SIEMENS

CORPORATE TECHNOLOGY

Enterprise JavaBeans 3.0

Public Review Version

Introduction and Overview

Michael.Stal@siemens.com

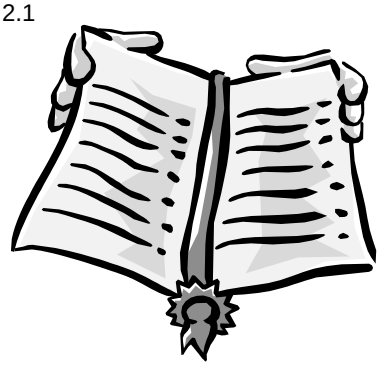
CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Content



CT SE 2

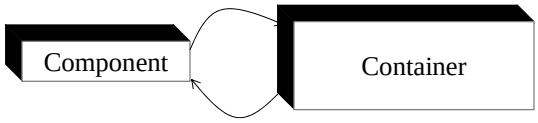
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Component/Container Approach

- **In order to shield components from the underlying infrastructure specifics, containers are introduced as standardized runtime systems.**
- **Containers:**
  - manage components and notify components about events such as activation, passivation, transactions.
  - provide components access to services such as transactions, security, persistence.
  - help to register and deploy components.



CT SE 2

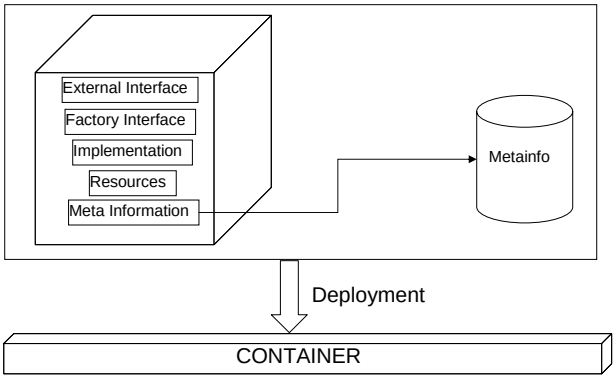
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Deployment Contract

- **Component is provided in a special package format and provides specific meta information.**



CT SE 2

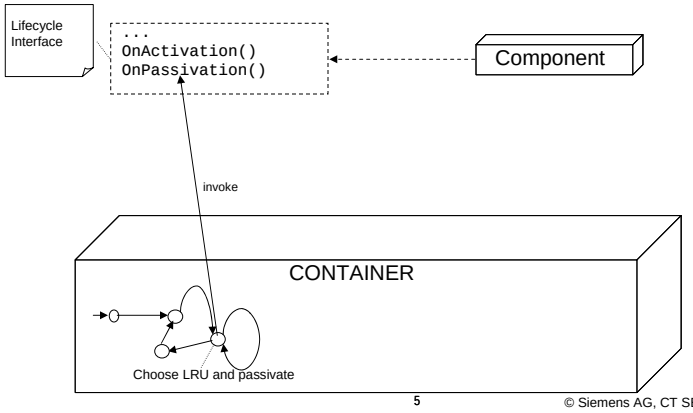
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Lifecycle Contract

- **Component implements lifecycle event methods the container „automagically“ invokes.**



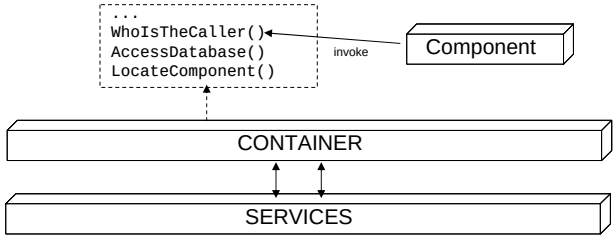
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Container Service Contract

- **Container provides a context object or interface to the component so that the component can transparently consume services.**



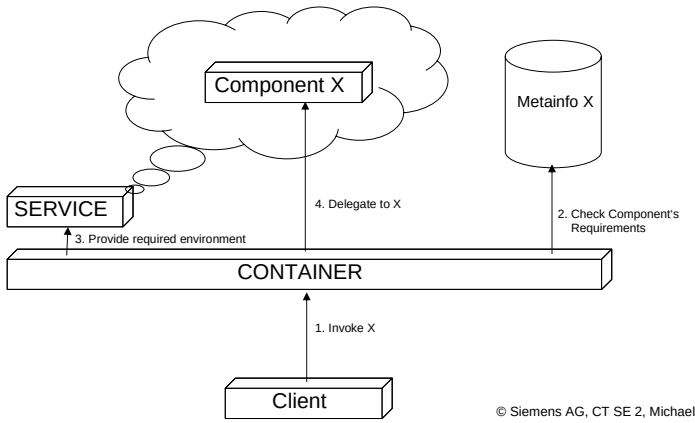
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Environmental Contract

- **Container always provides appropriate environment for component according to deployment info.**



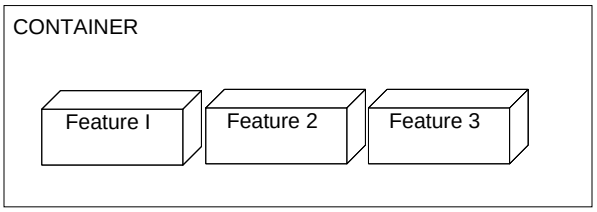
CT 2 S&E
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Extension Contract

- **Container might itself be extensible according to the Open-Close-Principle.**
- **Possible approaches:**
  - Allow custom interceptors
  - Support runtime configuration of entities
  - Provide your own services
  - Change/extend sources
  - Use aspect-oriented approaches



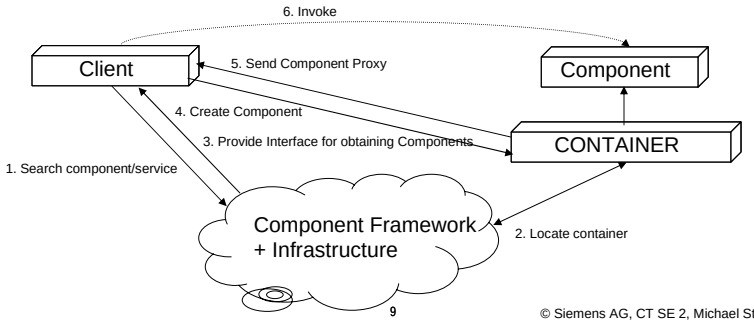
CT 2 S&E
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Client/Container Contract

- **Client accesses components using standard transport**
- **Component Framework provides mechanisms to locate component: centralized mechanisms or decentralized (P2P, trading, ...) mechanisms**
- **Approach can be service-based or entity-based**

