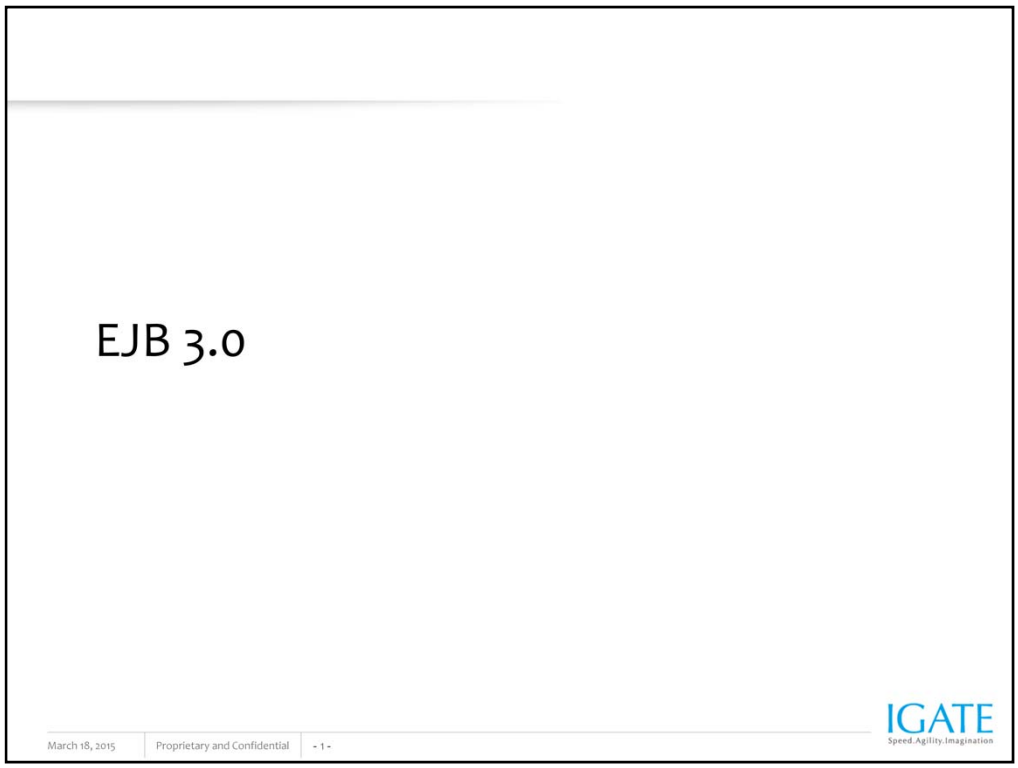

EJB 3.0
March 18, 2015
Proprietary and Confidential
- 1 -
IGATE
Speed.Agility.Imagination

Day-wise Schedule

- ***Enterprise JavaBeans Concept***
  - 1.1 Introduction to Distributed Objects System
  - 1.2 Enterprise JavaBeans
  - 1.3 JavaBeans and Enterprise JavaBeans
  - 1.4 EJB's Architecture
  - *Java™ 2 Enterprise Edition (J2EE) Platform*

Day-wise Schedule

- *Session Beans*
- *Java Persistence API [JPA]*
- *Database Connections*
- *Transactions*
- *Security*
- *Clients*

Enterprise JavaBeans Concept

- **Enterprise JavaBeans (EJBs) are write-once, run–anywhere, middle-tier components.**
- **Transaction Processors**
  - Allows several statements to be executed together as one logical unit.
  - Manages the execution of programs and sharing of resources.
  - Maintain a pool of running instances and hand them out as needed in response to user request.

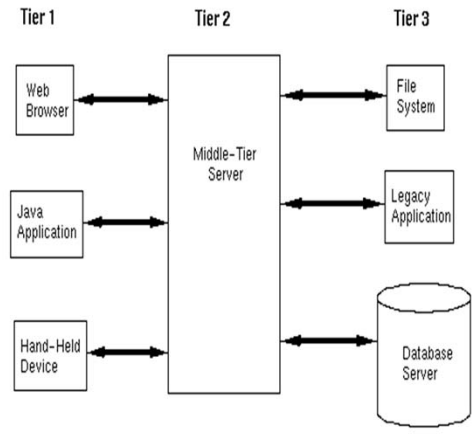
Two Tier Architecture

- In two-tier system, transaction integrity is usually guaranteed by the database management system itself. The application runs on the client machine and it contains both presentation logic as well as the business logic.
- Hence the moment the business logic changes the application needs to be redistributed.

