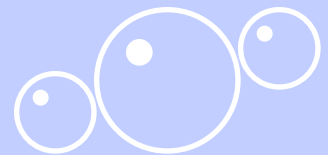
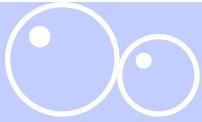


Software Component Model

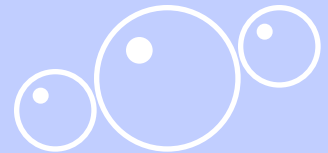
Java Beans vs. ActiveX

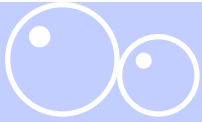




Overview

- Software Component Model
- ActiveX
- Java Beans

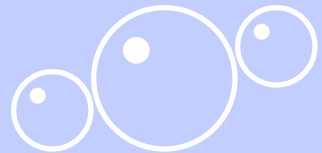


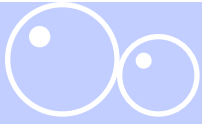


Overview

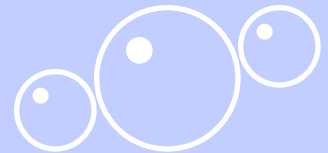
> Software Component Model

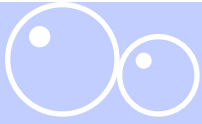
- ActiveX
- Java Beans



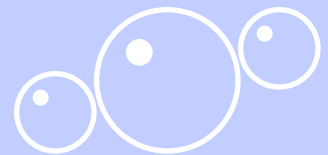


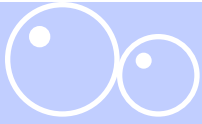
- Definition
- Why?
- Three Major Elements
 - Components
 - Containers
 - Scripting





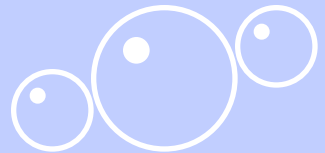
- Component-based software Services
 - Interface Publishing and Discovery
 - Event Handling
 - Persistence
 - Layout Control
 - Application Builder Support

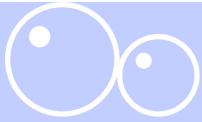




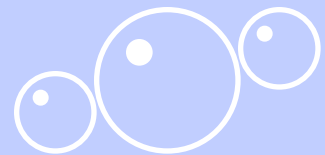
Overview

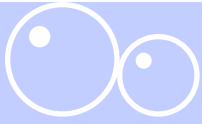
- Software Component Model
 - > ActiveX
- Java Beans



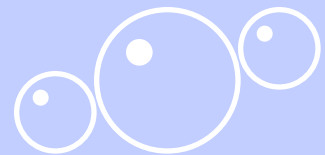


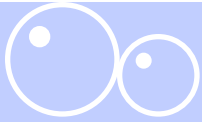
- What is ActiveX?
- Elements of ActiveX
 - Active control
 - Active Document
 - Active Scripting
 - ActiveX Server Framework
 - Java Virtual Machine





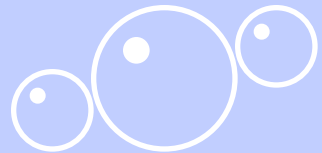
- Platform Support
 - 32-bit Windows platform only, namely, Windows 3.x, Windows 95, Windows NT
 - Macintosh, beta testing started in Oct. 96
 - UNIX, scheduled for first quarter of 97
- Security
 - Depends on Digital signature on Components
 - No limitation on the functionality accessible

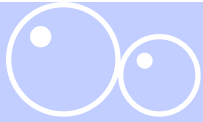




Overview

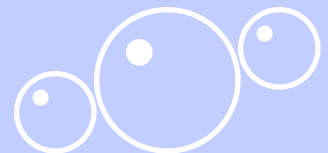
- Software Component Model
- ActiveX
- > Java Beans

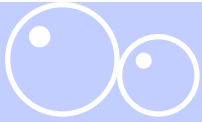




Goal

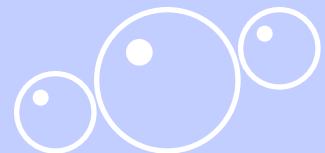
- “To Define a software component model for Java, so that third party ISVs can create and ship Java components that can be composed together into applications by the end users.”