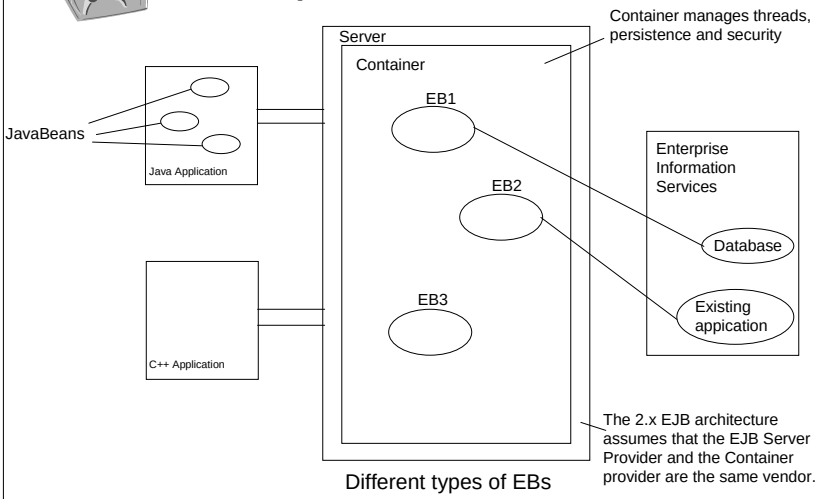


Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Short History of Time
Enterprise JavaBeans 2.x



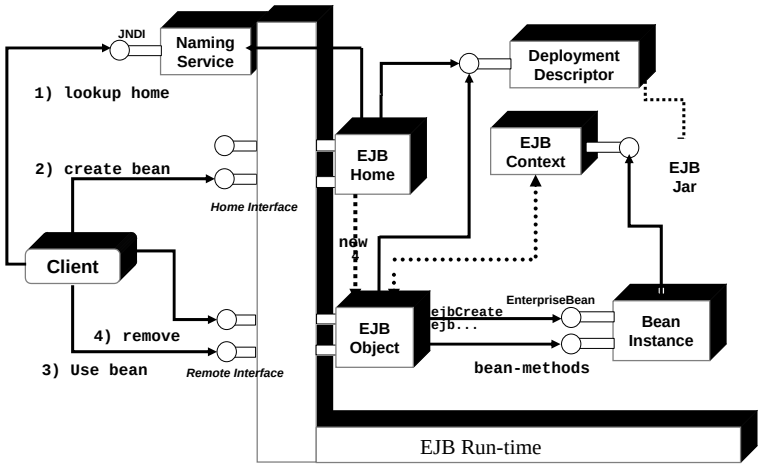
Different types of EBs

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

EJB Servers/Containers



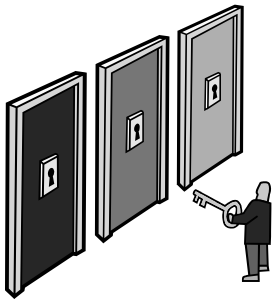
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Flavors of Enterprise Beans

- There are different flavors of Enterprise JavaBeans:
  - **Session Bean**
    - **stateless**
    - **stateful**
  - **Message-driven Bean**
  - **Web Services with EJBs**
  - **Entity Bean**



Software & Engineering Architecture

CORPORATE TECHNOLOGY

A Sample Stateful Session Bean

```
public interface ShoppingCartHome extends
  javax.ejb.EJBHome {
  public ShoppingCart create(int owner_id)
       throws CreateException, RemoteException;
}

public interface ShoppingCart extends javax.ejb.EJBObject
{
  public void addItem(int item, int quantity)
       throws RemoteException;
}

public class ShoppingCartBean implements SessionBean {
  public void addItem(int item, int quantity) {
     // ...
  }
  …
}
```

CT SE 2
Software & Engineering Architecture

CORPORATE TECHNOLOGY

A Sample Stateful Session Bean (cont'd): Callback Methods

```
public void setSessionContext(SessionContext sc) {
     this.ctx=sc;
}
public void ejbActivate()  {...}
public void ejbPassivate() {...}
public void ejbRemove() {...}
public void ejbCreate() throws
     javax.ejb.CreateException {...}
```

CT SE 2
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Deployment Descriptor for a Stateful Bean

```
<enterprise-beans>
 <session>
      <ejb-name>ShoppingCartEJB</ejb-name>
      <home>ShoppingCartHome</home>
      <remote>ShoppingCart</remote>
      <ejb-class>ShoppingCartBean</ejb-class>
      <session-type>Stateful</session-type>
 </session>
</enterprise-beans>
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

SessionContext

- **The SessionContext provides the following methods:**
  - **`getEJBObject`** returns the associated EJB object
  - **`getEJBHome`** returns home interface
  - **`getCallerPrincipal`** identifies caller
  - **`isCallerInRole`** checks role of caller
  - **`getTimerService`** (EJB 2.1)
- **Example:**

```
boolean isManager=ctx.isCallerInRole("Manager");
if (isManager) {
   …
}
```

Software & Engineering Architecture

CORPORATE TECHNOLOGY

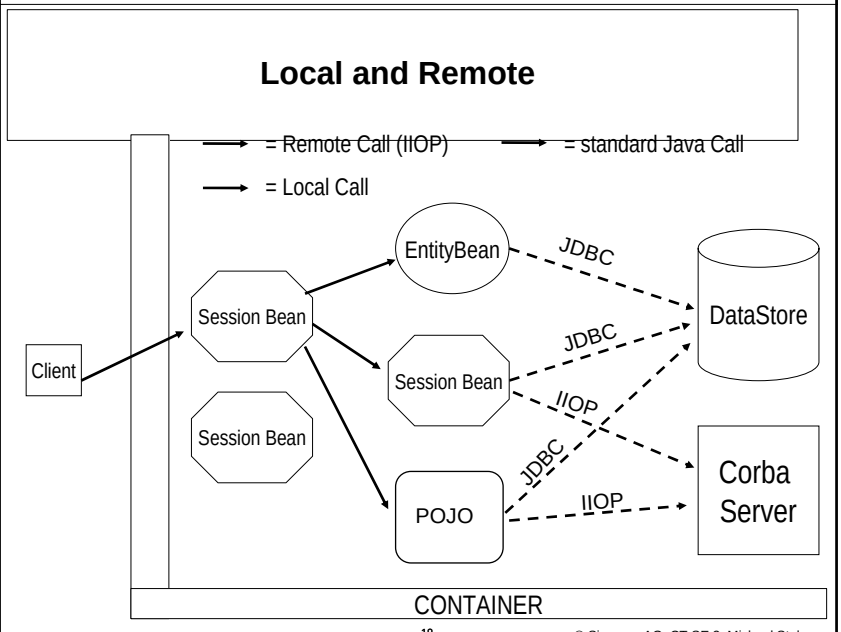
Stateful Session Bean Client Example

```
// Client looks up the Bean's home interface:
Context initialContext = new InitialContext();
CartHome cartHome =
   (CartHome)javax.rmi.PortableRemoteObject.narrow
    (initialContext.lookup("java:comp/env/ejb/shopping-cart"),
    CartHome.class);

// creates an EJB object:
Cart myCart = cartHome.create("Mike", "1234-5");

// und uses it:
myCart.add("Harry Potter Vol. 6"); // call business method
myCart.add("JavaSpektrum 1/2006"); // call business method
myCart.pay();

