Session: 8

Transactions

Objectives

- Explain the concept of transactions
- Define various properties of transactions
- Explain Java Transaction API (JTA)
- Describe the interfaces of JTA that can be used in the enterprise applications
- Explain programmatic versus declarative demarcation
- Explain container-managed transaction
- Explain bean-managed transaction
- Explain how to manage transactions in messaging

Transaction Basics

- Transaction is a group of operations to be executed as a single unit.
- For example, purchase made through an e-commerce portal involves different operations such as:
  - Choosing the product
  - Adding it to the shopping cart
  - Making payment for the product
  - Finalizing the transaction
- All these operations are an integral part of an e-commerce transaction.

Transaction and Data Integrity 1-3

- An enterprise application stores critical information for business operations in databases.
- If multiple users access the same data at the same time then, the integrity of the data is lost.
- If there is a system failure then, business transaction would partially update the data.
- Thus, transactions ensure data integrity of the data stored in the database.

Transaction and Data Integrity 2-3

- Transactions on the database are managed by the transaction management mechanism provided by the DBMS.
- Enterprise applications require transaction management in the middle layer of the application.

The transaction management mechanism in the middle layer is provided by the Java Transaction API (JTA).

Transaction and Data Integrity 3-3

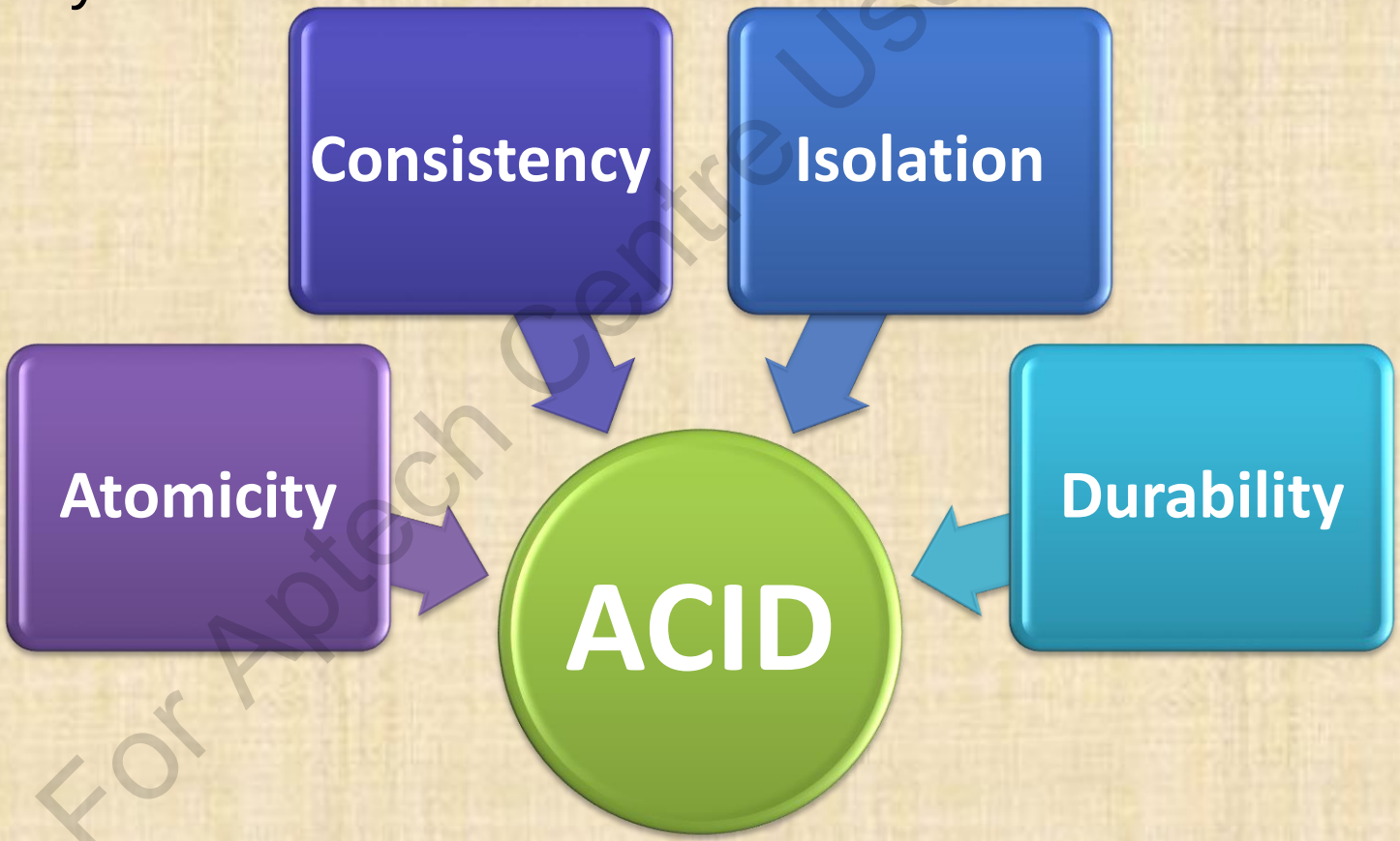
- Following figure illustrates the concept of transaction using the example of an e-commerce portal:

Transaction Scope

- Transaction scope extends from the begin operation of the transaction and ends at the commit.
- All operations between the begin and commit have to successfully execute.
- Transaction is rolled back when any one of the operations in transaction fail.
  - Rollback is the process of restoring the system to previous consistent state.

Transaction Properties 1-2

- Transaction properties are referred to as ACID properties and they are:



Transaction Properties 2-2

Atomicity

- Transaction should execute as a single unit.
- Either all of the operations execute or none of them execute.

Consistency

- All the validation constraints should hold good after the transaction execution.

Isolation

- All transactions should independently execute.

Durability

- Changes made by the committed transaction should be persistent.

