Lecture topics

- J2EE architecture continued
- Security policies for EJBs
- Description of the project

Session beans

- A session bean represents a single client inside the J2EE server
- The client invokes the session bean’s methods
- A session bean is not shared and not persistent
- Two types of session beans
  - Stateful
    - The state of the bean is preserved as its fields
  - Stateless
    - Although a bean may have fields, they are not used to hold data across method invocations on this bean
    - Usually, these are more efficient than stateful beans
**Entity beans**

- An entity bean is an object in a persistent storage mechanism
  - Typically, each entity bean corresponds to a table in a relational DB
  - Each instance of an entity bean corresponds to a row in a DB table
- Two types of bean persistence:
  - Bean-managed
    - The bean code accesses the DB explicitly
  - Container-managed
    - The EJB container automatically generates the DB access calls
- Entity beans may be shared by multiple clients
  - Have to work in transactions
- Each entity bean has a unique object identifier
  - Can be used by the client to locate a particular entity bean
- An entity bean can be related to other entity beans

**Message-driven beans**

- A message-driven bean that allows a J2EE application to process messages asynchronously
  - Messages can be sent by any J2EE component (client, EJB, servlet) or even a system not based on J2EE technology
- Message-driven beans are similar to stateless session beans
  - Message-driven bean instances retain no state for a specific client
  - All instances of a message-driven bean are equivalent
    - Allows pooling
    - A single message-driven bean can process data from multiple clients
**The contents of an EJB**

- **Interfaces**
  - Define methods that are accessible to clients
- **Bean class**
  - Implements the methods defined in the interfaces
  - Can contain additional methods
- **Deployment descriptor**
  - An XML file that specifies information about the bean
    - Persistence
    - Transaction attributes
- **Helper classes**
  - Any other utility classes that are used by this bean

**Lifecycle of a stateful session bean**

```
Does not exist

create
setSessionContext
ejbCreate

Ready

esebPassivate
ejbActivate

Passive

remove
ejbRemove
```
**Lifecycle of a stateless session bean**

- **Does not exist**
  - `setSessionContext` \(\rightarrow\) `ejbRemove` \(\rightarrow\)

- **Ready**

**Lifecycle of an entity bean**

- **Does not exist**
  - `setEntityContext` \(\rightarrow\) `unsetEntityContext` \(\rightarrow\)

- **Pooled**
  - `remove` \(\rightarrow\) `ejbRemove`
  - `ejbActivate` \(\rightarrow\)
  - `ejbPassivate` \(\rightarrow\)
  - `create` \(\rightarrow\)
  - `ejbCreate` \(\rightarrow\)
  - `ejbPostCreate` \(\rightarrow\)

- **Ready**
Can we see an example already?

```java
public class TestSessionBean implements SessionBean {
    private SessionContext context;

    public void ejbCreate() throws CreateException {}
    public void ejbActivate() throws EJBException {}
    public void ejbPassivate() throws EJBException {}

    public void setSessionContext(SessionContext newContext) throws EJBException {
        context = newContext;
    }
    public void myMethod() throws EJBException {
        System.out.println("my method got called");
    }
}
```

A bean must have interfaces

Remote interface:

```java
public interface TestSessionBean extends javax.ejb.EJBObject {
    public void myMethod() throws java.rmi.RemoteException;
}
```

Home interface:

```java
public interface TestSessionBeanHome extends javax.ejb.EJBHome {
    public static final String COMP_NAME= "java:comp/env/ejb/TestSessionBean";
    public static final String JNDI_NAME="ejb/TestSessionBeanHome";

    public TestSessionBean create()
    throws javax.ejb.CreateException, java.rmi.RemoteException;
}
```
Remote interfaces

- These are interfaces that expose functionality to remote clients
  - If a bean class declares a public method, it does not mean that a client can call it
    Only if this method is declared in the remote interface
- Remote interfaces usually extend interface `javax.ejb.EJBObject`
  - Defines some useful methods for EJBs:
    - `Object getPrimaryKey()`
      - Should be used only for entity beans
    - `boolean isEqual(EJBObject other)`
      - Determines if the given EJB object is identical to the one on which the call is made
    - `void remove()`
      - Destroys the EJB reference used by the calling client