// before getting rid of the bean:
myCart.remove(); // remove session object
```

Software & Engineering Architecture

CORPORATE TECHNOLOGY

Local and Remote



Software & Engineering Architecture

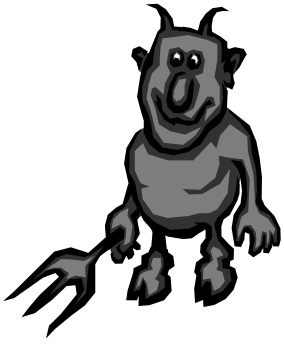
Deployment Descriptor

- **In the deployment descriptor information the bean provider must supply information such as**
  - Bean name, bean class, home interface, remote interface
  - Bean type (entity, session, message-driven)
  - Re-entrancy (session/message-driven beans are never re-entrant)
  - State-management type for session beans (stateful, stateless)
  - Transaction demarcation type for message-driven bean
  - Persistence management (managed, container)
  - Primary key class: prim-key-class element. Bean-managed: class name
  - Abstract schema name (for cmp-version 2.x)
  - cmp-fields, crm-relationships, dependent classes
  - Finder/select queries
  - Environment entries, resource manager factory referencesm, resource env. References, EJB references, security role references, Message-bean destination, Message-selector, Message-acknowledgement mode

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>  CustomerEJB    </ejb-name>
      <home>      CustomerHome   </home>
      <remote>    CustomerRemote </remote>
      <ejb-class> CustomerBean   </ejb-class>
      <persistence-type> Container </persistence-type>
      <prim-key-class> java.lang.Integer </prim-key-class>
      <reentrant> False </reentrant>
      <cmp-version> 2.x </cmp-version>
      <abstract-schema-name> CustomerExt </abstract-schema-
    name>
      <cmp-field> <field-name> id        </field-name> </cmp-
    field>
      <cmp-field> <field-name> lastName  </field-name> </cmp-
    field>
      <cmp-field> <field-name> firstName </field-name> </cmp-
    field>
      <primkey-field> id </primkey-field>
    </entity>
    <entity> ...
    </entity>
  </enterprise-beans>
  ... relationships ...
  ... assembly-descriptor ...
</ejb-jar>
```


EJB 2.x Summary

- **EJB 2.1 is like hell since developers must**
  - Implement a lot of artifacts such as home interfaces, callback methods, remote interfaces
  - Use only one business interface per EJB bean
  - Follow a lot of conventions when building enterprise beans such as using RMI conventions, using specific base interfaces
  - Abide to a different scheme when developing EJBs compared to developing regular Java classes and interfaces
  - Configure beans and applications with giant XML-based deployment descriptors
  - Introduce dependencies to the environment
  - Deal with the complexities of persistence
  - Manually introduce tracing, logging and other home-grown methods to test components
  - Manually locate and use beans as clients



SIEMENS

CORPORATE TECHNOLOGY

Goals of EJB 3.0

- Use Meta-Data
- Use Dependency Injection
- Reducing the number of artifacts developers must provide
- Configuration by exception
- Reducing environmental dependencies
- Simplification of EJB types
- Getting rid of many interfaces
- Simplification of persistence
- Improved testing outside of the container

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Session and Message-Driven Beans

- **Using POJOs (Plain Old Java Object) and POJIs (Plain Old Java Interface)**
  - Business Interface as Java interface
  - No more home interface
  - Using metadata annotations for configuration of EJB Type, Local/Remote, Transactions, Security
- **Dependency Injection**
  - For fields/properties, resource usage
- **Advanced Lifecycle Support**
  - Custom callbacks
- **Interceptors**
  - To intercept business methods in an AOP like way

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Programming Model: Bean Class

- The bean class now serves as the main programming artifact
- Home interfaces have disappeared ☺

```
@Stateful public class ShoppingCartBean
  implements ShoppingCart
{
  public int add(long item){…}
  public void remove(int id){…}
}

public interface ShoppingCart {
  public int add(long item);
  public void remove(int id);
}
```

- Bean class methods may throw application exceptions
- They should not throw `java.rmi.RemoteException`

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Programming Model: Lifecycle Events

- Life cycle events notified by container through callback methods which are annotated:

```
@Stateful public class ShoppingCartBean
  implements ShoppingCart
{
  ...
  @PreDestroy remove() { ... }
}
```

- Callbacks may throw runtime exceptions causing transactions to roll back, but must not throw application exceptions
- Possible to introduce callback listener classes instead (`CallBackListener` annotation)

CT SE 2

Software & Engineering Architecture

Programming Model: Interceptors

- **Interceptors used to intercept business method invocations (session beans, message-driven beans, EJB 2.1 entities)**
- **Marked with annotation `AroundInvoke`**
- **Interceptor methods defined in bean class or within an interceptor class:**
  - May throw runtime exceptions or application exceptions defined in throw clause of intercepted business method
  - Execute in same security/transaction context like business methods
  - May invoke JNDI, JDBC, JMS, other beans and the EntityManager
  - Supported by dependency injection
- **Multiple interceptors may be defined for same business method**
- **Within interceptor `proceed()` used to call actual implementation**

Interceptors (cont'd)

- **Method signature of an interceptor method:**
  - `public Object methodName(InvocationContext) throws Exception`
- **`InvocationContext` defined as follows:**

```
public interface InvocationContext {
  public Object getBean();
  public Method getMethod();
  public Object[] getParameters();
  public void setParameters(Object[]);
  public EJBContext getEJBContext();
  public java.util.Map getContextData();
  public Object proceed() throws
  Exception;
}
```

- **This data is shared across all interceptors for same method invocation**

SIEMENS

CORPORATE TECHNOLOGY

Important Annotations and Defaults (1)

- Classes/interfaces may be annotated with `@Local` or `@Remote`
- `@Remote`: bean is remotely accessible
- `@Local`: bean is local. Can be left out as it is the default setting

```
@Remote public interface ShoppingCart
{
  public int add(long item);
  public void remove(int id);
}
```

- Transaction management (container versus bean-managed) specified with `@TransactionManagement` (defaults to `TransactionManagementType.CONTAINER`) and is assigned to bean class.

