Universidad de los Andes

Transaccionalidad
Java Transaction API – JTA 1.1

Por: Rafael Gustavo Meneses M.Sc.

Departamento de Ingeniería de Sistemas y Computación
Especialización en Construcción de Software
Bogotá, COLOMBIA

Universidad de los Andes

Agenda

- Introducción
- Modelos Transaccionales
- Java Transaction API (JTA)
- Tipos de Transacciones
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- Referencias

Universidad de los Andes

Agenda

- **Introducción**
- Modelos Transaccionales
- Java Transaction API (JTA)
- Tipos de Transacciones
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- Referencias

Transacciones

- Los datos son cruciales para el negocio, y deben ser precisos independientemente de:
  - Las operaciones que se realicen con/sobre ellos
  - El número de aplicaciones que los acceden concurrentemente

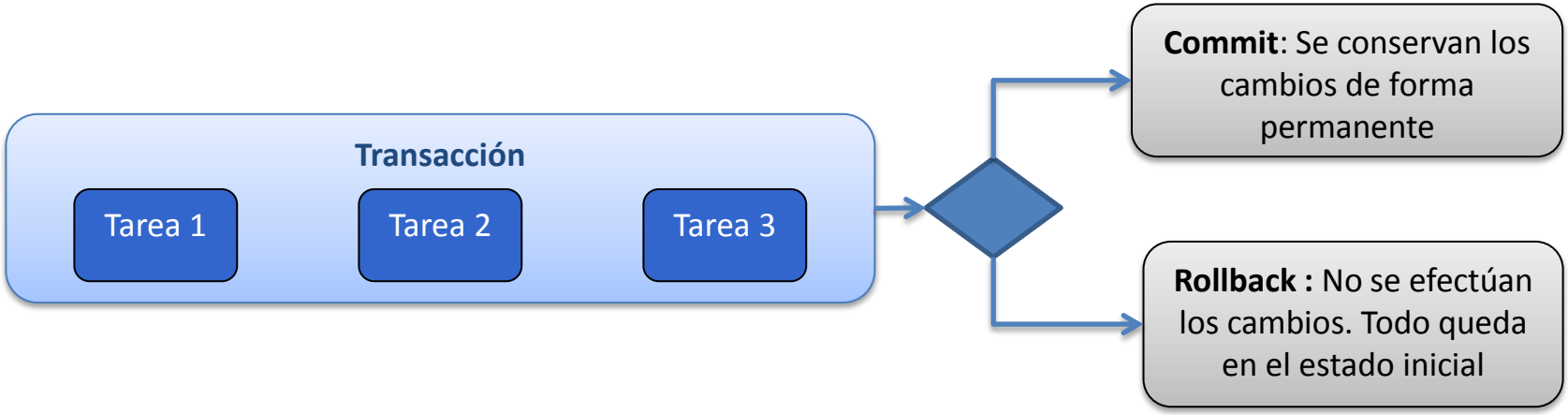
- Una *transacción* es utilizada para asegurar que los datos permanezcan en un *estado consistente*

Transacciones (2)

- Una transacción:
  - *"…es un grupo de tareas que deben ser procesadas en una sola unidad."* [4]
  - *"…es una secuencia de pasos que adiciona, actualiza o borra datos persistentes."* [5]
  - *"…representa un grupo lógico de operaciones que debe ser ejecutado como una única unidad, también conocida como unidad de trabajo."* [3]

Transacciones (3)

- En una transacción:
  - Cada operación que la compone debe tener éxito para que ella tenga éxito → Se confirma la transacción (**Committed**)
  - Si una de las operaciones que la compone falla, ella falla también → La transacción se deshace (**Rolled Back**)



Propiedades ACID

- Conjunto de propiedades que define una transacción confiable:
  - Atomicidad *(**A**tomicity)*
  - Consistencia *(**C**onsistency)*
  - Aislamiento *(**I**solation)*
  - Durabilidad *(**D**urability)*

Atomicidad → *Atomicity*

- Garantiza que todos los pasos de una transacción se hacen o no se hacen
- En caso de presentarse fallas en la ejecución, los datos no cambian
- En caso de ejecutarse exitosamente, se mantienen los cambios

Consistencia → *Consistency*

- Al final de la transacción, los datos se mantienen en un estado consistente
- Se ejecutan aquellas operaciones que no van a romper la reglas de negocio y directrices de integridad de la base de datos

Aislamiento → *Isolation*

- El estado intermedio de una transacción no es visible para otras transacciones antes de hacer ***commit***
- Asegura que dos transacciones sobre la misma información, son independientes y no generan ningún error

Durabilidad → *Durability*

- Se asegura que una vez realizadas las operaciones de la transacción, los cambios hechos a los datos persistirán, no se podrán deshacer y quedarán visibles a otras aplicaciones

