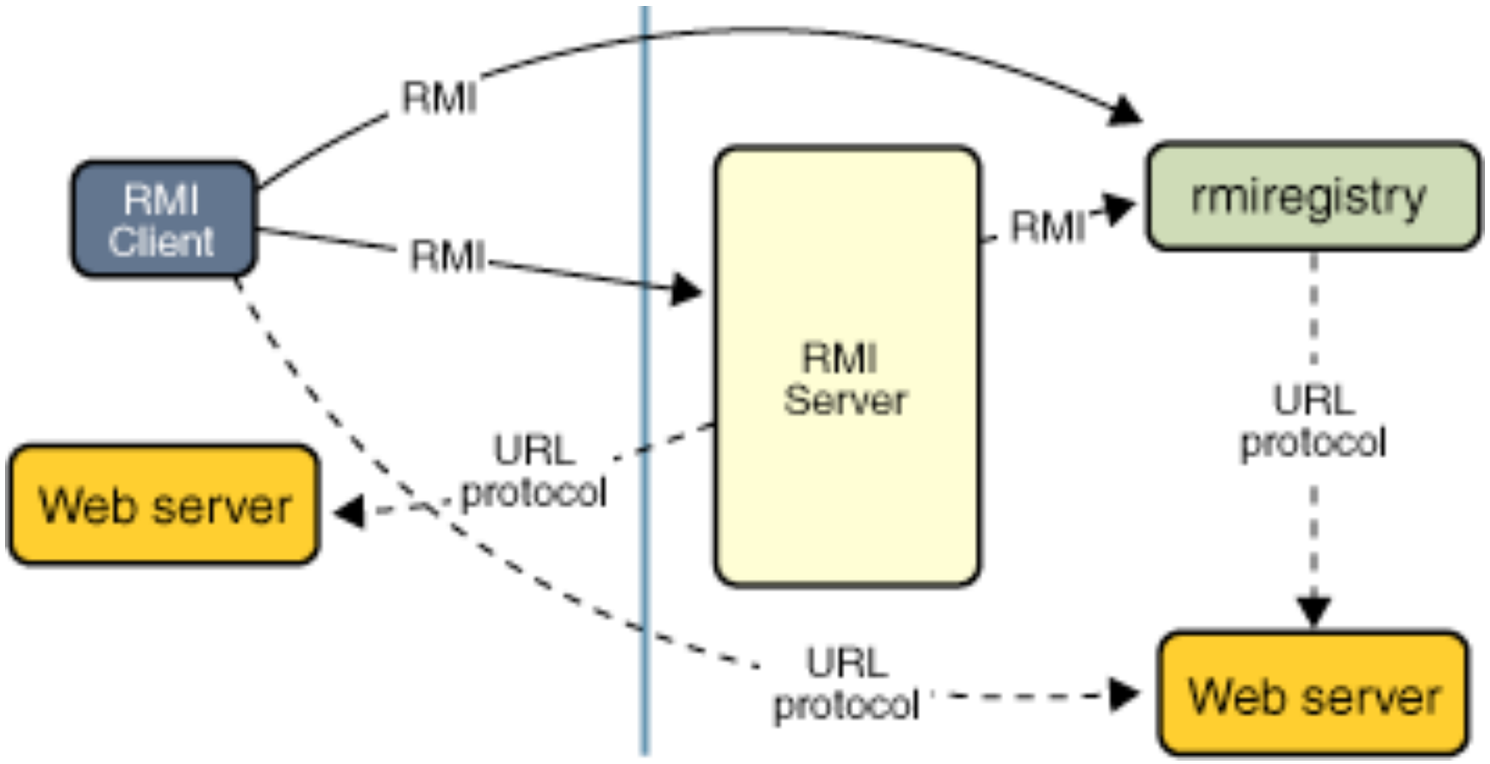
Enterprise Java Beans

RMI

- Thread running in one JVM can invoke a method on an object inside another JVM



Implement Interface, Server, Client

```
public interface RemoteCalculator extends Remote {
    BigDecimal calculateSin(Integer val) throws RemoteException;
    User getUser(Integer id) throws RemoteException;
}

public class RemoteCalculatorImpl implements RemoteCalculator{

    @Override
    public BigDecimal calculateSin(Integer val) throws RemoteException {
        System.out.println("Calculating the sin...");
        double retVal = Math.sin(val);
        BigDecimal bd = new BigDecimal(retVal);
        return bd;
    }

    @Override
    public User getUser(Integer id) throws RemoteException {
        System.out.println("Getting the user...");
        User u = new User();
        u.id=1;
        u.name = "Manav";
        u.email="manav@gmail.com";
        return u;
    }

}
```

Server & Client JVMs

```
public class RMIServer {
    public static void main(String[] args) throws RemoteException {
        RemoteCalculator stub =
                (RemoteCalculator) UnicastRemoteObject.exportObject(new
                    RemoteCalculatorImpl(), 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind("myapp/remotecalc", stub);
            System.out.println("Remore calculator bound");
    }
}

public class RMIClient {
    public static void main(String[] args) throws AccessException, RemoteException,
NotBoundException {
        String name = "myapp/remotecalc";
        Registry registry = LocateRegistry.getRegistry("localhost",1099);
        RemoteCalculator calc = (RemoteCalculator) registry.lookup(name);
        System.out.println(calc);
        User u = calc.getUser(1);
        System.out.println(u.getName());
    }
}
```