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Important Annotations and Defaults (2)

- `@TransactionAttribute` is assigned to whole class or single methods and determines bean behavior in transactions
  - MANDATORY, REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED, NEVER
  - Defaults to REQUIRED
- `@Timeout` used to define timeout method of bean
- `@ApplicationException` to designate application exceptions
- Several security-related annotations:
  - `@RolesReferences`
  - `@RolesAllowed`
  - `@DenyAll`
  - `@RunAs`
  - `@SecurityRoles`

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateless Session Beans

- **Properties:**
  - Implement plain Java business interfaces
  - Do not provide a home interface
  - Comprise an implementation class annotated with `@Stateless`
  - Callbacks defined `@PostConstruct`, `@PreDestroy` and executed in unspecified security, transaction context
  - Interceptors might be applied
  - Dependency Injection before first business method execution

```
@Stateless
@Interceptors( {MyInterceptor.class} )
public class FlightBookingBean implements
   FlightBooking {
   public void bookFlight(long flightInfo) { ... }
   @PostConstruct public void Init() {
   }
}

public interface FlightBooking {
   public void bookFlight(long flightInfo);
}
// somewhere else:
public class MyInterceptor {
   @AroundInvoke
   public Object measure(InvocationContext ic) throws
   Exception {
     long time = System.currentTimeMillis();
     try {
       return ic.proceed();
     } finally {
       long total = time  - System.currentTimeMillis();
       System.out.println(inv.getMethod.getName() + „
  took “ + total + „ milliseconds“);
     }
   }
}
```

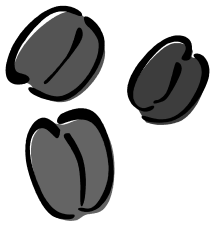
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateful Session Beans

- **Properties I:**
  - Implements plain Java business interfaces
  - Does not need a home interface
  - Is annotated with `@Stateful`
  - May implement `SessionSynchronization` interface
  - Callbacks defined: `PostConstruct`, `PreDestroy`, `PostActivate`, `PrePassivate`
  - Dependency Injection before first business method execution
  - Interceptors available: `afterBegin` before and `beforeCompletion` after any `AroundInvoke`
- **Properties II:**
  - Clients obtain references to session bean either
    - through JNDI, OR,
    - Using dependency injection
  - `@Remove` annotation for marking method that is invoked for bean removal



Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateful Session Bean Example (1)

- Bean is stateful
- One business interface is implemented
- Resources such as the context object are provided by the container using dependency injection (@Resource)
- After initial construction and after each re-activation the method `init()` gets invoked

```
@Stateful
public class
  ShoppingCartBean
implements ShoppingCart
{
  Connection rc;
@Resource SessionContext
  sc;
  @PostConstruct
  @PostActivate
  public void init() {
    rc =  Connection.Open();
  }
  ...
```

```
public interface ShoppingCart {
  public int addTem(long item);
  public void removeItem(int id);
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateful Session Bean Example (2)

- A specific method is annotated which is invoked before passivating or destroying the session bean
- The `getRidOfIt`-method will cause the container to remove the bean after completion

```
...
@PreDestroy
@PrePassivate
public void close() {
  try {
       rc.close();
  }
  catch (CloseException e)
  { /* no-op */ }
}
@Remove
public void getRidOfIt() {
  // whatever necessary
  // before removing
}
...
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateful Session Bean Example (3)

- The business interface ShoppingCart introduces two methods
- An interceptor method is defined getting invoked for every business method

```
...
public int addItem(int id, int
quantity) { }
public void removeItem(int id) { }
@AroundInvoke
public Object
monitor(InvocationContext ic) {
  try {
        Object res = inv.proceed();
        if ((OPResult)res
            ==OpResult.SUCCESS) {
           System.out.println(„Ok“);
        }
        return res;
  }
  catch(Exception ex) {
        throw ex;
  }
 }
} // end of class
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Stateful Session Bean Example (4)

- **Clients:**
  - Obtains session bean by lookup (JNDI)
  - Or by dependency injection
  - Uses special methods of a stateful session bean to initialize the bean bacause there are no create methods anymore

```
public static void testBean() throws Exception {
      InitialContext ctx = new InitialContext();
      ShoppingCart test =
          ctx.lookup(ShoppingCart.class.getName());
      // or using injection when in another bean:
      // @EJB
      // ShoppingCart test;
      int id = test.addItem(1234, 2);
      test.removeItem(id);
      test.getRidOfIt(); // Remove bean method
}
```

Software & Engineering Architecture

Message Driven Beans

- **Properties:**
  - Implements message-lsistener interface determined by messaging type (e.g., `javax.jms.MessageListener`)
  - Message driven bean class is annotated with `@MessageDriven`
  - It does not need implementing `javax.ejb.MessageDrivenBean`
  - Lifecycle event callbacks defined are `PostConstruct` and `PreDestroy`
  - Dependency Injection before first business method execution
  - Interceptors (methods or classes) are supported and apply to listener method invocations

Message Driven Bean Example

- **Taken from JBoss EJB 3.0 RC3**

```
import javax.ejb.MessageDriven;
import javax.ejb.ActivationConfigProperty;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activateConfig = {
        @ActivationConfigProperty(propertyName="destinationType",
                                  propertyValue="javax.jms.Queue"),
        @ActivationConfigProperty(propertyName="destination",
                                  propertyValue="exampleQueue")
        })
public class ExampleMDB implements MessageListener
{
   public void onMessage(Message recvMsg)
   { /* ... */ }
}
```

SIEMENS

CORPORATE TECHNOLOGY

Bean Context and Environment (1)

- **Bean declares dependency upon resource through dependency annotation**
- **A dependency annotation specifies**
  - type of object or resource a bean depends on
  - resource or object characteristics
  - name with which resource or object is accessed
- **Examples:**

```
@EJB (name=„mySessionBean,
  beanInterface=MySessionIF.class)

@Resource(name=„DB“,
  type=„javax.sql.DataSource.class)
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Bean Context and Environment (2)

- **Developer annotates instance variables of bean to specify dependencies**
- **Container automatically initializes these annotated variables with the appropriate external references**
- **Initialization occurs before any business method invocation and after `EJBContext` was set**

```
@Stateless
public class MySessionBean
  implements MySession {
  // type deferred
  @Resource(name=„DB“)
  public DataSource myDB;
  @EJB
  public AddressHome addrH;
  public void getData() {
    // try-catch omitted!
    Connection c =
     myDB.getConnection();
    Address a =
     addrH.create(„...“);
  }
}
```

CT SE 2

Software & Engineering Architecture

Bean Context and Environment (3)

- **Setter Injection**
  - Is an alternative to field injection
  - Container invokes setter method instead of setting instance variable

```
@Resource(name=„myDB“)
public void setDataSource(DataSource myDB) {
  this.MyDB = myDB;
}

@Resource
public void SetSessionContext(SessionContext ctx) {
  this.ctx = ctx;
}
```


Persistence

- **An entity in EJB 3.0 is a lightweight persistent object**
- **Annotation to mark bean class is `@Entity`**
- **`Must have public no-arg constructor`**
- **`Must implement Serializable` if being transmitted by value to other objects**
- **Entity class**
  - may be abstract or concrete
  - may inherit from other classes
  - may be inherited by other classes
  - state defined by instance variables must not be accessible to clients
  - Persistence provider accesses either instance variables or getter/setter methods
  - Annotations for persisted fields (`access=VALUE` or `access=PROPERTY`).
  - Default is Property access. All non-transient elements are persisted
  - Per default mapped to DB table with same name

