Today we will briefly cover:

- Introduction to EJB
- EJB basics
- EJB Architecture
- Session Beans
- EJB Clients
- EJB Development Process

Next lesson we will cover:

- Entity Beans
- Message Driven Beans
- EJB issues
- Indepth coverage

So far we have developed multi-tier applications:

- Web-based Applications (Servlets/JSPs/JDBC)
  - "thin" client – using browser
  - Model-View-Controller architecture
  - simple business logic
  - Some services provided by web container
- Distributed Applications (RMI/JDBC)
  - "fat" client – using Java application
  - Simple client/server architecture
  - You have to write all other services

However... what about:

- Reusability of components?
- Security?
- Resource management?
- Scalability – Load Balancing, Clustering?
- Fault tolerance? Error handling?
- Deployment in other environments?
- Persistence? Independence from specific databases?

With web applications & RMI, we have to program this.

EJB containers provide the above functions.

EJB Introduction

Why EJB?

J2EE architecture
EJB Introduction

- EJB is the J2EE server-side component model
- The component software model is based on the idea of creating reusable components that 'plug in' to 'containers'
- It addresses some important software development goals:
  - reuse
  - high level development focus
  - development automation via tools
  - simplified deployment

- JavaBeans, EJB and ActiveX/COM are examples of component models

EJB Basics

- There are 3 types of Enterprise Java Beans:
  - Session Beans
  - Entity Beans
  - Message Driven Beans

J2EE Architecture

- Component models come in two basic flavours
  - client-side; and
  - enterprise

- Client-side component models such as JavaBeans are specialised to handle presentation and user-interface issues
- Enterprise component models such as Enterprise JavaBeans are specialised to provide a framework for developing, deploying and executing distributed, transaction-aware middleware components

EJB – Types of Beans

- There are 3 types of Enterprise Java Beans:
  1. Session beans
     - stateless
     - stateful
  2. Entity beans
     - BMP – bean managed persistence
     - CMP – container managed persistence
  3. Message Driven beans

- This lesson focuses on session beans
- Next lesson we will cover:
  - Introduction to session beans
  - Session Beans
  - EJB Clients
  - EJB Architecture
  - EJB Development Process

- Next lesson we will briefly cover:
  - Introduction to EJB
  - Session Beans
  - EJB Clients
  - EJB Architecture

- Today we will briefly cover:
  - Component models come in two basic flavours
  - Client-side and enterprise

- This lesson focuses on session beans
- Next lesson on enterprise beans and intro to message beans
- Enterprise beans

- Enterprise beans
  - CEP – container managed persistence
  - BMP – bean managed persistence
  - stateless
  - stateful

EJB Architecture

- Component models come in two basic flavours
  - client-side; and
  - enterprise

- Client-side component models such as JavaBeans are specialised to handle presentation and user-interface issues
- Enterprise component models such as Enterprise JavaBeans are specialised to provide a framework for developing, deploying and executing distributed, transaction-aware middleware components

EJB – Types of Beans

- There are 3 types of Enterprise Java Beans:
  1. Session beans
     - stateless
     - stateful
  2. Entity beans
     - BMP – bean managed persistence
     - CMP – container managed persistence
  3. Message Driven beans

- This lesson focuses on session beans
- Next lesson we will cover:
  - Introduction to EJB
  - Session Beans
  - EJB Clients
  - EJB Architecture
  - EJB Development Process

- Next lesson we will briefly cover:
  - Introduction to EJB
  - Session Beans
  - EJB Clients
  - EJB Architecture

- Today we will briefly cover:
  - Component models come in two basic flavours
  - Client-side and enterprise

- This lesson focuses on session beans
- Next lesson on enterprise beans and intro to message beans
- Enterprise beans

- Enterprise beans
  - CEP – container managed persistence
  - BMP – bean managed persistence
  - stateless
  - stateful
EJB Basics

- EJB interfaces and classes are defined in the `javax.ejb` package.
- EJBs are configured and deployed in a similar manner to web applications using deployment descriptors in XML form (e.g., `META-INF/ejb-jar.xml`).
- We will focus on EJB 2.0, but many books/tutorials look at EJB 1.1.
- Main difference is that JPA is a database-specific alternative for entity beans.

