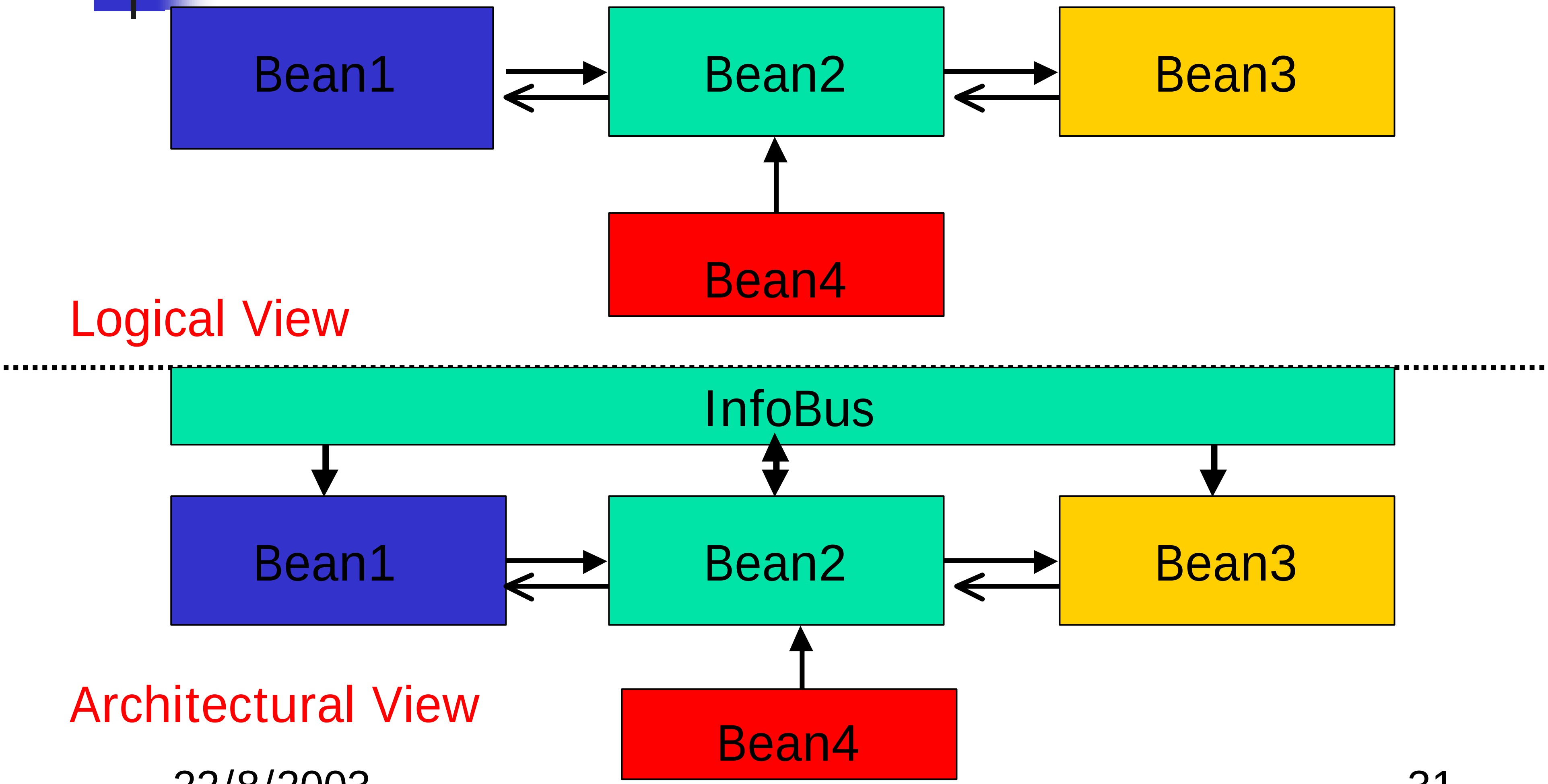
- Beans are included in the **ToolBox** window of the **BDK** only if they are in JAR files in the directory **c:\bdk\jars**.

```
 jar cfm ..\jars\colors.jar
   colors.mft
   sunw\demo\colors\*.class
```

InfoBus

- Two or more JavaBeans can dynamically exchange data through the Information Bus a.k.a. InfoBus.
- However, communicating Beans must implement required interfaces defined by InfoBus.
- The InfoBus is a Java API created by Lotus Development Corporation and Sun Microsystems's JavaSoft division.

InfoBus Structure



Bean1

Bean2

Bean3

Bean4

Logical View

InfoBus

Bean1

Bean2

Bean3

Bean4

Architectural View

InfoBus cont.

- Communicating Beans can be located in a Java application or on a Web page.
- We can distinguish three different roles in an InfoBus:
- **Data producers**: Beans mainly responsible for accessing data from their native store, such as files, DBMS, etc.
- **Data consumers**: Beans responsible for retrieving data from the bus for analysis or visual display.

InfoBus cont.

- **Data controllers**: an optional component that regulates or redirects the flow of events between data producers and consumers.
- A JavaBean can be both a consumer and producer.

InfoBus Communication Protocol

- **Membership**: Any Java class can join the InfoBus provided that it implements the InfoBusMember interface.
- **Rendezvous**: An InfoBus application supplies an object that implements `InfoBusDataProducer` or `InfoBusConsumer` interfaces to listen for events appropriate to a component’s role as a producer or consumer.

InfoBus Communication Protocol cont.

- **Data access**: InfoBus specifies a number of standard interfaces to provide direct data transfer between a producer and consumer:
  - **ImmediateAccess**: provides an InfoBus wrapper for a simple data item
  - **ArrayAccess**: provides access functions for an array with arbitrary dimensions
  - **RowAccess**: provides a row and column interface to support database solutions

InfoBus Communication Protocol cont.

- **Change notification**: a consumer, which receives data from a producer, can request notifications of all changes to the data by registering a `DataItemChangeListener` on the data item. As the producer detects changes, it will announce the changes to all listeners.

Implementing InfoBusMember

- Example 4:

```
public class infobusDemo extends Applet
  implements InfoBusMember,
   InfoBusDataProducer, ActionListener {

//IBMS holds our InfoBus
private InfoBusMemberSupport IBMS;
//data is a simple data item String
//data is the name of the InfoBus to
  which we connect
private SimpleDataItem data;
```

Implementing InfoBusMember

```
//The name of the InfoBus to which we
  connect
private String bus=null;
privte String guest;
private Object available = new
  Object();

//Delegates all calls to our
  InfoBusMemberSupport, IBMS
public InfoBus getInfoBus() {
   return IBMS.getInfoBus();
 }
```

Implementing InfoBusMember

```
//The InfoBusMemberSupport instance
  must be created before any class
  are delegated to it. This can be
  done in the init() method.

public void init() {
   super.init();
IBMS=new
          InfoBusMemberSupport(this);
```

Implementing InfoBusMember

```
IBMS.addInfoBusPropertyListener(this);
bus=getParameter("InfoBusName");
guest=getParameter("DataItemName");
   if(guest==null)
      guest="Guest";
  if(bus != null)
     IBMS.joinInfoBus(bus);
  else
    IBMS.joinInfoBus(this);}
```

References

- http://java.sun.com/products/javabeans/software/bdk_download.html
- http://java.sun.com/products/javabeans/faq/faq.help.html#Q28
- http://java.sun.com/beans/infobus