

Object – Oriented Design with UML and Java

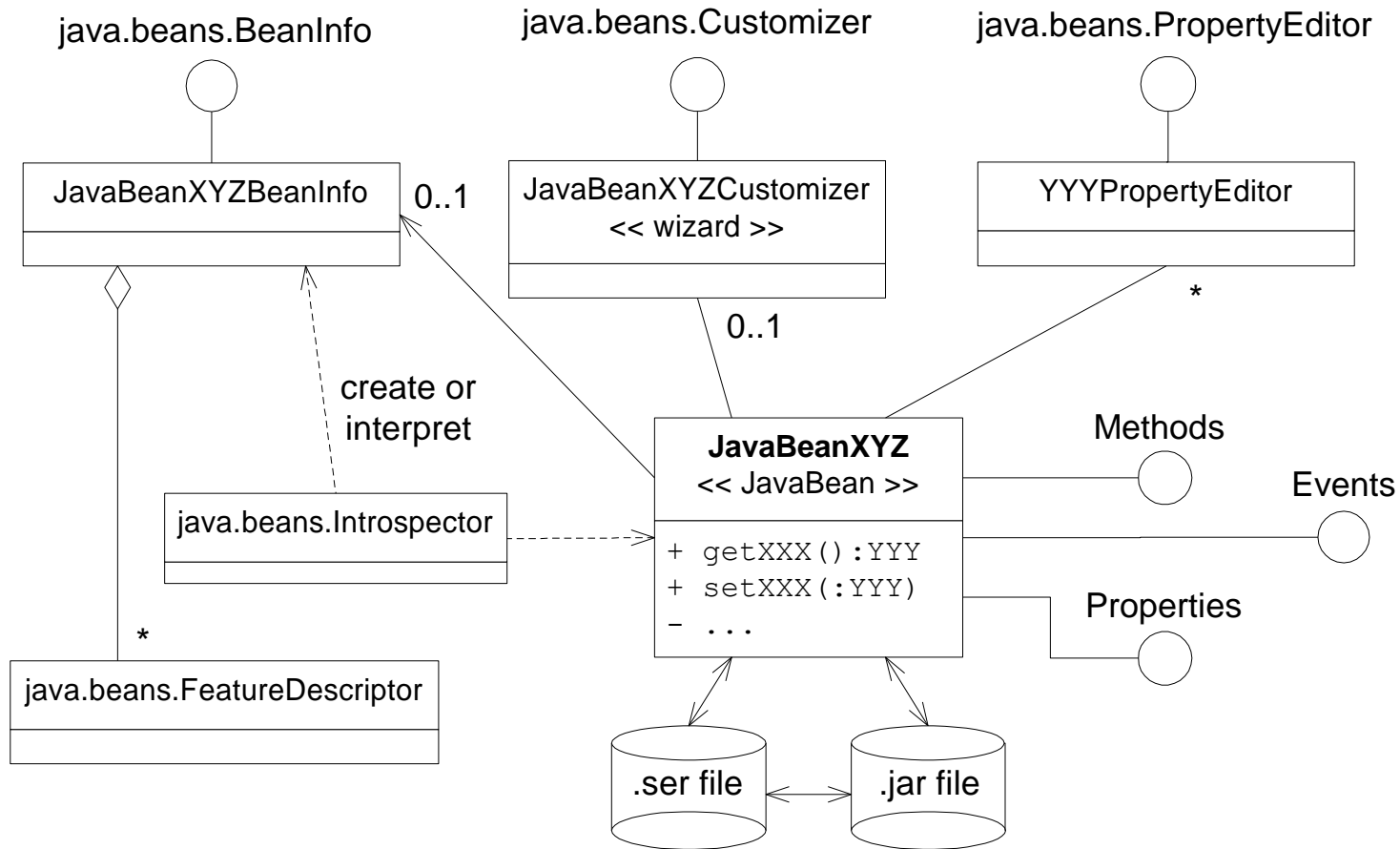
Part XVII - (Enterprise) Java Beans

JavaBeans

A *JavaBean* (pre-*Enterprise*) is a reusable component bundled with supporting files (in a `.jar` file) that adheres to the *JavaBean* standard.

- Note the word “*JavaBean*” has two meanings.
- The official definition from Sun Microsystems: “A Java Bean is a reusable software component that can be manipulated visually in a builder tool.”
- JavaBeans are used primarily on the client side of a distributed application. They are often visual components, like, say, for a Spreadsheet or Calendar widget. But they can be “invisible” and still be quite useful, like, say a proxy for a legacy system, or a PGP Bean, providing encryption services.
- Reflection allows the Bean’s properties to be determined at “customization time” (design time) by a tool.