EJB Standards Evolution

- **EJB 1.0**: Originally defined in 1998 as part of the Java 1.1 platform.
  - Defined for Session beans and (optional) Entity beans.
- **EJB 1.1**: Part of the J2EE 1.x specification released in 1999.
  - Includes mandatory support for Entity beans.
  - Enhancement to deployment descriptors, including support for an XML format.
  - Significant tightening of specification to improve EJB-based application portability.
- **EJB 2.0**: J2EE 1.3 standard.
  - JMS (Java Message Service) integration with Message Driven beans.
  - Improved support for container-managed persistence (CMP).
  - Support for RMI/IIOP protocol for network interoperability.
- **EJB 3.0**: Java EE 5 standard.
  - Significantly simplified EJB development. Now use Java 5 annotations.
  - Replaced Entity EJB with JPA.

EJB Containers

- Containers provide services to deployed components so that developers can concentrate on business logic instead of software infrastructure.
- An EJB container is an environment in which Enterprise Java Beans execute. Its primary role is to serve as a buffer between EJBs and the outside world.
- The container simplifies component development by providing services:
  - Transactions
  - Scalability (management of multiple instances)
  - Persistence
  - Security
  - Synchronization
  - Distributed object invocation
  - Monitoring and management of EJB instances.

EJB Runtime Environment

- The EJB specification defines an EJB runtime environment as a server and a set of containers.
- The EJB specification defines the server and runtime environment.
- This is usually implemented as a single application server and containers.

EJB and Application Servers

- We are using a "web application server".
- Note that it has two different types of container:
  - Web container (for servlets and JSPs)
  - Business-logic container (for EJBs)
- They are logically two separate types of container.
- Web container for servlets and JSPs.
- EJB container for beans.
- Note that these two types of container:
  - Expose different types of APIs.
  - Have different sets of responsibilities.

EJB and Application Servers

- Main difference is that EJB 2.0 supports different types of beans.
- Many books/tutorials look at EJB 1.1.
  - We will focus on EJB 2.0.
- We are using a "web application server".
  - There are two different types of container:
    - Web container (for servlets and JSPs)
    - Business-logic container (for EJBs)
  - Each type of container provides different services.
- EJBs are configured and deployed in a similar manner to web applications using deployment descriptors in XML format.
- EJB interfaces and classes are defined in the `javax.ejb` package.
### EJB Containers - services

- **Declarative transaction management:**
  - An application's use of transactions can be declared in the XML deployment descriptor, rather than having a programmer write code to manage transactions.
  - Less code to write.
  - Less chance of programmer-generated errors.

- **Declarative security management:**
  - An application's use of authentication (login) and confidentiality (encryption) can be declared in the XML deployment descriptor, rather than having a programmer write code to manage security.
  - Less code to write.
  - Less chance of programmer-generated errors.

- **Management of multiple instances:**
  - EJBs are written as if they are single threaded classes.
  - However, other beans may be accessed by only one client.
  - EJBs are written as if they are single threaded classes.

- **Pre-cached instances:**
  - The EJB container must ensure that each EJB instance can be accessed by multiple clients.
  - This is true for some beans (e.g., session beans).
  - An EJB container must ensure that each EJB instance can be accessed by multiple clients.

- **Optimised method invocations:**
  - The EJB container must ensure that each EJB instance can be accessed by multiple clients.
  - This is true for some beans (e.g., session beans).
  - An EJB container must ensure that each EJB instance can be accessed by multiple clients.

- **Database connection pooling:**
  - The creation of a pool of database connections, so that connections can be shared between multiple beans, rather than a new connection being created for every client request.

- **Instance passivation:**
  - The temporary swapping of an EJB out to storage. If a container needs resources, it may choose to temporarily passivate a bean.

- **Instance pooling:**
  - The creation of a pool of bean instances, so that beans can be shared between multiple clients, rather than a new bean being created for every client request.

- **Pre-cached instances:**
  - The caching of EJB state information to accelerate the initial creation of EJBs.

- **Optimised method invocations:**
  - The short-circuiting of method calls between two beans in the same container in order to avoid the overhead of a full-blown remote method call.

- **Database connection pooling:**
  - The creation of a pool of database connections, so that connections can be shared between multiple beans, rather than a new connection being created every time the database is accessed.