Transaction Demarcation

- It is the process of determining where the transaction begins and ends.
- Transaction demarcation can be defined as:
  - Declarative
  - Programmatic
- Declarative demarcation is done through annotations and deployment descriptor.
- Programmatic demarcation is done by the developer by explicitly beginning and ending the transaction.
- Transaction demarcation ends with a commit or rollback operation.
  - The commit request directs all the participating components to store the effects of the transaction operations permanently.
  - The rollback request makes the components to undo the effects of all transaction operations.

Programmatic Vs. Declarative Demarcation

Programmatic Demarcation	Declarative Demarcation
• Used for small number of transactional operations • Programmer determines where the transaction begins and ends • Referred as **Bean-Managed Transactions (BMT)**	• Used for numerous transactional operations • Container places entire method into a transaction • Referred as **Container-Managed Transactions (CMT)**

Transaction Processing 1-3

Following components are involved in transaction processing:

- **Application Components** – EJB business methods which invoke transactions.
- **Resource Managers** – Components which manage the persistent data storage usually drivers.
- **Transaction Managers** – Core component that creates and maintains transactions.

Transaction Processing 2-3

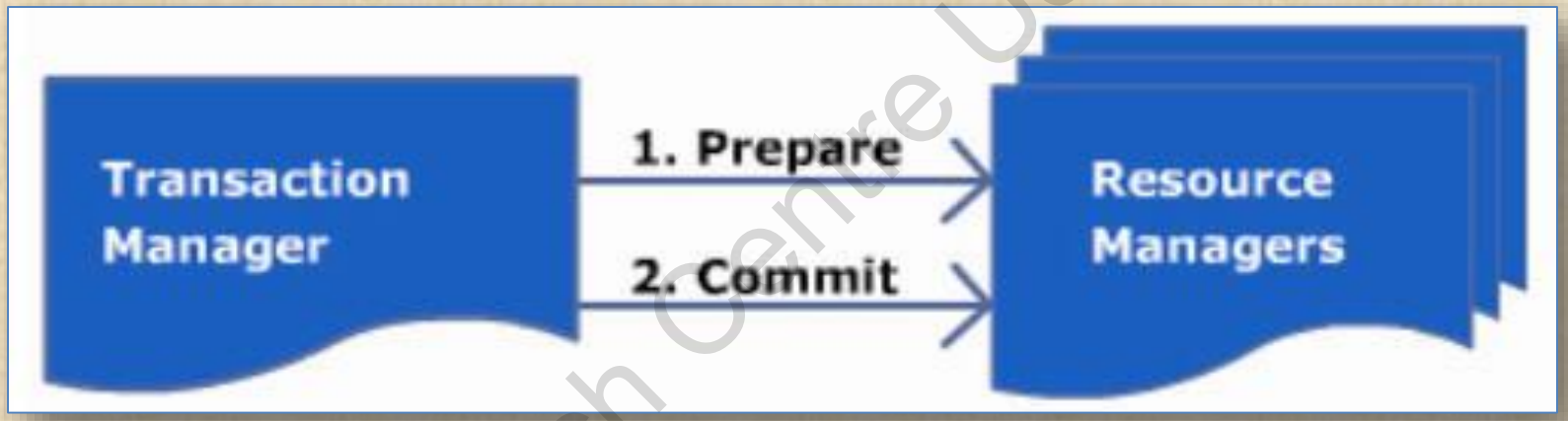
- When the transaction manager receives a request from the application component to commit a transaction, the transaction manager performs the activity in two phases.

Requests all the involved resource managers to be ready for the transaction, updates all the resources and makes an entry in the transaction log.

The transaction manager sends a commit request to all the resource managers.

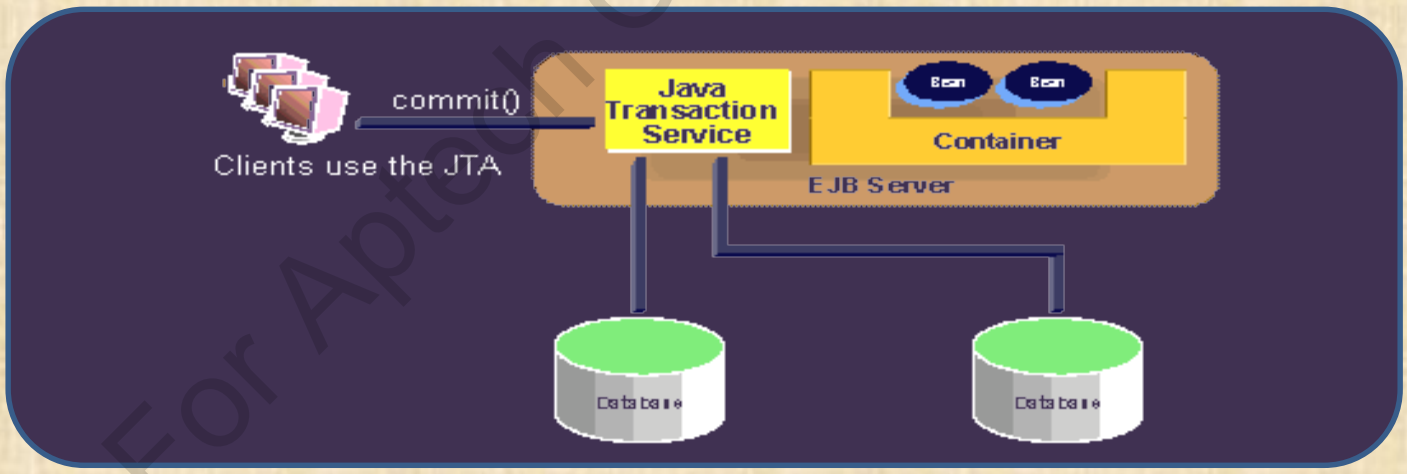
Transaction Processing 3-3

- Following figure shows the execution of two-phase commit protocol:



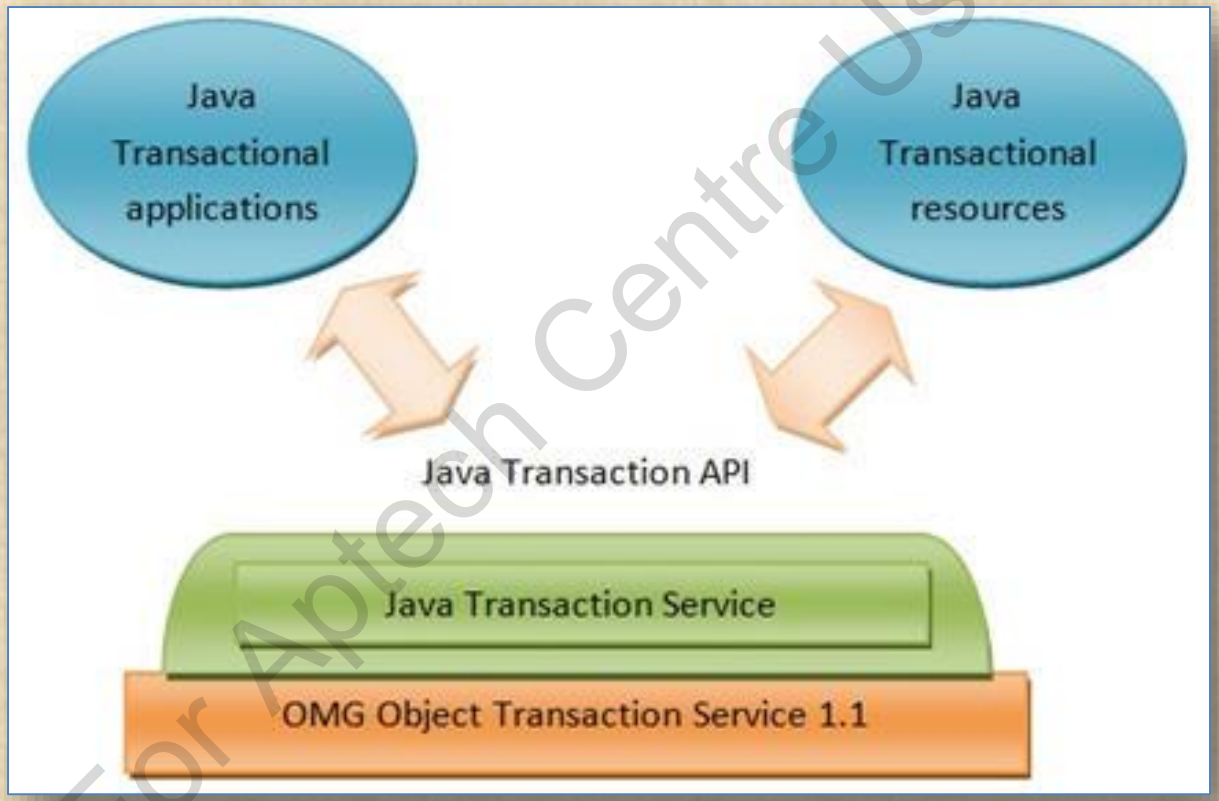
Transaction API 1-3

- Java transactions are managed through:
  - Java Transaction Service (JTS)
  - Java Transaction API (JTA)
- JTA is an interface between a transaction manager and components involved in distributed transaction system.
- JTS is a specification for the implementation of a transaction manager in the application.



Transaction API 2-3

- Following figure demonstrates how JTS and JTA are interrelated:



Transaction API 3-3

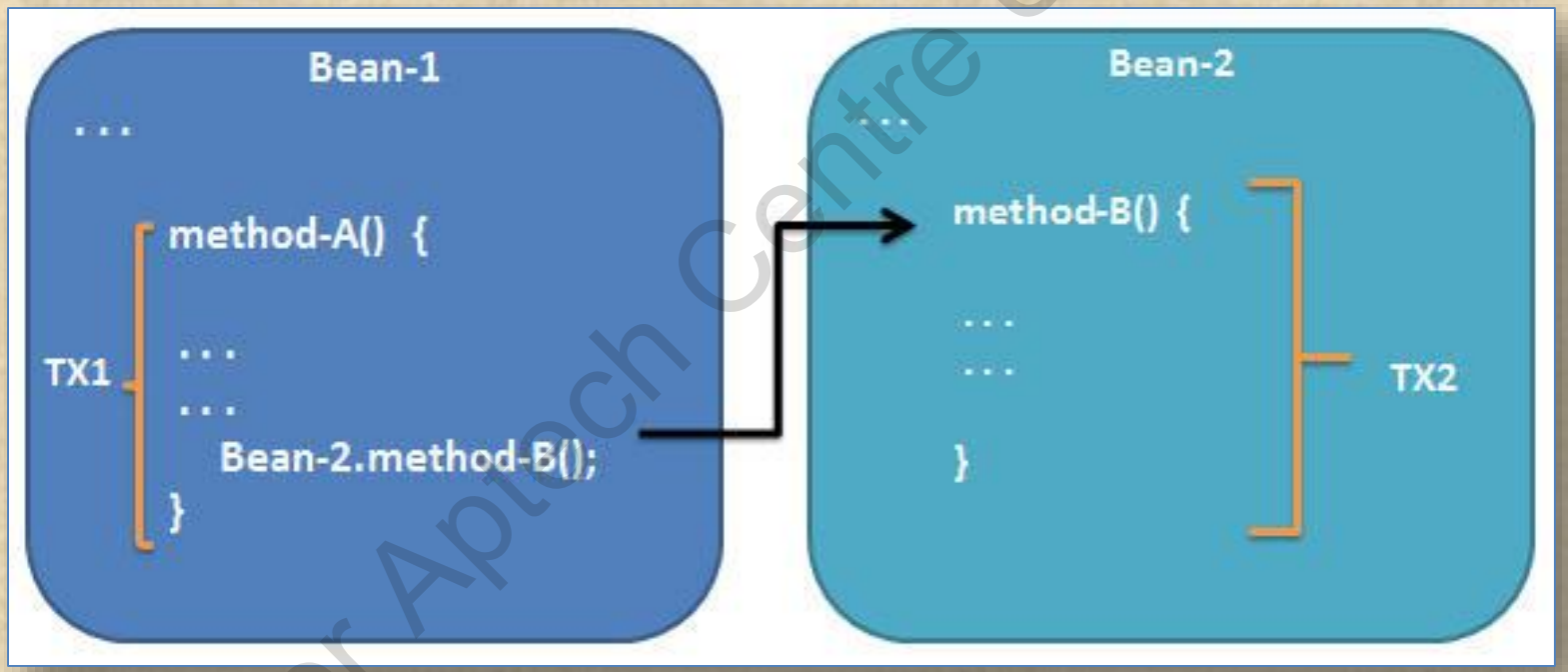
- JTA API comprises the following three sets of interfaces:
  - **`javax.Transaction.xa.XAResource`**
    - Is used by JTA to communicate with X/Open XA-enabled resource managers.
  - **`javax.transaction.TransactionManager`**
    - Is used by JTA to communicate with the application servers.
  - **`javax.transaction.UserTransaction`**
    - Is used by the enterprise beans and other Java classes to work with the EJB transactions.