Three-Tier Architecture

- In this approach the database is the ultimate repository of information.
- The client remains responsible for providing user interface logic.
- The business rules, however, are separated into middle tier.
- These kinds of architectures can be implemented using wide variety of approaches, including the following:
  - Sockets -RPCs -CORBA
  - RMI - OLE/DCOM -Message Queues.

A 3-Tier Client/Server Appl.



Shields back-end complexity.

Support variety of clients.

Provide Remote Access.

Session & Transaction Mgmt.

Security enforcement.

Resource pooling.

IGATE
Speed.Agility.Imagination

What is Enterprise JavaBeans

- It is an architecture for component-based distributed computing.
- Enterprise beans are components of distributed transaction-oriented enterprise applications.

Enterprise JavaBeans (EJB)

- Defines a standard to building distributed serve-side systems
- Free the EJB developer to only concentrate on programming the business logic
- Handles transactions, security, connection pooling etc., by delegating it to the vendor
- Defines a framework for creating distributed enterprise middleware applications

Enterprise JavaBeans (EJB)

- Beans contain the application's business logic so client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases.
- The clients become thinner which is a benefit important for clients that run on small devices.

Differences between JavaBeans & Enterprise JavaBeans

- **Can be either visible or non-visible**
- **Intended to be local to a single process on the client side**
- **Uses the BeanInfo classes, Property Editors & Customizes to describe itself**
- **Can also be deployed as ActiveX controls**

- Are Non-Visible remote objects
- Remotely executable components deployed on the server
- Uses the Deployment Descriptor to describe itself
- Can't be deployed as ActiveX control since OCXs run on desktop

IGATE
Speed.Agility.Imagination

Why Enterprise JavaBeans ?

- Architectural Independence from middleware
- WORA for server side components
- Establishes roles for application development
- Takes care of Transaction Management
- Provides Distributed transaction support
- Helps create Portable & Scalable solutions
- Integrates seamlessly with CORBA

WHEN EJB ?

- Scalable Application with large number of users
- For Application with large number of transactions [OLTP] to ensure data integrity
- Application with variety of clients.

EJB Architecture

- **EJBs run in EJB container, a runtime environment within the Application Server**
- **The Enterprise Java beans architecture defines following types of EJB objects :**
  - Session Object
  - Message-driven Object
  - Entity Object

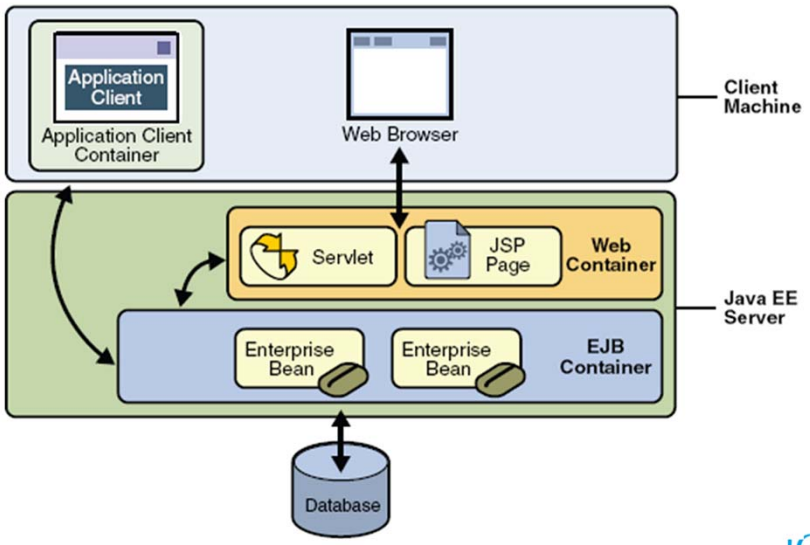
Container

- *Container is* the interface between a component and the low-level platform-specific functionality that supports the component.
- Web application, enterprise applications are assembled into a Java EE module and deployed into its container before they are executed.

