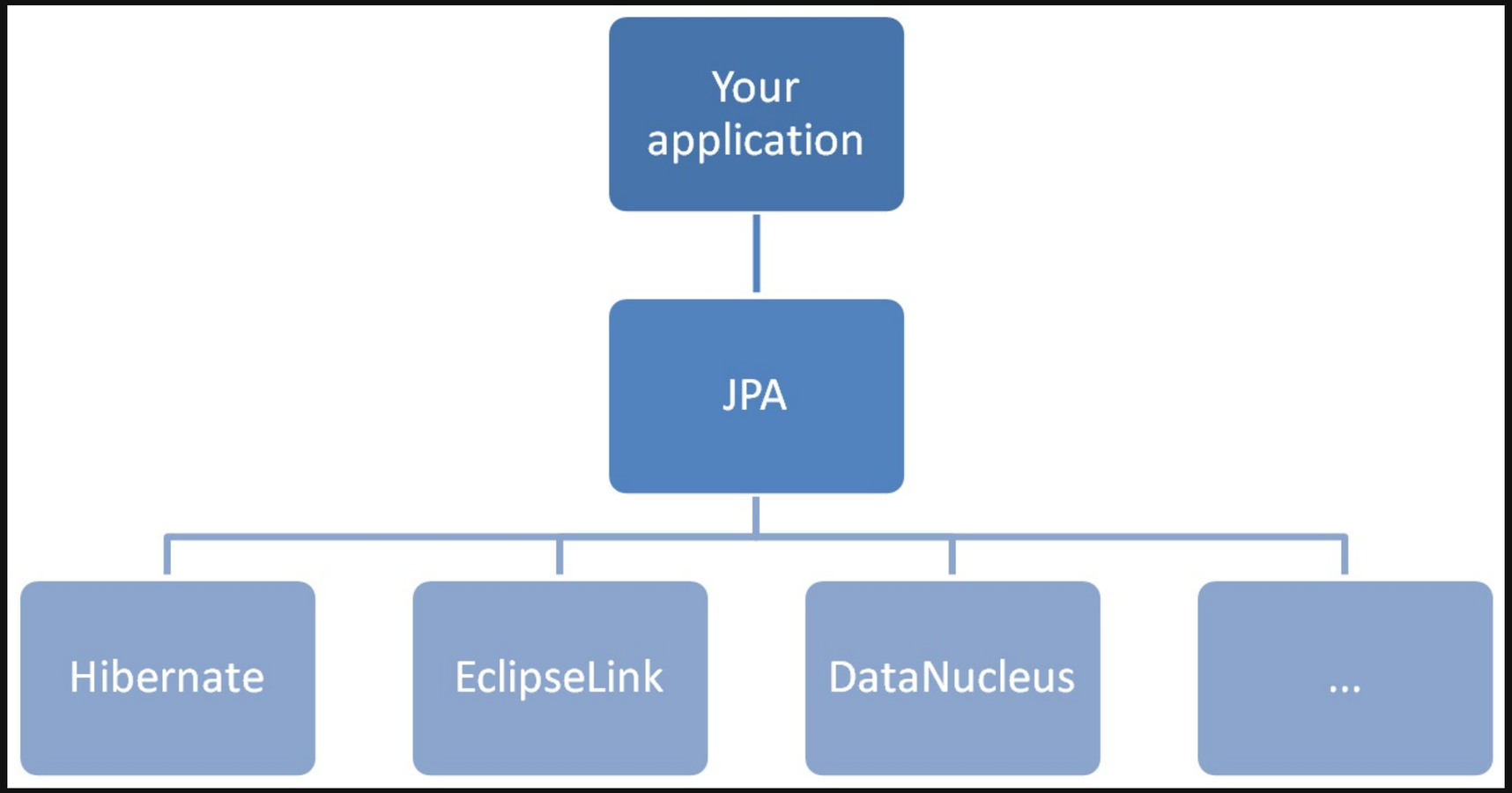
Persistência

Java Persistence API

- **Interface comum para frameworks de mapeamento OR**
- **POJOs (Entidades)**
- **Cada provedor fornece sua implementação (EclipseLink é a implementação de referência)**
- **Áreas:**
  - API propriamente dita
  - Linguagem de consulta
  - API Criteria
  - Metadados para mapeamento objeto-relacional


Your application
JPA
Hibernate
EclipseLink
DataNucleus
...