Example for an Entity

- **The following entity defines a primary key, its fields are persisted through getters/setters:**

```
@Entity
@Table(name = „CustomerList")
public class CustomerList implements java.io.Serializable
{
   private int id;
   private Collection<Customer> custList;
   ...
   @Id(generate = GeneratorType.AUTO)
   public int getId() { return id; }
   public void setId(int id) { this.id = id;}
   @OneToMany
   public Collection<Customer> getCustomers() { return custList; }
   public void SetCustomers(Collection<Customer> custList)
    { this.CustList = custList;}
}
```


Primary Keys

- **A primary key may be simple or composite**
  - Simple primary key by marking single field/property
  - Composite primary key by introducing primary key class
    - types/names of constituents must be the same in entity class and primary key class
    - Both can have different access types (field, property)
- **Primary key class must define consistent `equals` & `hashCode` contract**

SIEMENS

CORPORATE TECHNOLOGY

Embedded and Embeddable

- **Embedded classes are collections of data that belong to one single entity (standard whole-part relationship). Annotation `@Embeddable`**

```
@Entity
@Table(name = „Customer")
public class Customer implements java.io.Serializable
{// instances and embedded objects map to one single table
  privat Address addr;
  ...
  @Embedded
  @AttributeOverrides({
    @AttributeOverride(name=„city“, column=@Column(name=„city“)),
    @AttributeOverride(name=„country, column=@Column(name=„country“))})
  public Address getAddress() { return addr; }
  public void setAddress(Address a) { addr = a; }
}
@Embeddable
public class Address implements java.io.Serializable {
  private String city;
  private String country;
}
```

Software & Engineering Architecture

SIEMENS

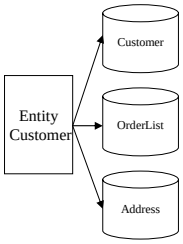
CORPORATE TECHNOLOGY

Additional Mapping Issues

- **One entity bean may map to multiple data base tables**

```
@Entity
@Table(name = „Customer")
@SecondaryTables({
@SecondaryTable(name=„Address“)
@SecondaryTable(name=„OrderList“)})
public class Customer implements
   java.io.Serializable
{
   // from Customer DB
   @Column(name=„name“)
   public String getName() { ... }
   ...
   @Column(name=„street“,
   secondaryTable=„Address“)
   public String getStreet(){ ... }
}
```

If column annotation is missing, derive from name



Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Relationships

- Relationships can be `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`
- Relationships are either unidirectional (only owning side) or bidirectional (additional inverse side)
- When bidirectional, inverse side must specify its owning side with `mappedBy`
- Many-side in `@OneToMany`, `@ManyToOne` must be the owning side
- The owning side of `@OneToOne` is the one containing the foreign key

```
@Entity public class Employee {
  private Cublicle assignedCubicle;
  @OneToOne
  public Cubicle getAssignedCubicle() {
         return assignedCubicle;
  }
  public void setAssignedCubicle(Cubicle c)
  {
         this.assignedCubicle = c;
  }
}

@entity public class Cubicle {
  private Employee employee;
  @OneToOne(mappedBy="assignedCubicle")
  public Employee getEmployee() {
         return employee; }
  public void setEmployee(Employee e) {
         this.employee = e;
  }
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Inheritance

- Entities can be involved in inheritance relationships
- Both abstract and concrete classes can be entities
- Entity class may inherit from on-entity class: behavior-inheritance, parent state non-persistent
- Inheritance mapping strategies:
  - Single table per class hierarchy (discriminator values)
  - Single table per concrete entity class
  - Fields of sub-class and parent class in separate tables. Join to instantiate subclass (see example)

```
@Entity
@Inheritance(strategy =
  InheritanceType.JOINED)
public class Pet implements
  java.io.Serializable
{
 ...
}
@Entity
@Inheritance(strategy =
  InheritanceType.JOINED)
public class Cat extends Pet
{
  ...
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

EntityManager

- **EntityManager responsible to manage (lifecycle of) entities**
- **EntityManager API supports**
  - Creation and removal of entities
  - Find entities by primary keys
  - Query over entities
- **EntityManager is associated with persistence context, i.e. set of entities**
- **Set of managable entity classes (classes related or grouped by appliccation and collocated in mapping to same database defined by persistence unit**

```
@Stateless public class OrderEntry {
  @PersistenceContext EntityManager em;
  public void enterOrder(int custID, Order newOrder) {
         Customer cust = (Customer)em.find("Customer", custID);
         cust.getOrders().add(newOrder);
         newOrder.setCustomer(cust);
  }
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

EntityManager API

```
public interface EntityManager {
  public void persist (Object entity);
  public <T> T merge(T entity);
  public void remove(Object entity);
  public <T> T find(Class<T> entityClass, Object primaryKey);
  public <T> T getReference(Class<T> entityClass, Object primaryKey);
  public void flush();
  public void refresh(Object entity);
  public boolean contains(Object entity);
  public Query createQuery(String ejbqlString);
  public String createNamedQuery(String name);
  public String createNativeQuery(String sqlString);
  ...
  public void close();
  public boolean isOpen();
  public EntityTransaction getTransaction();
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

More About Entity Managers

- Lifecycle of entity managers either managable by J2EE container (container-managed) or by application itself (application managed: termination by calling `close()`)
- Last option typical for J2SE environments
- Obtaining entity managers possible through dependency injection or JNDI:

```
@Stateless
@PersistenceContext(name="orderEM"
  unitName="Order")
public class MySessionBean implements MyInterface {
  @Resource SessionContext ctx;
  public void foo() {
        EntityManager em =
              (EntityManager)ctx.lookup("OrderEM");
  }
}
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Manager Factories

- Application-managed entity managers obtained via a factory `EntityManagerFactory`
- In J2EE container the factory is retrieved by injection:

```
@PersistenceUnit
EntityManagerFactory emf;
```

- In J2SE:

```
EntityManagerFactory emf =
javax.persistence.Persistence.createEntityManagerFactory(
  "Order");
EntityManager em = emf.createEntityManager();
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Example: Resource-Local EntityManager

```
import javax.persistence.*;
public class PasswordChanger {
  public static void main (String[] args) {
        EntityManagerFactory emf =
                Persistence.createEntityManagerFactory();
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        user = em.createQuery
                ("SELECT u FROM User u WHERE u.name=:name AND
                        u.pass=:pass")
                .setParameter("name", args[0])
                .setParameter("pass", args[1])
                .getSingleResult();
        if (user!=null)
        user.setPassword(args[2]);
        em.getTransaction().commit();
        em.close(); // application ends lifecycle
        emf.close ();
  }
}
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Transaction Control

- **Transaction control either via JTA or by using resource-local `EntityTransaction`**
- **In J2EE both options mandatory (container-managed environments must use the first)**
- **In J2SE resource-local option only!**