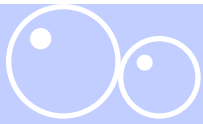




What is a Bean?

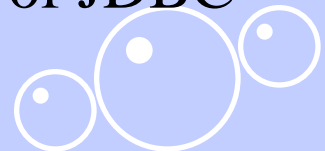
- Initial Definition
 - “A Java bean is a reusable software component that can be manipulated visually in a builder tool.”
- Typical features of a Java Bean
 - Introspection, Customization, Events, Properties, and Persistence
- Examples of a Java Bean
 - Simple GUI elements like buttons and sliders
 - Database viewers

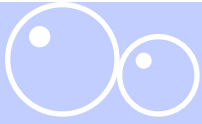




Java Beans vs. Class Libraries

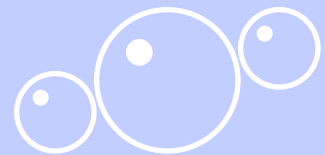
- Beans are appropriate for software components that can be visually manipulated and customized to achieve some effect
- Class libraries provide functionality that is useful to programmers but doesn't benefit from visual manipulation
- Example
 - Provide the JDBC database access API as a class library
 - Provide database access beans on top of JDBC

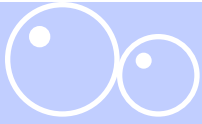




3 important features of Beans

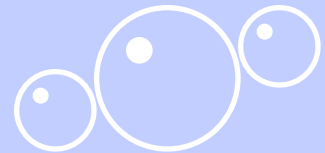
- **Properties:** Named attributes exposed by a bean that can be read or written by calling appropriate methods on the bean
- **Methods:** Normal Java methods which can be called from other components or from a scripting environment
- **Events:** Provide a way for one component to notify other components that something interesting has happened

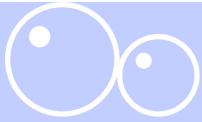




Design Time vs. Run-Time

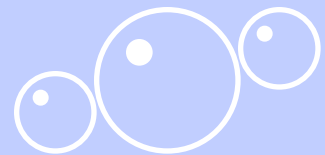
- Within a design environment, ie. builder tool, the bean should provide design information to the builder tool and allow the end-user to customize the appearance and behavior of the bean
- Each bean must be usable at run-time within the generated application
- There is a clear split between the design-time aspects of a bean and the run-time aspects, so that a bean at run-time can be deployed without needing to download all its design time code.

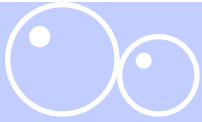




Invisible Beans

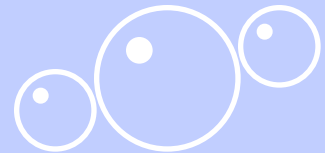
- Beans that have no GUI representation but should still be editable in a GUI builder
- Able to call methods, fire events, save persistent state, etc.
- Can be used to share resources within GUI applications or as components in building server applications that have no GUI appearance at all
- Some beans may be able to run with or without GUI appearance depending where they are instantiated.

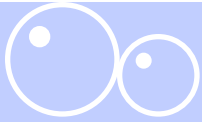




Persistence

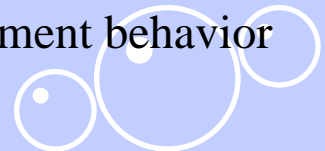
- Why?
 - Needs to store away appropriate parts of its internal state so that it can be resurrected later with a similar appearance and behavior.
- Java Object Serialization Mechanism
 - Provide an automatic way of storing out and restoring the internal state of a collection of Java objects
- “Externalization” Mechanism
 - Allow object complete control over the writing of their state, so that they can choose to mimic arbitrary existing data formats.

