Lecture 10

Enterprise Java Beans
Question

• No one writes their own OSes, Databases, Browsers, Wordprocessors, Spreadsheets, hardware, etc. anymore. Why?
  – They buy the commodity software and build the stuff they can’t easily buy. Exception is extreme situations of execution speed or problem domain.
  – Cheaper, quicker, economies of scale, vendors concentrate expertise, well defined interfaces to/from purchased products, support, documentation, upgrades, ....
Enterprise Application Issues

- Platform Independence
- Scaleability
- Maintainability
- Fault tolerance / Fault Resilience
- Shorten time to market
- Lower development costs
- Security
- Legacy Integration
- Business process integrity
- Runtime tuning of distributed applications
Platform Independence

- Code once run anywhere
- Mix technology platforms
- Lower cost
- Flexibility
- Migrate applications easily when changing platforms
- Code on one platform deploy on another
Scalability

- Need to make applications serve many users
- Ability to just add hardware (boxes), package software (OSes), instances of existing software and serve more users
- Web makes estimating the number of users of a system and their usage patterns (time of day) very difficult.
  - Ex: stock trading system does 67% of total daily volume in first 15 minutes of the day and the last 15 minutes of the trading day.
Scaleability Example
Maintainability

• 4/5 to 19/20 the cost of a software project can be AFTER the code is written, tested and deployed!
  – Administration
  – Bug fixes
  – Enhancements
  – Disaster recovery
  – Training Users and Admins
Maintainability (2)

- Economies of scale – many customers can share the cost of bug fixes, support, enhancements, upgrades, …
- Package software has management and administration interfaces built in: MBeans / JMX, SNMP, Tivoli, …
Fault tolerance / Fault Resilience

- Hardware fails – CPU, memory, disk, I/O, network
- Software fails – OS crashes, application crashes, memory leaks, GPF, unhandled exceptions
- Fail over – switch from one running application to another copy on the same box or a different box without loss or interruption of the user operations.
Fault tolerance / Fault Resilience (2)

- **Storing state** (user name, address, account number) in memory complicates failing over (data could get lost so needs to be dealt with by code). Writing everything to disk makes failover easier but impedes scalability.

- **Session affinity** – if a HTTP connection times out the client machine could reconnect to a different HTTP server (Server B) but pass in the existing session Id. Server B must somehow get the information about the session and initialize its application components.
Shorten time to market

• Make valid and scalable code easier to write and debug
• Things that are hard for the average developer:
  – Transactions
  – Threads
  – Concurrency – simultaneous access to objects or database rows
  – Object persistence – storage and reactivation
  – Security
  – Remoting – RMI, CORBA, Sockets, …
  – Messaging – sending msgs between processes or boxes
Lower development costs

• Simpler to code
• Simpler to debug
• Simpler to reuse
• Simpler to understand
• Simpler to manage in runtime
• Simpler to extend, enhance, migrate
• Lower the skill level required to build my solution
Security

- **Identification** – who do you say you are - login
- **Authentication** – verifying your are that entity – password, id token, fingerprint
- **Authorization** – what can you see and do
- **Encryption** – preventing others from seeing the data
- **Integrity** – proving that the data wasn’t tampered with.
- **Non-repudiation** – sent me order and now you are trying to say you didn’t – digital signatures
- **User Access Management** – granting and revoking access privileges and entitlements.
Legacy Integration

• Existing systems written in numerous:
  – **Languages**: C, COBOL, ForTran, Smalltalk, C++, Powerbuilder, VisualBasic
  – **OS**: Mainframe (IBM OS/390, Unisys, Cyber, …), VAX, Unix, Windows, …
  – **Hardware**: IBM, HP, DEC, Compaq, Dell, …
  – **Component Models**: DCE, CICS, IMS, CORBA, DSOM, DCOM, COM+, …
  – **Datastores**: RDB, VSAM, QSAM, Sybase, Oracle, Ingress, Postgress, IMS, IMDB, DB2, MS SQL Server, …
  – **Data format**: fixed position/length, comma delimited, tagged, XML, Serialized Object, …
Runtime tuning of distributed applications

