EBU6501 - Middleware
Week 4, Day 2: Architecture of Enterprise JavaBeans

Gokop Goteng & Ethan Lau

Queen Mary
University of London

Original version created by: Dr. Gokop Goteng

Lecture Aim and Outcome

- Aim
  - The aim of this lecture is to extend the Java programming students from J2Se to J2EE using EJB
- Outcome
  - At the end of this lecture students should be able to:
- Identify the important differences between EJB and POJO (Plain Old Java Objects)
- Describe the lifecycle of EJB
- Write simple stateless and stateful EJB applications

Lecture Outline

- EJB: Enterprise JavaBeans
- EJB Architecture
- EJB Bean
- The Professional Roles in EJB Implementation
- Types of Enterprise Beans
- Lifecycles of Stateless Session Bean, Stateful Session Bean and Entity Bean
- The Bean Class
- Support for Distributed Transactions
- The Role of the EJB Container
- The Contracts Between EJB, Container and Client
- Persistence in EJB Beans
- Benefits of using EJB Technology
- Some Example Codes
- Class Task

EJB: Enterprise JavaBeans

- Enterprise JavaBeans (EJB) is a **component java architecture and specification** used for the **server-side management** and **development** of **modular enterprise applications** based on the **J2EE** (Java 2 Enterprise Edition) specifications
  - EJB architecture defines the functions of the following EJB entities:
    - EJB Clients
    - EJB Servers
    - Enterprise Beans
    - EJB Containers