```
public interface EntityTransaction {
  public void begin();
  public void commit();
  public void rollback();
  public boolean isActive();
}
```

CT SE 2

Software & Engineering Architecture

Persistence Contexts

- **A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance**
- **Within the persistence context, the entity instances and their lifecycle are managed by the entity manager**
- **A persistence context may be either a transaction-scoped persistence context or an extended persistence context**
- **A persistence context is either container-managed or application-managed (such as the `EntityManager`)**
- **Container-managed PersistenceContexts:**
  - *Transaction-scoped*: entity manager created in JTA transaction and no other context exists. Ends with transaction termination
  - *Extended*: starts with creation of entity manager until it is closed. Omly applicable within the scope of a stateful session bean
- **Application-managed PersistenceContexts:**
  - *Transaction-scoped*: for JTA transactions same a sabove. For resource-local entity managers starts with new resource-transaction and ends when transactions end
  - *Extended*: start/end directly depending on lifetime of entity manager.

Example: Application-managed Transaction-scoped Persistence Context

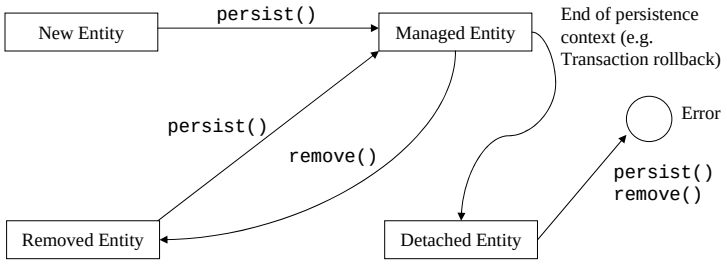
```
public class ShoppingCart {
   private EntityManager em;
   private EntityManagerFactory emf;
   public ShoppingCart() {
            emf = Persistence.createEntityManagerFactory();
            em = emf.createEntityManager();
   }
   public Order getOrder(Long id) { return em.find(Order.class, id); }
   public Product getProduct() {
            return (Product) em.createQuery("select p from Product p
                                    where p.name = :name")
                                    .setParameter("name", name)
                                    .getSingleResult();
   }
   public LineItem createLineItem(Order order, Product product, intquantity) {
            em.getTransaction().begin();
            LineItem li = new LineItem(order, product, quantity);
            order.getLineItems().add(li);
            em.persist(li);
            em.getTransaction().commit();
            return li;
   }
   public void destroy() { em.close(); emf.close();}
}
```

SIEMENS

CORPORATE TECHNOLOGY

Entity Instance Life Cycle

- Synchronization with database always performed using transaction commits (not shown below)
- Persistence Context may either be scoped to a transaction (default) or may have extended lifetime
- The Persistence context is defined upon creation of the EntityManager



CT SE 2 Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Listeners and Callback Methods

- Container may notify beans about lifecycle events using bean callback methods ir separate event listener classes
- Callbacks may throw runtime exceptions (rollback transaction)
- Callbacks must not throw application exceptions
- Callbacks can not call entity managers
- Events supported: `@PrePersist, @PostPersist, @PreRemove, @PostRemove, @PreUpdate, @PostUpdate, @PreLoad`

```
@Entity
@Table(name="CUSTOMER")
@EntityListener(CustomerCallbackListener.class)
public class Customer implements java.io.Serializable {…}
public class CustomerCallbackListener{
  @PrePersist public void doPrePersist(Customer customer) {...}
  ...
}
```

CT SE 2 Software & Engineering Architecture

Query API

- Supports dynamic and static queries
- Also supports named parameter binding, pagination control
- Queries are polymorphic (subclasses are also considered)

```
public interface Query {
  public int ExecuteUpdate();
  public List getResultList();
  public Object getSingleResult();
  public Query setParameter(string name, object value);
  ...
}
```

```
// Example code fragment:
public List findWithName(String name) {
  return em.CreateQuery(„SELECT c FROM Customer c WHERE c.NAME LIKE
  :custName“)
  .setParameter(custName, name)
  .setMaxResults(10)
  .getResultList();
}
```


Named Query

- EntityManager API also allows to instantiate named query
- Special annotation `@NamedQuery` available

```
@NamedQuery(name=„findAllCustomersWithName“,
  queryString=„SELECT c FROM customer.c WHERE
  c.name LIKE :custName“)
...
@PersistenceContext
public EntityManager em;
...
Customers =
  em.createNamedQuery(„findAllCustomersWithName“)
  .setParameter(„custName“, „Smith“)
  .getResultList();
```

SIEMENS

CORPORATE TECHNOLOGY

Entity Manager and Query Example

```
@@Stateless
@Remote(CustomerDAO.class)
public class CustomerDAOBean implements CustomerDAO {
  @PersistenceContext  private EntityManager manager;
  public int create(String first, String last, String street, String
  city, String state, String zip)   {
     Customer customer = new Customer(first, last, street, city,
                                                    state, zip);
     manager.persist(customer);
     return customer.getId();
  }
  public Customer find(int id)   {
     return manager.find(Customer.class, id);
  }
  public List findByLastName(String name)   {
     return manager.createQuery("from Customer c where c.name.last =
     :name").setParameter("name", name).getResultList();
  }
}
```

CT SE 2 Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

EJB QL (Query Language)

- **EJB QL used to define queries over entities and their persistent state**
  - Select entities or values with Query API
  - Select entities in EJB 2.1 container-managed beans using finders
  - Select entities or values in EJB 2.1 container-managed beans using select methods
- **Specifies queries in a portable way**
- **Augments previous versions with bulk update and delete, join, group by, having, projections, sub-queries**
- **Now also applicable to dynamic queries**
- **Is complied to target language such as SQL**
- **Queries use abstract persistence schemas as defined by entities**

```
SELECT C
FROM CUSTOMER c JOIN c.orders o JOIN c.address a
WHERE a.STATE = ,CA‘
ORDER BY, o.quantity, o.totalcost
```

CT SE 2 Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Packaging

- **A persistence unit is a logical grouping that includes:**
  - A named entity manager together with its provider and configuration information
  - The set of managed classes included in the persistence unit for the specified entity manager
  - Mapping metadata (in the form of metadata annotations and/or XML) that specifies the mapping of the classes to the database
- **Within J2EE, the persistence archive, or .par file, is the packaging artifact or deployment vehicle for persistence units.**
- **Each persistence archive houses a single persistence unit.**
- **Persistence archives may be referenced by J2EE application metadata.**

```
<application>
        <module>
          <persistence>orderEntities.par</persistence>
        </module>
</application>
```

- **Any number of persistence archives may be deployed within a J2EE application (EAR)**

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Entity Packaging (cont'd)

- **Defined in a persistence.xml file located in the META-INF directory of the persistence archive are**
  - The configuration information for the entity manager and its entity manager factory
  - the managed classes included in the persistence unit
  - and the object/relational mapping information for a persistence unit

```
<entity-manager>
   <name>em1</name>
   <provider>com.acme.persistence</provider>
   <jta-data-source>jdbc/MyDB</jta-data-source>
   <mapping-file>ormap.xml</mapping-file>
   <jar-file>MyApp.jar</jar-file>
   <class>com.widgets.Order</class>
   <class>com.widgets.Customer</class>
   <properties>
           <property name="sql-logging" value="on"/>
   </properties>
</entity-manager>
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Summary