Universidad de los Andes

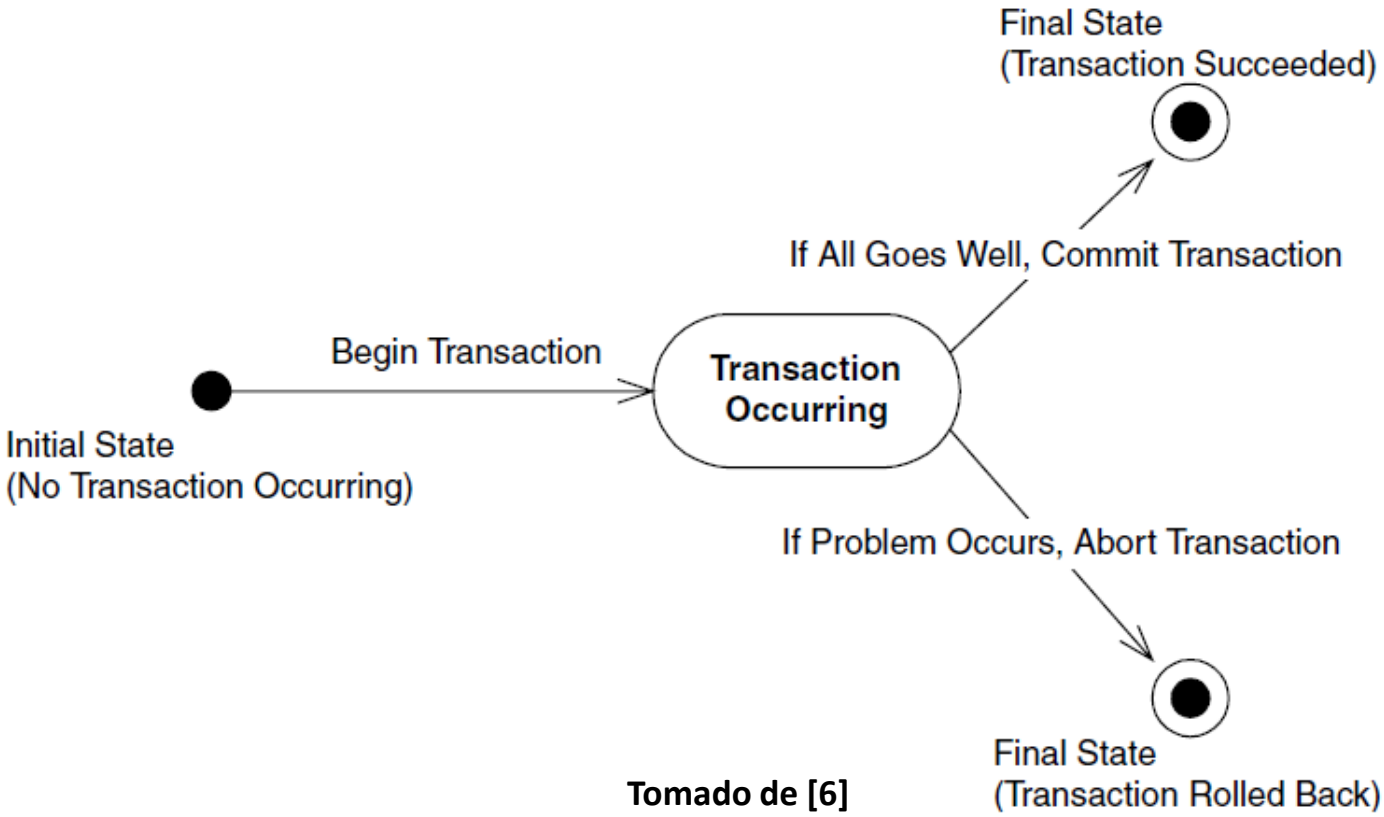
Agenda

- Introducción
- **Modelos Transaccionales**
- Java Transaction API (JTA)
- Tipos de Transacciones
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- Referencias

Modelos Transaccionales

Transacciones Planas → *Flat Transactions*

- Es una serie de operaciones que se realizan de forma atómica como una sola *unidad de trabajo*



Tomado de [6]

Transacciones Anidadas → *Nested Transactions*

- Transacciones que permiten integrar unidades atómicas de trabajo (*subtransacción*) en otras unidades de trabajo *(padre)*
- Una *subtransacción* puede hacer *roll back* sin forzar el *roll back* de su transacción *padre* → La unidad *padre* puede intentar volver a ejecutar la subtransacción
- El efecto de una *subtransacción* es provisional sobre el *commit/roll back* de su(s) transacción(es) *padre* → Si el *padre* falla, toda la transacción hace *roll back*, así la *subtransacción* haya hecho *commit*

Universidad de los Andes

Agenda

- Introducción
- Modelos Transaccionales
- **Java Transaction API (JTA)**
- Tipos de Transacciones
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- Referencias

Administración Interna de una transacción

- ***Resource:*** Almacenamiento persistente donde se pueden leer o escribir datos → Base de datos, cola de mensajes
- ***Resource Manager:*** Administra las operaciones de los recursos y los registra en el *Transaction Manager* → Driver de base de datos, recurso JMS, un conector Java
- ***Transaction Manager:***
  - Coordina las operaciones transaccionales
  - Crea las transacciones en la aplicación
  - Notifica al *Resource Manager* como participante en una transacción (*enlistment*)
  - Conduce el commit/roll back en el *Resource Manager*