Principais estruturas

Entity

- **Objeto persistente que representa um conceito do domínio**
- **Entidade ↦ Tabela**
- **Instância da Entidade ↦ Linha da Tabela**
- **POJOs com** get **e** set

Entity

- **Persistent fields vs Persistent properties**
- **Suporta herança e polimorfismo**
- **Configurável por anotação e / ou metadado**
  - @Entity
  - Persistence.xml

Entity

- Quando não indicado, mapeamento adota nome da classe e campos
- Mapeamento Top-Down vs Bottom-Up
- Relacionamento
  - @OneToOne
  - @OneToMany
  - @ManyToMany

Entity

Exemplo

```
@Entity
@NamedQuery(name="f.findAll", query="SELECT f FROM Fornecedor f")
public class Fornecedor implements Serializable {
    ...
    @Id
    private Integer id;
    private String nome;
    ...
    @ManyToOne
    @JoinColumn(name="id_situacao")
    private SituacaoFornecedor situacaoFornecedor;
}
```

Entity Manager

- Ǵerencia as entidades
- Principal interface da API JPA
- Oferece uma API simples para inserção, busca e deleção de registros

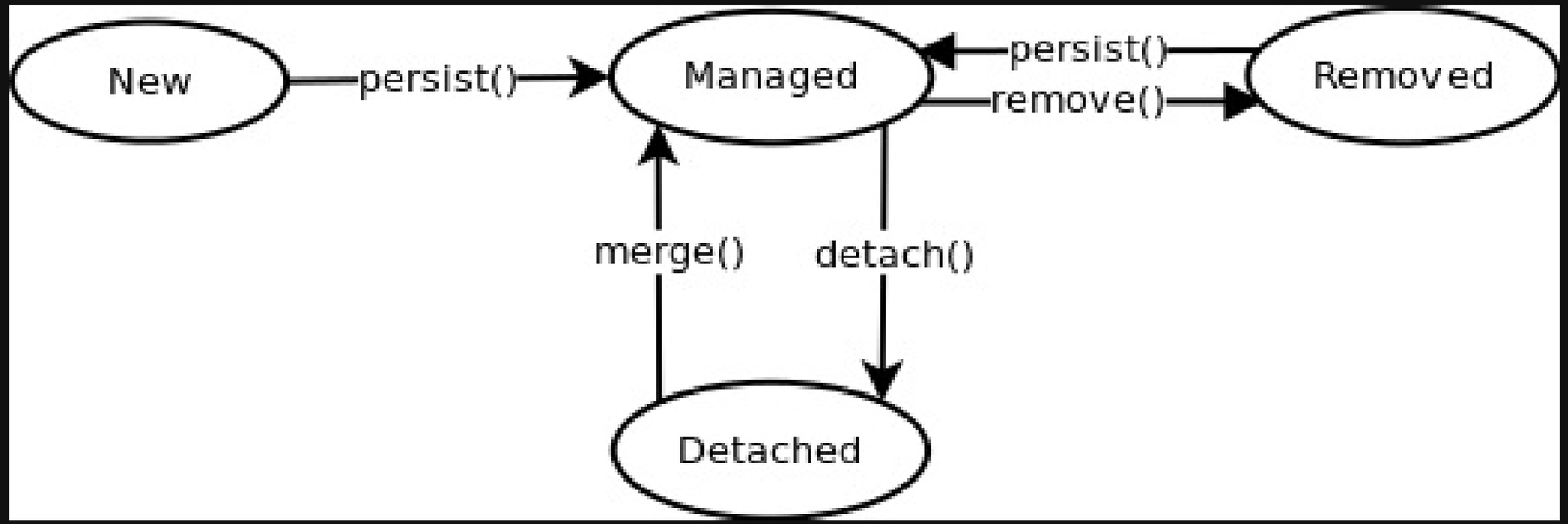
Entity Manager

Principais operações

- persist - **Salva uma instância transiente tornando-a gerenciada**
- detach - **Desassocia uma instância gerenciada de seu contexto**
- merge - **Associa uma instância não gerenciada ao contexto**
- remove - **Remove a instância do Banco**
- find - **Acha um tupla em banco e a associa com instância em memória**