• Sharing boxes
• Sharing databases
• Sharing network
• Different applications may have different usage patterns. Ex:
  – Email gets used most heavily in the morning
  – Time recording gets used most heavily just before close of time period
  – Stock trading application gets used most heavily start of day and end of day
  – Quake gets used most actively during lunch
Enterprise Application Issues

- Platform Independence
- Scaleability
- Maintainability
- Fault tolerance / Fault Resilience
- Shorten time to market
- Lower development costs
- Security
- Legacy Integration
- Business process integrity
Solution

• A component framework that takes the hard parts of coding and does them for the developers
  – Security
  – Transactions
  – Threading

• A framework that supports multiple hardware and OS configurations
Solution (2)

• A component framework that runs application components and can control their behavior and lifecycle dynamically
• An easy to learn component framework.
• A component framework supported by tool vendors
  – IDE
  – Debuggers
  – Profilers
  – Runtime
Solution (3)

• A component framework that supports interfacing with legacy systems
  – Via standard interfaces SAP, PeopleSoft, …
  – Via messaging – MQ Series
  – Via CORBA
  – Via DCOM
  – Via web services interfaces
AKA

• Application Server
• App Server
• Object Server
• Object Transaction Server (older term)
• Examples:
  – Orbix OTS - Component Broker
  – COM+ - MTS
  – J2EE EJB Servers – WebSphere, WebLogic
• The evolution of CORBA, RPC, Tuxedo, Encina, DCE, CICS an IMS.
Java J2SE – Standard Edition

- I/O – Files, Streams, Pipes
- Swing(GUI)
- Applets – run in a web page on the client
- RMI (Remote Method Invocation) – remotely calling other object’s methods
- Math
- JavaBeans – client side component model
- Security
- 2D Graphics
- Internationalization / Localization
J2EE – Enterprise Edition

- **EJB (Enterprise Java Beans)** – server side component model. Business logic and data logic components.
- **Servlets** – server side active web pages written in java. Runs on web server.
- **JSP (Java Server Pages)** – HTML + java mingled together that gets compiled into servlet automatically by the web server.
- **JDBC** – database access API
- **JNDI (Java Naming and Directory Interface)** – finding other objects and database tables, etc. Looking up distributed things in a central repository.
- **JavaMail** – email API
- **JMS (Java Message Service)** – message oriented middleware
- **JCA (Java Connector Architecture)** – standard adapters to legacy systems like mainframes or SAP r/3
Java Technologies

copyright SUN © 2002
Java API Relationships

copyright SUN © 2002

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
Enterprise Java Beans!

• J2EE EJB is spec not a product.
• Multiple vendors have EJB Application Server products that implement the spec
  – JBoss (freeware)
  – BEA WebLogic
  – IBM WebSphere
  – Oracle 9aIS
  – Many more
EJB

- Beans are components
- One or more classes
- Contain business logic or data manipulation/persistence logic
- Standard interfaces must be defined in your EJB (that's what makes it an EJB)
  - Lifecycle of the bean - interface
  - Client access to business logic - interface
EJB’s are managed by containers

- The EJB server provides a container to manage the EJB’s.
- The EJB is instantiated and lives inside the container.
- A container:
  - Intercepts communication between the client and the component to allow automation of infrastructure code, such as transactions and security.
  - Communicates with the component using direct function calls.
Types of EJBs

• Session Beans
  – Statefull
  – Stateless
  – Message Driven

• Entity Bean
Statefull Session EJB

• **Session Bean**
  – **Statefull** – state is maintained in memory between client calls. Multiple method calls can reference instance variable because they survive between calls. A client gets the same bean that they called previously in the *session*.
  – Ex: **Shopping cart**
Stateless Session EJB

