EJB 3 in Action

Major annotations

- @Resource : dependency injection of resources
- @EJB : dependency injection of session beans
- @WebServiceRef : dependency injection of web services
- @PersistenceContext : dependency injection of container-managed EntityManager
- @PersistenceUnit: dependency injection of EntityManagerFactory
- @PostConstruct: lifecycle method
- @PreDestroy: lifecycle method

Session Beans

- Annotations @Stateless and @Stateful are not inherited, but lifecycle annotations (defining callback methods) are by the bean class.
- All business methods must be public, not final and static, and should not begin with ejb.
- All session beans have @PostConstruct and @PreDestroy.
- Stateful session beans have @PrePassivate and @PostActivate.
- @Local: designed for client of the session beans in the same container (JVM).
- @Remote: enable a session bean to be accessible via RMI (all parameters and return objects must be Serializable).
- @WebService: Make a bean SOAP accessible.
- Only one access type annotation per interface.
- @Remove: used in stateful session beans to indicate the end of the session.
- @EJB: used to inject session beans into client code.

Message Driven Beans

- Two messaging models: point-to-point (P2P) and publish-subscribe.

- P2P destination = queue; pub-sub destination = topic
- JMS has two administrative objects: `javax.jms.ConnectionFactory` and `javax.jms.Destination`
- A `MessageProducer` is used to send messages.
- A `message` can contain properties (metadata).
- Spring's `JmsTemplate` simplifies the code.
- JCA (Java Connector Architecture): allows MDB to receive messages from any enterprise information system (EIS). JCA endpoint = JMS destination = MDB proxy.
- @`MessageDriven`
- Acknowledge mode: AUTO_ACKNOWLEDGE (advised by default).
- Durability: normally, a subscriber (topic) not connected does not received a copy of the message, unless otherwise specified.
- `MessageSelector`: it is possible to filter a message according to its properties
- By default, the container starts a transaction before the `onMessage`() is invoked and commits it when the method returns."
- Poison messages: an incorrect message will be delivered and roll backed indefinitely. Solutions exists (dead message queue) but they are vendor specific.

Advanced Concepts

- `EJBContext`: an interface that provides access to the container-provided run-time context of an enterprise bean instance.
- To invoke the contexts: `@Resource SessionContext ctx; @Resource MessageDrivenContext ctx;` (both extends `EJBContext`)
- @`Resource`(name="…") => JNDI name: `java:comp/env/…`
- ENC = Environment Naming Context
- Resource must be specified in a JNDI file, except for `EJBContext` and `TimerService`.

Access environment entries

```
@Resource private boolean censorship;
```

```
<env-entry>
   <env-entry-name>censorship</env-entry-name>
```

```
   <env-entry-type>java.lang.Boolean</env-entry-type>
   <env-entry-value>true</env-entry-value>
</env-entry>
```

Interceptors

- Point-cut: place in the code where the interception takes place.
- EJB3 = around invoke advice (triggered at the beginning of the method, check the return values)
- Annotations: `@Interceptors, @AroundInvoke, @ExcludeDefaultInterceptors, @ExcludeClassInterceptors.`
- Interceptors: Object <method> (`InvocationContext`) throws Exception

Timer services

- Only in stateless session beans and MDB.
- @`Timeout`

Transactions and Security

- ACID: Atomicity, Consistency, Isolation, Durability
- Isolation levels: Read uncommitted (dirty read possible), Read committed (often the default), Repeatable Read, Serializable.
- Transaction Manager: component that coordinates a transaction over multiple distributed resources.
- Two-phase commit: there is a first step where the TM asks every resource manager if the current transaction can be successfully committed.
- **XA protocol**: most popular distributed transaction protocol, develop by the X/Open group.
- JTA: Java Transaction API.
- CMT: Container Managed Transaction – transactions are managed though annotations or a deployment descriptor.
- BMT: Bean Managed Transaction – transactions are managed programmatically.
- JTS: Java Transaction Service: implementation of JTA.
- `@TransactionManagement(TransactionManagementType.CONTAINER)` to explicitly declare the use of CMT.