EJB Roles

- **EJB Architecture defines seven distinct roles in *the application development and deployment life cycle.***
  - Enterprise Bean Provider
  - Application Assembler
  - Deployer
  - EJB Sever Provider
  - EJB Container Provider
  - Persistence Provider
  - System Administrator

Java Enterprise Server & Container



The EJB Server

- Provides an organized framework for EJB Containers to execute in
- Provides system-services like multiprocessing, load balancing, device access, JNDI accessible naming and transaction services available to the container
- Makes EJB Containers visible

The EJB Container

- Provides a distributed object infrastructure
- Deployment of beans
- EJB Life Cycle Management
- Declarative transaction management
- Bean Activation and Passivation
- Bean state management
- Provides security

Session Bean

- ***A typical session object has the following characteristics:***
  - *Executes on behalf of a single client.*
  - *Represents business process*
  - *Can be transaction-aware.*
  - *Updates shared data in an underlying database.*
  - *Is not persistent object by itslef, although it may perform database operations.*
  - *Is relatively short-lived.*
  - *Is removed when the EJB container crashes. The client has to re-establish a new session object to continue computation.*

Message-Driven Object

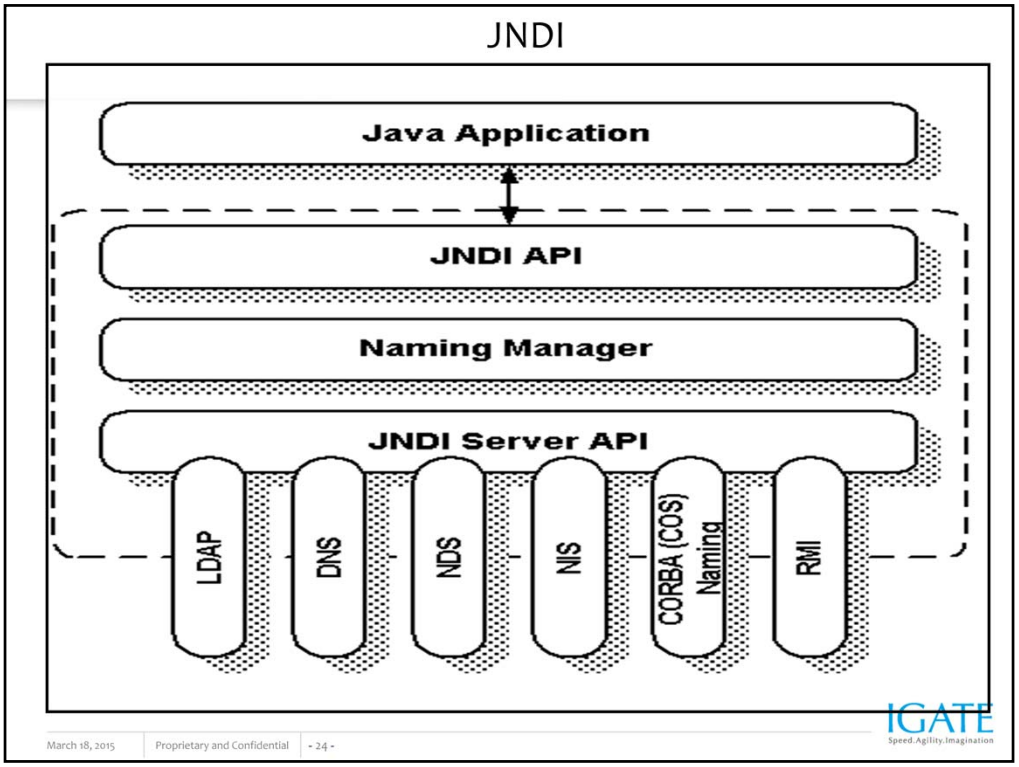
- *Executes upon receipt of a single client message.*
- *Is asynchronously invoked.*
- *Can be transaction-aware.*
- *May update shared data in an underlying database.*
- *Does not represent directly shared data in the database, although it may access and update such data.*
- *Is relatively short-lived.*
- *Is stateless.*
- *Is removed when the EJB container crashes. The container has to re-establish a new message-driven object to continue computation.*