- **Session Bean**
  - **Stateless** – no state is maintained between method calls on the bean. Any client can use any bean. Independent of any client. VERY SCALABLE!
  - No client information is passed to the bean (hence stateless)
  - Ex: the **Catalog** model of the catalog in the database with read only access via client
Entity EJB

- **Entity Bean**
  - Bean is identified with a specific piece of data i.e. CustomerX’s information.
  - Has a unique key that is used to “lookup” the bean by a client.
  - Last longer than a session maybe forever! (actually live in database. In memory is a cached version of the disk based representation.)
  - Usually a given instance of bean.
  - Ex: **customer info** model or **customer order (PO)** model of the data stored in the database
Message Driven Bean EJB

- **Message Driven Bean (MDB)**
  - A stateless session bean that gets invoked by a message coming in over JMS not by a call from a client application.
  - Used to invoke other beans.
  - Ex: Mainframe invoking an EJB

![Diagram showing Legacy System, Queue, and EJB App Server connections](image-url)
Session, Entity and Message Driven Beans

- Clients normally talk to session beans.
- Common pattern to mask entity beans with session beans.

![Diagram](image-url)
Comparison of Session EJBs

• Stateless
  – Supports heavy loads.
  – No conversational state to manage.
  – Not synchronized with transactions.
  – No parameters when created.
  – Performs tasks independent of the client connected.

• Stateful
  – Consumes more resources on the server.
  – Allows you to manage the conversational state.
  – The conversational state can be synchronized with transactions.
  – Takes parameters when created.
  – Performs tasks for a single client.
• **Session EJB.**
  – Represents a conversation between the client and the server.
  – Does not survive a server restart.
    • Example: Browsing a product catalog.
    • Example: A shopping cart allowing you to buy items.

• **Stateful and stateless session EJB’s.**
  – Stateful: The session EJB stores values relating to the user’s session.
  – Stateless: the session EJB does not store any values relating to the session.

• **A better name for stateless session beans might be “service beans.”**
To write an EJB

- Developer needs to define:
  - Interface for the home (so the container can create the ejb)
  - Remote interface for the bean (so the container can create a factory object for us)
  - Implement the bean

```
<<interface>>
MyBeanHome

create()

<<interface>>
MyBean

businessMethod1()
businessMethod2()

MyBeanImpl

ejbCreate()
businessMethod1()
businessMethod2()
```
To write an EJB (2)

- Define the home interface for your bean and the container will generate a `ejbHome` object for you.
- `ejbHome` object is a class factory for your bean.
Cart implementation – statefull session EJB

import java.util.*;
import javax.ejb.*;
public class CartBean implements SessionBean {
    String customerName;
    String customerId;
    Vector contents;

    public void ejbCreate(String person) throws CreateException {
        //implementation here
    }

    public void ejbCreate(String person, String id) throws CreateException {
        //implementation here
    }

    public void addBook(String title) { contents.addElement(title); }

    public void removeBook(String title) throws BookException,
    RemoteException {//impl;}

    public Vector getContents() throws RemoteException {//impl;}

    public CartBean() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
Cart Home interface

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CartHome extends EJBHome
{
    Cart create(String person) throws RemoteException, CreateException;
    Cart create(String person, String id) throws RemoteException,
       CreateException;
}

Cart Remote interface

```java
import java.util.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Cart extends EJBObject {

    public void addBook(String title) throws RemoteException;
    public void removeBook(String title) throws BookException,
        RemoteException;
    public Vector getContents() throws RemoteException;
}
```
FundsTransfer EJB – stateless session bean

public class FundsTransferEJBImpl implements SessionBean
{
    private SessionContext context;

    public void ejbCreate() throws CreateException { }
    public void ejbRemove() { }
    public void setSessionContext(SessionContext context)
    {
        context = this.context;
    }
    public void ejbPassivate() { }
    public void ejbActivate() { }

    public void transferMoney(String from, String to, double amount)
    { // business logic in here
        System.out.println("Transfer of "+ amount + " from "+ from + " to "+ to);
        // do work here
    }
}
import javax.naming.;
import javax.rmi.;
public class FundsTransferClient {
    public static void main(String[] args) {
        try {
            Context myContext = new InitialContext();
            Object objref = myContext .lookup("TransferHome"); //JNDI lookup
            FundsTransferEJBHome homeForEJB = (FundsTransferHome)PortableRemoteObject.
                Narrow (objref, FundsTransferEJBHome.class);