### Container Services - Instances

- In order to accomplish the goal of timely responses to client requests, the server may perform several tasks:

  - **Instance passivation**
    - The temporary swapping of an EJB out to storage. If a container needs resources, it may choose to temporarily passivate a bean.

  - **Instance pooling**
    - The creation of a pool of bean instances, so that beans can be shared between multiple clients, rather than a new bean being created for every client request.

  - **Database connection pooling**
    - The creation of a pool of database connections, so that connections can be shared between multiple beans, rather than a new connection being created every time the database is accessed.

### Container Services - Transactions

- **Declarative transaction management:**
  - An application's use of transactions can be declared in the XML deployment descriptor, rather than having a programmer write code to manage transactions.
  - Less code to write.
  - Less chance of programmer-generated errors.

- **Declarative security management:**
  - An application's use of authentication (login) and confidentiality (encryption) can be declared in the XML deployment descriptor, rather than having a programmer write code to manage security.
  - Less code to write.
  - Less chance of programmer-generated errors.

### EJB Containers - services

- **Multiple instance support:**
  - The creation of a pool of classes that an application can choose to use.

- **Security management:**
  - The creation of a pool of classes that an application can choose to use.

- **Transaction management:**
  - The creation of a pool of classes that an application can choose to use.
Today we will briefly cover

- Introduction to EJB
- EJB Basics
- EJB Architecture
- Session Beans
- EJB Clients
- EJB Development Process

Next lesson we will cover

- Entity Beans
- Message Driven Beans
- EJB issues

Application-specific elements of the EJB architecture shown in the diagram are:

- **EJB Client**: EJB clients use JNDI to look up home interfaces, and use home and remote interfaces to utilise all EJB-based functionality. EJB clients can be external to the application server (standalone Java client applications or components running in another application server) or running in the same application server (or on a component running in another application server) of the application server (and depend on that component to communicate with EJBs). EJB clients can be external to the application server, and use home and remote interfaces to utilise remote EJBs.

- **EJB Home interfaces (and Stubs)**: EJB home interfaces provide operations for clients to create, remove and find handles to EJB remote object interfaces. Underlying stubs marshal home interface requests and unmarshal home interface responses for the client.

- **EJB Remote interfaces (and Stubs)**: EJB remote interfaces provide the business-specific methods defined for a particular EJB. Underlying stubs marshal remote interface requests and unmarshal remote interface responses for the client.

- **EJB implementations**: EJB implementations are the application components implemented by developers to provide any application-specific business logic, including business methods and EJB creation, removal, finding, activation, passivation, storage and loading.

- **Container EJB implementations (Skeletons and Delegates)**: The container manages the distributed communication skeletons used to marshal and unmarshal data sent to and from the client. Containers may also store EJB implementation instances in a pool, and use delegates to perform service management operations related to a particular EJB before calls are delegated to the EJB implementation instance.
Today we will briefly cover

- Introduction to EJB
- EJB Basics
- EJB Architecture
- Session Beans
- EJB Clients
- EJB Development Process

Next lesson we will cover

- Entity Beans
- Message Driven Beans
- EJB issues

Session Beans

Session beans are EJBs that are created to perform some action on the enterprise system, and possibly return a result set to the client. They model a connection (session) with a single client.

Session beans can be grouped into 2 distinct classifications:

- Stateless session beans
- Stateful session beans

Stateless session beans represent EJBs that are created with no regard for the maintenance of any state between calls by a client. Stateless session beans represent pure input / output engines.

Stateful session beans represent session beans that are created and maintain state during a session. The session is subject to time-out. Stateful session beans represent input / output engines that can utilise state information maintained for a client.

Session beans persist only for the life of the connection with the client – if the server crashes, the session bean dies.

Entity Beans

Entity beans are EJBs that are created to encapsulate some data contained by the system (such as a row in the database or, more generally, to an entry that exists in persistent storage).

Entity beans can be thought of as the "nouns" of a particular problem domain that they are implemented to solve.

The data can be created, searched for (found) or removed by clients.