Entity Manager

Ciclo de vida das entidades



Entity Manager

Gerenciado pelo Container

- O container automaticamente propaga o contexto para todas as aplicações
- O contexto transacional é garantido pelo container
- Injetado por meio de `@PersitenceContext`

Entity Manager

Gerenciado pelo Container

```
@Stateless
public class DistribuidoraAS {
...
@PersistenceContext
EntityManager em;
}
```

Entity Manager

Gerenciado pela Aplicação

- O contexto não é propagado para as aplicações
- A aplicação deve explicitamente criar e destruir a instância do contexto
- Injetado por meio de `@PersistenceUnit`

Entity Manager

Gerenciado pela Aplicação

```
@Stateless
public class DistribuidoraAS {
...
@PersistenceUnit
EntityManagerFactory ef;
...
EntityManager em = ef.createEntityManager();
}
```

Persistence Unit

- **Define as entidades gerenciadas por um EntityManager**
- **É definida no** persistence.xml
- **Empacotado no WAR ou EJB-JAR**

Persistence Unit

```
<persistence-unit name="DistribuidoraEJB">
    <jta-data-source>java:/DistribuidoraDS</jta-data-source>
    <class>fa7.distribuidora.persistence.Estoque</class>
    <class>fa7.distribuidora.persistence.Fornecedor</class>
    <class>fa7.distribuidora.persistence.Mercadoria</class>
    <class>fa7.distribuidora.persistence.Reserva</class>
    <class>fa7.distribuidora.persistence.SituacaoFornecedor</class>
</persistence-unit>
```