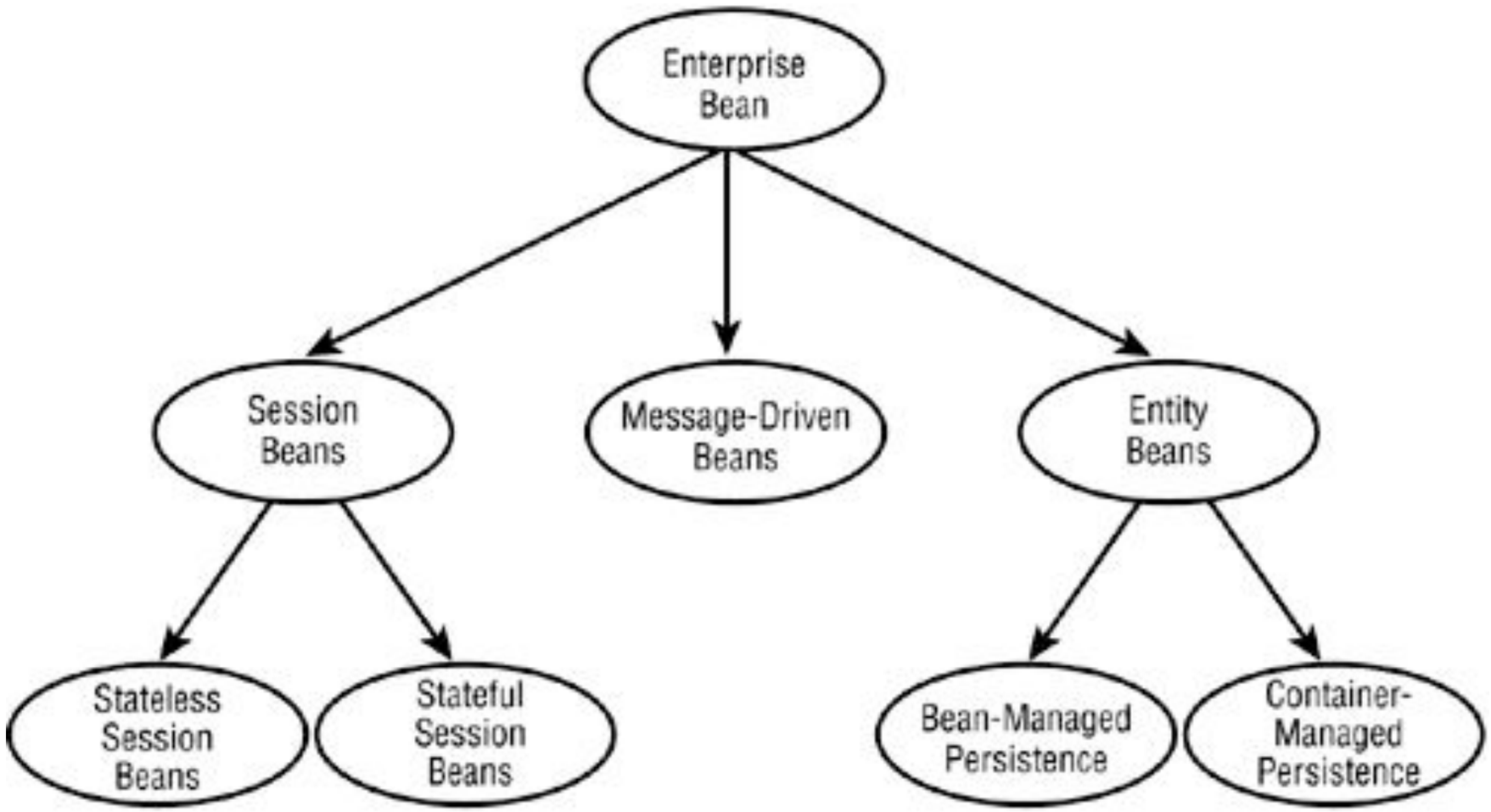
Proxy

```
public class ProxyImpl implements java.lang.reflect.InvocationHandler {

    private Object obj;

    public static Object newInstance(Object obj) {
        return java.lang.reflect.Proxy.newProxyInstance(obj.getClass()
                .getClassLoader(), obj.getClass().getInterfaces(),
                    new ProxyImpl(obj));
    }

    private ProxyImpl(Object obj) {
        this.obj = obj;
    }

    public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
        Object result;
        try {
            //do something before
            result = m.invoke(obj, args);
            //do something after
        } catch (Exception e) {
            throw new RuntimeException("unexpected invocation exception: "
                    + e.getMessage());
        }
        return result;
    }
}
```

What Is EJB

- Enterprise Java Beans (EJB) is a development architecture for building highly scalable and robust enterprise level applications to be deployed on JEE compliant Application Server such as JBOSS, Web Logic etc.