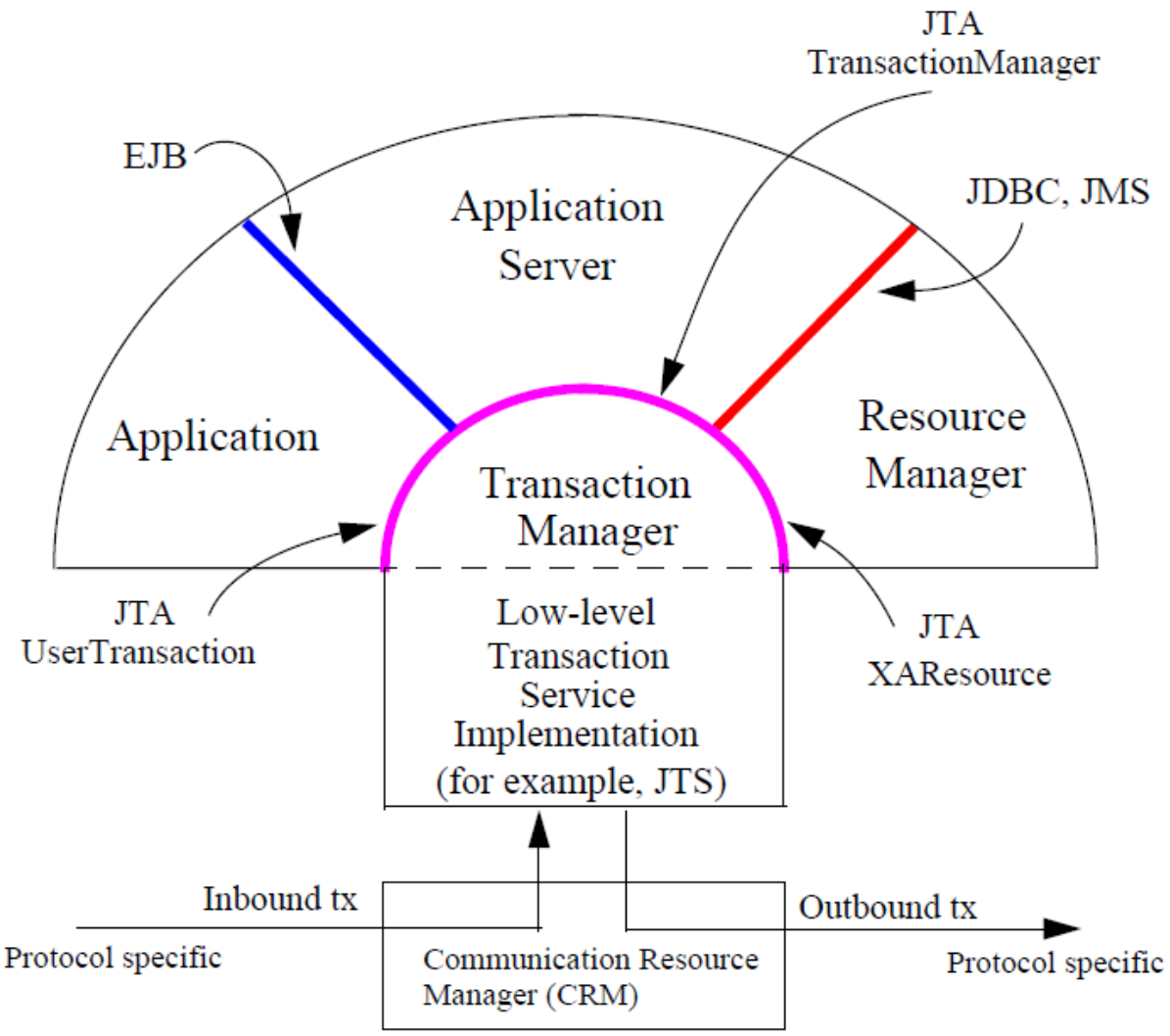
Universidad de los Andes

Java Transaction API (JTA)

- JSR (*Java Specification Request*) 907
- *"JTA defines a set of interfaces for the application to demarcate transactions' boundaries, and it also defines APIs to deal with the transaction manager."* [3]
- Principales Interfaces → Paquete ***javax.transaction***
  - ***UserTransaction***. Métodos para controlar programáticamente las fronteras de la transacción
  - ***TransactionManager***. Permite al contenedor EJB demarcar las fronteras de la transacción en nombre del EJB
  - ***Transaction***. Representa una transacción
  - ***XAResource***. Es un mapeo Java del estándar *XA* del *X/Open group*. Define el contrato entre un *Resource Manager* y un *Transaction Manager* en un ambiente de procesamiento distribuido de transacciones

Java Transaction API (JTA)

Especificación JTA



Tomado de [7]

Universidad de los Andes

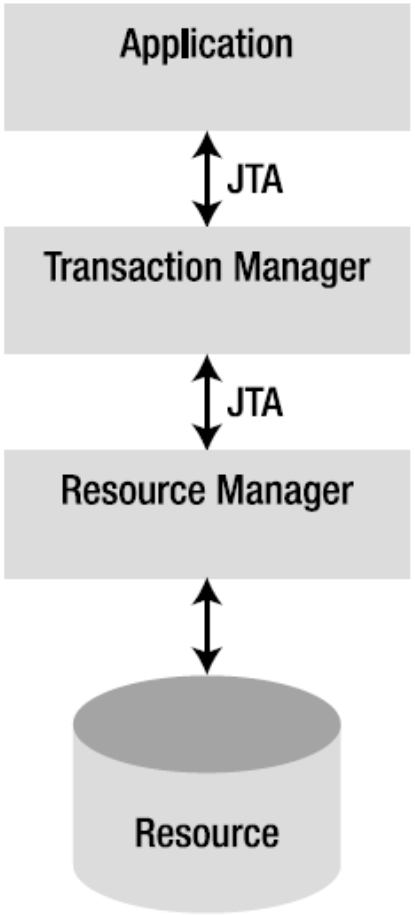
Agenda

- Introducción
- Modelos Transaccionales
- Java Transaction API (JTA)
- **Tipos de Transacciones**
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- Referencias

