EJB 2 Final thoughts

- Not object-oriented
  - Data and operations separated
  - Entity beans encapsulate data (DB record-oriented)
  - Session beans encapsulate functionality (no data)
- No component inheritance (EJB 2.0)
  - EJB 3.0 – Beans are POJOs
    - Component inheritance implemented as classical OO inheritance
- Development and Application
  - Strict architecture
  - Complex
  - Beans are difficult to test
    - Container required
    - Deployment errors mistaken for Business Logic errors

EJB 3

- EJB 3 – POJO configurable through metadata annotations
  - Remote interface
    - Bean’s business methods
    - Amenable to Reflection, metadata analysis
  - Bean class
    - Business logic

\[ \text{J} + \text{Annotations} = \text{EJB} \]

EJB 3 Architecture

EJB 3 Bean Types

- Session
  - Stateless
  - Stateful
- Message-Driven

- Entity = JPA Entity
EJB 3 Entities (1)

- **Java Persistent API Entities (1)**
  - **Not EJB Beans**
    - Not distributed objects, i.e., not EJB components
    - Have no access to Container services, e.g., transactions, etc.
    - Clients must always be local to JVM
    - No remote access
    - Mitigated by Facade design pattern
  - Can run outside EJB container
    - EJB Container
    - Web Container
    - Java SE Applications

EJB 3 Entities (2)

- **Java Persistent API Entities (2)**
  - **POJOs**
    - Fully Object-oriented
      - Can encapsulate both data and operations
    - Cf. EJB 2 Entity Beans
      - Data carriers (a.k.a. DAOs)
      - No business logic beyond getters/setters and data validation
  - Standard way of doing Object-relational mapping

EJB 3 Entities (3)

- **Java Persistent API Entities (3)**
  - Managed by Entity Manager
  - **persistence.xml**
    - Describes Persistence Module
    - Defines Persistence Units

EJB 3 Entities (4)

- **Entity**
  - `@Entity(name="JpaUser")`
    - `public class JpaUser implements Serializable {`
      - ... (code snippet)
    - `}`
  - `@Entity(name="JpaRole")`
    - `public class JpaRole implements Serializable {`
      - ... (code snippet)
    - `}`
  - `@Entity(name="JpaEmployee")`
    - `public class JpaEmployee implements Serializable {`
      - ... (code snippet)
    - `}`

EJB 3 Entities (5)

- **JPA**
  - Metadata-driven POJO technology
  - **Persistence Unit** – a group of entities packaged in an application module
    - Requires JPA engine
    - GlassFish – Oracle TopLink
    - toplink-essentials.jar

EJB 3 Entities (6)

- **Entity Manager API**
  - Most important API in JPA
  - Manages lifecycle of entities
  - Bridge between O/R and relational worlds
  - Translates Entities into table records and back

- **Example**
  - `@Entity(name="JpaUser")`
    - `@Entity(name="JpaRole")`
    - `@Entity(name="JpaEmployee")`
  - `public class JpaUser {`
    - `@Id
      @GeneratedValue(strategy=GenerationType.AUTO)`
    - `}`
  - `public class JpaRole {`
    - `}`
  - `public class JpaEmployee {`
    - `}`
  - `return db.getJpaUser();`
EJB 3 Entities (7)

- Packaging
  - Entity in Java EE Application
    - Package in a standard Java EE Module
    - JAR or WAR
    - Place into target EAR
    - Root or Library directory
  - Entity in EJB Module
    - Merge JPA code into EJB module
    - Put entity together with Bean classes
    - Put persistence.xml in META-INF
    - Package as a single EJB JAR

EJB 3 Remote Interface (1)

- Similar to EJB 2
- Acts as proxy
- Defines business methods available to clients

- New Features
  - New annotation-based declaration
  - New retrieval protocol
    - No need to use EJBHome factory
    - Classical JNDI lookup or Dependency Injection (DI)

```
import javax.ejb.Remote;

public interface SimpleSession
{
    public String sayHello();
}
```