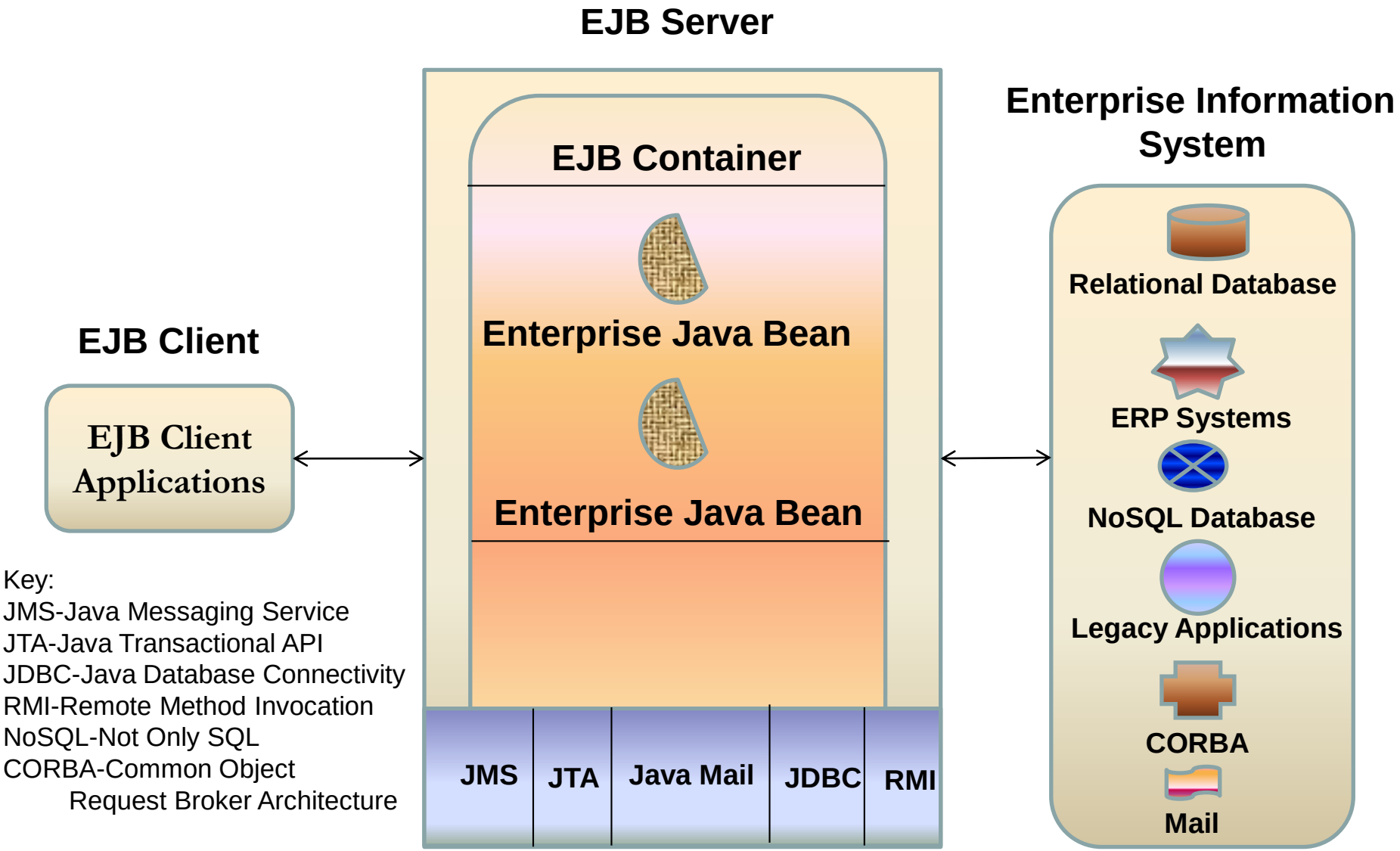
EJB: Enterprise JavaBeans

- EJB **consists** of **reusable Java components**, **Java classes** and **XML files**
  - All these **function** as **one single unit**
- EJB components encapsulate the use case and business logic of its implementation so that it hides the details of system implementation from the programmer
  - This improves the efficiency and productivity of programmers
- EJB specification is one of the J2EE APIs (Application Programming Interfaces) specifications

EJB responsibilities

- Transaction processing
- Integration with the persistence services offered by the Java Persistence API (JPA)
- Concurrency control
- Event-driven programming using Java Message Service and Java EE Connector Architecture
- Asynchronous method invocation
- Job scheduling
- Naming and directory services (JNDI)
- Security
- Deployment of software components in an application server

EJB Architecture



Key:
JMS-Java Messaging Service
JTA-Java Transactional API
JDBC-Java Database Connectivity
RMI-Remote Method Invocation
NoSQL-Not Only SQL
CORBA-Common Object Request Broker Architecture

EJB Architecture

- **EJB Client**
  - The client **interacts** with Enterprise Beans using **IIOP** and proxy objects **protocols**
    - IIOP (Internet Inter-ORB Protocol) is the protocol that allows **distributed applications** written in **different languages** to **communicate/interact** over the internet
    - IIOP is an **important component** of **CORBA**
    - IIOP enables the bean to **integrate RMI/IIOP** applications
- **EJB Server**
  - This provides the **environment** for the **EJB containers** to be **executed** and **run**
  - It **provides** the **systems services**
    - Load balancing, multi-processing and device access
  - Manages **system resources** for **containers**
    - Network connections, threads, memory, databases
  - Provides **concurrent access** to **beans**
  - Provides the **implementation** of **java naming directory interface** (JNDI) server
  - It can **host one or more containers**

EJB Architecture

- EJB Container
  - It is the **interface** between **EJB** and **low-level functionalities** that support the Bean (EJB Server)
  - Helps the EJB Client to **access** the **Bean** through **container-generated class** which **invokes** the **bean** methods
  - The security management is based on the **declarations** in the **deployment descriptor** (DD) rather than being **hard-coded**
  - It **manages** the **life cycle** of **beans** to use server resources such as memory and threads
  - It makes the services required to beans through an interface defined in the EJB specification

EJB Bean

- It is the **EJB Server**
- EJB Bean consists of:
  - **Home Interface**
    - This enables clients to create, remove and find bean instances
  - **EJB Class**
    - This is what provides the **implementation of actual functionalities** and **business logics** using **methods** defined in the **Remote Interface** in addition to implementing the creation and finder methods in the **Home Interface**
  - **Remote Interface**
    - This **exposes** and **makes** the **business logic** of the bean available to the clients
  - **Deployment Descriptor (DD)**
    - This is where the **configuration** of the **properties** of the **bean** are **declared**
      - The security, transactional and persistent state management policies are declared in the DD