Sumário

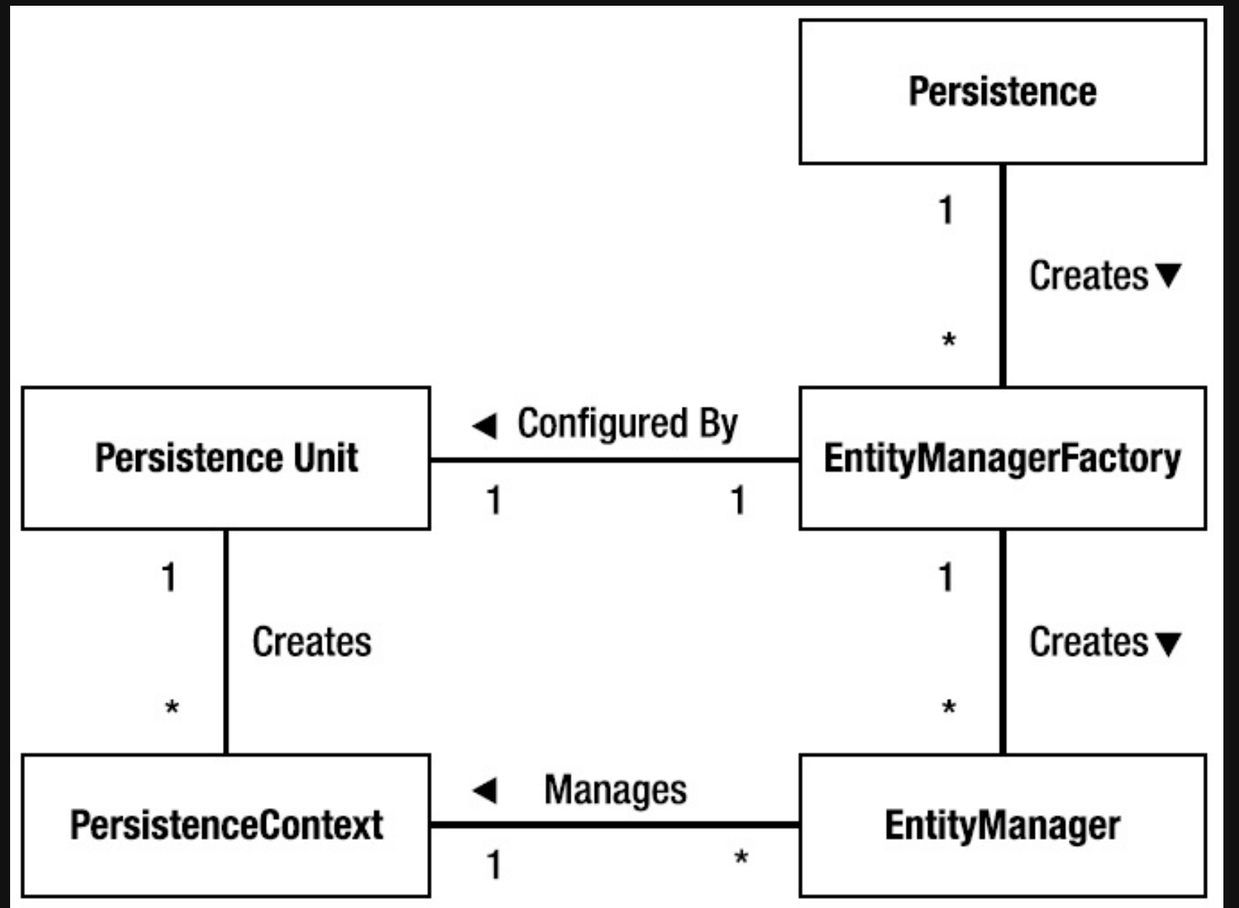


Figure 2-1. Relationships between JPA concepts

Consultas

JPQL

- **Evolução do EJBQL**
- **Consultas operam em objetos ao invés de tabelas relacionais**
- **Permite junções, projeções e outras funcionalidades existentes no SQL**
- **Suporte parâmetros posicionais e nomeados**
  - ?1 e :param
  - setParameter(posicao, valor)
  - setParameter(nome, valor)

Consultas

Consulta nomeada vs Consulta Dinâmica

Consultas

Consulta Dinâmica

Uma consulta passada para o método createQuery do EntityManager no código da aplicação.

```
public long queryEmpSalary(String deptName, String empName) {
    String query = "SELECT e.salary " +
    "FROM Employee e " +
    "WHERE e.department.name = '" + deptName +
    "' AND " + "e.name = '" + empName + "'";
return em.createQuery(query, Long.class).getSingleResult();
}
```

Consultas

Consultas Nomeadas

Consulta associada a um nome e localizada na definição de uma classe \ entidade da aplicação.

```
@Entity
@NamedQuery(name="findSalaryForNameAndDepartment",
    query="SELECT e.salary " +
    "FROM Employee e " +
    "WHERE e.department.name = :deptName AND " +
    "e.name = :empName")
public Department implements Serializable {...}
```

Consultas

Consultas Nomeadas

Multiplas consultas nomeadas requerem a anotação @NamedQueries.

```
@Entity
@NamedQueries({
    @NamedQuery(name="Employee.findAll",
        query="SELECT e FROM Employee e"),
    @NamedQuery(name="Employee.findByPrimaryKey",
        query="SELECT e FROM Employee e WHERE e.id = :id"),
    @NamedQuery(name="Employee.findByName",
        query="SELECT e FROM Employee e WHERE e.name = :name")
})
```