- `@TransactionAttribute`: Required, Requires_New, Supports, Mandatory, Not_Supported, Never. For MDB, only supported attributes are Required and Not_Supported.
- To mark a transaction for roll-back: `sessionContext.setRollbackOnly()`.
- `@ApplicationException`: An application exception is one that the client is expected to handle. Setting the attribute `rollback` to true tells the container that it should rollback the transaction before the exception is passed to the client.
- JAAS: Java Authentication and Authorization Service. The JAAS API knows how to talk to underlying authentication systems like LDAP.
- The `java.security.Principal`: entity shared by the application tiers (the user is identified at the web tier).
- Web tier security is mainly configured using the `login-config` and `security-constraint` elements of the `web.xml`.
- Authentication method: BASIC, FORM or CLIENT-CERT

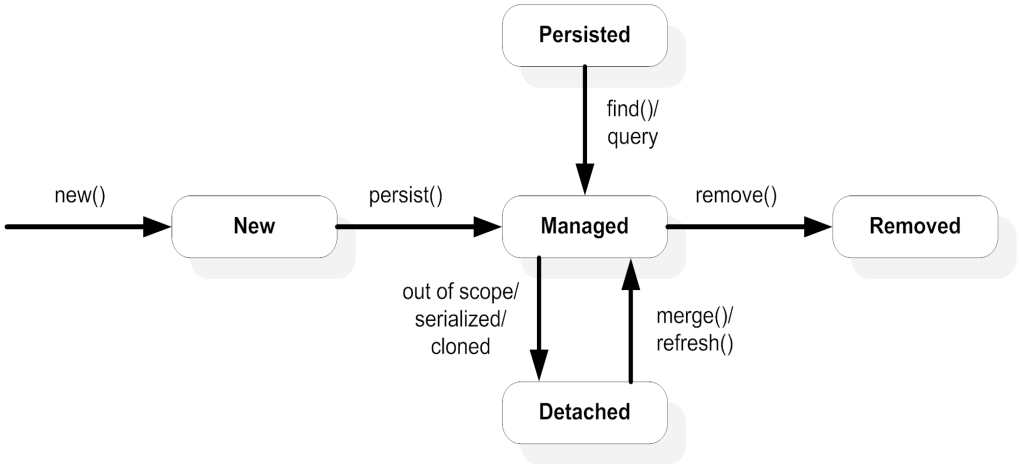
```
<login-config>
	<auth-method>BASIC</auth-method>
	<realm-name>ActionBazaarRealm</realm-name>
</login-config>

<login-config>
	<auth-method>FORM</auth-method>
	<form-login-config>
		<form-login-page>/loginPage.do</form-login-page>
		<form-error-page>/loginErrorPage.do</form-error-page>
	</form-login-config>
</login-config>

<security-constraint>
	<web-resource-collection>
		<web-resource-name>Admin Component</web-resource-name>
		<url-pattern>/admin/*</url-pattern>
	</web-resource-collection>
	<auth-constraint>
		<role-name>CSR</role-name>
	</auth-constraint>
</security-constraint>

<security-role>
	<description>Login</description>
	<role-name>CSR</role-name>
</security-role>
```

- Security annotations: `@DeclareRoles("CSR"), @RolesAllowed("CSR3"), @PermitAll, @DenyAll and @RunAs("ADMIN")`

- Programmatic security: `EJBContext.getCallerPrincipal() and isCallerInRole(rolename)`

Java Persistence API (JPA)

- `@Entity`: a domain object that can be uniquely identified.
- `@Transient`: prevent a field from being persisted.
- `@Id`: identity field; `@IdClass`: define a class for composite keys.
- `@Embeddable`: for persistent objects that need not have an identity of their own.
- `@OneToOne`: used to mark uni- and bidirectional one-to-one relationships.
- `@OneToMany`, `@ManyToOne`: one-to-many relationship (default: LAZY)
- `@ManyToMany`
- `EntityManager`: bridge between the OO and relation database. Methods: persist(entity), merge(entity), remove(entity), find(class, key), flush(), refresh(entity), createQuery(jpql), close(), isOpen(), getTransaction(), …
- Entity lifecycle:



- Injecting the Persistence Manager: `@PersistenceContext(unitName="...")`
- Cascade Type: ALL, MERGE, PERSIST, REFRESH, REMOVE
- `@Basic(fetch=FetchType.LAZY)`: prevent the loading of an object until it is required.
- Default fetch modes: `@OneToOne`, `@ManyToOne` = eager <> `@OneToMany`, `@ManyToMany` = lazy.