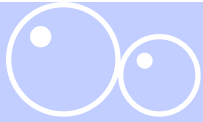




Events

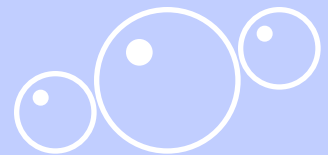
- Event notifications are propagated from sources to listeners by Java method invocations on the target listener objects
- Each distinct kind of event notification is defined as a distinct Java method. These methods are then grouped in `EventListener` interfaces that inherit from `java.util.EventListener`
- Event listener classes identify themselves as interested in a particular set of events by implementing some set of `EventListener` interfaces
- The state associated with an event notification is normally encapsulated in an event state object that inherits from `java.util.EventObject` and which is passed as the sole argument to the event method
- Event sources identify themselves as sourcing particular events by defining registration methods that conform to a specific design pattern and accept references to instances of particular `EventListener` interfaces
- In circumstances where listeners cannot directly implement a particular interface, or when some additional behavior is required, an instance of a custom adaptor class may be interposed between a source and one or more listeners in order to establish the relationship or to augment behavior

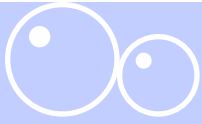




Events

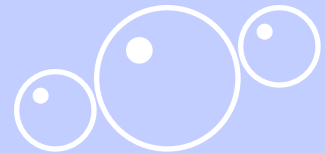
- Unicast/Multicast
 - Normal event delivery is multicast
 - Unicast is also permitted for event registration

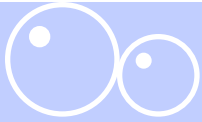




Properties

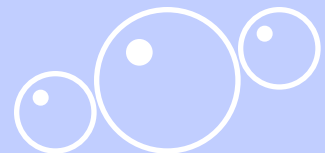
- May be exposed in scripting environments as though they were fields of objects
- Can be accessed programmatically by other components calling their getter and setter methods
- As part of the process of customizing a component, its properties may be presented in a property sheet for a user to edit
- A bean's properties will be persistent

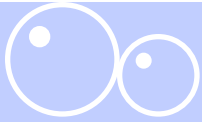




Customization

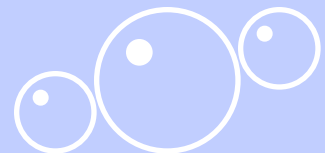
- Two different ways for customization
 - Construction of a GUI property sheet and providing a property editor for each property
 - Inclusion of a customizer class that control the customization of the bean within a bean

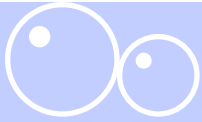




Packaging

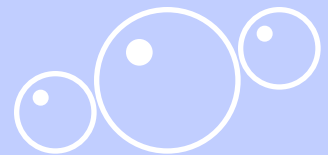
- Goals and non goals
 - Specify the format and conventions needed so that a packaged Java Bean can be used by tools and applications
 - A format to be used by an application to store the Java Bean it uses is not prescribed

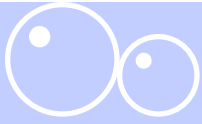




Packaging

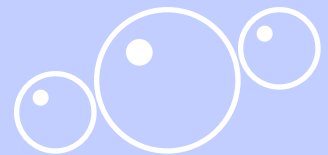
- Based on two technologies available in JDK1.1
 - Java object serialization
 - JAR files

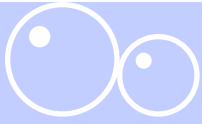




Development tools

- Borland's JBuilder
- IBM's AppletAuthor
- IBM's Visual Age for Java
- Penumbra's Mojo
- Sunsoft's Project Studio
- Sunsoft's Java Workshop
- Sybase's "Jato"
- Symantec's Visual Cafe





Future Direction

- Menubar Merging
- Extended Persistence Support