---

Home interfaces

- Home interfaces are used to create initial references to beans for clients
  - For session beans, this includes creating a bean object
- Home interfaces enable EJB containers to manage how to allocate EJB resources
- For people familiar with design terminology, home interfaces are factories
- Clients use JNDI to get a handle to an EJB home interface
**EJB client example**

```java
public class Test {
    public static void main(String[] args) {
        try {
            Properties properties = new Properties();
            properties.put(Context.INITIAL_CONTEXT_FACTORY,
                            "org.jnp.interfaces.NamingContextFactory");
            properties.put(Context.PROVIDER_URL, "host:1099");
            Context initial = new InitialContext(properties);
            Object objectRef = initial.lookup("TestSessionBean");
            TestSessionBeanHome testBeanHome =
                (TestSessionBeanHome) PortableRemoteObject.narrow(
                    objectRef, TestSessionBeanHome.class);
            TestSessionBean testBean = testBeanHome.create();
            testBean.myMethod();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**EJB security architecture terminology**

- A **principal** is the EJB container’s view of a user of the application
  - Mapping of users to principals is not described by the EJB spec
  - Authentication of principals is not described by the EJB spec
- A **permission** is an authorization to perform some action
- A **security role** is a grouping of principals and associated permissions
  - A principal is assigned a role
  - A role is granted permission to execute specific methods in the EJB
**Declarative security vs. programmatic security**

- **Programmatic security**
  - Security is handled on the level of Java code in the J2EE components
    - E.g., check if the grade being accessed is actually for the student doing accessing

- **Declarative security**
  - Security settings of J2EE components are done in the deployment descriptor
  - Specify what roles can call what methods of what beans
    - This level of access can also be implemented programmatically

- **So which approach is better?**
  - Declarative is more flexible for deployment
    - Can alter security settings to match the environment, without changing the code
  - Programmatic is more flexible in the sense of functionality

---

**Defining security roles**

- Done in the deployment descriptor
  - Refers only to components in the same JAR

```xml
<assembly-descriptor>
  <security-role>
    <description>
      Any student at the university
    </description>
    <role-name>student</role-name>
  </security-role>
  <security-role>
    <description/>
    <role-name>professor</role-name>
  </security-role>
</assembly-descriptor>
```
Mapping roles to EJB methods

```xml
<method-permission>
  <role-name>professor</role-name>
  <method>
    <ejb-name>GradebookBean</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>

<method-permission>
  <role-name>student</role-name>
  <method>
    <ejb-name>GradebookBean</ejb-name>
    <method-name>getGrade</method-name>
    <method-name>complainAboutGrade</method-name>
  </method>
</method-permission>
```

Security for resources

- Specifies access privileges to resources controlled by EJBs
- Defines JNDI namespace for resources
  - `java:comp/env/jdbc/GradesDB` in this example

```xml
<resource-ref>
  ...
  <description>The JDBC driver</description>
  <res-ref-name>GradesDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```
**J2EE application for the project**

- A Web application for auctioning textbooks
- Only books authorized for use in some course can be posted for auctions
  - Only administrator is allowed to
    - Create courses
    - Create books
    - Assign books to courses
- All auctions for the same textbook are pooled
  - Matching of bidders to sellers is hidden from the users
  - A seller (bidder) specifies the minimal rating of the bidder (seller) that (s)he is willing to deal with
  - An auction terminates after a pre-set amount of time passed since the last bid on a specific copy of the textbook
- Both sides have to “sign off” on the transaction
  - Includes entering a rating for the other party for this transaction

**Architecture of bookauction**

- Tomcat container
  - Servlets
    - HTML
  - Client browser
- JBoss container
  - Bookauction bean
    - serialization
    - RMI
  - Client-side application
- Server filesystem

CIS 916, Application Security © Gleb Naumovich