Container Services

Client with DI

```
import javax.ejb.EJB;

public class SessionBeanClient
{
    private static SessionBean simpleSession;

    private void InvokeSessionBeanMethods()
    {
        System.out.println( simpleSession.sayHello( "John" ));
        System.out.println( "
```

EJB 3 Remote Interface (2)

```
    + simpleSession.getClass().getName());
    }

    public static void main( String[] args )
    {
        new SessionBeanClient().invokeSessionBeanMethods();
    }
}
```

EJB & Java EE Containers

- Process client requests
  - Generates stubs and skeletons
  - Creates EJB instances as needed
  - Persists entity beans (EJB 2)
  - Handles security and transactions
- Provide middleware services

EJB & Java EE Containers

- Process client requests
  - Generates stubs and skeletons
  - Creates EJB instances as needed
  - Persists entity beans (EJB 2)
  - Handles security and transactions
- Provide middleware services
Middleware
- Software that connects components and applications
  - Explicit middleware (CORBA):
    - Programmer writes to API
    - Difficult to develop, maintain, support
  - Implicit middleware (EJB):
    - Programmer writes business logic
    - Declarative middleware specification
    - Middleware generated automatically

EJB & Java EE Container Services
- A.k.a. EJB middleware services:
  - Remote Method Invocation (RMI-IIOP)
  - Transactions (JTA)
  - Persistence (JPA)
  - Resource pooling
  - Client state management
  - Messaging (JMS)
  - Interceptors (AOP)
  - Integration (DI)
  - Authorization
  - Security
  - Thread safety

EJB & Java EE Container Services
- Integration (1)

EJB & Java EE Container Services
- Integration (2)
  - Reminds “early binding” but is not one
  - Reminder:
    - Early binding
      - Static linking of objects – compile-time linking
    - Late binding
      - Run-time reference resolution a.k.a. dynamic loading
  - Integration is run-time activity
    - Java resolves variable references and method calls dynamically
    - Run-time exceptions if fails

EJB & Java EE Container Services
- Integration (3)
  - JNDI Lookup example
EJB & Java EE Container Services

Integration (4)  Dependency Injection example (1)

```java
public class SimpleClient {
    ...
    EMF mapName="cohort:timestamp:123.0.0.1:370001SingleSession"
    public static SingleSession bean2();
    public void use2() throws Exception {
        String result = bean2().sayHello("John");
    }
    ...
}
```

EJB & Java EE Container Services

Integration (5)  Dependency Injection example (2)

```java
public class BookService implements BookService {
    private DataSource database;
    public Integer getQuantity(String bookId) {
        try {
            Connection conn = database.getConnection();
            PreparedStatement stmt = conn.prepareStatement("SELECT quantity FROM books WHERE id = ?");
            stmt.setString(1, bookId);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                qty = rs.getInt("quantity");
            } else {
                throw new RuntimeException("getQuantity()");
            }
            return qty;
        } catch (Exception e) {
            throw new RuntimeException(e.getMessage());
        }
    }
}
```

EJB & Java EE Container Services

Integration (6)  Dependency Injection

- Container injects resources using Java Metadata Annotations
- EJBs via @EJB
- Persistence Context via @PersistenceContext
- Persistence Unit via @PersistenceUnit
- Other resources via @Resource

Applies
- Session Beans
- Message-driven Beans
EJB & Java EE Container Services

Integration (7)
- Dependency Injection
  - Limitations
    - No Container = No Injection!
    - Once Injected = Always Injected!
    - Configuration fixed
    - Can be done at runtime (e.g., JNDI access needed) but
    - No procedural composition if (...) then @Resource
    - Can't inject POJOs, only properly declared EJBs
    - See Spring Framework and AOP if you need it