Types

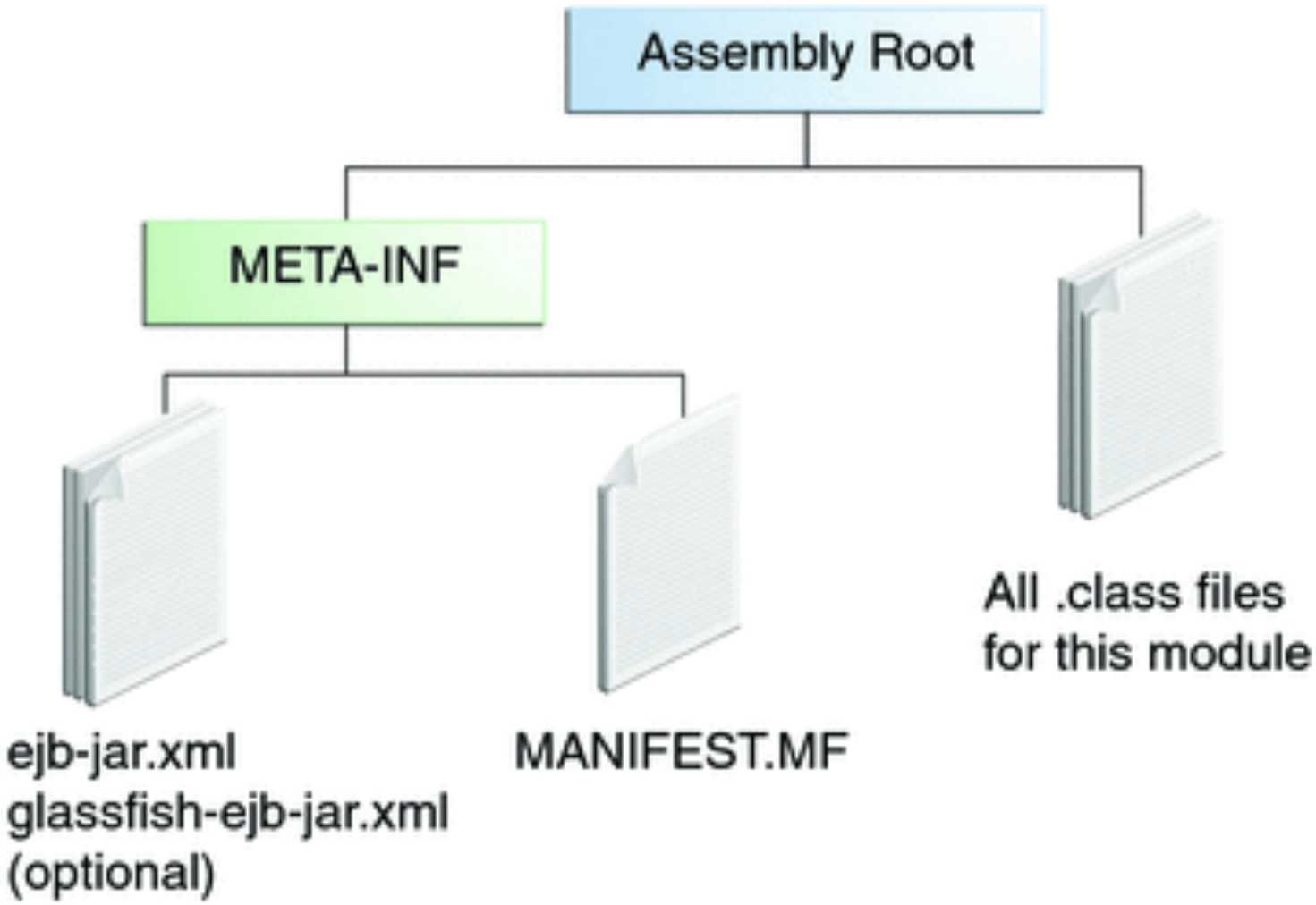


EJB Types

SLSB Structure

```
@Stateless
@LocalBean
public class Calculator implements CalculatorRemote, CalculatorLocal {

    public Integer add(int i, int j) {
        return i+j;
    }
}

@Local
public interface CalculatorLocal {
    Integer add(int i, int j);
}

@Remote
public interface CalculatorRemote {
    Integer add(int i, int j);
}
```