Tipos de Transacciones

Transacciones Locales

- Transacciones que involucran un único recurso

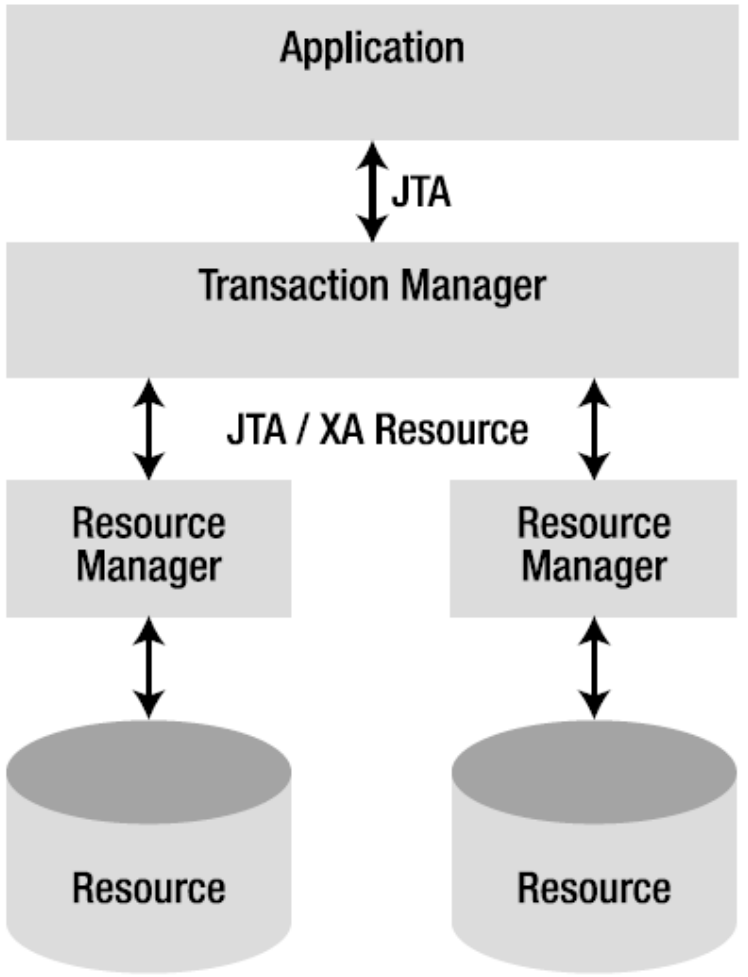


Tomado de [3]

Tipos de Transacciones

Transacciones Distribuidas

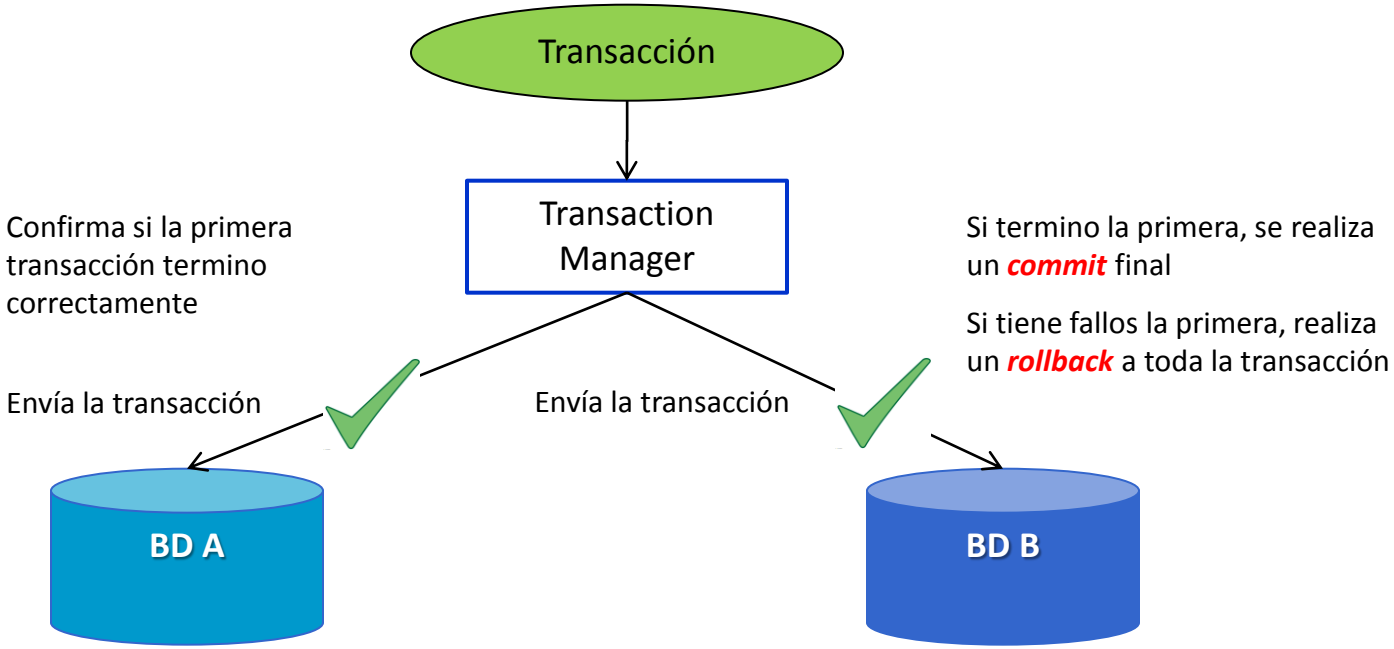
- Transacciones que involucran más de un recurso
- Necesitamos la administración de transacciones entre *recursos de diferente tipo* y/o *recursos distribuidos en la red*