EJB & Java EE Container Services

Remotability
- Remote Method Invocation over Internet Inter-Orb Protocol (RMI / IIOP)
- Enables remote access to EJBs
- Applies
  - Session Beans
  - Location Transparency
    - Invoke Local and Remote EJBs same way
    - Local - deployed and executed in the same JVM
    - Remote
    - Best practices
      - Define EJBs as Local whenever possible

EJB & Java EE Container Services

Logging
- Process by which Enterprise Server captures data about
  events that occur during it's operation
- Records in log file
- Implementation
  - Vendor specific to a standard
  - Alternative: Apache Commons Logging Library
- Obs! [JAR 308 J2EE, Version 1.4, Sec. 20.1.3 Programming/Restoration, p. 565]
  - EJB must not use AWT functionality to output to a display, or to
    input from a keyboard
  - EJB must not use java.io package functionality to access
    files and directories in the file system
  - Use resource manager API, such as JDBC, to store data

EJB & Java EE Container Services

Transactions
- Sequence of one or more steps that add, modify, or
  delete persistent data
- ACID
- Java Transaction API (JTA)
  - Declarative configuration
  - Any method can be transactional
  - Container commits if completes properly
  - Applies
    - Session Beans
    - MDBs

EJB & Java EE Container Services

Resource pooling and Management (1)
- Instance Pooling
- Passivation / Activation
EJB & Java EE Container Services

Resource pooling and Management (2)

1. Instance Pooling
   - Applies
     - Stateless Session Beans
     - MDBs
   - EJB 3 a pool of component instances
2. Goal
   - Reuse and Resource optimization
   - Faster than garbage collection

Pooling and Reuse - Stateless (1)

Pooling and Reuse - Stateless (2)

Pooling and Reuse - Stateless (3)

EJB & Java EE Container Services

Resource pooling and Management (3)

1. Passivation / Activation
   - Applies
     - Stateful Session Beans

Pooling and Reuse - Stateful (1)
Pooling and Reuse - Stateful (2)

State management
- Applies Stateful Session Beans
- Container maintains conversational state
- Container maintains session
- Programmer works with instance variables

Pooling and Reuse - Stateful (3)

State management
- Applies Stateful Session Beans
- Container maintains conversational state
- Container maintains session
- Programmer works with instance variables

EJB & Java EE Container Services

Interceptors (1)
- Objects automatically triggered when an EJB method is invoked
- Applies Session Beans
- Message-driven Beans
- Interceptor
  - Regular Java class with annotated methods
  - Attached to either EJB class or method
  - Triggered when EJB method execution begins
  - Can complete after EJB method returns
  - Can inspect whatever EJB method returns
  - In AOP terms – an around invoke advice
  - @AroundInvoke, @PostConstruct, @PrePassivate, @PostActivate, @PreDestroy

EJB & Java EE Container Services

Interceptor (2)

```java
public class ActionLogger {
    public ActionLogger() {
        // Constructor
    }

    public void logMethodEntry(String method) {
        // Log method entry
    }

    public void logMethodExit(String method) {
        // Log method exit
    }
}
```

EJB & Java EE Container Services

Message
- Applies Message-driven Beans
- Container supports JMS
- Programmer can abstract from implementation details

EJB & Java EE Container Services
EJB & Java EE Container Services

- **Timers**
  - Time-delayed callbacks
  - Timeout method invoked after time interval elapses
  - Persistent – can survive a crash
  - Transactional – a transaction failure in a timeout method rolls back the actions taken by the timer

```java
public class MyEJB {
    Resource TimerService timerService;
    public void addSchedule(Record record) {
        timerService.createTimer("JVM103899", "JVM103899", record);
        Create a timer
    }
    ...
    public void monitorRecord(Timer timer) {
        Thread method
        Record record = timer.getInfo();
    }
    ...
}
```

- **Thread safety**
  - **Applies**
    - Session Beans
    - Message-driven Beans
  - Container makes components thread-safe
  - Container optimizes performance
  - Programmer writes beans to single-threaded paradigm

- **Security**
  - **Applies**
  - Session Beans
  - Container integrates with JAAS
    - All JPA entities external to EJBs
  - Simple configuration instead of complex coding
    - Rules
    - Realms

- **Persistence**
  - **Applies**
  - JPA Entities
  - Process of preserving consistency between EJB properties and Database tuples
  - Declarative mapping
  - Declarative configuration
  - Alternative to low-level JDBC/SQL

How container vendors compete?