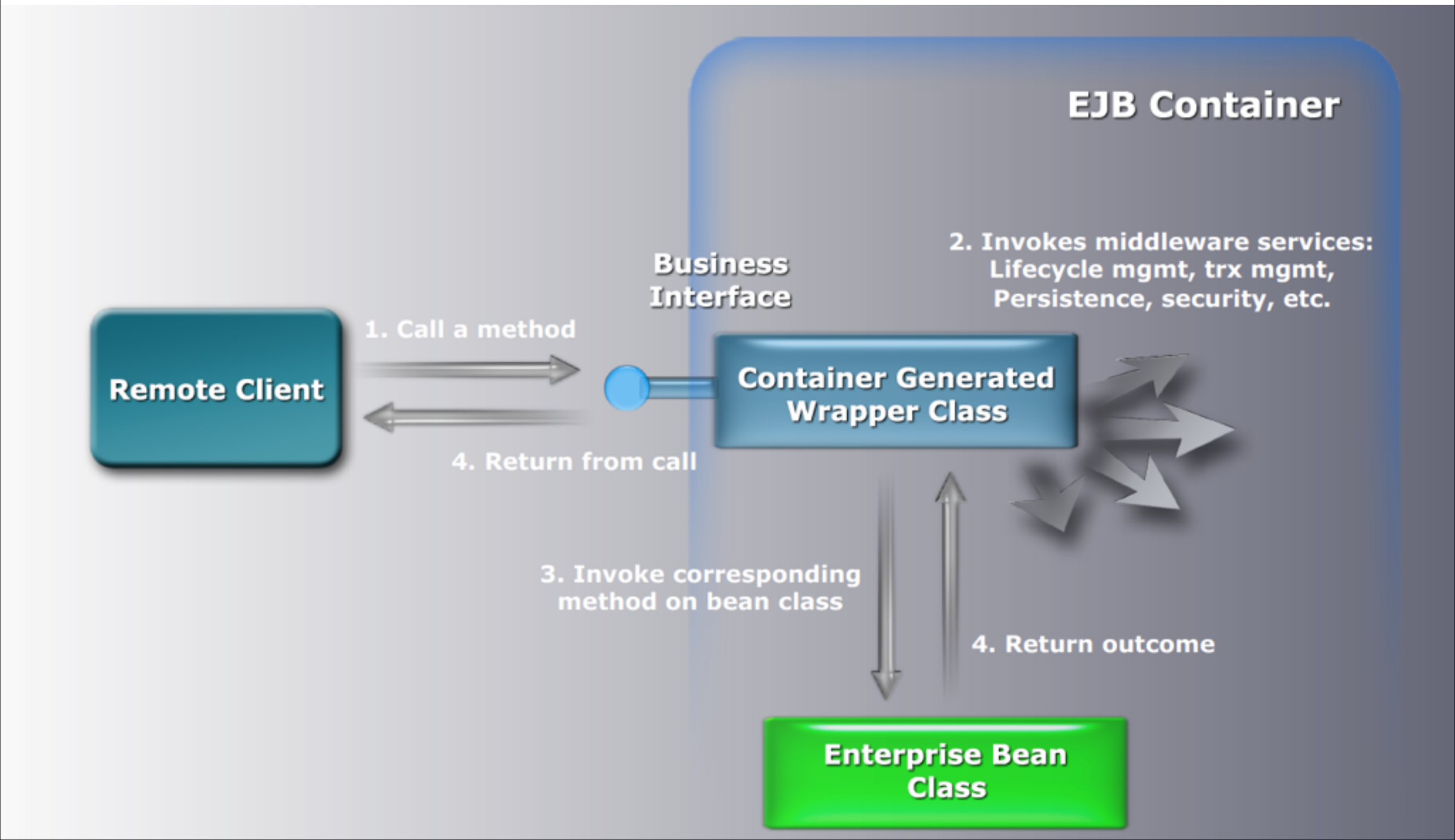
EJB Deployment Structure



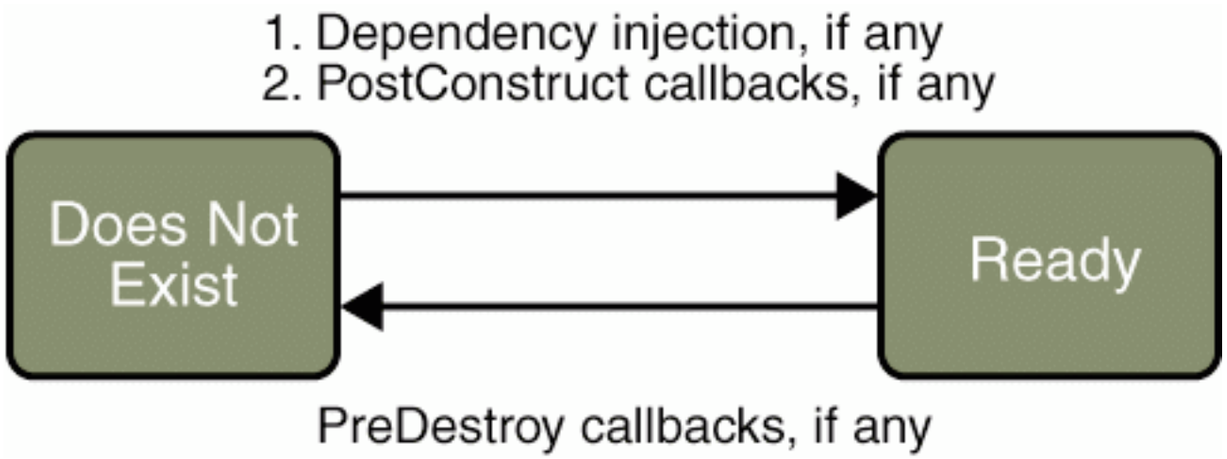
EJB Client

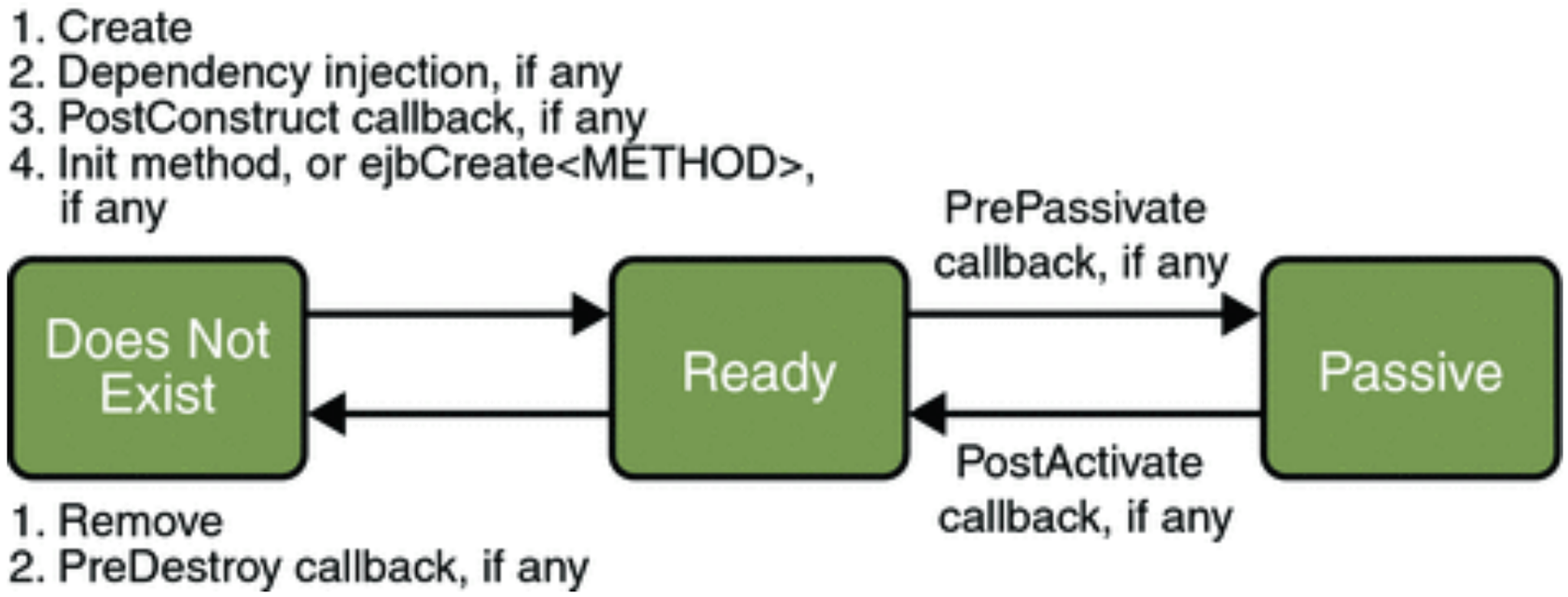
```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,”org.jboss.ejb.client.naming
”);
jndiProperties.put("jboss.naming.client.ejb.context", true);
final Context context = new InitialContext(jndiProperties);
CalculatorRemote remote = (CalculatorRemote) context.lookup("ejb:/EJB//
Calculator!com.mydomain.CalculatorRemote");
System.out.println("Sum = " + remote.add(3, 5));
```

EJB Flow


EJB Container
Business Interface
2. Invokes middleware services: Lifecycle mgmt, trx mgmt, Persistence, security, etc.
Remote Client
1. Call a method
Container Generated Wrapper Class
4. Return from call
3. Invoke corresponding method on bean class
4. Return outcome
Enterprise Bean Class