- **EJB 3.0 significantly improved over previous versions**
  - Empowers developers due to more possibilities
  - Increases productivity by reducing required artifacts, configuration by exception, dependency injection, annotations
  - Much easier to handle through POJOs and POJIs
  - Developers might use deployment descriptor or annotations or both
  - Extensibility of container using callback concept
- **Persistence APIs**
  - Follows best practices from JDO, Hibernate, etc
  - POJO model for entities
  - Dynamic queries
  - More OO than ever: inheritance and polymorphism
  - EntityManagers and PersistenceContexts
  - Flexible O/R Mapping support
- **In summary, makes good impression**

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Resources

- **Documents:**
  - JSR 220 specification (EJB Simplified API, Java Persistence API, EJB Core Contracts and Requirements) download from http://jcp.org/aboutJava/communityprocess/pr/jsr220/index.html
- **Application Servers with alpha/beta EJB 3.0 support**
  - JBoss: http://www.jboss.org
  - Oracle: http://otn.oracle.com/ejb3
  - Joint venture of Sun and Oracle: https://glassfish.dev.java.net/
  - Eclipse ORM project: http://www.eclipse.org/proposals/eclipse-ejb30-orm/index.html

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Q&A

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Thanks for Attending

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

BACKUP MATERIAL

- DEPENDENCY INJECTION
- EJB 2.1 ENTITY BEANS EXAMPLE

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

Dependency Injection Explained (1)

- Consider the following example:

```
public class A {
  public B cb;
  public A() { cb = get_B_Object(...); }
  ...
}
public class B { ... }
public class Container {
  public A createA() {
         return new A();
  }
}
```

- Liability:
  - A depends on way to create or obtain a B instance

Software & Engineering Architecture

Dependency Injection Example (2)

- **Here is another possibility:**

```
public class A {
  public B cb;
  public A() { ... }
  ...
}
public class B { ... }
public class Container {
  public A createA() {
         A a = new A();
         a.cb = get_B_Object(...);
         return a;
  }
}
```

- **Benefits:**
  - A is not dependent anymore on how to create or obtain B instance
  - This is called field injection and is normally committed by reflection

Dependency Injection Example (3)

- **Other possibilities include setter injection and constructor injection**

```
public class A {
  protected B b1;
  protected B b2;
  public A(B b1) { this.b1 = b1; }
  public setB2(B b2) { this.b2 = b2; }
  ...
}
public class B { ... }
public class Container {
  public A createA() {
         B b1 = get_B_Object(...);
         A a = new A(b1); // constructor injection
         B b2 = get_B_Object(...);
         a.setB2(b2); // setter aka property injection
         return a;
  }
}
```

Dependency Injection Example (4)

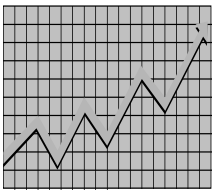
- **To further detach class to be injected from container we can use metadata annotations:**

```
public class A {
  protected B b1;
  protected B b2;
  @MyFieldInjection
  public    B b3;
  @MyConstructorInjection
  public A(B b1) { this.b1 = b1; }
  @MySetterInjection
  public setB2(B b2) { this.b2 = b2; }
  ...
}
```

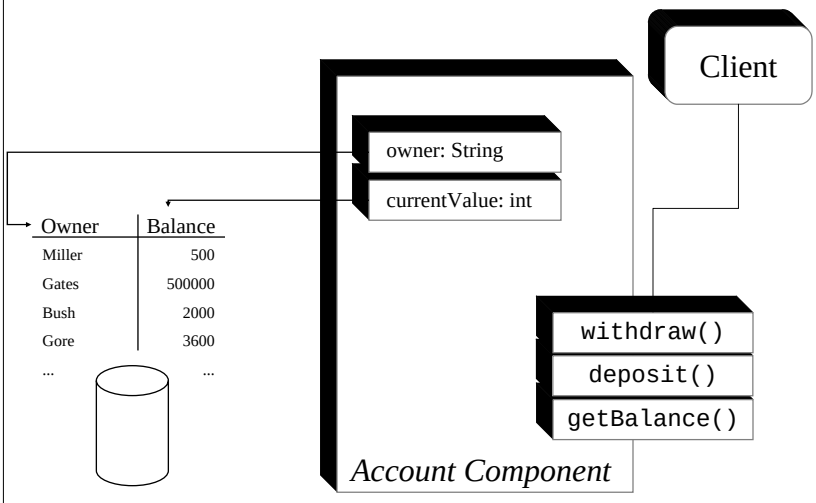
- **Or XML-based configuration files**

A Sample Entity Bean

- Let us build a simple Enterprise Bean that represents a bank account.
- A client deposits or withdraws money from her/his account.
- The bean is designed as an Entity Bean.
- Persistence is automatically handled by the run-time environment.



A Sample Entity Bean (cont'd)



A Sample Entity Bean (cont'd)

Steps to develop an entity bean:

1. define enterprise bean's business purpose (remote interface)
2. define how enterprise bean can be created, located and destroyed (home interface)
3. provide implementation (entity bean)
4. create deployment descriptor and ejb-jar
5. create database table
6. deploy the entity bean
7. create a client application

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Remote Interface

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

public interface Account extends EJBObject {
    public void deposit(int amount)
           throws RemoteException;
    public void withdraw(int amount)
           throws RemoteException;
    public int getBalance() throws RemoteException;
}
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Home Interface

```
import java.rmi.RemoteException;
import javax.ejb.FinderException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface AccountHome extends EJBHome{
    public Account create(java.lang.String owner)
       throws CreateException, RemoteException;

    public Account findByPrimaryKey(String owner)
       throws FinderException, RemoteException;
}
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Bean class

```
import java.rmi.*;
import javax.ejb.EntityBean;
import javax.ejb.CreateExcept

public abstract class AccountBean implements EntityBean
{
     public String ejbCreate(String owner)
                   throws CreateException{
            setOwner(owner);
            return null; // CMP
     }
     public void ejbPostCreate(String owner)
                   throws CreateException{
            // do nothing
     }
     ...
```

- signatures must match parameters of the home interface's create method
- findByPrimaryKey() method generated at deployment and implemented by the container

CT 2 S&E Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Callback Methods to inform Bean about Events in its Lifecycle

```
...
public void setEntityContext(EntityContext ec) {
      // interface to the container
}
public void unsetEntityContext() {
}
public void ejbLoad() {
      // called after container loads persistent state
}
public void ejbStore()  {
      // called before container stores persistent state
}
public void ejbRemove() {
      // deletes database entry
}
public void ejbActivate()  {
      // allocate resources
}
public void ejbPassivate() {
      // deallocate resources
}
```

CT 2 S&E Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Declarations for Container-Managed fields