Some persistent problem domains that they are implemented to represent include:

- Bank customers
- Employees
- Database records

Entity beans are EJBs that are created to encapsulate some data.
You need to create three files:
- Remote interface (declares your business logic)
- Home interface (methods to create the bean)
- EJB implementation class (method implementations)

Let's start with a stateless session bean
- the easiest kind

Remote interface
```java
package bank;
import java.rmi.*;
import javax.ejb.*;

interface BankTeller extends EJBObject {
    String sayHello() throws RemoteException;
    void deposit(String acctnum, Float amount) throws RemoteException;
    void withdraw(String acctnum, Float amount) throws RemoteException;
}
```

Home interface
```java
package bank;
import java.rmi.*;
import javax.ejb.*;

interface BankTellerHome extends EJBHome {
    public BankTeller create() throws RemoteException, CreateException;
}
```

Bean implementation class
```java
package bank;
import javax.ejb.*;

public class BankTellerBean implements SessionBean {
    SessionContext sessionContext;
    // "Standard" session bean methods
    public void setSessionContext (SessionContext sc) {
        this.sessionContext = sc;
    }
    public void ejbRemove() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbCreate() { }

    // Argument list must match create() method in home interface
    // continued on next page
}
```

Bean implementation class (2)
```java
// continued from previous page
// Now the business logic methods (as defined in the Remote interface)
public String sayHello() {
    String message;
    if (Math.random() > 0.5) { message = "Hi, how are you today?"; }
    else { message = "Yeah, whaddya want?"; }
    return (message);
}
```
Stateless Bean – code notes (2)

- Your bean implementation class must implement the javax.ejb.SessionBean interface (which extends javax.ejb.EnterpriseBean).

- This requires you to implement the following methods (as well as your business methods) which are used by the container to create and manage your bean during its life:
  - Parameterless constructor – called by the container to create a new instance of the bean (Class.newInstance()).
  - setSessionContext() – called by the container to pass environment context information to the bean.
  - One or more ejbCreate() methods – called by the container to initialise the bean with the parameters from the client when it is created. A stateless session bean can only have a single, no-parameter method.
  - ejbRemove() – called by the container to remove the bean. Stateless session beans are removed by the container at its own discretion.
  - ejbPassivate() – called by the container when about to passivate the bean to conserve resources (passivated beans may be removed completely if a time-out period elapses). This method has no meaning for a stateless session bean, and is generally left empty.
  - ejbActivate() – called by the container when about to activate a passivated bean (passivated beans are activated if a client request is received). This method has no meaning for a stateless session bean, and is generally left empty.

Stateless Beans – how not to use

- Stateless session beans do NOT represent the data.

Stateful Session Beans

- Implementation-wise, a stateful session bean is basically identical to a stateless session bean.

- The difference? For stateful beans:
  - You should define some state, i.e. instance variables.
  - Inside your business methods, you can manipulate the state (read and change the variable values).

- When a client creates a stateful session bean, that client is given their own personal instance of the bean that is used by the same client on successive method calls.

StatefulBeans – same basic rules

- To implement a stateful session bean, you need to create the bean’s home interface, its remote interface and the bean implementation – the same as for a stateless session bean.

- Again, your bean implementation class must implement the javax.ejb.SessionBean interface (which extends javax.ejb.EnterpriseBean).

- Again, your bean implementation class must implement the SessionBean (as well as your business methods) which are used by the container to create and manage your bean during its life.

Stateful Beans – differences!

- For stateful session beans, you may wish to perform some state management tasks in the ejbPassivate() and ejbActivate() methods.
  - E.g. save the state to a disk file during passivation, and load it back again during re-activation.

- Stateful session beans are only removed when the client calls ejbRemove() – the container will not remove stateful session beans unless they time-out.

- Stateful sessions are not closed when a client disconnects. The container will need to remove them at its own discretion.

- Stateful beans are removed by the container at its own discretion.

- Stateless beans do NOT represent the data.

Stateful Beans – same basic rules

- Stateless and stateful beans are equivalent – they come and go. Data is meant to be persistent.

- If a server crashes unexpectedly, a stateful session bean will be destroyed with the crash (same for stateless session beans). Data is meant to be persistent.

- Stateless beans do NOT represent the data. Data is represented by Entity Beans.

- Stateful session beans are transient – they come and go.

Stateful Bean – same basic rules

- Which a client creates() a stateful session bean, the bean’s home interface is passed to the client. The client interacts with the bean via its remote interface.