Lifecycles

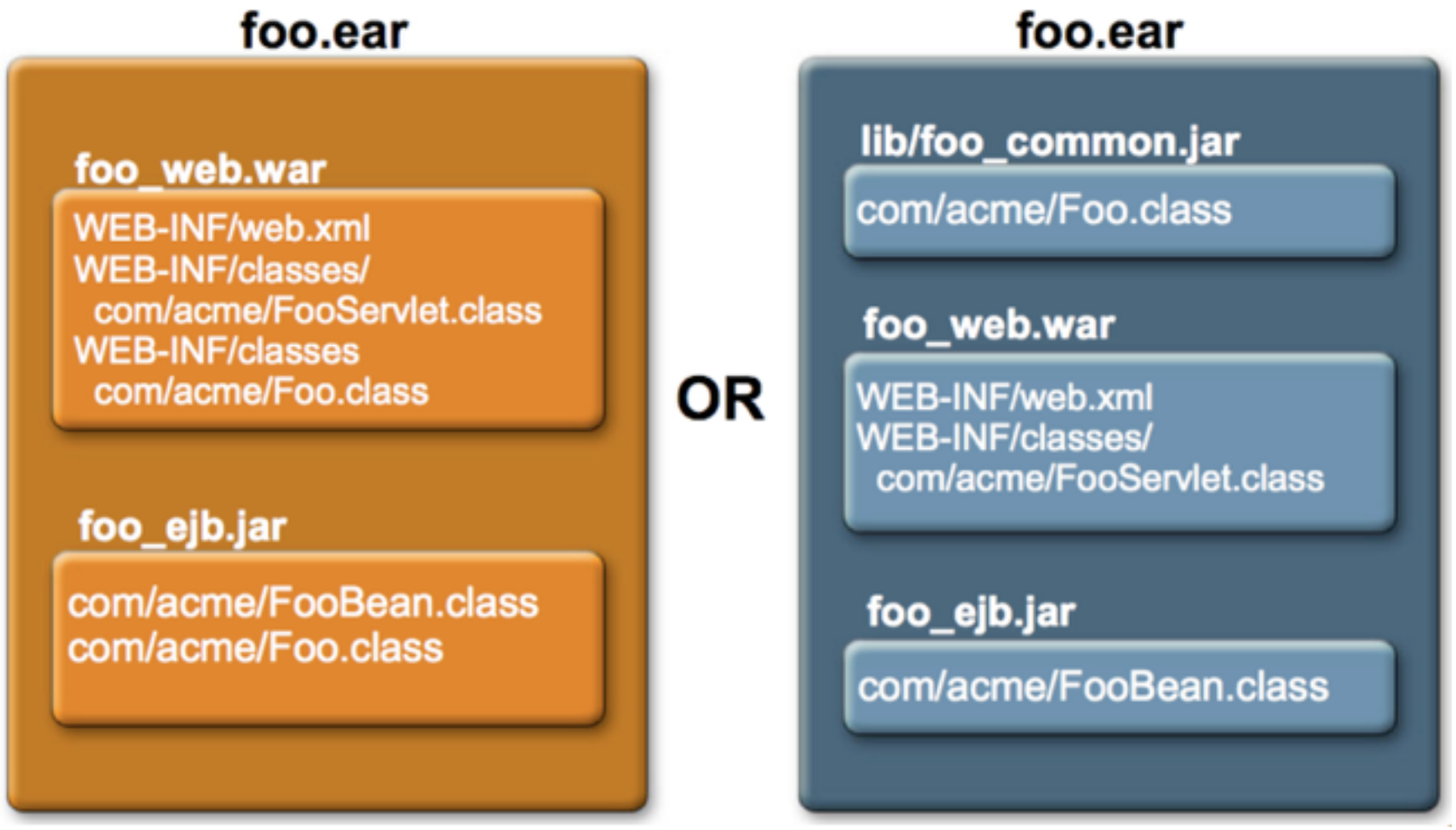

1. Dependency injection, if any
2. PostConstruct callbacks, if any
Does Not Exist
Ready
PreDestroy callbacks, if any


1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any
PrePassivate callback, if any
Does Not Exist
Ready
Passive
PostActivate callback, if any
1. Remove
2. PreDestroy callback, if any

Container Services

- Bean Pooling
- Lifecycle Management
- DB Connection Pooling
- Transactions
- Security
- High Availability
- Persistance
- Session Management for Stateful

Packaging Options



EJB Lite

- EJB Lite:
  - Local Session Beans
  - CMT / BMT
  - Declarative Security
  - Interceptors
  - Web Profile also includes JPA 2.0
- Full = Lite + ...
  - Message Driven Beans
  - EJB Web Service Endpoints
  - RMI-IIOP Interop
  - 2.x / 3.x Remote view
  - 2.x Local view
  - Timer Service
  - CMP / BMP Entities

Lets Try EJB Lite

- Convert UserManager to SLSB

```
@Stateless
public class UserManagerBean implements UserManagerLocal
```

```
@Local
public interface UserManagerLocal {
```

```
In servlet:
@EJB
UserManagerLocal userManager;
```

  - Use the userManager in doGet/doPost

DB Connection Pools

- App Servers provide for a database connection pool
- Reduces connection creation overhead
- Limits max connections to db
- Keeps connections under app server control

Configure A DataSource

- Deploy the derbyclient.jar in modules\org\apache \derby\main along with a module.xml (Shared)

- Setup a driver and datasource segment in standalone.xml (Shared)

- Obtain connection using datasource in EJB

```
InitialContext ic = new InitialContext();
DataSource ds = (DataSource) ic.lookup("java:/DerbyDS");
con = ds.getConnection();
```

Implement Bulk Delete

- Add checkboxes for each user

```
<input type="checkbox" name="selectedIds" value="<%=user.getId()%>"/>
```

- Get selected user ids on server

```
String[] idStrArr = request.getParameterValues("selectedIds");
```

- Loop thru the ids and call delete on userManager Bean

- Break one entity delete by inserting an order for that user

```
Insert into orders (ship_to_address,order_value,user_id) values
('Bangalore',300,<id>)
```

Implement Bulk Delete Method

- Create a deleteAll(List<Integer> ids). This method calls the delete(id) method in a loop
- Call this method from bulk delete servlet
- Test… finally wrap the Exception and throw a RuntimeException

Create Orders For Each User

- Write a new EJB method to loop over all the users and for each user insert 5 orders (shared).
- Throw an exception if the user name starts with “Error”
- Observe transaction behaviour

Method Transaction Attributes

- MANDATORY
- NEVER
- NOT_SUPPORTED
- REQUIRED
- REQUIRES_NEW
- SUPPORTS

Bean Managed Transactions

- Transaction Boundaries are demarcated

```
@Resource
private SessionContext sctx;
```

- Session context methods

```
sctx.getUserTransaction().begin();
sctx.getUserTransaction().commit();
sctx.getUserTransaction().rollback();
```

- BMT Beans are one way - no incoming transactions

Transaction Methods in CMT

- Transaction Status can be set and obtained

```
sctx.setRollbackOnly();
sctx.getRollbackOnly();
```

Transfer Users

- Transfer selected user to another database.
    - Create a copy of the database and create another datasource for that in standalone.xml
    - Create a function to transfer user

```
User u = getUser(id);
createUser(con2, u);
deleteUser(id);
```

    - Test the transfer
    - Cause an exception in deleteUser

Distributed Transactions

- Distributed transactions are performed on multiple transactional resources involved in the same transaction

- Two phase commits are used to improve the chances of transaction success

Async Calls

- EJB methods can be marked with @Asynchronous annotation to make them async

- Make the orders generation method async

Creating Timers

- Methods can be marked to run on a timeout by using the @Timeout annotation

- Then the following code can be used to initialise the timer

```
ScheduleExpression sc = new ScheduleExpression();
sc.second("*/10");
sc.minute("*");
sc.hour("*");
TimerConfig tc = new TimerConfig();
tc.setPersistent(false);
context.getTimerService().createCalendarTimer(sc,tc);
```

Creating a Schedule

- Scheduled methods dont need to be initialised like a timeout method

- An annotation like below is sufficient to create the schedule

```
@Schedule(minute="*/1", hour="*", second="0", persistent=false)
```

Interceptors

- Interceptors on EJB are very similar to filters on servlets.
- Interceptor methods have a special signature and annotation. The interceptor methods can be defined on the bean class or in a separate class
- There can be only one interceptor method per class

```
@AroundInvoke
public Object methodInterceptor(InvocationContext ctx) throws Exception {}
```