Consultas
Uso

```
@Entity
@NamedQueries({
    @NamedQuery(name="Employee.findAll",
        query="SELECT e FROM Employee e"),
    @NamedQuery(name="Employee.findByName",
        query="SELECT e FROM Employee e WHERE e.name = :name")
})
public EMployee {...}
```

```
public class EmployeeService {
    @PersistenceContext
    EntityManager em;
    ...
    public Employee findEmployeeByName(String name) {
        return em.createNamedQuery("Employee.findByName",
        Employee.class).setParameter("name", name).getSingleResult();
    }
}
```

Consultas

Resultados

- Tipos básicos
- Entidades da aplicação
- Array de objetos
- Tipos definidos pelo usuário

Consultas

Array de Objetos

```
public void displayProjectEmployees(String projectName) {
    List result = em.createQuery(
    "SELECT e.name, e.department.name " +
    "FROM Project p JOIN p.employees e " +
    "WHERE p.name = ?1 " +
    "ORDER BY e.name")
    .setParameter(1, projectName)
    .getResultList();
    int count = 0;
    for (Iterator i = result.iterator(); i.hasNext();) {
    Object[] values = (Object[]) i.next();
    System.out.println(++count + ": " +
    values[0] + ", " + values[1]);
}
```

Consultas

Tipos definidos pelo usuário

```
public void displayProjectEmployees(String projectName) {
    List<EmpMenu> result =
        em.createQuery("SELECT NEW example.EmpMenu(" +
            "e.name, e.department.name) " +
            "FROM Project p JOIN p.employees e " +
            "WHERE p.name = ?1 " +
            "ORDER BY e.name", EmpMenu.class)
        .setParameter(1, projectName)
        .getResultList();
    for (EmpMenu menu : result) {
    System.out.println(menu.getEmployeeName() + ", " +
        menu.getDepartmentName());
    }
}
```

Consultas

Boas práticas

- **Usar DAOs somente quando necessário**
  - DAO vs Repository
- **As máximas do mundo SQL valem para as consultas JPQL**
- **Cuidado com as funcionalidades específicas do Provedor**
- **Tente usar NamedQuery sempre que possível**

Java Transaction API

JTA

O que é transação?

Sequência de ações que devem ser executadas de forma atômica.
Do contrário, elas são revertidas.