Entity Object

- *Is part of a domain model, providing an object view of data in the database.*
- *Lives as long as the data in the database*
- *The entity and its primary key survive the crash of the EJB container. If the state of an entity was being updated by a transaction at the time the container crashed, the entity's state is restored to the state of the last committed transaction when the entity is next retrieved.*

What is a Naming Service?

- A Naming Service provides a natural, understandable way of identifying and associating names with data.
- The Java Naming and Directory Interface is a client API that provides naming and directory functionality
- It is designed to provide a common interface for accessing existing services like DNS, NDS, LDAP, CORBA, or RMI.


JNDI
Java Application
JNDI API
Naming Manager
JNDI Server API
LDAP
DNS
NDS
NIS
CORBA (COS) Naming
RMI
IGATE
Speed.Agility.Imagination
March 18, 2015
Proprietary and Confidential
- 24 -

The Java Transaction Service (JTS)

- The Java Transaction Service (JTS) plays the role of a transaction coordinator for all the constituent components of the EJB architecture.
- When an application begins a transaction, it creates a transaction object that represents the transaction.
- The application then invokes the resource managers to perform the work of the transaction.

The Java Transaction Service (JTS)

- The participants in the transaction that implement transaction-protected resources, such as relational databases, are called resource managers.
- As the transaction progresses, the transaction manager keeps track of each of the resource managers enlisted in the transaction.

Passivation/Activation

- To manage its working set container temporarily transfers the state of an idle bean instance to some form of secondary storage.
- Passivation - saves the state of the bean to secondary storage
- Activation - Restores the state of the bean from secondary storage

Session Bean Types

- ***Stateless Session Bean***

  The session bean instances contain no conversational state between methods; any instance can be used for any client.

- ***Stateful Session Bean***

  The session bean instances contain conversational state which must be retained across methods and transactions.

Stateless vs. Stateful beans

- **Stateless**
  - –no internal state
  - –do not need to be passivated
  - –can be pooled to serve multiple clients
- **Stateful**
  - –possess internal state
  - –need to handle passivation/activation
  - –one bean per client

IGATE
Speed.Agility.Imagination

Writing Enterprise Bean

- **Enterprise bean class:**
  Implements the methods defined in the business interface and any life cycle callback methods.
- **Business Interfaces:**
  The business interface defines the methods implemented by the enterprise bean class.
- **Helper classes:**
  Other classes needed by the enterprise bean class, such as exception and utility classes.
- **Package the above classes in EJB jar file.**

Stateless Session Bean

- A *stateless* session bean does not maintain a conversational state with the client.
- When the method is finished, the client-specific state is not retained

Writing Stateless Session Bean

- **The *business interface* defines the business methods that a client can call. The business methods are implemented in the enterprise bean class.**
- **Source code for Hello interface with business method greet()**

```
package patni.ejb;
import javax.ejb.Remote;
@Remote
public interface Hello {
public String greet(String name);
}
```

Writing Enterprise Bean Class

```
package patni.ejb;
import javax.ejb.*;
@Stateless
public class HelloBean implements Hello {
    public String greet(String name) {
    return "Hello" + name;
}
```