            FundsTransferEJB transferEJB = homeForEJB.create(); //EJB class factory

            transferEJB.transferMoney("Bill Gates", "Logan Poelman", 1000.00);
            transferEJB.remove(); //says we are done with the bean
        }
        catch (Exception e) {
            System.err.println("Unexpected exception!"+ e);
        }
    }
}
Message Driven Beans

- public void onMessage(Message inMessage)
- This is the only “business” method on a message driven bean. It gets called when a message comes in to the queue that is configured with this bean.
EJB lifecycle methods

```plaintext
ejbCreate()     //constructor
ejbRemove()     //going to be destroyed, **not**
                 //guaranteed to get called

```

```

```plaintext
ejbActivate()    //get ready to work
ejbPassivate()   //get ready to get swapped out
                 //possible multiple passivate/activate
                 //cycles before destruction

```

```

```plaintext
ejbLoad()       //refresh the data from the DB
ejbStore()      //update the DB

```

---

All EJB Types

Statefull Session

Entity
EJB lifecycle – Stateless Session Bean

1. setSessionContext
2. ejbCreate

Does Not Exist

ejbRemove

Ready
EJB lifecycle – Statefull Session Bean

Does Not Exist

1. create
2. setSessionContext
3. ejbCreate

1. remove
2. ejbRemove

Ready

Passive

ejbPassivate

ejbActivate
EJB lifecycle – Entity Bean

- **Does Not Exist**
  - setEntityContext
  - unsetEntityContext

- **Pooled**
  - ejbActivate
  - ejbPassivate
  - 1. create
  - 2. ejbCreate
  - 3. ejbPostCreate

- **Ready**
  - 1. remove
  - 2. ejbRemove
EJB lifecycle – Message Driven Bean

1. setMessageDrivenContext
2. ejbCreate

onMessage

Does Not Exist

ejbRemove

Ready
What happens when my EJB gets passivated?

- I should close any connections I’ve created (database), free any resources I’m holding.
- My state will get serialized to disk (maybe a RBDMS or a file)
- The memory used gets freed up.
What happens when my EJB gets activated?

- My state will get deserialized from disk into memory (from the RBDMS or a file)
- I should reopen any connections I’ve closed (database), allocate any resources I’m need.
- I am now in memory though I have not been executed yet
Entity Bean Methods

- `ejbCreate()` //creates a record in the database
- `ejbRemove()` //deletes a record from the database
- `ejbPassivate()` //serializes it out
- `ejbActivate()` //deserializes it into memory
- **New methods**
  - `ejbLoad()` and `ejbStore()`
- Synchronize the state of the bean with the disk or vice versa.
- Persistence existence of the bean even beyond shutting down the app server and restarting it
EJB lifecycle – Entity Bean

- ejbCreate
  - ejbLoad
    - ejbActivate
    - ejbPassivate
  - ejbStore
- ejbDelete
Entity Bean Methods

`findByPrimaryKey(String accid)`
Allows finding an entity EJB causing instance variables to be set from storage
Transactions

- Transactions are a way of grouping together operations into a single unit of work.
  - Classic example: `transfer()` consists of `withdraw()` and `deposit()`.
    ```java
    beginTransaction();
    acct1.withdraw(100);
    acct2.deposit(100);
    commitTransaction();
    ```
  - Either both methods should execute or neither should.

