

# Transaction Definition

## ❖ Unit of work that accesses one or more shared resources (usually databases)

- Set of one or more activities related to each other
- Must be completed together or not at all
- Cohesion of unit is normally mission critical
- Examples
  - ATM
    - Withdraw from one source, deposit to another
  - Order System (e.g. online book store)
    - Locate item, charge account, schedule shipment
  - Medical System
    - Identify medical state, dispense prescription

# Transaction Properties (ACID)

## ❖ Atomic

- Transaction must execute completely or not at all

## ❖ Consistent

- Data in database is always in a consistent state (data integrity makes sense)
- A transactional system fulfills its responsibility for consistency by ensuring that a transaction is atomic, isolated and durable
- Application developer must ensure that database has appropriate constraints (primary keys, referential integrity, etc.) and that unit-of work doesn't result in inconsistent data

## ❖ Isolated

- Transaction executes without interference

## ❖ Durable

- Changes are not lost if the system crashes

# EJB Transaction Support

- ❖ **EJB is designed to support transactions automatically.**
- ❖ **Declarative (CMT)**
  - Easy
  - Transaction management controlled through deployment descriptor ( transaction attributes for individual enterprise bean methods)
- ❖ **Programmatic (BMT)**
  - Complex
  - Explicit transaction demarcation
  - Direct programming calls to Java Transaction Service (JTS) API
  - Transaction management code mixed with business code (change in transactional behavior requires change in business code)

# Transaction Scope

- ❖ One or more tasks that operate as a unit of work; succeed or be rolled back together
- ❖ Tasks
  - EJB methods
- ❖ Unit of Work
  - Each bean visited by the methods during a thread of execution depending on bean transaction attributes
  - Ends when thread of execution
    - ❖ Completes normally
    - ❖ An exception is thrown ( type)
    - ❖ Transaction is rolled back

# Transaction Attributes

- ❖ EJB servers can manage transactions implicitly, based on the transaction attributes established at deployment time.
- ❖ **Supports Transactions**
  - Supports
  - Required
  - RequiresNew
  - Mandatory
- ❖ **Transactions not Supported**
  - NotSupported
  - Never
- ❖ Set a transaction attribute for the entire EJB or for individual methods (flexibility)