JavaBeans



The JavaBean Standard

In order for a Java object to be a JavaBean:

- The Bean's configurable properties have public “getter and setter” methods. For example, the property “String foo” must have the methods `getFoo()` and `setFoo()`. (This naming convention is referred to as “design patterns”).
- This Bean's companion *Introspector* determines the configurable properties using Java's Reflection API. The Introspector constructs a *property sheet* for the Bean, which can be edited using visual tools at design time.
- The Bean might have a companion *BeanInfo* object that describes the Bean's configurable properties; if a BeanInfo object exists, the Introspector will use it instead of using reflection.
- The Bean might have a companion *Customizer* object, which is Java code (adhering to a standard interface) designed to assist the user with Bean customization at design time. Within the Bean Box tool, the Customizer acts like a “wizard” to guide the bean configuration use case and/or constrain the range of possible values for the Bean's properties.

JavaBeans

- The fact that the Bean can be customized at design /deployment time implies that the Bean must have a means to “remember” the values for all of the customizable properties. This is accomplished using *Serialization*. A persistent Bean is saved to a `.ser` (Serialized) file, deployed with the Bean inside of a `.jar` (Java Archive) file. This is an example of the *Memento* design pattern.
- Individual Bean properties can have their own custom PropertyEditors. For example, a Color object is often described by three integers representing red, blue, and green; a visual ColorEditor can be provided.
- Bean properties may be “indexed”, “bound” or “constrained”... *Indexed* properties have array values, and it is expected that the Bean provides methods to get and set individual elements of the array. *Bound* properties, when updated, will cause the appropriate Bean *Observers* to get notified. These observers have the right to “veto” proposed updates of *constrained* properties.

JavaBeans

To summarize, JavaBeans have the following core features:

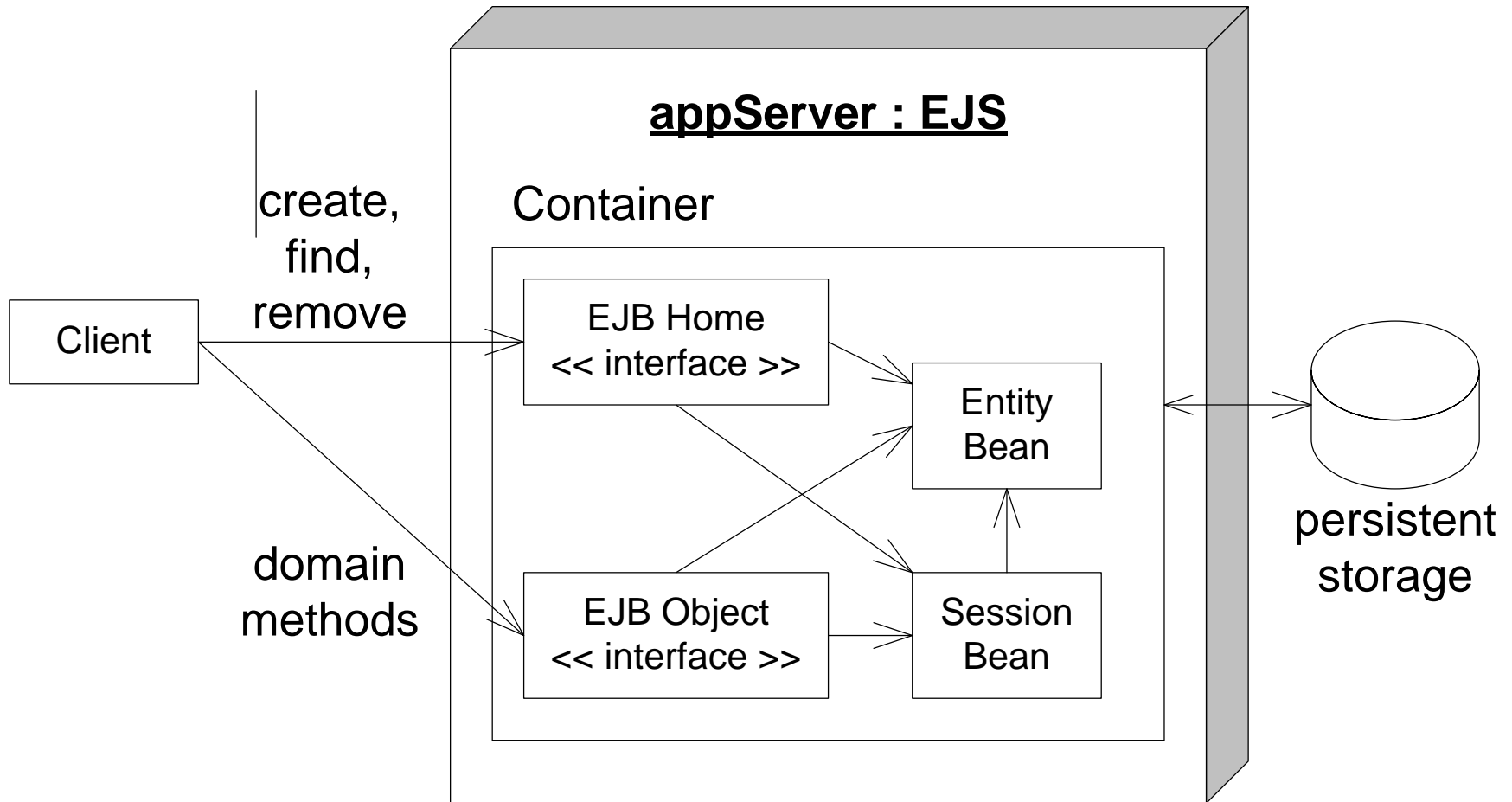
- Ability to be manipulated visually in a builder tool.
- Introspection, to allow the Bean's properties to be determined by the tool.
- Customization, of both appearance and behavior.
- Event handling, using the *Observer* design pattern (listener model).
- Properties, for customization and other programmatic uses.
- Persistence, allowing the Bean's state to be stored for later retrieval.