Container-Managed Transaction

- Implements declarative transaction demarcation.
- Container is responsible for starting and managing the transaction.
- Container is the demarcation of the transaction.
- Can work with a Message-driven bean or a session bean.
- Each transaction is associated with a bean method.
- Multiple transactions or nested transactions are not allowed in bean methods.
- Following methods are not used in container-managed transactions:
  - **`commit`**, **`setAutoCommit`**, and **`rollback`** of **`java.sql.Connection.`**
  - **`getUserTransaction`**() method of **`javax.ejb.EJBContext.`**
  - All methods of **`javax.transaction.UserTransaction.`**

Transaction Attributes 1-5

- A transaction attribute controls the scope of a transaction.
- Following figure demonstrates the transaction scope:



Transaction Attributes 2-5

- A transaction attribute can have any one of the following values:

Required	RequiresNew	Mandatory
Never	Supports	Not Supported

Transaction Attributes 3-5

- Following table shows the list of transaction attributes permitted according to the EJB type:

Transaction Attribute	Stateless Session Bean	Stateful Session Bean	Entity Bean	Message-Driven Bean
`Not Supported`	Yes	Yes	Yes	Yes
`Required`	Yes	Yes	Yes	No
`Supports`	Yes	Yes	Yes	No
`RequiresNew`	Yes	No	No	No
`Mandatory`	Yes	No	No	Yes
`Never`	Yes	No	No	No

Transaction Attributes 4-5

- A transaction attribute can be set through the annotation `@javax.ejb.TransactionAttribute.`
- A transaction attribute is defined for entire bean class.
- The `@TransactionAttribute` annotation accepts the constant values provided by the `@javax.ejb.TransactionAttributeType` constants.