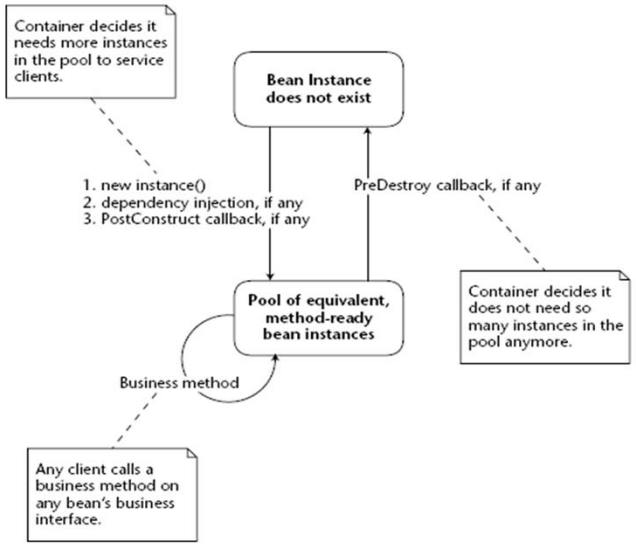
Invoking Business Method

[Stateless Session Bean]

- **Client Code**

```
package patni.client.ejb;
import javax.ejb.EJB;
public class HelloClient {
@EJB
public static void main(String[] args) {
HelloClient clientObj = new HelloClient();
Hello helloObj = new Hello();
helloObj.greet(args[0]);
}
```

Life Cycle of Stateless Session Bean



IGATE
Speed.Agility.Imagination

Stateful Session Bean

- **Maintains client specific information between method calls for a client.**
- **Methods in the bean class may be declared as a life-cycle callback method by annotating the method with the following annotations:**
  - javax.annotation.PostConstruct
  - javax.annotation.PreDestroy
  - javax.ejb.PostActivate
  - javax.ejb.PrePassivate

Callback Method Annotations

- @PostConstruct methods are invoked by the container on newly constructed bean instances after all dependency injection has completed and before the first business method is invoked on the enterprise bean.
- @PreDestroy methods are invoked after any method annotated @Remove has completed and before the container removes the enterprise bean instance.
- @PostActivate methods are invoked by the container after the container moves the bean from secondary storage to active status.
- @PrePassivate methods are invoked by the container before the container passivates the enterprise bean, meaning the container temporarily removes the bean from the environment and saves it to secondary storage.

Writing Stateful Session Bean

- **Business Interface Count.java**

```
package patni.stateful;
public interface Count {
  public int count();
  public void set(int val);
  public void remove();
}
```

Stateful Bean Class

```
package patni.stateful;
import javax.ejb.*;
@Stateful
@Remote(Count.class)
@Interceptors(CountCallbacks.class)
    public class CountBean implements Count {
    /** The current counter is our conversational state. */
      private int val;
    public int count() {
        System.out.println("count()");
        return ++val;
      }
```


IGATE
Speed.Agility.Imagination

Stateful Bean Class [contd.]

```
public void set(int val) {
      this.val = val;
      System.out.println("set()");
   }

   /**
    * The remove method is annotated so that the container knows
    * it can remove the bean after this method returns.
    */
   @Remove
   public void remove() {
      System.out.println("remove()");
   }

}
```

Overriding Callback Methods

```
package patni.stateful;

import javax.ejb.InvocationContext;
import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

/**
 * This class is a lifecycle callback interceptor for the Count bean.
 * The callback methods simply print a message when invoked by the container.  */
```

Overriding Callbacks Methods[contd]

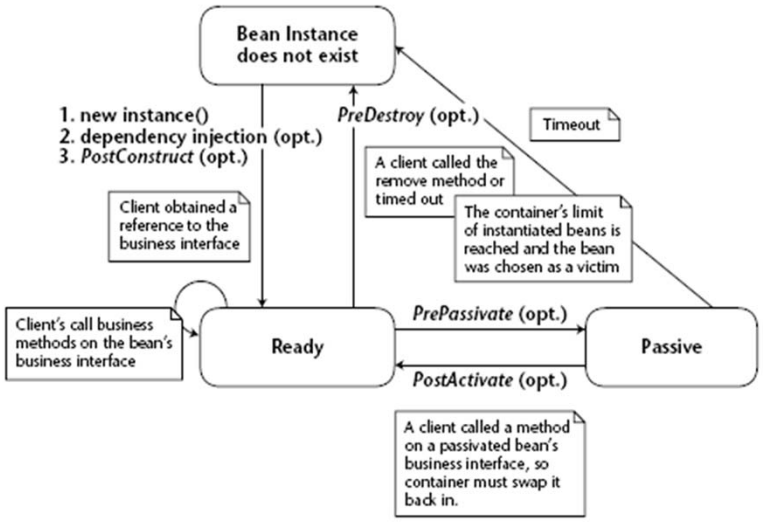
```
public class CountCallbacks {
  public CountCallbacks() {}
  /**   * Called by the container after construction   */
  @PostConstruct
  public void construct(InvocationContext ctx) throws Exception {
    System.out.println("cb:construct() ");
    ctx.proceed();
  }
  /**   * Called by the container after activation    */
  @PostActivate
  public void activate(InvocationContext ctx) throws Exception {
    System.out.println("cb:activate()");
    ctx.proceed();
  }
```