# Setting a Transaction Attribute

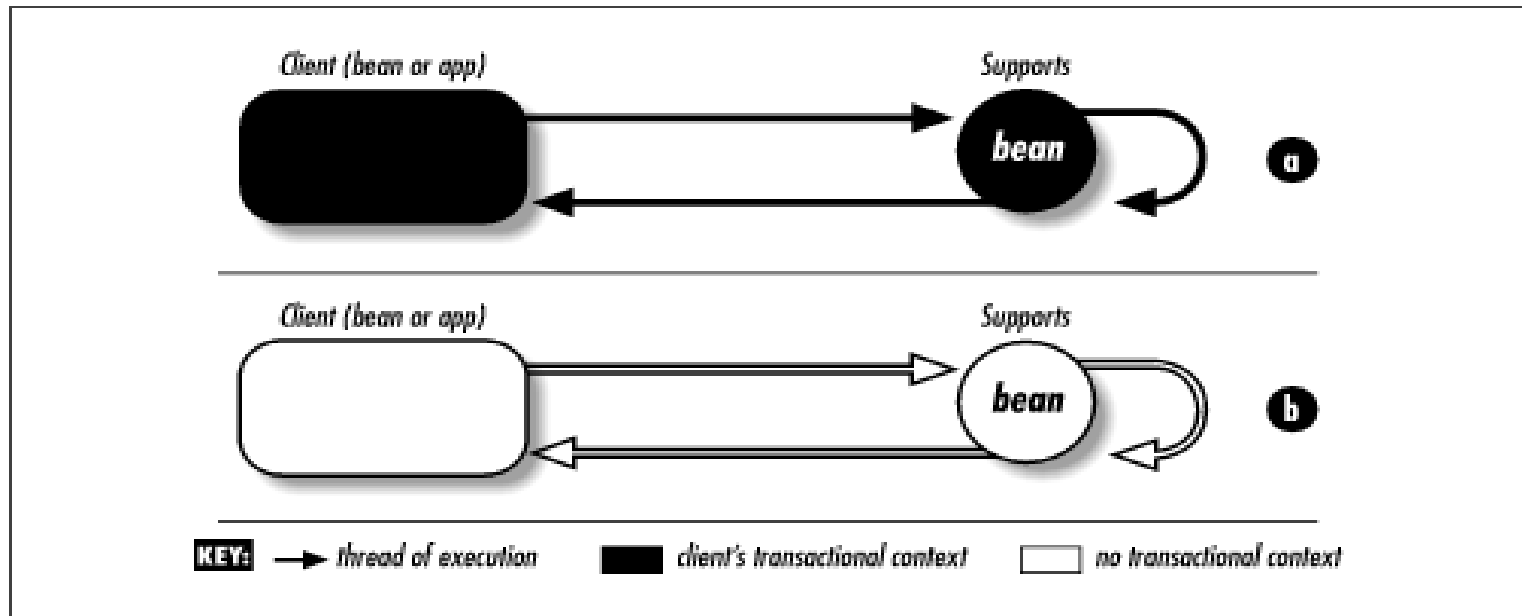
```
<ejb-jar>
...
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>MyEJB</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

# Not Supported



- ❖ Transaction is suspended during the method of the invoked Bean;
- ❖ Resumes when method complete
- ❖ Transaction scope is not propagated to invoked EJB or anything it invokes

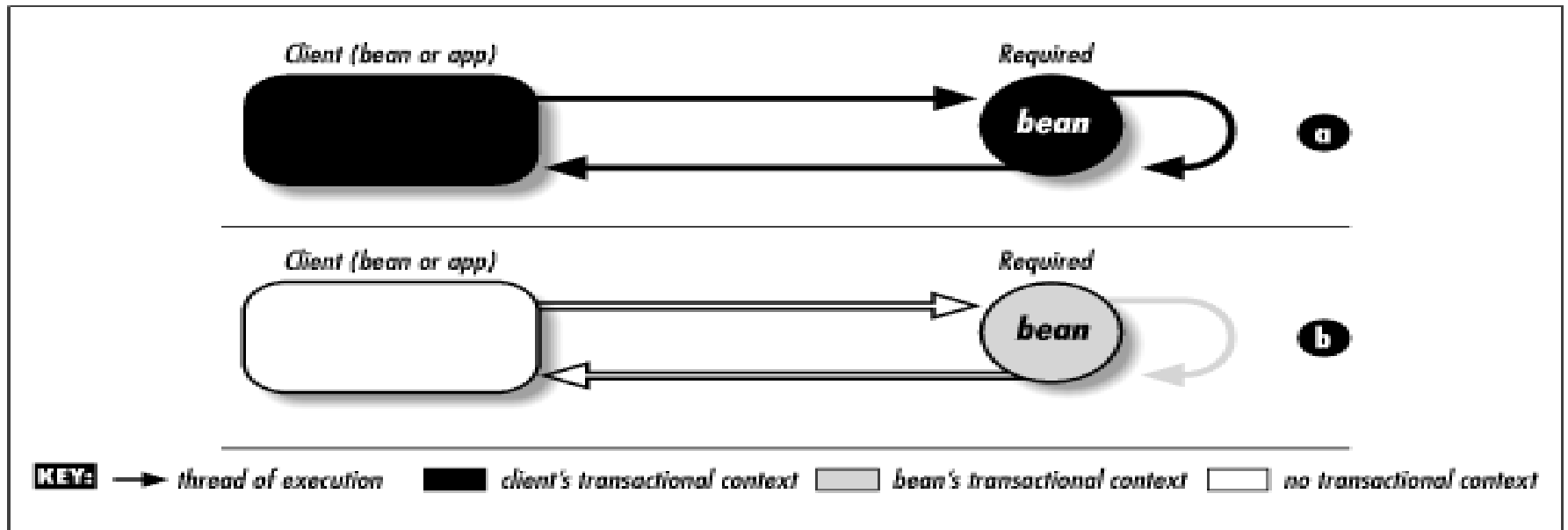
# Supports



- ❖ Joins the transaction context if invoked as part of a transaction (A)
- ❖ Does not require a transaction; can be invoked outside of a transaction context (B)

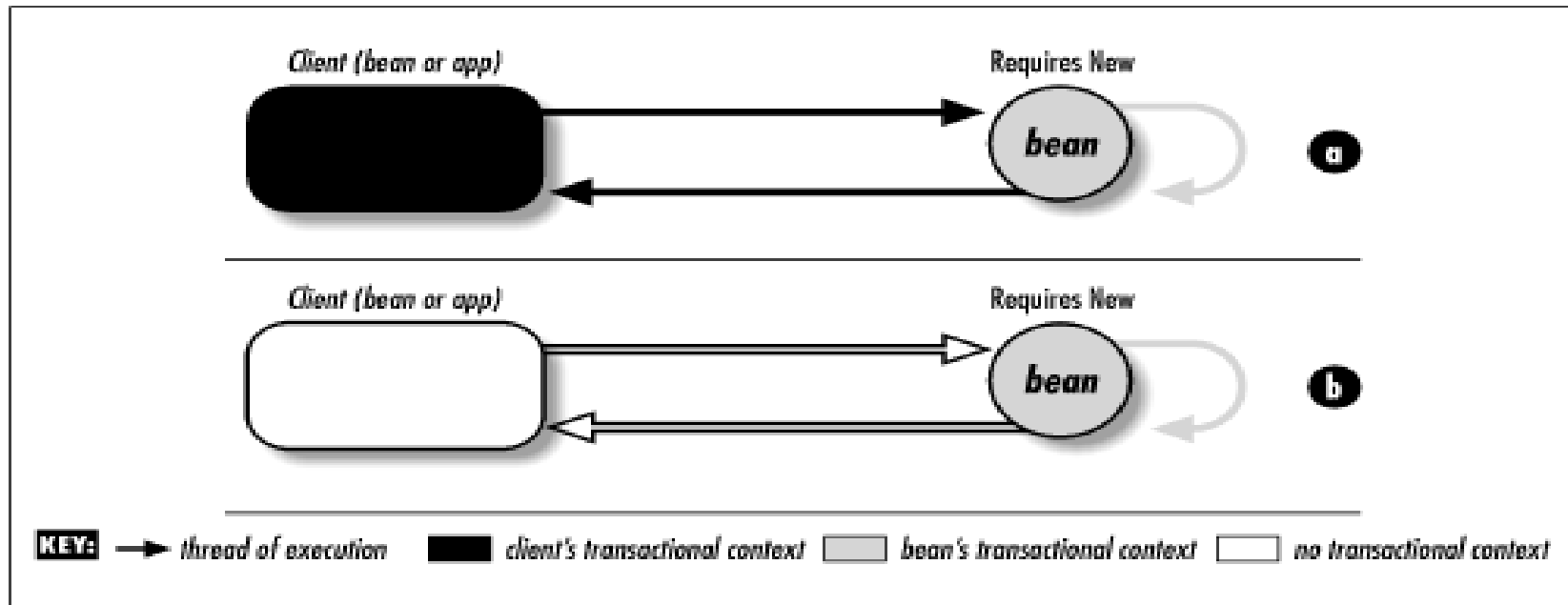


# Required



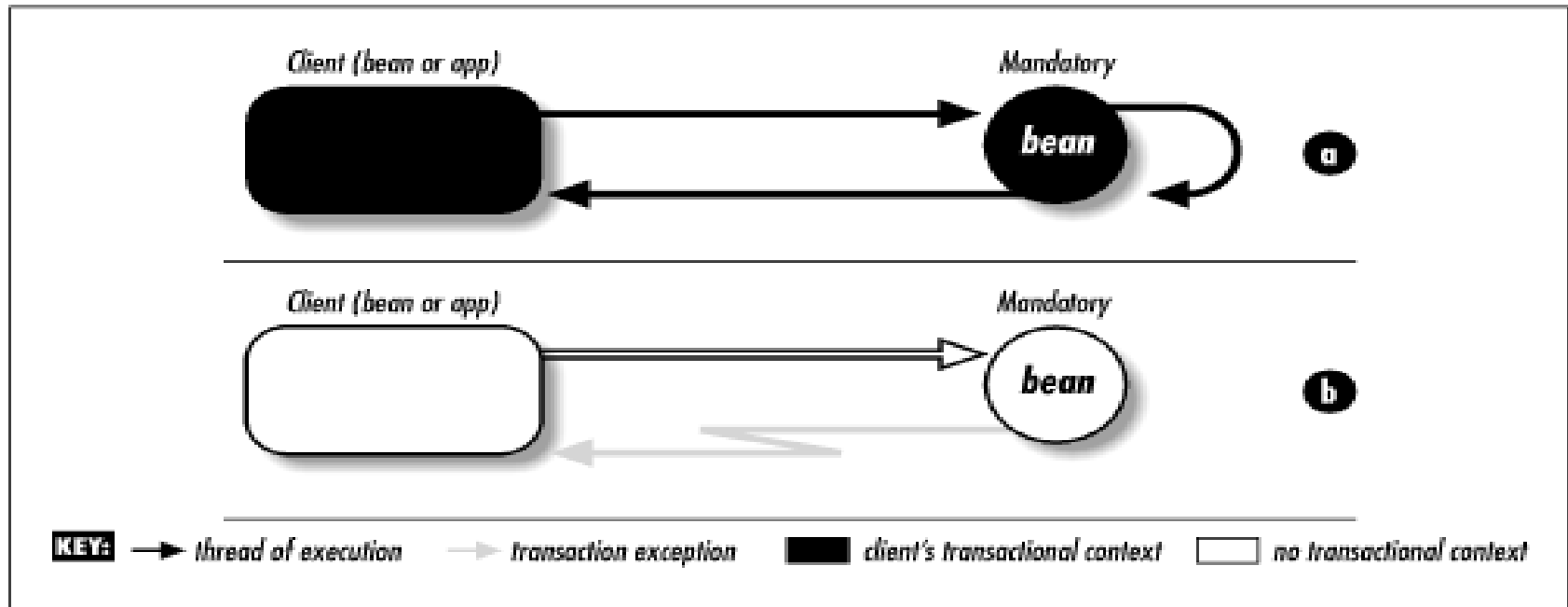
- ❖ Enterprise bean method must be invoked within the scope of a TX
- ❖ Joins the transaction context if invoked as part of a transaction (A)
- ❖ Initiates its own transaction context if invoked outside of a transaction (B)

# RequiresNew



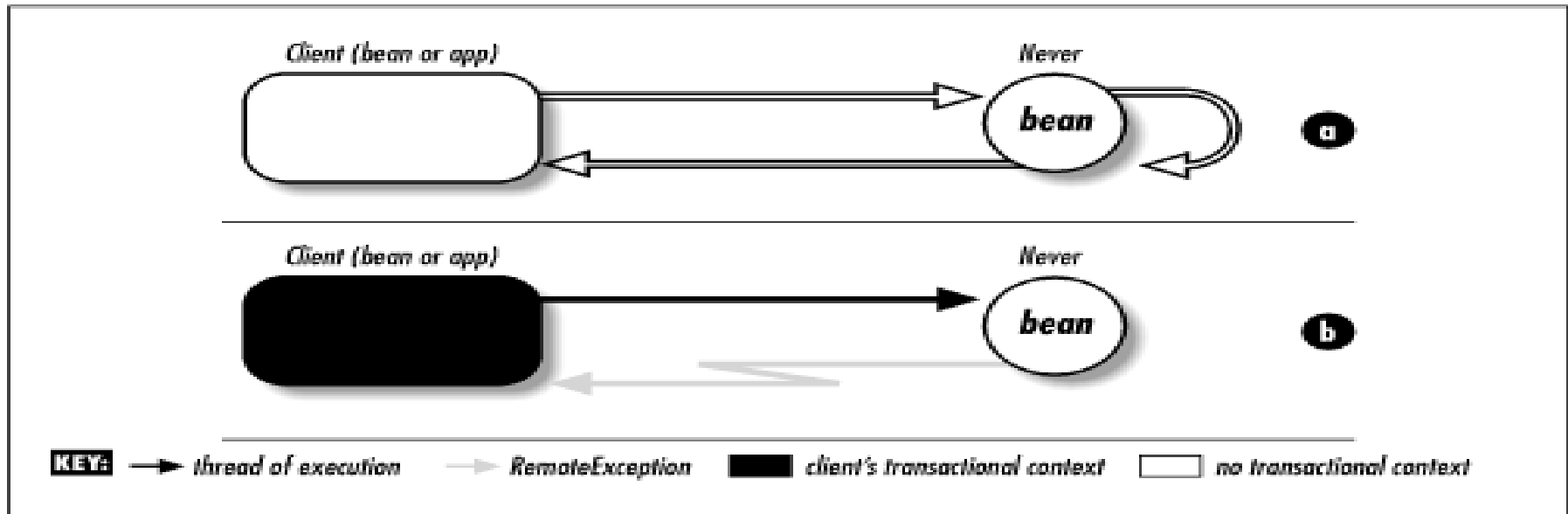
- ❖ A new transaction is always started.
- ❖ Initiates its own transaction context whether called within an existing transaction context (A) or outside of a transaction context (B)
- ❖ Initiated transaction completes prior to returning to caller

# Mandatory



- ❖ Joins the transaction context if invoked as part of a transaction (A)
- ❖ Throws a Transaction Exception if not called within a transaction context (B) (TransactionRequiredException for Remote Clients; TransactionRequiredLocalException for Local Clients)

# Never



- ❖ Bean method must not be invoked within the scope of a transaction  
Throws a Transaction Exception if called within a transaction context
- ❖ (A) (RemoteException to Remote Clients; EJBException to Local Clients)
- ❖ Must be invoked outside of a transaction context (B)

# Isolation and Database Locking

❖ **What happens when two or more transactions attempt to access the same Data:**