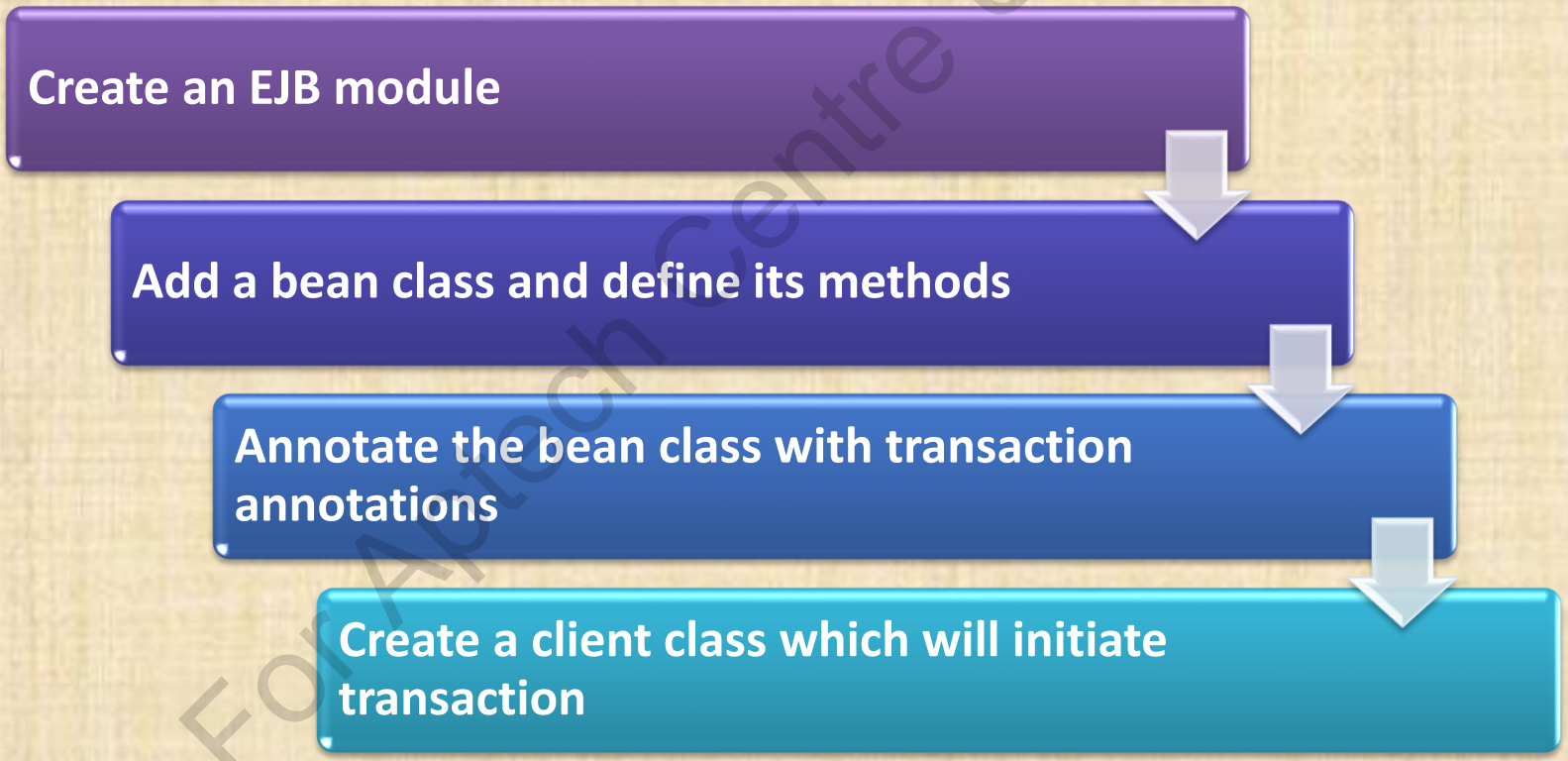
Transaction Attributes 5-5

- Following table shows different **`TransactionAttributeType`** constants:

Transaction Attribute	Constant
Not Supported	`TransactionAttributeType.NOT_SUPPORTED`
Required	`TransactionAttributeType.REQUIRED`
Supports	`TransactionAttributeType.SUPPORTS`
RequiresNew	`TransactionAttributeType.REQUIRES_NEW`
Mandatory	`TransactionAttributeType.MANDATORY`
Never	`TransactionAttributeType.NEVER`

Implementing CMT 1-7

- CMTs are initiated, managed, and closed by application container. Following are the steps in implementing a container managed transaction:



Implementing CMT 2-7

- Following code snippet demonstrates a container managed transaction:

```
. . .
@TransactionManagement(TransactionManagementType.CONTAINER)
@Stateless
@LocalBean
public class CMT {
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public String add(Connection c){
String message;
try{
        c.createStatement();
        CachedRowSet st = new CachedRowSetImpl();
        String query = "insert into Customers
                                values(1007,'Martha')";
```


Implementing CMT 3-7

```
message=”Row Inserted”;
}catch(SQLException e){ message=e.toString(); }
return message;
}

@TransactionAttribute(TransactionAttributeType.REQUIRED)
public CachedRowSet display(Connection c) throws
                                        SQLException{
      Statement stmt = c.createStatement();
      CachedRowSet st = new CachedRowSetImpl();
      String quer = “select * from Customers”;
      st.setCommand(quer);
      st.execute(c);
return st;
}
}
```


Implementing CMT 4-7

- The code shows a bean class with two bean methods defined.
- The annotation in the code defines the transaction management type as Container.
- This implies that the transaction management is taken care of by the application container.
- Annotations define the transaction attributes for each of the bean methods.
- The bean method `add()` takes a Connection object as a parameter.
- The bean method `display()` also connects to the database through a Connection object and displays all the rows present in the Customers table.

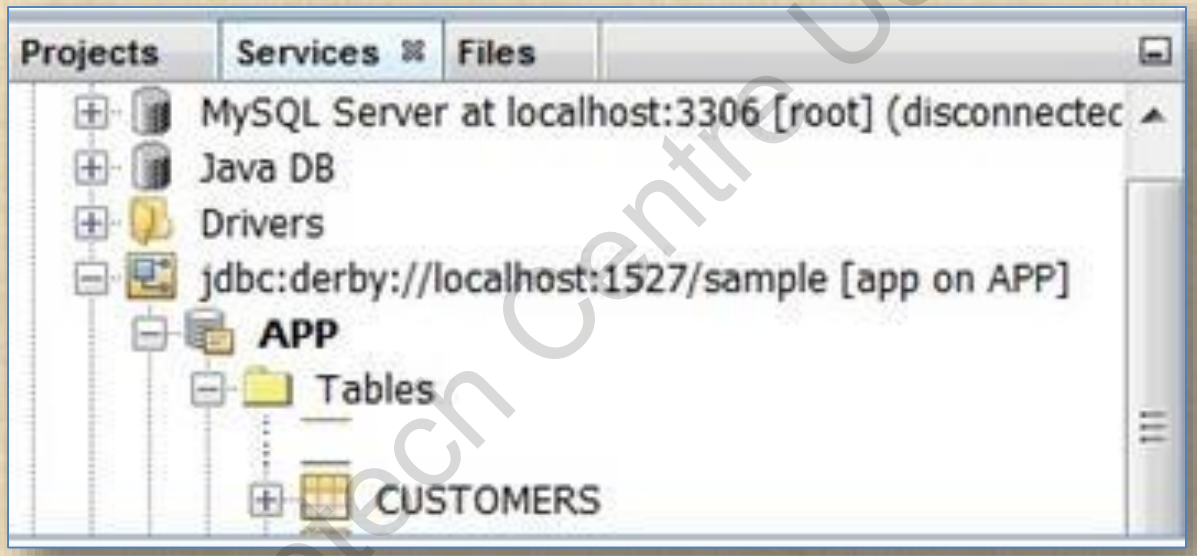
Implementing CMT 5-7

- Following code snippet shows the client code which accesses bean methods:

```
. . .
public class CallCMT {
public static void main(String[] args)
try {
  String driver = "org.apache.derby.jdbc.ClientDriver";
  String url = "jdbc:derby://localhost:1527/sample";
  String username = "app";
  String password = "app";
  Class.forName(driver).newInstance();
  Connection conn = DriverManager.getConnection(url,
                                  username, password);
   CMT c = new CMT();
   c.add(conn);
   c.display(conn);
}catch(){ . . . }
}
```