- Better and more efficient container services
- Caching strategies
- Development tool integration
- Database access optimization
- Performance
EJB Patterns

Publish - Subscribe

Point-to-Point

Façade

EJB Generations Comparison

EJB 2 vs. EJB 1.1

- **EJB 2 advantages (1)**
  - **Local interfaces**
    - Can declare Beans as local (same JVM). No need to wrap as remote. Arguments passed by reference, not by value. Improves performance.
  - **ejbHome methods**
    - Entity beans can declare ejbHomeXXX(...) methods that perform operations related to the EJB component but that are not specific to a bean instance. The ejbHomeXXX(...) method declared in the bean class must have a matching home method XXX(...) in the home interface.
EJB 2 vs. EJB 1.1

- **EJB 2 advantages (2)**
  - **Message Driven Beans (MDB)**
    - New enterprise bean type to handle incoming JMS messages
  - **New Container Managed Persistence (CMP) Model**
    - New contract called the abstract persistence schema. The container handles persistence automatically at runtime.
  - **EJB Query Language (EJB QL)**
    - SQL-based language that allows to implement and execute finder methods. EJB QL also used in new query methods `ejbSelectXXX(…)` which is similar to `ejbFindXXX(…)` methods except that it is only for the bean class to use and not exposed to the client (i.e., it is not declared in the home interface)

- **EJB 2 disadvantages**
  - Must create certain component interfaces
  - Must implement certain call-back methods
  - EJB deployment descriptors are complex
  - EJB components are not truly object oriented
  - Use of inheritance and polymorphism restricted
  - EJB modules cannot be tested outside EJB container
    - Debugging inside container difficult

EJB 3 vs. EJB 2

- **Simplified EJB implementation cycle**
  - EJB 3.0 eliminates the need for home and component interfaces and the requirement for bean classes for implementing `javax.ejb.EnterpriseBean` interfaces.
  - EJB bean class is a simple Java class – POJO.
  - EJB interface is a simple business interface.

- **Use of Java Annotations Instead of XML-based Deployment Descriptors**
  - Metadata annotation as an alternative to deployment descriptors.
  - Annotations can be used to specify bean types, different attributes such as transaction or security settings, O-R mapping and injection of environment or resource references.
  - Deployment descriptor can be used to override metadata annotations.
EJB 3 vs. EJB 2

- Interceptor capabilities
  - An interceptor is a method that intercepts a business method invocation.
  - An interceptor method may be defined in a Stateless Session Bean. Stateful Session Bean or an interceptor class may also be used instead of defining the interceptor method in the bean class.

- Simplified JNDI lookup of EJB
  - Lookup of EJB has been simplified and clients do not have to create a bean instance by invoking create method on EJB and can directly invoke a method on the EJB.
  - DI FIXME! EJB 3 In Action 2007, p. 40, 22, 26

EJB Best Practices

Best Practices

- Session Beans
  - Choose Stateless Session Bean
    - Container will not persist stateless session bean
  - Choose Local interface over Remote if same JVM
    - Remote interface implies network access – expensive
  - Choose HttpSession over EJB Stateful Beans
    - Can do if Client is a Web-tier Client
    - Separate crosscutting concerns
    - Place Logging and auditing in Interceptors (see below)
  - See EJB 3 In Action 2007, p. 108

References


Resources

- Szyperski, chapter 14
- TDD05 course page with links to EJB tutorials and books
- JBoss, Open source EJB Container
  - http://www.jboss.org
- Sun Microsystems. EJB 3.0 Specification Final Release
- Sun Microsystems. The Java EE 5 Tutorial, 2006