- Transactions are ACID.
  - Atomic – all operations occur or none (they get rolled back)
  - Consistent – a transaction should transform the system from one consistent state to another consistent state.
  - Isolated – changes made during the transaction aren’t visible outside the code in the transaction until the transaction commits.
  - Durable – changes get written to disk and survive a server crash.
Transaction are important for:

• Money transfers
• Stock trades
• Air line reservations
• Business value operations
• Inventory systems
• Payroll systems
• Fulfillment systems
• Most things that cost or generate money
J2EE and Transactions

• Transactions can be controlled by by:
  – The client.
  – The container.
  or
  – The bean.
Deployment

• Transaction requirements of a bean can be set in the deployment descriptor (parameter file) instead of changing the code

EJB Transaction Attributes

– RequiresNew – bean always starts a new transaction
– Required – starts a txn if none is passed to the bean
– Supports – if a txn is passed to it it will use it otherwise it won’t create
– NotSupported – bean doesn’t do things transactionally but will accept a txn
– Mandatory – you must send a txn in to the bean
– Never - throws an error if you pass it a txn
Container “tricks”

• Pool of Stateless Session Beans kept in memory. Request comes in a bean is taken out of the pool. – **Thread reuse**.

• Concurrent requests for a given entity bean are sequenced so they each get access but in order not simultaneously. Means other requests are blocked while someone is using the entity bean. – **Concurrent management/serialization**
Container “tricks”

• A EJB can be invoked by a client. The invocation is intercepted by the container. The container then loads the EJB into memory and then calls the method. The client doesn’t know the difference. – Autoactivation

• Beans that are in memory but not used recently can be either destroyed (Stateless) or passivated (statefull or entity) to disk. Then the memory freed up. – Instance management.

• Number of threads in the app server can be controlled to tune performance to the platform – Managed thread pooling.
Application Server “tricks”

- App Server can be clustered and distribute requests to other servers in peak demand times - **Clustering**.
- Resources (EJBs) can be moved from one app server box to another to support failover - **Failover**.
- Multiple EJBs can be aggregated into a transaction without adding any code to the EJBs themselves to do distributed transactions easily – **Container managed transactions**.
Thing you CAN’T do in an EJB

• An EJB cannot:
  – Access the local disk using the java.io package.
  – Create threads.
  – Use the synchronized key word.
  – Use the java.awt or javax.swing packages.
  – Listen to a socket or create a socket server.
  – Modify the Java SocketFactory.
  – Use native libraries.
  – Use read/write static variables.

• To perform these processes, you must put them on an external RMI or CORBA server.
Questions

• How do you “find” a bean to invoke it?
  – Via a naming and lookup service - JNDI
• What if two clients want to change the same client object (entity bean)?
  – Concurrency management
• What if the app server needs to use memory due to a heavy influx of requests for some EJBs not currently in memory?
  – Passivate and persist to disk currently loaded EJBs
  – Load new EJBs into memory
Questions (2)

• What if the App Server has 20 copies of my EJB running on threads but there are 300 clients trying to use it?
  – Serialization (different use of the term) clients and only twenty at a time get access. When a client request is done, the thread gets returned to the pool and then the next client “in line” gets the thread.
Summary

• EJB – business logic and data management code.
• Reusable
• Manageable
• “Do the hard work” for developers:
  – Transactions
  – Concurrency
  – Resource balancing
  – Runtime management
Summary (2)

• EJB (J2EE) – based on J2SE.
• Based on RMI
• Developer needs to create:
  – Home interface definition
  – Remote interface definition
  – Implement the code of the EJB
• Server
  – Creates the EJBHome object factory
  – Manages the lifecycle of the EJB
Summary (3)

• Types of EJBs
  – Session
    • Stateless – ex: Catalog Bean
    • Statefull – ex: Shopping Cart bean
    • Message Driven – ex: adapter for legacy system access to EJB
  – Entity
    • Persistent
    • Not client related (not session related lifecycle)
    • Ex: PO, CustomerInfo
resources

• Transactions
  http://www.subrahmanyam.com/articles/transactions/NutsAndBoltsOfTP.html

• EJBs tutorials
  http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBJBConcepts.html

• JBoss
  http://www.jboss.org/