**– Dirty Reads**

- First transaction reads uncommitted changes from a second transaction that may eventually be rolled back

**– Repeatable Reads**

- **Data guaranteed to be the same if read multiple times during the same transaction; implemented with locks or snapshots**
- Opposite: nonrepeatable read

**– Phantom Reads**

- **First transaction sees new rows added by second transaction after beginning of the first transaction**

# Database Locks

- How to prevent overlapping transactions from viewing the other's data?
  - Read Lock
    - Data is prevented from changing while transaction is in progress
    - Current transaction is prohibited from making changes
  - Write Lock
    - Prevents other transactions from modifying the data
    - Permits dirty reads by other transactions (and by the current TX itself)
  - Exclusive Write Lock
    - Prevents other transactions from reading and modifying data
    - Prevents dirty reads
  - Snapshots - frozen view of data
    - Every transaction gets its own snapshot of the data

# Transaction Isolation Levels

- Defined in terms of the isolation conditions (dirty reads, repeatable reads, and phantom reads)
  - Used in DBS to describe how locking is applied to data within a transaction:
- Read Uncommitted
    - Can read data being modified by another transaction
    - Allows dirty, non-repeatable, and phantom reads
  - Read Committed
    - Cannot read data being modified by another transaction
    - Allows non-repeatable and phantom reads; prevents dirty reads
  - Repeatable Read
    - Cannot change data being read by another transaction
    - Allows phantom reads; prevents dirty and non-repeatable reads
  - Serializable
    - Transaction has exclusive read/write privileges to data
    - Different transactions can neither read or write same data

# Specifying Isolation Level

- EJB deployer sets transaction isolation levels in a vendor specific way if the container manages the transaction