Implementing CMT 6-7

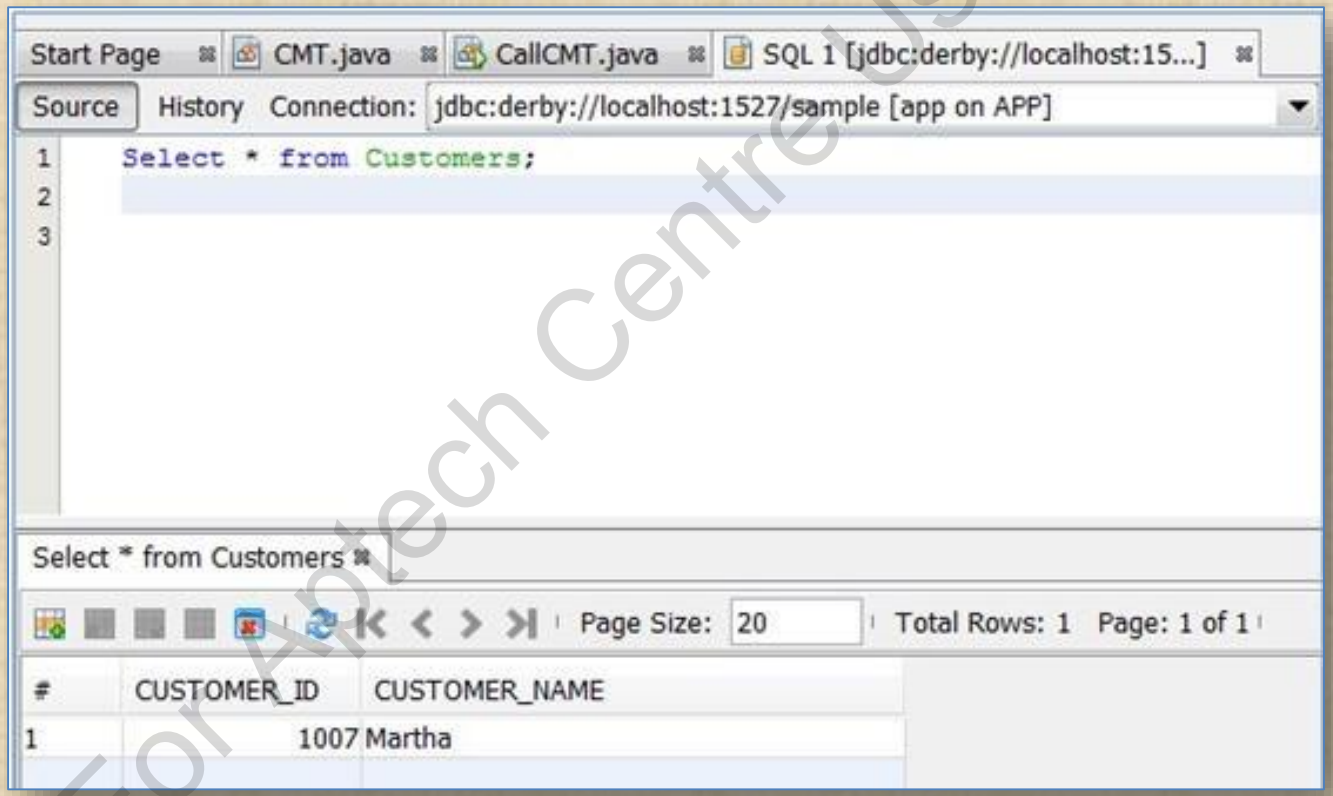
- Following figure demonstrates how to check the state of the database after the transaction executes:



  - Choose the '**Services**' tab in the NetBeans IDE.

Implementing CMT 7-7

- Following figure demonstrates how to execute a command on the database to check the transaction:



Rollback Transaction

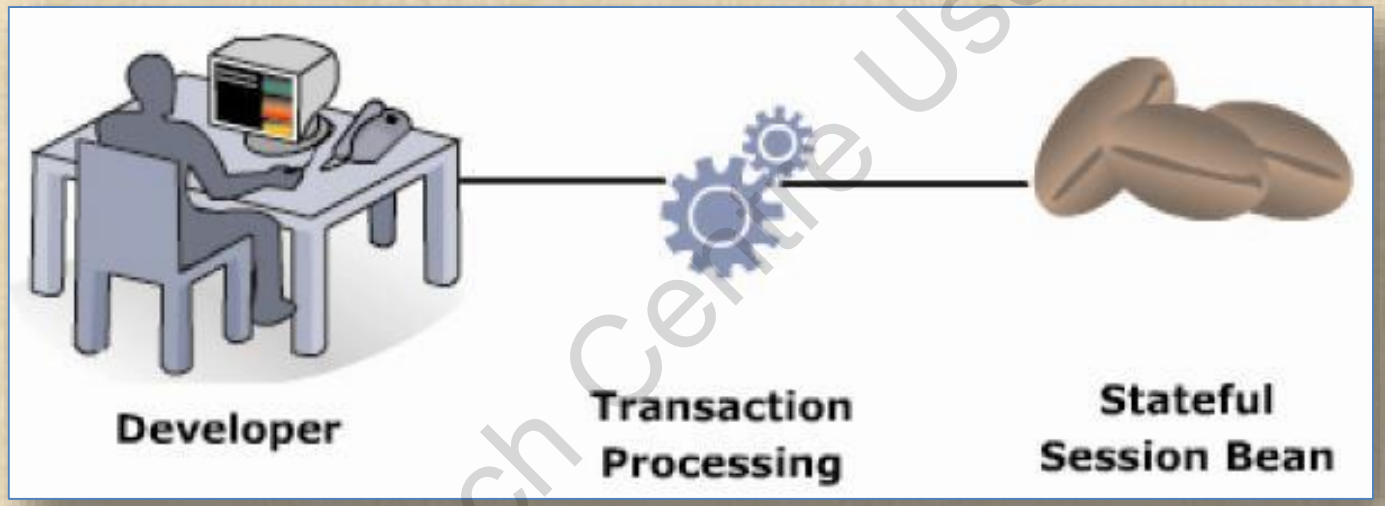
- A transaction committed by CMT can be rolled back in the following ways:
  - If a system exception is raised, then the container automatically rolls back the transaction.
  - If an application exception is raised, then the container will not rollback it automatically. The EJB must invoke the method `setRollbackOnly()` of the `EJBContext` interface to notify the container to roll back the transaction.