The Professional Roles in EJB Implementation

- **EJB Server Provider**
  - Does implementation and provision of **access** to **JNDI naming service**
  - Implements the **transaction service** with **CORBA/OTS** (Object Transaction Service)
  - Provides the **environment** to run **EJB containers**
- **EJB Container Provider**
  - Provides the **deployment tools** to generate **EJB object implementation**, create **stubs** and **classes** to provide access to EJB instance's Home object and EJB object
  - Does **installation** of **EJB Bean**
  - **Registers the reference** to the Home object in the JNDI namespace
  - Provides the **runtime classes** and **services** required by EJB Bean instances
- **EJB Provider**
  - Does the **development/coding** of the business logic server components
  - **Need to be knowledgeable** in the EJB specification and should be able to **describe** the **transactional requirements** of **EJB class methods** to the **EJB deployer**

The Professional Roles in EJB Implementation

- **EJB Deployer**
  - Is knowledgeable in the workings of the run-time server environment such as database
  - Does **installation and deployment of the bean** and classes on the EJB server
  - **Registers** the **Home interface** to the **JNDI**
- **Application Assembler**
  - Does **coding** of the **client applications** using pre-built EJB components
  - Configures and customise pre-built EJB components for the assembled applications
- **Systems Administrator**
  - A **person** that is an administer and manages the EJB application when it has been deployed to a target environment

Types of Enterprise Beans

- **Session Beans**
  - Created and destroyed by EJB client
  - **Non-persistent**
    - If you shutdown the server, it will go out of session (session terminated)
  - Stores data of a particular user for a single session
  - It is associated with **only one EJB Client**
  - There are two types of session beans
    - **Stateful** and **Stateless** session beans
- **Entity Beans**
  - Can be shared by **multiple EJB Clients**
  - It has **persistent** states and **can survive server shutdown or crash**
  - It **always** have a **state**
  - It encapsulates a unique ID that points to the state
  - The class that implements it implements EntityBean interface
- **Message-Driven Beans**
  - This is based on passing messages across different components for management, security and network connections.

Home Interface

- Consists of factory methods to **create**, **remove**, **locate EJB bean** instances
- It is **inherited** from the javax.ejb.EJBHome
- This is usually **created** by the **EJB Developer** for each bean type
- Example of how it inherits from EJBHome

```
public interface javax.ejb.EJBHome extends Remote {
    public abstract void remove (Handle handle)
   throws RemoteException, RemoveException;
public abstract void remove (Object primaryKey)
    throws RemoteException,RemoveException;
public abstract EJBMetaData getEJBMetaData ()
    throws RemoteException;
}
public interface EJBMetaData { EJBHome getEJBHome();
    Class getHomeInterfaceClass();
   Class getRemoteInterfaceClass();
   Class getPrimaryKeyClass();
   boolean isSession();
  boolean isStatelessSession();
}
```

Remote Interface

- The Remote Interface is **defined by the EJB Developer**
- It lists the **business logic methods provided** by the **Enterprise Bean to the Client**
- The remote interface can extend the EJBObject Interface:

```
public interface EJBObject extends java.rmi.Remote {
    public EJBHome getEJBHome() throws
        java.rmi.RemoteException;
    public Object getPrimaryKey() throws
        java.rmi.RemoteException;
    public void remove() throws java.rmi.RemoteException,
        java.rmi.RemoveException;
    public Handle getHandle() throws java.rmi.RemoteException;
    boolean isIdentical (EJBObject p0) throws java.rmi.RemoteException; }
```