```
<weblogic-ejb-jar>
<transaction-isolation>
  <!-- TRANSACTION_SERIALIZABLE
  TRANSACTION_READ_COMMITTED
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_REPEATABLE_READ
  -->
<isolation-level>TRANSACTION_READ_COMMITTED</isolation-level>
  <method>
    <ejb-name>Account</ejb-name>
    <method-name>*</method-name>
  </method>
</transaction-isolation>
```



# Message Driven Bean Transactions

- Valid Values
  - *NotSupported*
  - *Required*
- Client transaction context not propagated to MDB
  - *Supports, RequiresNew, Mandatory, and Never* are relative to client

# Exceptions within Transactions

- **System Exceptions (e.g. NullPointerException, EJBException)**
  - Container automatically rolls back transaction
  - Bean instance is discarded
- **ApplicationException (e.g. AccountOverdrawException)**
  - Container does not automatically rollback transaction
  - Exception delivered to client
  - Client optionally signals rollback
    - `EJBContext.setRollbackOnly()`

# Distributed Transaction

- In addition to managing transactions in its own environment, an EJB server can coordinate with other transactional systems
  - ❖ E.g. EJBs on different EJB servers
- EJB server would cooperate to manage the transaction as one unit-of-work
- Two-phase commit (2-PC) is required

# Enable JTS Transactions for Connection Pool

mydomain> JDBC Tx Data Sources> MyJDBC

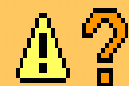
Connected to localhost:7001

Active Domain

Configuration

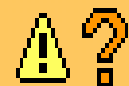
Targets

Notes



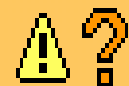
Name:

MyJDBC Tx Data Sou



JNDI Name:

weblogic.jdbc.dataSourc



Pool Name:

MyJDBC Connection Poo

# Enable JTS Transactions in Entity Bean