JTA

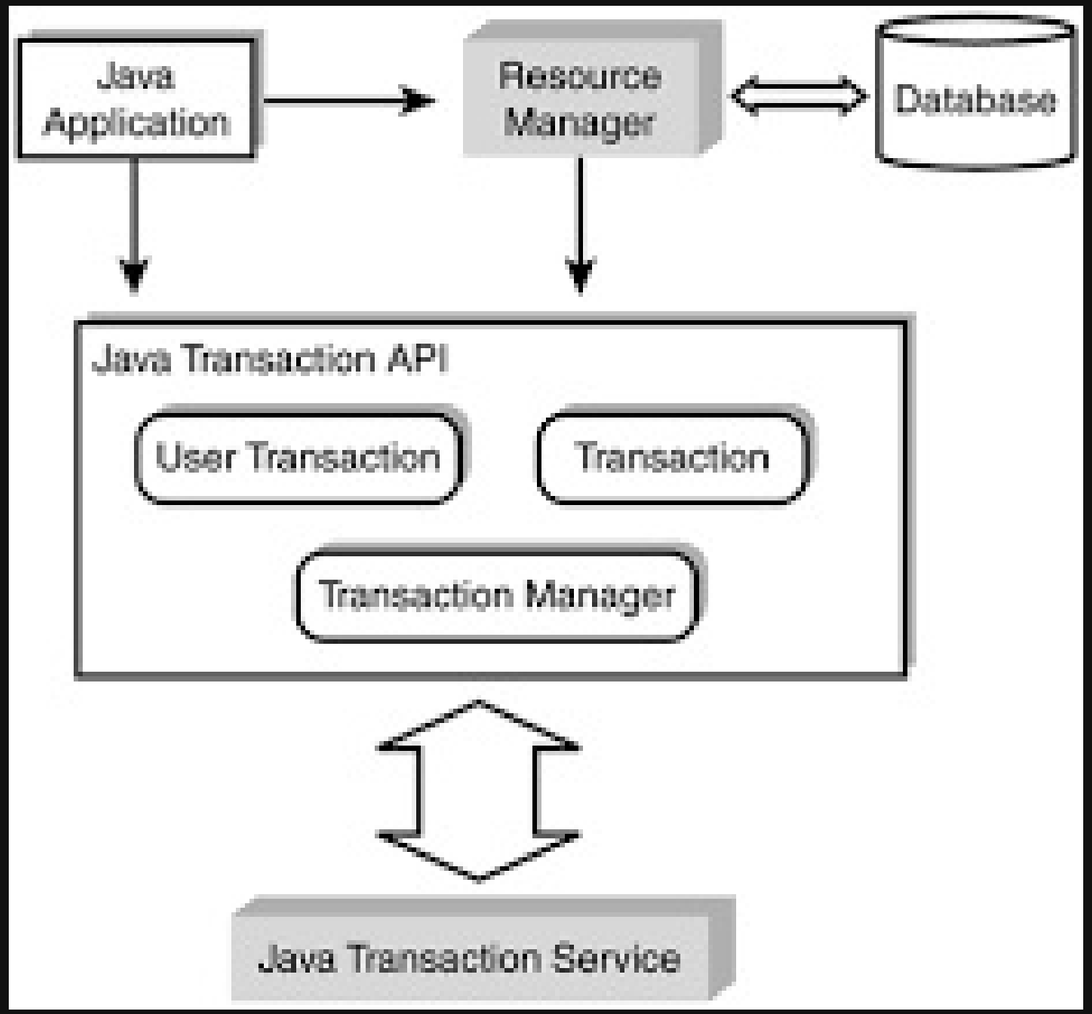
Transações Locais vs Transações Distribuídas

- JDBC
- JTA

JTA

- API de acesso às transações independente de implementações
- Permite o controle de transações distribuídas pelo Container ou Aplicação

JTA


Java Application
Resource Manager
Database
Java Transaction API
User Transaction
Transaction
Transaction Manager
Java Transaction Service

JTA

E no mundo do container?

JTA

Aplicações JavaEE podem fazer uso das transações gerenciadas pelo container (CMT) ou elas mesmo gerenciarem as transações (BMT)