- call ctx.proceed to chain the call. The EJB on which we need interception needs to have this annotation: @Interceptors({LoggingInterceptor.class})
- Log the time taken by a method in an interceptor

Interceptors

- Interceptors can be bound in three different ways:
- Default
  - is invoked whenever a business method is invoked on any bean within the deployment.
- Class level
  - Applied only to a class
- Method level
  - Applied only to a method

Default Interceptors

- Default interceptors can only be configured in ejb-jar.xml

```
<assembly-descriptor>
   <!-- Default interceptor that will apply to all methods for all beans
in deployment -->
   <interceptor-binding>
      <ejb-name>*</ejb-name>
      <interceptor-class>com.mydomain.MyInterceptor</interceptor-class>
   </interceptor-binding>
</assembly-descriptor>
```

Bean Pools

- Container by default manages a pool of beans for use for each Session Bean.
- Beans in a pool are all shared across different invocations
- In a Stateful Session bean, the beans in the pool are not shared across invocations.

Singletons

- One instance per App Server (Not cluster wide)
- Works via regular @EJB dependency injection aswell as look ups
- Class has to be annotated with @Singleton

Concurrency On Singletons

- Container and Bean managed concurrency options are available
- Container can provide method level locking
  - READ lock: allow any number of concurrent accesses to bean instance
    @Lock(LockType.READ)
  - WRITE lock (default): ensure single-threaded bean instance access @Lock(LockType.WRITE)

SOAP

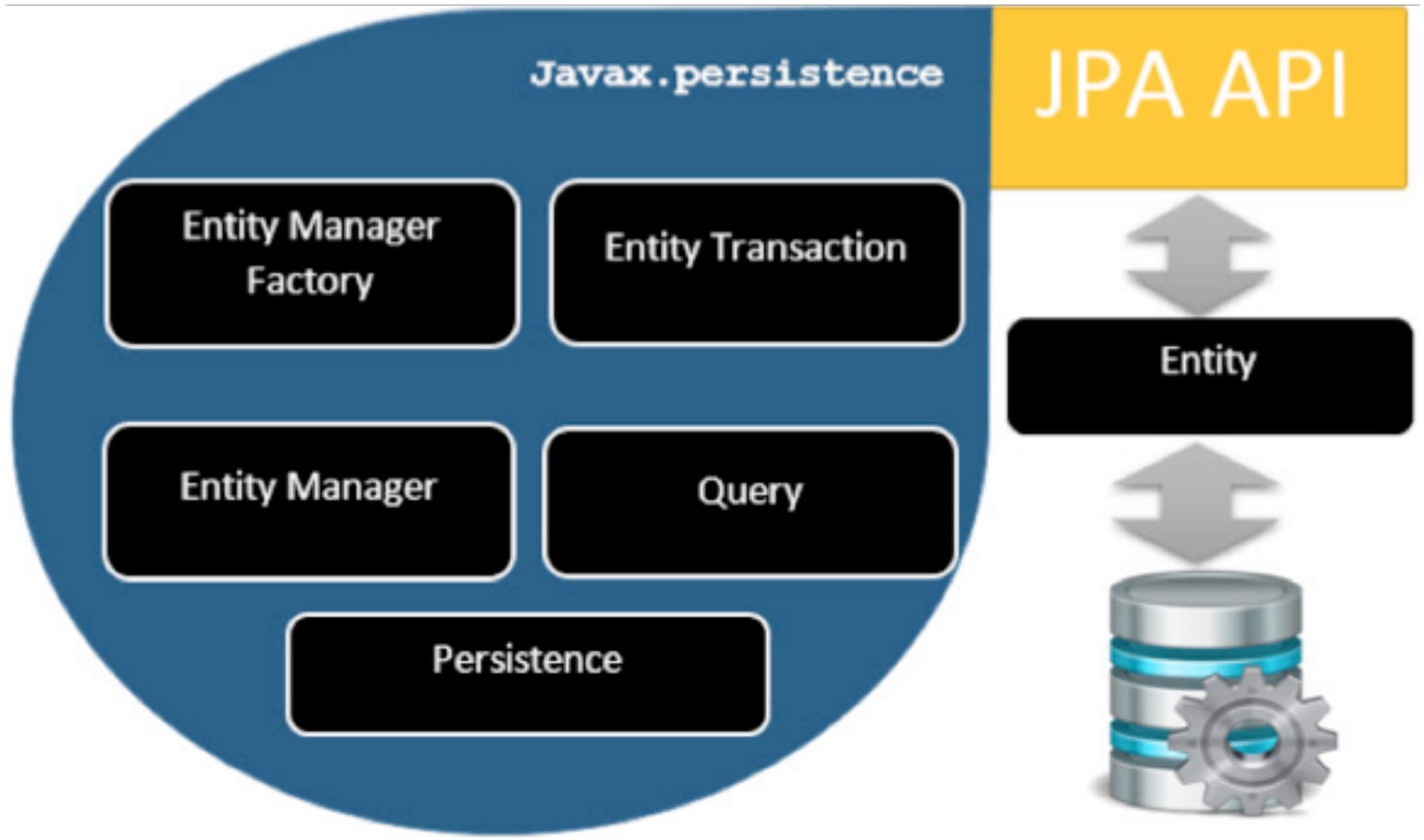
- SOAP stands for Simple Object Access Protocol
- Is a communication protocol supporting both RPC and Message style interaction
- Used for communication between applications
- communicates via Internet on http
- It is a W3C recommendation