The Declarative EJB XML File

- Name of the File: ebj-jar.xml

```
<?xml version="1.0"?>
<ejb-jar>
 <enterprise-beans>
   <session>
     <ejb-name>OrderManagement</ejb-name>
     <home>orderMgmt.OrderManagementHome</home>
     <remote>orderMgmt.OrderManagement</remote>
     <ejb-class>orderMgmt.OrderManagementBean</ejb-class>
     <session-type>Stateless</session-type>
     <transaction-type>Container</transaction-type>
   </session>
 </enterprise-beans>
<assembly-descriptor>
 <container-transaction>
   <method>
     <ejb-name>OrderManagement</ejb-name>
     <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
 </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Support for Distributed Transactions

- EJB Server **coordinates** the **distributed transaction**
- The **EJB Container handles** the **distributed transactions** for **beans**
- **Java Transaction Service** (JTS) **supports EJB transactions**
- JTS also takes responsibility of coordinating all resources participating in transactions
- The **deployment process** consists **of transaction policies** that are defined in the **declarative statements**
  - **EJB Container** manages the **start, commit** and **rollback** of transactions
- **Transactions** can be **controlled** by both the **client applications** and the **bean** themselves

The Role of the EJB Container

- **Components** are **executed** within the **container environment**.
- The container houses components as well as provides the services that the components need
- The container is a **subset** of **the application server** and the **application server** provides the **environment** for all containers to **run**
- The container **provides persistence** services and information during c**onnection** and **encapsulates** data access in components
- Provides data **caching**
- Provides **declarative capabilities** for security and transactions
- It handles **errors** arising from transactions
- Provides **scalability** and **fall-over** features
- Provides **portability** across multiple platforms and programming languages
- Provides **management functionalities** for EJB systems

The Role of the EJB Container

- The "**contract**", which is the different roles played by EJB Client, EJB Container and EJB Components allow the container to **manage the clients** and the **components**
- The container manages the **persistence transactions** and **security separate** from the **Java files** that implement the business logic to provide a **loosely coupled** and **modular system**
- The container **interposes** itself between the client and the EJB components to facilitate all business method calls between the client and the components

Persistence in EJB Beans

- **Persistence** in computing defines the **retention of data** or **information** for a **long time** even in the event that there is a **crash** or **shutdown** of the system or services
- **Bean-Managed Persistence**
  - The **Entity Bean saves and restores** its **own state**
  - The commands for saving and restoring the state are part of the bean's code
  - This is **not the best way** of implementing persistence in beans
- **Container-Managed Persistence**
  - This is the **recommended way** of implementing persistence
  - **EJB container saves the states** of the **beans**
  - The **variables** used to store and restore the state are declared in the **DD**
  - The Java serialisation stored the bean's instance
  - The connection parameters to database (datasource) is defined in the DD
  - The datasource is separate from the bean's codes that implements the business logic

EJB and RMI

- EJB **uses RMI** when it **makes method calls** to a **remote client** by:
  - The EJB initiates a call to the remote method invocation (RMI) stub
  - The **RMI stub** then comes in **between** (interposes) **the method call** and the **remote client** to send the information across the network
  - A **component** on the **server** then **obtains** the **information and parameters** and **send** them to the **EJB Container**

POJO and JavaBeans

- A POJO (Plain Old Java Object) is the **normal Java object created** from **normal Java Class**.
- A POJO **does not** need to **extend any class** or **implement any interface**.
- A **JavaBean** is a **POJO** that is **serialisable**
  - Serialisation is the **process** of **converting data structures** (Arrays, List, ArrayList, Maps, Table, etc) **or objects into a format** that can be **stored** and then **reconstructed again** in **another different environment**
- A JavaBean has **no-argument constructor**
- A JavaBean allows access to application properties using **getter** and **setter** methods using simple Java naming conventions
  - Because of this, declarative properties of arbitrary JavaBean references can be made in a program

JavaBean Declarative Properties File for Hello World

```
<enterprise-beans>
    <session>
        <ejb-name>helloWorld</ejb-name>
        <ejb-class>com.example.HelloWorldService</ejb-class>
     <session-type>stateless</session-type>
    </session>
</enterprise-beans>
```

Benefits of using EJB Technology

- **Portability**
  - EJB components are portable across clients written in different programming languages and runs on any operating system
- **Modularity**
  - The implementation of the bean is divided into different components
    - For example the Home Interface, Remote Interface, Java implementation classes and declarative XML are separate from each other
- **Scalability**
  - The implementation of EJB is scalable and multi-tier
- **Distributed Transactions**
  - The architecture of EJB is transactional in nature and supports distributed transactions
- **Security and Administration**
  - The declarative security settings make it secure and enhances easy administration and management of services and components

Class Work:

The architecture of Enterprise JavaBeans (EJB) shows the relationships between different EJB components. Using your knowledge of EJB architecture:

- Briefly describe the functions of EJB Client, EJB Server and EJB Container in Enterprise JavaBeans (EJB) architecture. **[6 marks]**
- Describe Data Persistency in EJB **[2 marks]**
- Describe TWO types of EJB persistency **[4 marks]**

Class Work:

Create the JavaBean declaration file as would appear in the deployment descriptor (DD) file of Apache Tomcat Container. The EJB name should be “myEJBFile”, the EJB class name should be “myEJBClass.” and session type should be “stateful” **[5 marks]**