```
<weblogic-rdbms-jar>  
  <weblogic-rdbms-bean>  
    <ejb-name>Account</ejb-name>  
    <data-source-name>java.jdbc.txDataSource  
  </data-source-name>  
    <table-name>Account</table-name>
```

# Enable JTS Transactions for JMS

mydomain > JMS Connection Factories > MyJMS Conn

Connected to localhost:7001

Active Domain: mydom

Configuration

Targets

Notes

General

Transactions

# Bean Types and Transactions

## ❖ Entity

- In EJB 1.1 must be container managed
- May force a rollback with `setRollbackOnly()`

## ❖ Stateless Session

- Transactions simple. Abort by throwing **EJBException**

## ❖ Stateful Session

- E.g. shopping cart ( Caching)
- What about the conversational state?
- Have to respond to transaction aborts to keep conversational state consistent

# Stateful Session Bean and Transactions

- ❖ • **Can implement SessionSynchronization interface**
  - `afterBegin()`, `beforeCompletion()`, `afterCompletion(boolean)`
- ❖ **If `afterCompletion` is false, you can roll back the conversational state**
- ❖ **SessionSynchronization can not be implemented with vbean-managed TX**



# Programmatic Transactions

- ❖ **Bean programmer is responsible for issuing begin(), commit(), and abort() calls in application code**
- ❖ **Discouraged**
  - ❖ Why not let the container do it?
  - ❖ Flexibility and fine grained control
- ❖ **Use Java Transaction API (JTA)**
  - **Simple layer over Java Transaction Service (JTS) which is an implementation of the CORBA Object Transaction Service (OTS)**

# User Transaction

- ❖ **Required EJB Transaction interface**  
(javax.transaction.UserTransaction)
- ❖ **Set transaction-type on bean-managed transaction (BMT) in deployment descriptor**
- ❖ **Provides access to transaction service**
  - Can start and commit
  - Can mark that transaction must be rolled back
- ❖ **Client and server components can use**
- ❖ **Container/Server does not have to expose JTS API to the bean**

# Code Example