Tomado de [3]

Transacciones Distribuidas (2)

- El administrador se encarga de iniciar, hacer commit o rollback a las transacciones
- La administración se realiza con el protocolo de dos fases (*two-phase commit*)



Comparación tipos de transacciones

Propiedad	Local	Distribuida
Número de recursos	Uno	Múltiple
Coordinador	Resource Manager	Transaction Manager
Protocolo Commit	Single-Phase	Two-Phase

Agenda

- Introducción
- Modelos Transaccionales
- Java Transaction API (JTA)
- Tipos de Transacciones
- **Soporte de Transacciones en EJB**
  - **Container-Managed Transactions (CMT)**
  - Bean-Managed Transactions (BMT)
- Referencias

Soporte de Transacciones en EJB

Container-Managed Transactions (CMT)

- El contenedor inicia la transacción, hace *commit* y *roll back*
- El inicio y fin de una transacción son definidos por los métodos del EJB
- Proceso:
  - El contenedor inicia la transacción JTA antes de iniciar la ejecución del método
  - Invoca el método
  - Dependiendo de la ejecución (exitosa ó fallida), hace *commit* o *roll back* a la transacción que administra

Soporte de Transacciones en EJB

Container-Managed Transactions (CMT)

- El contenedor puede administrar las transacciones a través de:
  - Anotaciones
  - Descriptores de despliegue

- En el siguiente ejemplo se inicia una transacción, se validan los datos de un cliente, si hay algún error se hace un roll back de la transacción

Container-Managed Transactions (CMT)

Ejemplo

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class OrderManagerBean {
   @Resource
   private SessionContext context;
   ...
   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   public void placeSnagItOrder(Item item, Customer customer){
      try {
         if (!bidsExisting(item)){
            validateCredit(customer);
            chargeCustomer(customer, item);
            removeItemFromBidding(item);
         }
      } catch (CreditValidationException cve) {
         context.setRollbackOnly();
      } catch (CreditProcessingException cpe){
         context.setRollbackOnly();
      } catch (DatabaseException de) {
         context.setRollbackOnly();
      }
   }
}
```

Utiliza CMT si no se define por defecto

Inyecta contexto EJB

Define transacción para el método

Rollback

Tomado de [4]

Container-Managed Transactions (CMT)

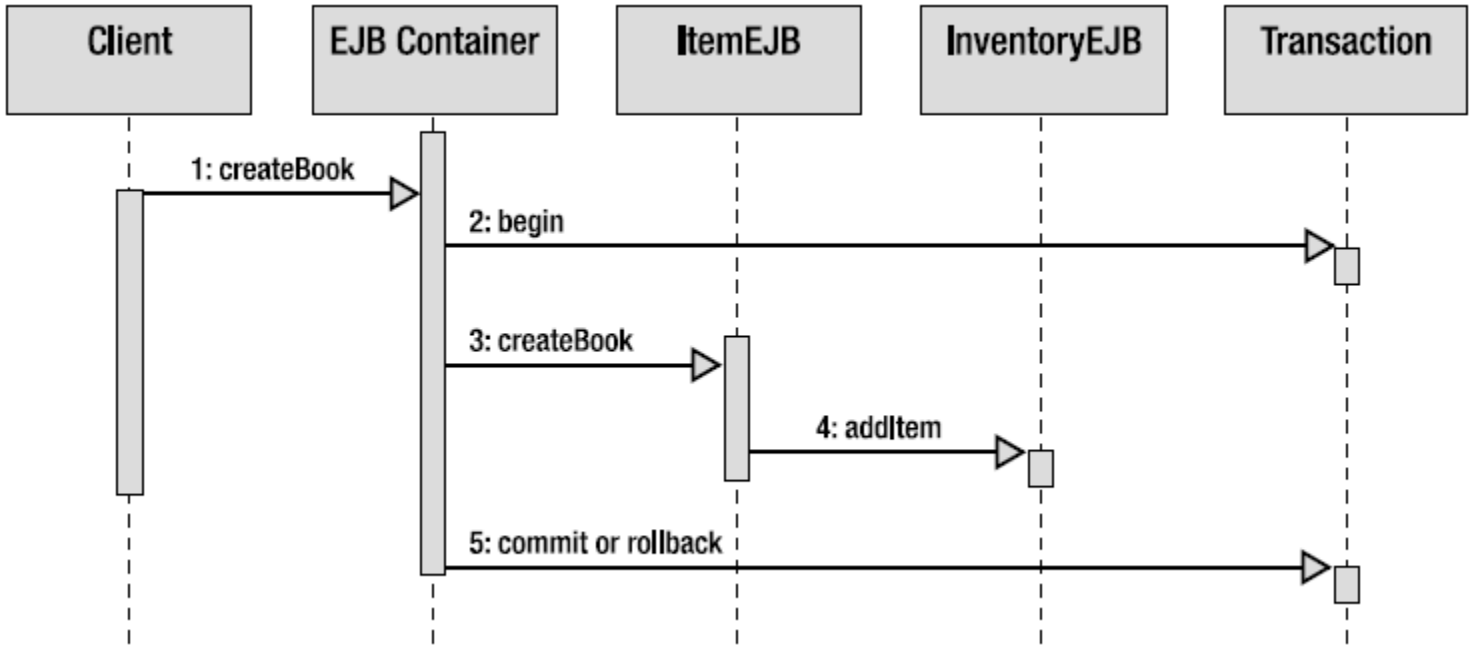
Ejemplo 2