Developer Responsibilities 1-2

- The developer has to set the transaction attributes in the deployment descriptor.
- The developer should handle the exceptions appropriately.
- The developer should implement **`javax.ejb.SessionSynchronization`** interface in case of Stateful Session bean.
- The methods used for handling transactions in Stateful Session beans are:
  - **`afterBegin()`**
  - **`beforeCompletion()`**
  - **`afterCompletion()`**

Developer Responsibilities 2-2

- Following figure shows the developer's responsibilities with respect to Stateful Session beans:



- During the creation of a Stateful Session bean, a developer can implement `javax.ejb.SessionSynchronization` interface, which provides methods for persisting the state at various stages.

Synchronized Stateful Session Bean

- Following code snippet shows an example of a synchronized Stateful Session bean:

```
. . .
@Stateful
public class MyStatefulEJB implements
SessionSynchronization {
protected int total = 0; // actual state of the bean
// value inside transaction, not yet committed
protected int newtotal = 0;
protected String clientUser = null;
protected javax.ejb.SessionContext sessionContext = null;
public void ejbCreate(String user)
{
  total = 0;
  newtotal = total;
  clientUser = user;
}
```


Container Responsibilities

- Based on the transaction attribute, the container can perform one of the following actions before performing the transaction:
  - Continue the current transaction.
  - Suspend the current transaction and run the method without a transaction.
  - Suspend the current transaction and begin a new one.
  - Refuse to execute the method at all.
- At the end of a method call, the container will attempt to commit or rollback any transaction.

Bean-Managed Transactions 1-2

- Bean-managed transactions are useful when a method has to associate with more than one transaction.
- Bean-managed transactions provide better control on the application execution.
- Developer is responsible for starting the transaction and then committing or rolling back the transactions to end it.
- Bean-managed transactions are implemented through Java Database Connectivity (JDBC) or JTA.
- Bean-managed transaction is an instance of **`UserTransaction`**.

Bean-Managed Transactions 2-2

- Following are various methods of bean managed transactions:

Implementing BMT 1-2

- Following code snippet shows a servlet code which explicitly initiates a transaction:

```
public class BMTServlet extends HttpServlet {
@EJB
Private BMT b;
@Resource
javax.transaction.UserTransaction utx;
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try{
/* TODO output your page here. You may use following sample code.
*/
  out.println("<!DOCTYPE html>");
  out.println("<html>");
  out.println("<head>");
```


Implementing BMT 2-2

```
   out.println("<title>Servlet BMTServlet</title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<h1>CUSTOMER DETAILS</h1>");
  String driver = "org.apache.derby.jdbc.ClientDriver";
  String url = "jdbc:derby://localhost:1527/sample";
  String username = "app";
  String password = "app";
  Class.forName(driver).newInstance();
  Connection conn = DriverManager.getConnection(url, username, password);
  System.out.println("Connection Done");
  utx.begin();
  out.println(BMT.add(conn));
 .. .
  utx.commit();
  out.println("</body>");
  out.println("</html>");
}
catch(Exception ex){
  out.println("Error: Other exception - " +ex);
  }
 }
}
. . .
```


Client-Managed Transaction Demarcation 1-3

- Apart from CMT and BMT, EJB also enables client managed transaction demarcation.
- The client can use **`UserTransaction`** class to initiate and manage transactions.

Client-Managed Transaction Demarcation 2-3

- Following figure shows client managed transaction demarcation:

Client-Managed Transaction Demarcation 3-3

- Following code snippet shows JNDI lookup of transaction from the client:

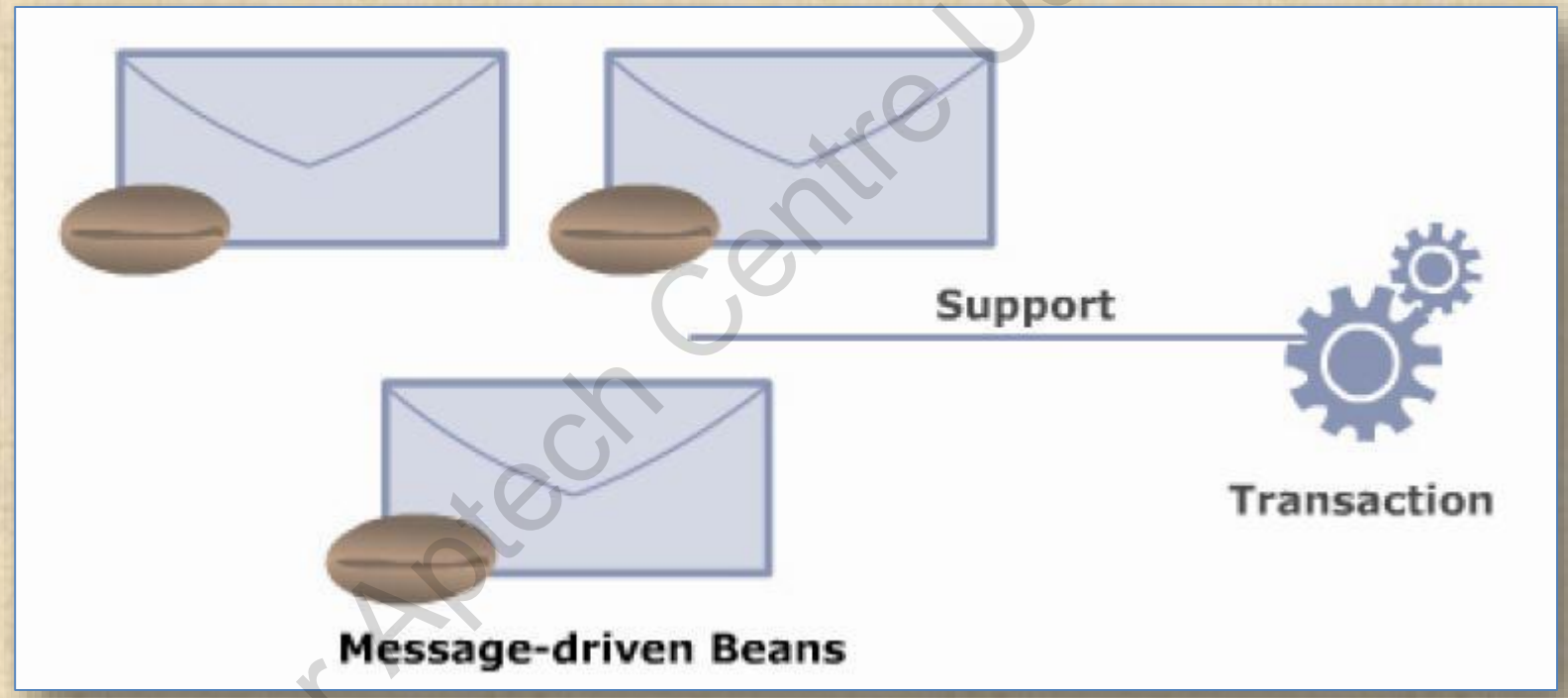
```
. . .
Context c = new InitialContext();
UserTransaction ut =
(UserTransaction)c.lookup("java:comp/
UserTransaction");
. . .
```