Overriding Callbacks Methods[contd]

```
/**
 * Called by the container before passivation     */
@PrePassivate
public void passivate(InvocationContext ctx) throws Exception {
   System.out.println("cb:passivate()");
   ctx.proceed();
}
/**
 * Called by the container before destruction     */
@PreDestroy
public void destroy(InvocationContext ctx) throws Exception {
   System.out.println("cb:destroy()");
   ctx.proceed();
}

}
```

Life Cycle of Stateful Session Bean



IGATE
Speed.Agility.Imagination

Persistence

- **The Java Persistence provides an object/relational mapping facility for managing relational data in Java applications.**
- **Java Persistence consists of three areas:**
  - The Java Persistence API
  - The query language
  - Object/relational mapping metadata

IGATE
Speed.Agility.Imagination

Java Persistent API [JPA]

JPA is made of six main parts:

- **Metadata**
- **Domain Object Model Requirements**
- **Application Programming Interface**
- **Queries**
- **Lifecycle Model, including detachment**
- **Callbacks**

JPA Metadata

- **Two types of objects in an Enterprise Application:**
  - **logical** component describing the domain object model
    - Handle business processes. [eg. Session beans]
  - **Persistent Data Objects** describing how the object model maps to the database schema
    - Represents business data

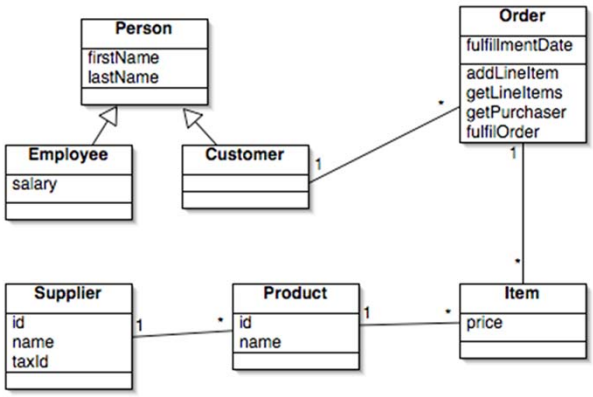
Entities

- Persistent objects are called Entities.
- They are POJOs which are persisted to durable storage or legacy system.
- Entities store data as fields
- They have methods associated with them

Domain Model: Entities

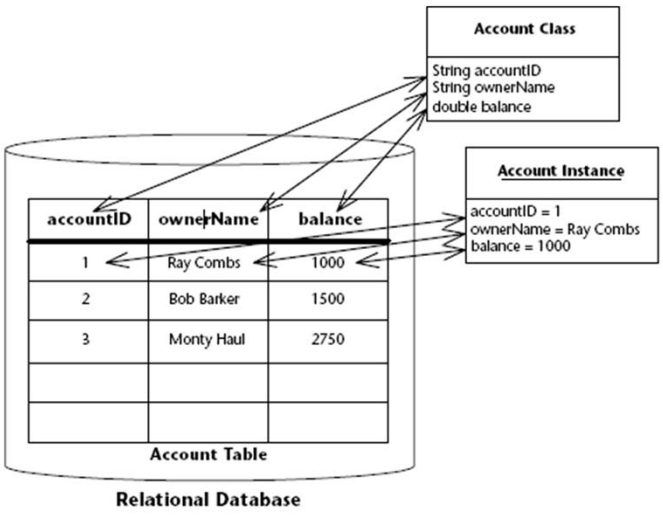
- **Entities support standard OO domain modeling techniques**
  - Inheritance
  - Encapsulation
  - Polymorphic Relationships
- **Can be created with the new operator**
- **There is no required superclass / interface in the EJB3 Persistence API**