```
@Stateless
public class ItemEJB {

    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;
    @EJB
    private InventoryEJB inventory;

    public List<Book> findBooks() {
        Query query = em.createNamedQuery("findAllBooks");
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        inventory.addItem(book);
        return book;
    }
}
```

Tomado de [3]

Container-Managed Transactions (CMT)

Ejemplo 2 (2)



Tomado de [3]

Container-Managed Transactions (CMT)

Anotaciones

- `@TransactionManagement` Especifica el tipo de administración de las transacciones
  - *TransactionManagementType.BEAN*
  - *TransactionManagementType.CONTAINER*
- `@TransactionAttribute` Define la manera como el contenedor debe administrar la transacción
  Puede tener asociado uno de los siguientes valores:
  - *REQUIRED*
  - *REQUIRES_NEW*
  - *SUPPORTS*
  - *MANDATORY*
  - *NOT_SUPPORTED*
  - *NEVER*

Container-Managed Transactions (CMT)

REQUIRED

- Atributo por defecto
- Indica que el método siempre debe ser invocado en un contexto transaccional
- Si es invocado por un cliente no transaccional el contenedor crea una transacción antes de que el método sea llamado y la termina cuando el método retorne
- Si es llamado en un contexto transaccional, el método se une a la transacción existente
- Soportado por MDBs

Container-Managed Transactions (CMT)

REQUIRED - Ejemplo

Como está a nivel de clase aplica a todos los métodos de la misma

```
@Stateless
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class AuditServiceBean implements AuditService
{
    @PersistenceContext(unitName="BankService")
    private EntityManager em;

    public void addAuditMessage (int auditId,
                                 String message) {
        Audit audit = new Audit();
        audit.setId(auditId);
        audit.setMessage(message);
        em.persist(audit);
    }
}
```

Tomado de [5]

Container-Managed Transactions (CMT)

REQUIRES_NEW

- El contenedor siempre debe crear una nueva transacción cuando se invoque el método
- Si es invocado en un contexto transaccional, la transacción del cliente se suspende hasta que el método retorne el valor
- Si la transacción falla, NO tiene ningún efecto en la transacción del cliente que lo invoca

Container-Managed Transactions (CMT)

REQUIRES_NEW - Ejemplo

El atributo **REQUIRES_NEW** sólo aplica al método **addAuditMessage**, para los otros métodos aplica el atributo **REQUIRED** que es por defecto

```
@Stateless
public class AuditServiceBean implements AuditService
{
    @PersistenceContext(unitName="BankService")
    private EntityManager em;

    @TransactionAttribute
    (TransactionAttributeType.REQUIRES_NEW)
    public void addAuditMessage (int auditId,
                                 String message) {
        Audit audit = new Audit();
        audit.setId(auditId);
        audit.setMessage(message);
        em.persist(audit);
    }
}
```

Tomado de [5]

Universidad de los Andes

Container-Managed Transactions (CMT)

SUPPORTS

- Hereda el entorno transaccional del cliente que lo invoca
- Si el cliente no es transaccional el método será invocado sin una transacción asociada
- Si el cliente es transaccional el método EJB se unirá a este y no lo suspenderá
- Es utilizado en métodos de solo lectura, por ejemplo cuando se consultan registros de una tabla

Container-Managed Transactions (CMT)

SUPPORTS - Ejemplo

```
@Stateless
public class BankServiceBean implements BankService {
   @PersistenceContext(unitName="BankService")
   private EntityManager em;
   ...

   public Long addCustomer(int custId, String firstName,
                           String lastName) {
      Customer cust = new Customer();
      cust.setId(custId);
      cust.setFirstName(firstName);
      cust.setLastName(lastName);
      em.persist(cust);
      Long count = countQuery(custId);
      return count;
   }

   @TransactionAttribute(TransactionAttributeType.SUPPORTS)
   public Long countQuery() {
      return (Long) em.createQuery("SELECT COUNT(c) from " +
                  "Customer c").getSingleResult();
   }
}
```

Tomado de [5]

Container-Managed Transactions (CMT)

MANDATORY

- El cliente que invoca el método debe tener una transacción asociada antes de llamarlo
- El contenedor nunca debe crear una transacción relacionada al cliente que invoca el método
- Si el método lo invoca un cliente no transaccional, el contenedor lanza la excepción **`EJBTransactionRequiredException`**
- Poco utilizado

Container-Managed Transactions (CMT)

NOT_SUPPORTED

- El método no puede ser invocado por un cliente transaccional
- Si es llamado por un cliente transaccional, se suspende la transacción, se ejecuta el método y luego del retorno del método se reanuda la transacción
- Soportado por MDBs

NEVER

- El método nunca puede ser invocado por un cliente transaccional, si lo invoca se lanza la excepción **`javax.ejb.EJBException`**

Container-Managed Transactions (CMT)

Recordemos…