Applying Transaction to Messaging 1-2

- Transactions may also include sending and receiving messages.
- Java application uses Message-driven beans and JMS messages to send and receive messages.
- Message-driven beans can work with both CMT and BMT.
- Message-driven beans support only two transaction attributes – **`Required`** and **`NotSupported`**.

Applying Transaction to Messaging 2-2

- Following figure shows using transactions with Message-driven beans:

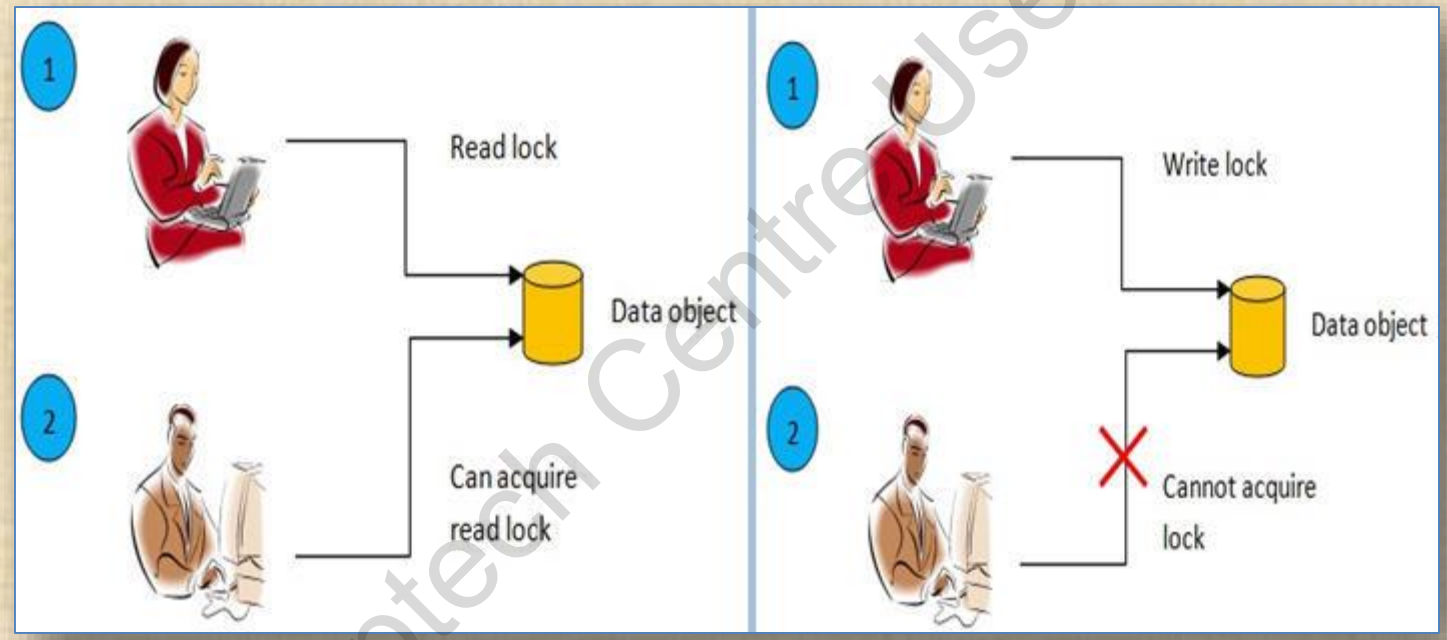


Database Locking and Isolation 1-2

- When multiple transactions are simultaneously executing, they are scheduled on the database.
- To ensure consistency, the database provider should ensure transaction serializability.
- Locking mechanism is used to provide transaction serializability.
- There are two types of locks which can be acquired by the transaction:
  - Shared or read lock
  - Exclusive or write lock

Database Locking and Isolation 2-2

- Following figure shows different types of locks:



Summary

- Transactions are a set of operations which has to execute as a single unit.
- Transactions should have properties such as atomicity, consistency, isolation, and durability.
- Transactions are supported in enterprise applications through Java Transaction Services and Java Transaction API.
- Transactions can be managed both declaratively and programmatically.
- Declarative transaction management makes use of annotations, deployment descriptors, and transaction attributes. They are also known as container-managed transactions.
- Explicit transaction management makes use of the JTA and its interfaces for transaction management, such transactions are also known as bean-managed transactions.
- Transactions can execute along with Stateful, Stateless, and Message-driven beans.
- Messages in transactions can be handled through both container-managed transactions and bean-managed transactions.