EJB WebService Endpoints

- EJBs can be exposed as WebService endpoint by adding this annotation to the top of the bean class
@WebService

- WSDL for the web service is available on the url http://localhost:8080/<APPNAME>/<BEANNAME>?wsdl

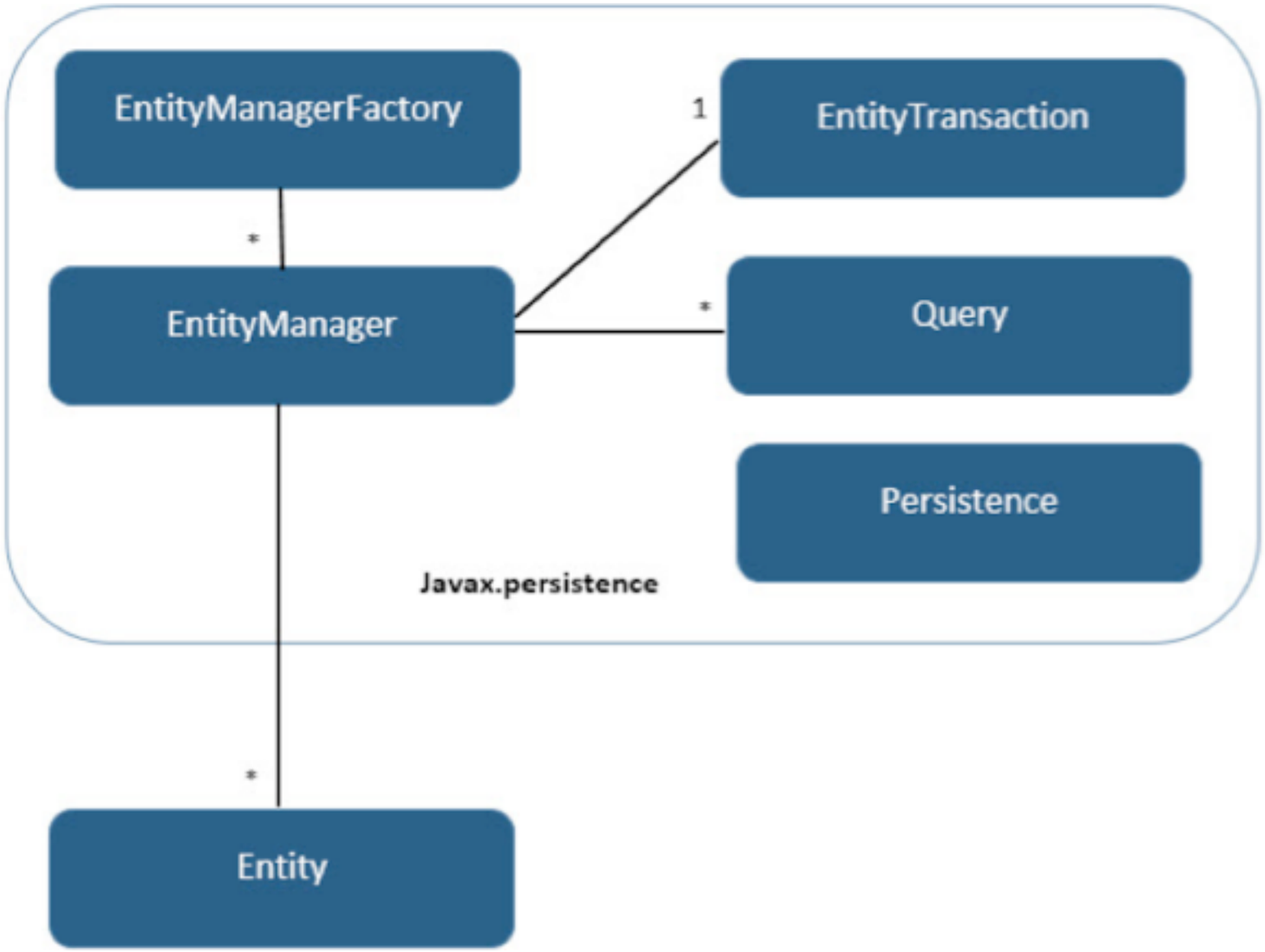
Persistance With JPA



JPA API

EntityManagerFactory	This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager	It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
Entity	Entities are the persistence objects, stores as records in the database.
EntityTransaction	It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class.
Persistence	This class contain static methods to obtain EntityManagerFactory instance.
Query	This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

JPA API Relationships



Lets USE JPA

- Define a persistence unit in persitance.xml in WEB-INF/classes/META-INF

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0">
    <persistence-unit name="UserPU">
        <description>Entity Beans for User</description>
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>java:/DerbyDS</jta-data-source>
        <class>com.mydomain.model.User</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="create" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.dialect"
value="org.hibernate.dialect.DerbyDialect" />
        </properties>
    </persistence-unit>
</persistence>
```

Define Entities

- Define Entity annotation on the model class

```
@Entity
@Table(name="users")
```

- Define the primary key and mark it as auto generated

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
```

- Define any columns that are different

```
@Column(name="EMAIL_ID")
@Column(name="JOIN_DATE")
```

Define Named Queries And Use

```
@NamedQueries({@NamedQuery(name = "AllUsers", query = "SELECT u
from User u”)})
```

In UserManager:

```
@PersistenceContext(unitName="UserPU")
    protected EntityManager em;

Query q = em.createNamedQuery("AllUsers");
List<User> userList = q.getResultList();
```