- Entity lifecycle listeners: `@PrePersist, @PostPersist, @PostLoad, @PreUpdate, @PostUpdate, @PreRemove, @PostRemove`.
- Best practices: Use container-managed entity beans; Avoid injecting entity managers into the web tier; Use DAO; Separate callbacks into external listeners.
- `EntityManager.find()` retrieves an entity using its ID.
- Creation of a query: `EntityManager.createQuery(sql) or em.createNamedQuery(name)`
- Named query: @`NamedQuery`(name="...", query="...")
- **There is no need of an active transaction to create or execute a query; if one does not exist, the retrieved entities become detached instances**.
- Retrieving entities: `query.getSingleResult() or query.getResultList()`
- JPQL: Java Persistence Query Language

Packaging EJB 3 applications

- CAR: client application archive (`application-client.xml`) – thick client for EJB
- EAR: enterprise application archive (`application.xml`) – über archive
- EJB-JAR (`ejb-jar.xml`): session beans, message-drive beans, entity beans
- RAR: resource adapter archive (`ra.xml`) – resource adapters
- WAR: web application archive (`web.xml`) – web application artifacts

application.xml (optional since Java EE 5)

```
<application>
	<display-name>(mandatory)</display-name>
	<module>
		<ejb>actionBazaar-ejb.jar</ejb>
	</module>
	<module>
		<web>
			<web-uri>actionBazaar.war</web-uri>
			<context-root>ab</context-root>
		</web>
	</module>
	<module>
		<java>actionBazaar-client.jar</java>
	</module>
</application>
```

- **Bootstrap** class loader: $JAVA_HOME/jre/lib/rt.jar
- **Extension** class loader: $JAVA_HOME/jre/lib/ext/*.jar (cryptographic + security classes)
- **System** class loader: $CLASSPATH
- **Application Server** class loader: $APP_SERVER_HOME/lib
- A class loader loads a class dynamically on an as-needed basis. It first looks at its local cache, then asks its parent to load the class, then attempt to load it from the local sources (`classpath`).
- For classes to be visible to all modules in the EAR file, best to package them as a library module in the EAR.

ejb-jar.xml

```
<ejb-jar version="3.0">
<enterprise-beans>
      <session>
            <ejb-name>BazaarAdmin</ejb-name>
            <remote>actionbazaar.buslogic.BazaarAdmin</remote>
            <ejb-class>actionbazaar.buslogic.BazaarAdminBean</ejb-class>
            <session-type>stateless</session-type>
            <transaction-type>Container</transaction-type>
      </session>
</enterprise-beans>

...

<assembly-descriptor>
      <container-transaction>
            <method>
                  <ejb-name>BazaarAdmin</ejb-name>
                  <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
      </container-transaction>
      <security-role>
            <role-name>users</role-name>
      </security-role>
</assembly-descriptor>
</ejb-jar>
```

Integrating EJB 3

- EAO: Entity Access Object

- Session Façade: 1/centralizes the business logic in a single place; 2/ clients do not have to know the internal design of the system; 3/ helps manage transactions and security 4/makes the client code simpler.

- Accessing session beans from the web tier: `@EJB` (not advised) or JNDI lookup:
```
Context context = new InitialContext();
context.lookup("java:com/env/ejb/<session bean name>")
```

Performance and Scalability

- **Dirty read**: a user see uncommitted transactions from another user.
- **Non-repeatable read**: a user is allowed to change data employed by another used in a separate transaction.
- **Phantom read**: a user receives inconsistent results in two different attempts of the same query.
- **Pessimistic locking**: lock all involved database rows for the entire span a time a user shows an interest in modifying an entity. Implemented using SELECT ... FOR UPDATE. No support within JPA.
- **Optimistic locking**: assumes concurrency problems rarely occur, up to the application to detect and solve those. Typically implemented by adding a column to the table, and either storing a version number or a timestamp (use `@Version)`.
- Improving entity performance: 1/merging tables; 2/dividing tables; 3/choosing the right inheritance strategy (single-table strategy / joined-table strategy / table-per-class strategy)
- Propertly size the connection pool; cache SQL statements; Use named queries; Reduce db operations (cf. JOIN FETCH);
- Query performances: avoid full-table scans, use indexes (also for relationship fields), do not use functions in the WHERE clause (because indexes are disabled), reduce round-trips to the db
- Use @Local rather than @Remote for the session beans.
- Use Stateless SB instead of Stateful.
- Use @remove for stateful beans.
- Control serialization by making variables transient.