```
@Stateless
public class ItemEJB {

    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;
    @EJB
    private InventoryEJB inventory;

    public List<Book> findBooks() {
        Query query = em.createNamedQuery("findAllBooks");
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        inventory.addItem(book);
        return book;
    }
}
```

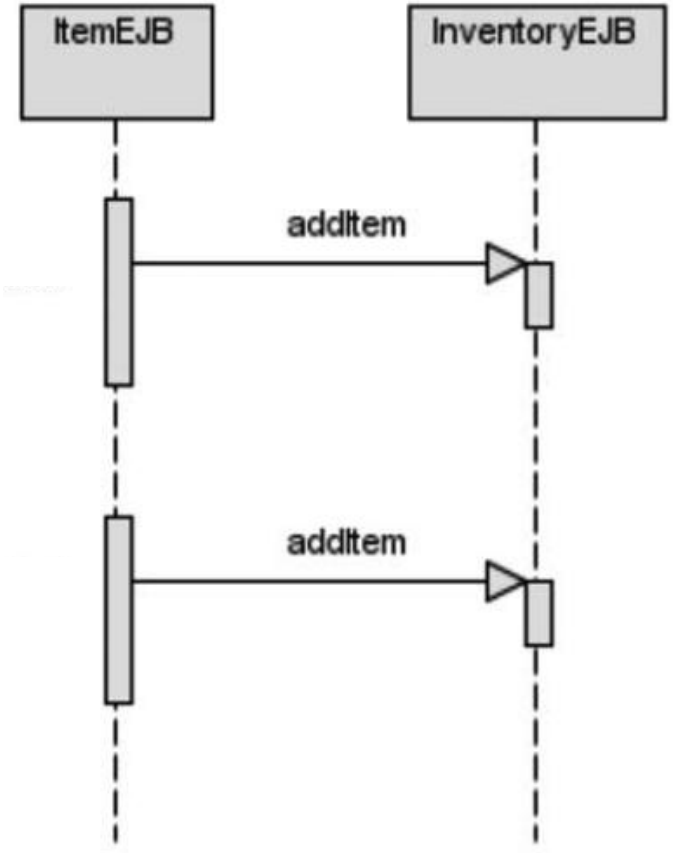
Tomado de [3]

Container-Managed Transactions (CMT)

¿Qué pasa si…?

1 Método ***createBook()*** **NO** está en una transacción

2 Método ***createBook()*** **SI** está en una transacción



Tomado de [3]

Atributo CMT	¿Qué pasa?
REQUIRED	Nueva transacción
REQUIRES_NEW	Nueva transacción
SUPPORTS	No hay transacción
MANDATORY	Excepción
NOT_SUPPORTED	No hay transacción
NEVER	No hay transacción

Atributo CMT	¿Qué pasa?
REQUIRED	Transacción del cliente
REQUIRES_NEW	Nueva transacción
SUPPORTS	Transacción del cliente
MANDATORY	Transacción del cliente
NOT_SUPPORTED	No hay transacción
NEVER	Excepción

Container-Managed Transactions (CMT)

Roll Back

- No se hace de inmediato, se genera un *flag* para que el contenedor lo tenga presente cuando se termine la transacción
- Utiliza métodos de la interfaz **`EJBContext`**
  - **`setRollbackOnly`**
  - **`getRollbackOnly`**
- Solamente se puede utilizar con los atributos *REQUIRED*, *REQUIRES_NEW*, o *MANDATORY*. Si se utiliza con otro atributo se lanza una excepción **`IllegalStateException`**

Manejo de Excepciones

- Definir y manejar las excepciones de la aplicación con `@ApplicationException(rollback=true)`
- Los bloques `try…catch` son reemplazados por `throws`
- Cuando se utiliza `throws`, las excepciones son relanzadas a quien invoca el método
- Con `try…catch`, las excepciones pueden ser capturadas y procesadas o relanzadas
- Si una excepción no se captura debe ser relanzada

Container-Managed Transactions (CMT)

```
public void placeSnagItOrder(Item item, Customer customer) throws
       CreditValidationException, CreditProcessingException,
       DatabaseException
{
   if (!bidsExisting(item)){
      validateCredit(customer);
      chargeCustomer(customer, item)
      removeItemFromBidding(item);
   }
}
```

Declara las excepciones con la clausula ***throws***. todas se relanzan al cliente