```
public abstract void setCurrentValue(int currentValue);
public abstract int getCurrentValue();
public abstract void setOwner(String owner);
public abstract String getOwner();
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Remote Interface Implementation

```
public void deposit(int amount) {
     setCurrentValue(getCurrentValue()+amount);
}
public void withdraw(int amount){
     setCurrentValue(getCurrentValue()-amount);
}
public int getBalance() {
      return getCurrentValue();
}
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd)

Steps to develop an entity bean:

1. ✓ define enterprise bean's business purpose (remote interface)
2. ✓ define how enterprise bean can be created, located and destroyed (home interface)
3. provide implementation (entity bean)
4. ✓ create deployment descriptor and ejb-jar
5. create database table
6. deploy the entity bean
7. create a client application

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Deployment

Component constituents are archived in ejb-jar file (e.g. account.jar):

```
Descriptor
      META-INF/ejb-jar.xml
Remote Interface:
      Account.class
Home Interface:
      AccountHome.class
Bean Class:
      AccountBean.class
...
```

CT SE 2

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): ejb-jar.xml

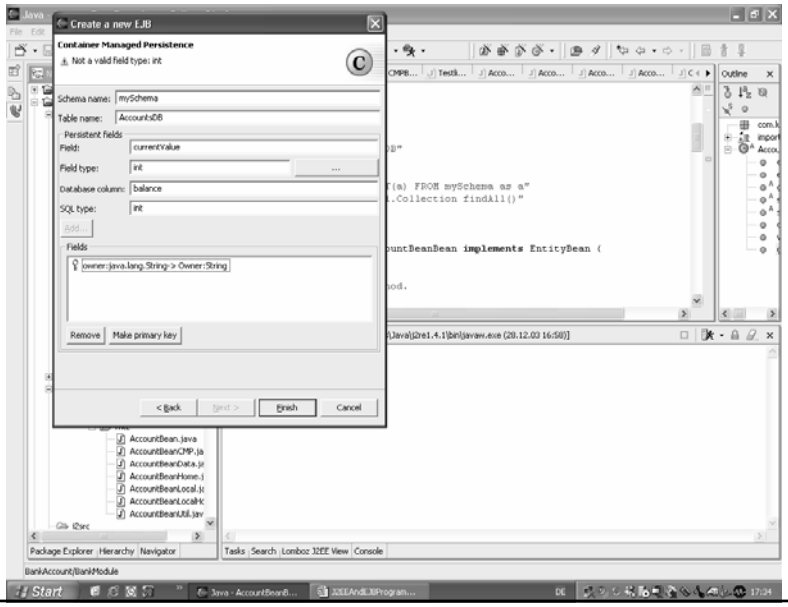
```
<enterprise-beans>
    <entity>
          <ejb-name>AccountEJB</ejb-name>
          <home>AccountHome</home>
          <remote>Account</remote>
          <ejb-class>AccountBean</ejb-class>
          <persistence-type>Container</persistence-type>
          <prim-key-class>java.lang.String</prim-key-class>
          <cmp-version>2.x</cmp-version>
          <cmp-field ><field-name>owner</field-name></cmp-field>
          <cmp-field ><field-name>currentValue</field-name></cmp-field>
          <primkey-field>owner</primkey-field>
     </entity>
</enterprise-beans>
```

Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

A Sample Entity Bean (cont'd): Database



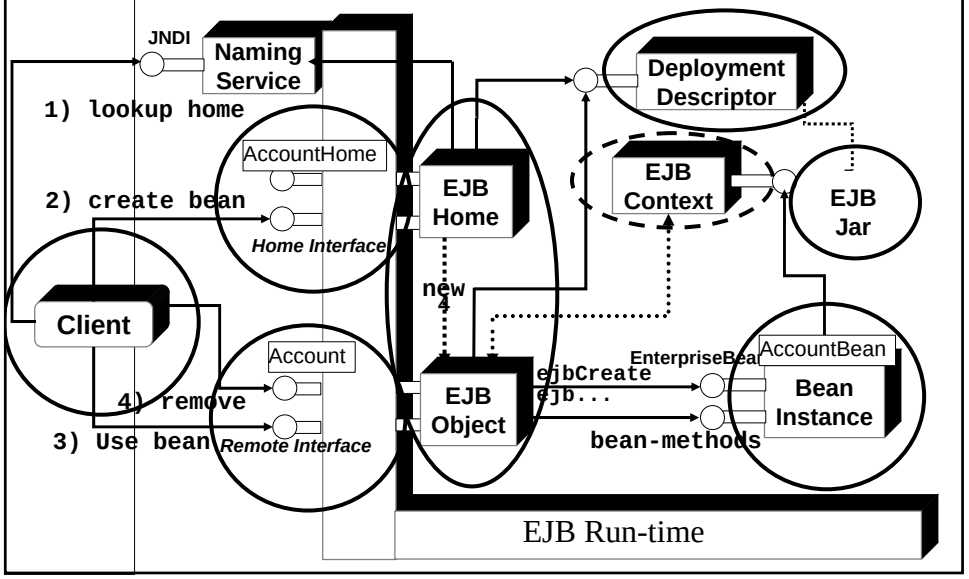
Software & Engineering Architecture

A Sample Entity Bean (cont'd): Client

```
Context jndiContext =  new InitialContext();   // Access JNDI
Object ref=jndiContext.lookup("AccountHome");
AccountHome aHome = (AccountHome)
    javax.rmi.PortableRemoteObject.narrow(ref, AccountHome.class);

Account myAccount1 = aHome.create("mike");
Account myAccount2 = aHome.findByPrimaryKey("johnson");
myAccount1.withdraw(500); // get money from first account
myAccount2.deposit(500);  // and deposit it in second account

System.out.println(myAccount1.getBalance());
System.out.println(myAccount2.getBalance());
```


SIEMENS

EJB Servers/Containers



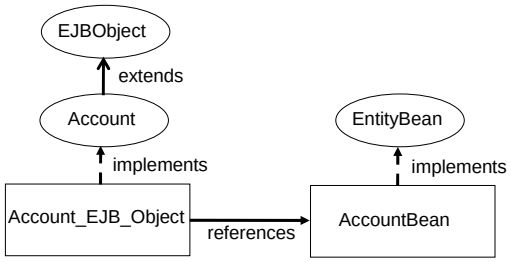
SIEMENS

CORPORATE TECHNOLOGY

EJB Object and EJB Home

EJB Object

- implements remote interface of the enterprise bean
- wraps enterprise bean instance



CT SE 2

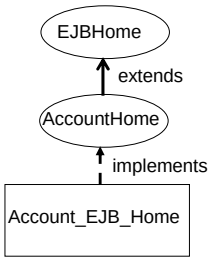
Software & Engineering Architecture

SIEMENS

CORPORATE TECHNOLOGY

EJB Object and EJB Home (cont'd)

EJB Home



EJB Home and EJB Object are implemented by the vendor.

CT SE 2

Software & Engineering Architecture

EJB Object and EJB Home (cont'd)

EJB Home

- working closely with the EJB container
- responsible for locating, creating and removing enterprise beans
- Example:
  - `Account myAccount1 = aHome.create("mike");`
  - EJB Home works with EJB Server to create AccountBean (or get it from pool)
  - data will be added to the database
  - EJB Server creates EJB Object to wrap the AccountBean instance
  - remote reference/stub to EJB Object will be returned to the client