```
public void transfer(AccountRemote from, AccountRemote to, float
    amount ) throws AccountException {
    UserTransaction tx = null;
    try {
        tx = ctx.getUserTransaction();
        tx.begin();
        // perform JDBC operations and transfer the money
        tx.commit();
    }
    catch (Exception ex) {
        tx.setRollbackOnly();
        throw new AccountException();
    }
}
```

# Programmatic Transactions in EJB

## ❖ Allowed in Session Beans

`<session>`

...

`<transaction-type>Bean</transaction-type>`

- Stateless Session Beans must begin/end in same method
- Stateful Session Beans may begin/end in separate methods (although not advised)

## ❖ EJB 1.1 disallows this for Entity Beans

## ❖ MDB - TX scope must begin and end within the `onMessage()` method

## ❖ Bean controls transactions programmatically

# Transactional Clients

## ❖ End user can also control transactions

```
UserTransaction ut = (UserTransaction);  
jndiContext.lookup("javax.transaction.UserTransaction");  
ut.begin();  
// perform multiple operations on beans  
account1.deposit( 34 );  
account2.withdraw( 34 );  
ut.commit();
```

# Client Issues

---

- ❖ Lots of network traffic required when client manages transactions
- ❖ More complicated
- ❖ Should have asked a session bean to perform the set of operations over on the server
- ❖ Transactions should be short

# Performance Implications

- ❖ **Avoid distributed transactions across multiple resource managers and/or multiple transaction managers**
  - **Consider JMS synchronization**
- ❖ **Avoid client control of transaction boundaries**
  - **Try to encapsulate transactional business operations in session beans**

# EJB Transaction Summary

- ❖ Supports EJB goal of removing middleware programming burden from bean programmer
- ❖ Declare transactional requirements in deployment descriptor
- ❖ • Transactions have performance implications
  - Limit client control, distributed transactions
  - Can control frequency of `ejbLoad()` and `ejbStore()`