```
@ApplicationException( rollback = true )
public class CreditValidationException extends
Exception {
...
@ApplicationException( rollback = true )
public class CreditProcessingException extends
Exception {
...
@ApplicationException( rollback = false )
public class DatabaseException extends
RuntimeException {
...
```

Especifica las excepciones Con ***@ApplicationException***, y serán tratadas como excepciones de aplicación.

El ***rollback=true*** indica que antes de pasar la excepción al cliente se debe hacer rollback

Container-Managed Transactions (CMT)

Manejo de Excepciones (3)

Extiende de	`@ApplicationException`	Transacción marcada para hacer *roll back*
`Exception`	Sin anotación	No
`Exception`	`rollback = true`	Si
`Exception`	`rollback = false`	No
`RuntimeException`	Sin anotación	Si
`RuntimeException`	`rollback = true`	Si
`RuntimeException`	`rollback = false`	No

Container-Managed Transactions (CMT)

Sincronización de Sesión

- Es posible notificar eventos durante el ciclo de vida de una transacción
- Un EJB debe implementar la Interfaz **`javax.ejb.SessionSynchronization`**
- Define los siguientes métodos
  - **`void afterBegin()`** → Invocado después de que el contenedor crea una nueva transacción y antes de la invocación del método
  - **`void beforeCompletion()`** → Invocado después de que el método retorna y antes de que el contenedor finalice la transacción
  - **`void afterCompletion(boolean committed)`** → Invocado después de finalizar la transacción

Universidad de los Andes

Agenda

- Introducción
- Modelos Transaccionales
- Java Transaction API (JTA)
- Tipos de Transacciones
- **Soporte de Transacciones en EJB**
  - Container-Managed Transactions (CMT)
  - **Bean-Managed Transactions (BMT)**
- Referencias

Soporte de Transacciones en EJB

Bean-Managed Transactions (BMT)

- A través de la interfaz `javax.transaction.UserTransaction` del API de JTA se especifica programáticamente:
  - El punto de inicio de la transacción → **`begin()`**
  - Los puntos de *commit* → **`commit()`**
  - Los puntos de *roll back* → **`rollback()`**
  - *Timeout de la transacción* → **`setTransactionTimeout()`**
  - Obtener el estado de la transacción → **`getStatus()`**
- Se maneja código más complejo
- El riesgo de errores de programación aumenta

Bean-Managed Transactions (BMT)

```
@Stateless)
@TransactionManagement(TransactionManagementType.BEAN)
public class OrderManagerBean {
   @Resource
   private UserTransaction userTransaction;

   public void placeSnagItOrder(Item item, Customer customer){
      try {
         userTransaction.begin();
         if (!bidsExisting(item)){
            validateCredit(customer);
            chargeCustomer(customer, item);
            removeItemFromBidding(item);
         }
         userTransaction.commit();
      } catch (CreditValidationException cve) {
         userTransaction.rollback();
      } catch (CreditProcessingException cpe){
         userTransaction.rollback();
      } catch (DatabaseException de) {
         userTransaction.rollback();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Indica que está utilizando **BMT**

Inyecta instancia **UserTransaction**

Inicia la transacción explícitamente

commit de forma explícita

rollback de forma explícita

Tomado de [4]

Bean-Managed Transactions (BMT)

Inyección

- La interfaz *UserTransaction* encapsula la funcionalidad del administrador de transacciones de JEE
- Puede ser inyectada con **`@Resource`** o por medio del JNDI. La localización se puede realizar de la siguiente forma:

```
Context context = new InitialContext();
UserTransaction userTransaction =
    (UserTransaction) context.lookup("java:comp/UserTransaction");
userTransaction.begin();
// Operaciones de la transacción
userTransaction.commit();
```

Bean-Managed Transactions (BMT)

```
@Stateless)
@TransactionManagement(TransactionManagementType.BEAN)
public class OrderManagerBean {
    @Resource
    private UserTransaction userTransaction;

    public void placeSnagItOrder(Item item, Customer customer){
    try {
        userTransaction.begin();
        if (!bidsExisting(item)){
            validateCredit(customer);
            chargeCustomer(customer, item);
            removeItemFromBidding(item);
        }
        userTransaction.commit();

        } catch (CreditValidationException cve) {
                userTransaction.rollback();
        } catch (CreditProcessingException cpe){
                userTransaction.rollback();
        } catch (DatabaseException de) {
                userTransaction.rollback();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }
}
```

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class OrderManagerBean {
        @Resource
        private SessionContext context;
        ...

        @TransactionAttribute(TransactionAttributeType.REQUIRED)
        public void placeSnagItOrder(Item item, Customer customer){
                try {
                    if (!bidsExisting(item)){
                        validateCredit(customer);
                        chargeCustomer(customer, item);
                        removeItemFromBidding(item);
                    }
                } catch (CreditValidationException cve) {
                        context.setRollbackOnly();
                } catch (CreditProcessingException cpe){
                        context.setRollbackOnly();
                } catch (DatabaseException de) {
                        context.setRollbackOnly();
                }
        }
}
```

Tomado de [4]

Universidad de los Andes

Agenda

- Introducción
- Modelos Transaccionales
- Java Transaction API (JTA)
- Tipos de Transacciones
- Soporte de Transacciones en EJB
  - Container-Managed Transactions (CMT)
  - Bean-Managed Transactions (BMT)
- **Referencias**

Referencias

1. **Java EE Enterprise Application Technologies.** http://www.oracle.com/technetwork/java/javaee/tech/entapps-138775.html
2. **The Java™ EE 6 Tutorial.** Eric Jendrock. Oracle Corporation. 2011.
3. **Beginning Java™ EE 6 Platform with GlassFish™ 3**, Second Edition. Antonio Goncalves. 2010.
4. **EJB 3 in Action.** Panda Debu, Rahman Reza, Lane Derek. Manning. 2007.
5. **EJB 3 Developer Guide.** Michael Sikora. 2008.
6. **Mastering Enterprise JavaBeans™ 3,0**. Rima Patel Sriganesh, Gerald Brose, Micah Silverman. Wiley Publising, Inc. 2006.
7. **Java Transaction API (JTA) Specification.** Susan Cheung, Vlada Matena. Sun Microsystems Inc. 1999.

Universidad de los Andes

Gracias !!!

Rafael Meneses

rg.meneses81@uniandes.edu.co