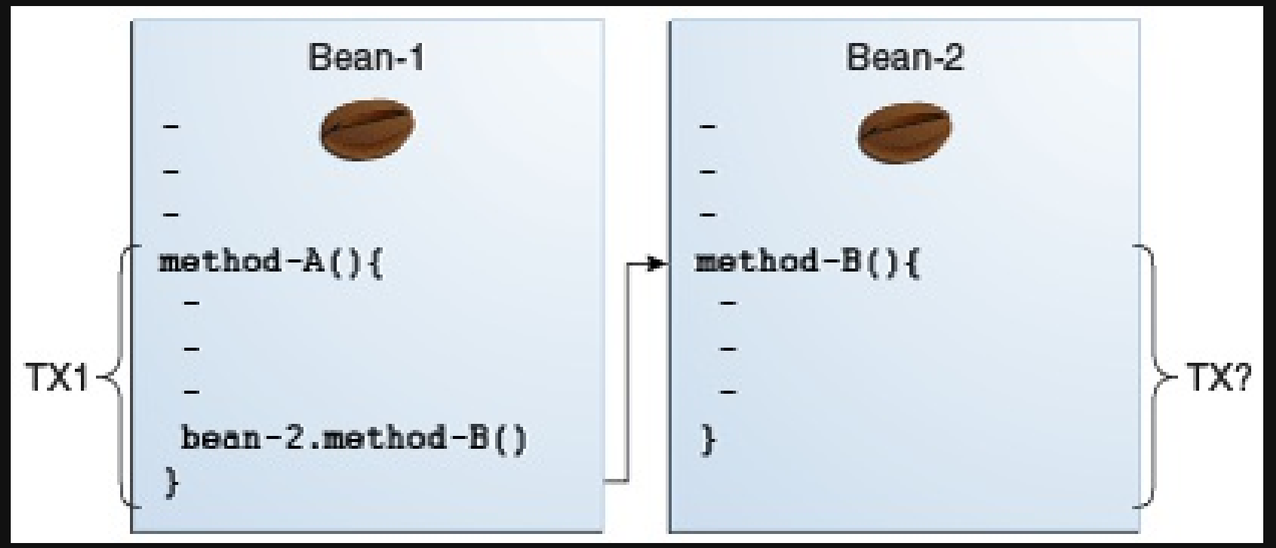
JTA

CMT

- **Demarcadas por anotações em classes ou métodos.**
- @TransactionAttribute vs @Transactional

JTA

TransactionAttribute



JTA

TransactionAttribute

- Required (Padrão)
- RequiresNew
- Mandatory
- NotSupported
- Supports
- Never

JTA

Exemplo

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

JTA

Rollback

- **Uma exceção de sistema é lançada**
- EJBContext.setRollbackOnly
- @ApplicationException(rollback=true)

Atributo	Cliente (B1)	M. Negócio (B2)
Required	Nenhuma	T2
Required	T1	T1
RequiresNew	Nenhuma	T2
RequiresNew	T1	T2
Mandatory	Nenhuma	Erro
Mandatory	T1	T1
NotSupported	Nenhuma	Nenhuma
NotSupported	T1	Nenhuma
Supports	Nenhuma	T1
Supports	T1	T1
Never	Nenhuma	Nenhuma
Never	T1	Erro

JTA

BMT

- **O Bean comanda a transação**
- javax.transaction.UserTransaction
  - begin
  - commit
  - rollback

JTA

BMT

- Stateless Session Bean não pode concluir o método sem realizar commit ou rollback
  - Não se aplica a Stateful Session Beans

JTA

UT via Injeção

```
@Stateless
public class ExampleBean {
    @Resource
    private UserTransaction utx;
    ...
    public void executaAlgo() {
        // Inicia a transacao
        utx.begin();
        ...
        // Acao
        utx.commit();
    }
}
```

JTA

UT via Contexto

```
@Stateless
public class ExampleBean {
    @Resource
    private SessionContext ctx;
    ...
    public void executaAlgo() {
        UserTransaction utx = ctx.getUserTransaction();
        utx.begin();
        // Fazendo algo
        utx.commit();
    }
}
```

Bizus

- Use JTA quando possível, principalmente CMT;
- Use transações locais (JDBC) em cenários que há necessidade de alto desempenho.

Exercício 9

Implementar as seguintes funções na aplicação distribuidora:

- **Uma listagem dos fornecedores e das mercadorias disponíveis;**
- **Uma ordem de reserva de mercadorias utilizando um stateful session bean**
  - O SFSB concentra os pedidos do cliente checando se há estoque disponível. Se não houver estoque disponível, o SSB elenca um dos distribuidores e envia uma mensagem via fila solicitando a quantidade faltante de produto
- **Crie uma aplicação FornecedorApp que implementa um MDB responsável por receber pedidos de mercadorias e processá-los.**

Exercício 9

Dicas

- **Usem o script de criação das tabelas que está disponível no material da aula**
- **Lembrem-se de configurar os serviços adequados para a execução da aplicação**
  - ActiveMQ
  - Postgresql e seus DataSources