- The difference? For stateful beans:
  - Stateless beans are managed by the container, whereas stateful beans are managed by the client.

- You should define some state, i.e. instance variables.

- Inside your business methods, you can manipulate the state (read and change the variable values).

- When a client creates a stateful session bean, that client is given their own personal instance of the bean that is used by the same client on successive method calls.

- A stateful session bean is used by the same client on successive method calls.

- The container will not remove stateful session beans unless they time-out.

- Stateless beans do NOT represent the data. Data is represented by Entity Beans.

- Stateless and stateful beans are equivalent – they come and go. Data is meant to be persistent.

- If a server crashes unexpectedly, a stateful session bean will be destroyed with the crash (same for stateless session beans). Data is meant to be persistent.
When do you use stateless beans vs. stateful beans?

- In most cases, it depends on how you choose to model your application.
- Consider the bank teller example on the next two slides:
  - You create the BankTeller EJB, you supply: nothing
  - Your account number comes from a transaction to the bank.
  - The bank's system remembers your account number.
  - When you call deposit() and withdraw(), you pass any number of parameters every time. Account number and amount.
  - When you call deposit() and withdraw(), you pass two parameters every time: account number and amount.
  - Statefull beans: every time, you have to supply the bank with your account number.

Stateless vs. stateful implications:

- Stateless beans do not remember: you can go to any other teller each time you perform a transaction.
- Stateful beans do remember: you must go back to the same teller each time.

Stateful Scenario:
- The teller never forgets your account number.
- When you create() the BankTeller EJB, you supply: the account number.
- When you call deposit() and withdraw(), you pass only one parameter: the amount.
- When you call deposit() and withdraw(), you pass any number of parameters every time.

Stateless Scenario:
- The teller never remembers your account number.
- When you create() the BankTeller EJB, you supply: nothing.
- When you call deposit() and withdraw(), you pass two parameters every time: account number and amount.
Today we will briefly cover:

- Introduction to EJB
- EJB Basics
- EJB Architecture
- Session Beans
- EJB Clients

Next lesson we will cover:

- Entity Beans
- Message Driven Beans
- EJB issues
Today we will briefly cover:

- Introduction to EJB
- EJB Basics
- EJB Architecture
- Session Beans
- EJB Clients
- EJB Development Process

Next lesson we will cover:

- Entity Beans
- Message Driven Beans
- EJB issues

EJB Development Process

- Complete the EJB code
- Implement business interface
- Implement the EJB implementation class
- Implement the home interface
- Implement EJB business methods
- Implement EJB deployment descriptors

The steps to be followed when developing an EJB:

1. Create the remote interface
   - Defines business methods
2. Create the home interface
   - Defines create/find methods
3. Create the EJB implementation class
   - Implements the bean
4. Compile the java code
5. Write the EJB deployment descriptor
   - Create an ejb-jar.xml file and maybe others
6. Package the EJB into a JAR file
   - EJBs get packaged in JAR files
7. Compile EJB stubs
   - Use an EJB compiler tool
8. Configure the EJB deployment descriptors
   - Complete the EJB descriptor
9. Package EJB JAR into an EAR file (optional)
   - Applications are packaged in EAR files
10. Deploy the EJB JAR or EJB application EAR
    - Deploy to the application server

EJB Development Steps

- 10 steps involved in server-side EJB Development:

- Deploy to the application server
- Complete the EJB descriptor
- Configure the J2EE application deployment descriptor
- Create an ejb-jar.xml file and maybe others
- Complete the EJB code
- Implement business interface
- Implement the EJB implementation class
- Implement the home interface
- Implement EJB business methods
- Implement EJB deployment descriptors
- Complete the EJB compiler tool
- Complete the EJB descriptor
- Create an ejb-jar.xml file and maybe others
- Complete the EJB code
- Implement business interface
- Implement the EJB implementation class
- Implement the home interface
- Implement EJB business methods
- Implement EJB deployment descriptors

Checkpoints:

- EJB issues
- Message Driven Beans
- Entity Beans

Next lesson we will cover:

- EJB Development Process
- EJB Clients
- Session Beans
- EJB architecture
- EJB beans
- Introduction to EJB
- Today we will briefly cover:

Enterprise Java Beans
Client Development Steps

1. Create Remote interface

```java
package bank;
import java.rmi *
import javax.ejb *
interface BankTeller extends EJBObject {
    String sayHello() throws RemoteException;
    void deposit(String acctnum, Float amount) throws RemoteException;
    void withdraw(String acctnum, Float amount) throws RemoteException;
}
```

2. Create Home interface

```java
package bank;
import java.rmi *
import javax.ejb *
interface BankTellerHome extends EJBHome {
    public BankTeller create () throws RemoteException, CreateException;
}
```

3. Create EJB implementation

```java
package bank;
import javax.ejb *
public class BankTellerBean implements SessionBean {
    // ... same as before ...
}
```

4. Compile Java code

```bash
javac -d . *.java
```

5. Write EJB deployment desc.

- Standard deployment descriptor (META-INF/ejb-jar.xml)
  - EJB name
  - Java class names for remote interface, home interface and bean implementation class
  - Whether bean is session or entity
  - JNDI name and EJB name for remote interface, home interface, and bean implementation class
- Container-specific deployment descriptor (META-INF/weblogic-ejb-jar.xml)
  - JNDI name (name by which the bean is advertised to the client)
  - Whether bean is session or entity
  - JNDI name for remote interface, home interface, and bean implementation class
  - EJB name
  - Size of bean pool
  - Whether bean is stateful or stateless
  - Whether bean is transactional or not
  - Whether bean is initializable or not

4 Steps for writing EJB classes:

- Create Remote interface
- Create Home interface
- Create EJB implementation
- Compile Java code
9. Package app as a JAR file

- Same concept as for WAR files
- but EJBs are packaged in JAR files
- EJB deployment descriptors must be contained in a sub-directory off the root called "META-INF"
- Directory structure in JAR must mirror Java package structure (as per normal Java rules)

```
jar  cf ../MyEJB.jar *
```
Deploy

• Deploy your JAR file into the application server
• In WebLogic, either:
  – copy the JAR file into your “autodeploy” subdirectory
  – run the command line tool to deploy the file
  – upload the file via the management console
• Exactly the same process as deploying WAR files
• If all goes well, in the management console you should see:
  – a new EJB appears under the “Deployments > EJB” menu
  – a new entry appears in the JNDI tree

Naming Conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise bean name</td>
<td><code>&lt;name&gt;EB</code></td>
<td>BankTellerEJB</td>
</tr>
<tr>
<td>EJB JAR deploy name</td>
<td><code>&lt;name&gt;JAR</code></td>
<td>BankTellerJAR</td>
</tr>
<tr>
<td>Enterprise bean class</td>
<td><code>&lt;name&gt;Bean</code></td>
<td>BankTellerBean</td>
</tr>
<tr>
<td>Home Interface</td>
<td><code>&lt;name&gt;</code></td>
<td>BankTellerHome</td>
</tr>
<tr>
<td>Remote Interface</td>
<td><code>&lt;name&gt;</code></td>
<td>BankTeller</td>
</tr>
<tr>
<td>Local home interface</td>
<td>Local&lt;name&gt;Home</td>
<td>LocalBankTellerHome</td>
</tr>
<tr>
<td>Local interface</td>
<td>Local&lt;name&gt;</td>
<td>LocalBankTeller</td>
</tr>
<tr>
<td>Abstract Schema</td>
<td><code>&lt;name&gt;</code></td>
<td>BankTeller</td>
</tr>
</tbody>
</table>

EJB 3.0

• Part of Java EE 5
• Significantly simplifies EJB development
• Just plain old Java Objects!
  ```java
  @stateless(name="BankTeller")
  public class BankTeller implements BankTellerRemote {
      public String sayHello(String name) {
          return "Hello, " + name;
      }
  }
  ```
• Note: no XML, no “housekeeping” code!

Summary – EJB Part 1

- Only 3 Java files, 2 very simple
- Plus 2 deployment descriptors
- Only 3 Java files, 2 very simple
- Reflect on how little code was required

- What the container does
  – How EJBs are mocked/looked etc.
  – The JEE architecture and how to model the real world
- Hard part about EJB is understanding
  – EJB makes even simpler
  – Plus 2 deployment descriptors
  – Only 3 Java files, 2 very simple
  – Reflect on how little code was required