Example Object Model



IGATE
Speed.Agility.Imagination

ORM



IGATE
Speed.Agility.Imagination

Writing Entity Class

- **An entity class must follow these requirements:**
  - The class must be annotated with the javax.persistence.Entity annotation.
  - The class must have a public or protected, no-argument constructor. The class may have other constructors.
  - The class must not be declared final.No methods or persistent instance variables must be declared final.
  - If an entity instance be passed by value as a detached object, such as through a session bean's remote business interface, the class must implement the Serializable interface.

Writing Entity Class [contd]

- Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Persistent instance variables must be declared private, protected, or package-private, and can only be accessed directly by the entity class's methods. Clients must access the entity's state through accessor or business methods.

Primary Keys in Entities

- Each entity has an unique identifier, every entity must have a primary key so that client can locate a particular entity instance.
- An entity may have either a simple or a composite primary key.
- Simple primary keys use the javax.persistence.Id annotation to denote the primary key property or field.

Primary Keys in Entities

- Composite primary keys must correspond to either a single persistent property or field, or to a set of single persistent properties or fields. Composite primary keys must be defined in a primary key class.
- Composite primary keys are denoted using the javax.persistence.EmbeddedId and javax.persistence.IdClass annotations.

Primary Key Class

- **A primary key class must meet these requirements:**
  - The access control modifier of the class must be public.
  - The properties of the primary key class must be public or protected if property-based access is used.
  - The class must have a public default constructor.
  - The class must implement the hashCode() and equals(Object other) methods.
  - The class must be serializable.

Primary Key Class

- A composite primary key must be represented and mapped to multiple fields or properties of the entity class, or must be represented and mapped as an embeddable class.
- If the class is mapped to multiple fields or properties of the entity class, the names and types of the primary key fields or properties in the primary key class must match those of the entity class.

Writing An Entity

```
package patni.entity;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Account implements Serializable {
```

Writing An Entity [contd]

```
/** The account number is the primary key for the persistent object */
  @Id
  public int accountNumber;
  public String ownerName;
  public int balance;
  /**  Entity beans must have a public no-arg constructor  */
  public Account() {
    accountNumber = (int) System.nanoTime();
  }
// business logic methods  withdraw() and deposit()
}
```

Managing Entities

- Entities are managed by the entity manager.
- The entity manager is represented by javax.persistence.EntityManager instances.
- Each EntityManager instance is associated with a persistence context.
- A persistence context is a set of managed entity instances that exist in a particular data store.

EntityManager

- **The EntityManager API**
  - creates and removes persistent entity instances
  - finds entities by the entity's primary key
  - allows queries to be run on entities.
- **Container Managed Entity Manager**
  - To obtain an EntityManager instance, inject the entity manager into the application component:

```
@PersistenceContext
EntityManager em;
```

EntityManager

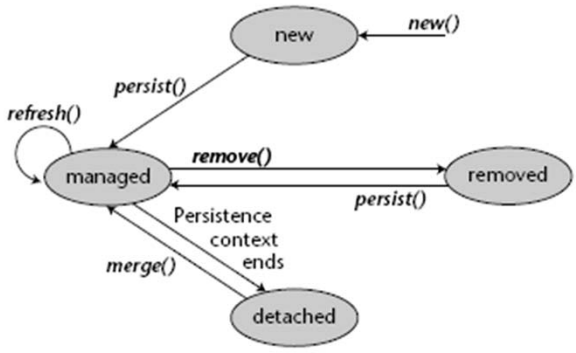
- **Application Managed Entity Manager**
  - To obtain an EntityManager instance, you first must obtain an EntityManagerFactory instance by injecting it into the application component by means of the javax.persistence.PersistenceUnit annotation:
    - @PersistenceUnit
    - EntityManagerFactory emf;
  - Then, obtain an EntityManager from the EntityManagerFactory instance:

```
EntityManager em = emf.createEntityManager();
```