Design-time customization code is NOT deployed with a run-time Bean.

Enterprise Java Beans (EJB)

- Powerful *Framework* for building fast, scalable, and secure servers.
- Configured with an XML deployment descriptors, or *annotations*.
- Part of the **Java *Enterprise Edition***, along with
 - Java Server Pages (JSP) / Java Server Faces (JSF2)
 - Container-Managed Persistence (CMP)
 - Servlets , Portlets, and more...
 - Java Naming and Directory Interface (JNDI)
 - Java Management Extensions (JMX) Managed Beans (MBeans)
 - Java Message Service (JMS)
 - Other API specifications: JDBC, RMI, XML, Web Services, ...
- EJB 1.0 was klunky and not well liked.
- EJB 2.0 was klunky but significantly better.
- EJB 3.0 is good ☺

EJB2 Architecture



EJB3 Framework

- Good for creating server-side applications that are:
 - Secure
 - Scalable
 - Persistent
 - Distributed
 - Configurable
 - Transactional

EJB3 provides a consistent component architecture for creating distributed middleware and applications, with minimal dependencies in your code to the framework.

Much of your code can be implemented as *POJOs*.

The revolutionary feature in Java that enabled EJB3 is the *annotation*.

There are 3 types of EJBs

Session - web-site user session

Entity - persistent business entity

Message-driven - intra-enterprise message

Session Beans

- Beans that represent *mediators, commands, or controllers*
- Define the scope of transactions involving multiple entity beans
- Two basic types:
 - **Stateless**
 - » Have no internal state
 - » Because of the fact that they are stateless, they can be pooled to service multiple clients
 - **Stateful**
 - » Possess internal states
 - » There can be only one **Stateful Session Bean** per EJB Client
 - » Must be thread safe or thread isolated.
 - » Frameworks can manage *session-scoped* variables, handy for stateful web-site user sessions (often mapping to a session-id in a browser cookie).

Entity Beans

- Entity Beans are business domain objects
- Can be a POJO (Plain Old Java Object) *annotated* with `@Entity`
- Stateful
- Persistent
 - can survive system shutdowns
 - can be shared by multiple EJB Clients
- There are two types of entity bean persistence
 - Container-Managed Persistence (CMP)
 - Bean-managed persistence – the Bean has code to map itself to some persistent storage, usually using JDBC.

`javax.persistence.EntityManager` has an interface to find, create and remove entity beans from the data store.

Message-Driven Beans

Ala the **Java Message Server** (JMS) standard.

The most useful tool for **Enterprise Application Integration**?

Commercial **Message Queues** are good.

Beyond the basic idea of messages on message queues, there are patterns:

- Producer – Consumer
- Publish – Subscribe
- Delivery Channels
- Persistent SOAP

Container-Managed Persistence (CMP)

The EJB container is responsible for saving the Bean's state.

- The container generates all the database calls (or however it chooses to manage the bean's persistence).
 - The implementation is independent of the data source.
-
- The EJB 3.0 CMP implementation is nearly identical to work done on the *Hibernate* project. See also the *Spring* framework.
 - May be configured using XML or annotations (see chapter XVIII).
 - CMP using annotations is a huge leap forward in simplifying the code required to implement object-to-relational mapping.

The Spring Framework

The **Spring Framework** provides sub-frameworks:

1. **Inversion of Control (IoC)** – Configure components using **dependency injection**. Manage the lifecycles of EJBs.
2. **Aspect-oriented programming (AOP)** – for cross-cutting concerns such as exception handling, distributed transactions, logging, and security.
3. Support for **object-relational mapping** (with annotations).
4. Support for enterprise queuing using **Java Message Service (JMS)**.
5. **Java Server Faces (JSF2)** can be the View technology for **Spring MVC** (a quality web-application framework).
6. Spring makes it easier to test your code.

Further discussion of *Java Enterprise Edition* is out of scope for this course.

Enterprise Architecture

EJB3 technology is powerful, but it is not the only tool in your toolkit.

- Commercial middleware *components* are configurable.
- Free and Open Source Software (FOSS) has come of age.
- *Python* is a better language for many tasks related to production deployment, scripting, and small jobs that need to be done quickly.
- The key is to *design* an elegant solution using “best fit” technologies.
- Keep it simple, and use “the right tool for the right job.”