Web Services

- WSDL: Web Service Description Language (service description)
- UDDI: Universal Description, Discovery and Integration registry (www.uddi.org)

- Three ways to implement a web service: REST, XML-RPC, SOAP.
- Every SOAP message contains several XML documents: Envelope, Header, Body. The Header contains meta-information (security, network routing, …)
- WSDL gives the message type, port, operations, data types, location, return information, ...

SOAP Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header/>
      <SOAP-ENV:Body>
            <addBid xmlns="http://actionbazaar.com/Bidding">
                  <user-id>viper</user-id>
                  <item-id>100</user-id>
                  <bid-price>2000.24</bid-price>
            </addBid>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

```
<definitions xmlns="..." xmlns:soap="..." xmlns:xsd="..." xmlns:mime="..."
      xmlns:tns="http://ejb3inaction.example.buslogic/"
      name="PlaceBidBeanService"
      targetNamespace="http://ejb3inaction.example.buslogic/">

      <!-- Data Types -->
      <types>
            <schema xmlns="..." ...>
                  <element name="addBid" type="tns:addBid"/>
                  <complexType name="addBid">
                        ...
                  </complexType>
                  <element name="addBidResp" type="tns:addBidResp"/>
                  <complexType name="addBidResp">
                        ...
                  </complexType>
            </schema>
      </types>

      <!-- Messages the service support -->
      <message name="PlaceBidBeanPortType_addBid">
            <part name="parameters" element="tns:addBid" />
      </message>
      <message name="PlaceBidBeanPortType_addBidResp">
            <part name="parameters" element="tns:addBidResp" />
      </message>
```

```
    <!-- Port Types: defines operations and messages -->
    <portType name="PlaceBidBean">
        <operation name="addBid">
            <input message="tns:PlaceBidBeanPortType_addBid"/>
            <output message="tns:PlaceBidBeanPortType_addBidResp"/>
        </operation>
    </portType>

    <!-- Binding -->
    <binding name="PlaceBidBeanSoapHttp" type="tns:PlaceBidBean">
        <soap:binding style="document" transport="(http)"/>
        <operation name="addBid">
            <soap:operation soapAction="" />
            <input>...</input>
            <output>...</output>
        </operation>
    </binding>

    <!-- Top-level Service with binding and port definition -->
    <service name="PlaceBidBeanService">
        <port name="PlaceBid" binding="tns:PlaceBidBeanSoapHttp">
            <soap:address location="${...}/PlaceBid" />
        </port>
    </service
</definitions>
```

- JAXB (Java Architecture for XML Binding): binding between Java and XML data types
- Better to use EJB session beans rather than POJO to build web services.
- JAX-WS: Java API for XML based Web Services
- To expose a session bean as a web service:
  `@WebService`(targetNamespace="..." serviceName="..." portName="...")
  `@SOAPBinding`(style=DOCUMENT)
  `@WebMethod`
  `@WebResult`
  `@WebParam`(name="...")
- Calling a WS: `@WebServiceRef(wsdlLocation="http://...?WSDL)`
- A tool is necessary to read the WSDL document and generates the interface and the proxy classes that can be used to invoke methods on the web service as a local object.

Spring

- Spring classes for JPA: `JpaTemplate, JpaDaoSupport, JpaTransactionManager...`

- `LocalContainerEntityManagerFactoryBean` : reads the `persistence.xml` to configure the persistence unit by using the data source supplied.
- `LocalEntityManagerFactoryBean` : outside a JEE container.

RMI and JNDI

- `@Remote`: the container makes the bean callable via RMI as a remote object.
- `@EJB`: the remote bean is injected through RMI.
- **Marshaling:** turning an object into a byte steam
- If interoperability is important (outside Java), use the web services.
- JNDI provides a uniform abstraction over a number of different naming services such as LDAP, DNS, RMI, CORBA, … Any naming service with a JNDI Service Provider Interface (SPI) can be plugged seamlessly.
- JNDI is the central repository for resources managed by the container. Every bean managed by the container is automatically registered with JNDI.

```
Context context = new InitialContext();
DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/myDS");
```

- Empty constructor for `InitialContext` only useful while looking up resources within the same JVM.
- The `InitialContext` uses the file `jndi.properties` for remote access.

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
      "oracle.j2ee.rmi.RMIInitialContextFactory");
properties.put(Context.PROVIDER_URL, "ormi://192.168.0.6:23791/appendixa");
Context context = new InitialContext(properties);
```

where `Context.INITIAL_CONTEXT_FACTORY = "java.naming.factory.initial"` et `Context.PROVIDER_URL = "java.naming.provider.url".`