Life Cycle Callbacks of Entities

- **JPA specification defines following callbacks for entities**
  - PrePersist
  - PostPersist
  - PreRemove
  - PostRemove
  - PreUpdate
  - PostUpdate
  - PostLoad

Life Cycle of Entity



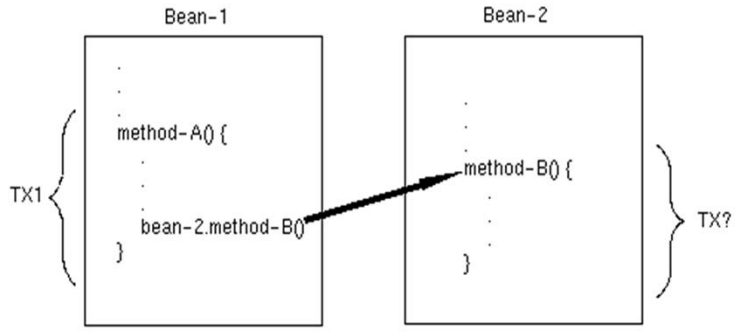
IGATE
Speed.Agility.Imagination

Database Synchronization

- Update to local entities are synchronized with the underlying database at transaction commit time.
- Flush mode is set on specific methods or fields using setFlushMode() to enforce synchronization before query is executed.
- To refresh state from database refresh() operation is invoked explicitly.

Transactions

- Container-Managed Transactions
- Bean-Managed Transaction
- Transaction Attributes



IGATE
Speed.Agility.Imagination

Transaction Attributes

- **Transaction Attribute Values**
  - Required
  - RequiresNew
  - Mandatory
  - NotSupported
  - Supports
  - Never

Transaction Attributes and Scope

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

Using Transaction Annotation

- **The following code snippet demonstrates usage of @TransactionAttribute annotation:**

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
...
@TransactionAttribute(REQUIRES_NEW)
public void firstMethod() {...}
@TransactionAttribute(REQUIRED)
public void secondMethod() {...}
public void thirdMethod() {...}
public void fourthMethod() {...}
}
```

Methods Not Allowed in Container-Managed Transactions

- commit, setAutoCommit, and rollback methods of java.sql.Connection
- getUserTransaction method of javax.ejb.EJBContext
- any method of javax.transaction.UserTransaction

Methods Not Allowed in Bean-Managed Transactions

- **Do not invoke the getRollbackOnly and setRollbackOnly methods of the EJBContext interface.**

Transaction Isolation Levels

- **Four Transaction isolation levels**
  - READ UNCOMMITTED mode does not guarantee isolation but offers the highest performance
  - READ COMMITTED does not allow dirty reads
  - REPEATABLE READ mode does not allow dirty reads as well as solves problem of unrepeatable read
  - SERIALIZABLE mode solves all problems of dirty read, unrepeatable read and also phantom reads

Security

- **EJB provides granular security.**
- **You can protect enterprise beans by doing the following:**
  - Accessing an Enterprise Bean Caller's Security Context
  - Declaring Security RoleNames Referenced from Enterprise Bean Code
  - Defining a Security View of Enterprise Beans
  - Using Enterprise Bean Security Annotations
  - Using Enterprise Bean SecurityDeploymentDescriptor Elements
  - Configuring IOR Security
  - Deploying Secure Enterprise Beans

Security API

The javax.ejb.EJBContext interface provides two methods that allow the bean provider to access security information about the enterprise bean's caller.

- **java.security.Principal getCallerPrincipal();**
  - allows the enterprise bean methods to obtain the current caller principal's name.
- **boolean isCallerInRole(String roleName);**
  - tests whether the current caller has been assigned to a given security role.

Development Phases of J2EE Applications

- Enterprise Bean Creation
- Web Component Creation
- J2EE Application Assembly
- J2EE Application Deployment

Clients

- Stand-Alone Java Applications
- J2EE Application Clients
- Servlets
- JavaServer Pages Components
- Other Enterprise Beans