Spring Transaction

Definición de transacción .-

- Una transacción se define como un conjunto de operaciones que o bien se ejecutan todas o no se ejecuta ninguna.

Propiedades ACID .-

- **Atomicidad**; todas las operaciones se ejecutan como un todo.
- **Consistencia**; Si el sistema es consistente antes de la transacción, debe seguir siéndolo al finalizar esta.
- **Aislamiento**; Nadie puede acceder a los datos involucrados en una transacción mientras esta esté en marcha.
- **Durabilidad**; Los datos de la transacción tras efectuar el commit, deben ser permanentes.

TransactionManager .-

- Puede ser:
  - DataSourceTransactionManager
  - HibernateTransactionManager
  - JpaTransactionManager

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

Transacciones en Spring .-

- Tenemos 3 formas de crear una transacción:

1. Programativa
2. Declarativa sin anotaciones
3. Declarativa con anotaciones

1. Programativa .-

- Para el modelo programativo necesitamos declarar un TransactionTemplate.

```
<bean id="transactionTemplate"
      class="org.springframework.transaction.support.TransactionTemplate">
    <property name="transactionManager" ref="transactionManager" />
</bean>
```

- En el dao inyectamos el TransactionTemplate como propiedad:

```
<bean id="daoProducto" class="datos.DAOFactory"
                       factory-method="getProductoDAO">
    <property name="jdbcTemplate" ref="jdbcNamedTemplate" />
    <property name="mapeador" ref="mapeador" />
    <property name="transactionTemplate" ref="transactionTemplate" />
</bean>
```

Utilización en el dao .-

- Todo lo que hagamos dentro del método doInTransaction estará en la misma transacción.

```
transactionTemplate.execute(new TransactionCallbackWithoutResult() {

    protected void doInTransactionWithoutResult(TransactionStatus ts) {
        try {
            Iterator<Producto> it = lista.iterator();
            while (it.hasNext()) {
                Producto p = it.next();
                insertarProducto(p);
            }
        } catch (Exception ex) {
            ts.setRollbackOnly();
            ex.printStackTrace();
        }
    }
});
```

2. Declarativa sin anotaciones .-

- Es necesario incluir los espacios de nombres:

```
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
```

- No hay que definir un TransactionTemplate

- Utiliza AOP, para ello es necesario 2 nuevos componentes:
  - <tx:advice>
  - <aop:config>

Continuación .-

```
<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name="altaProductos" propagation="REQUIRED"
                   isolation="DEFAULT" />
        <tx:method name="*" propagation="SUPPORTS" read-only="true" />
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:advisor pointcut="execution(* *..ServicioProductos.*(..))"
                 advice-ref="txAdvice" />
</aop:config>
```

OJO!! .- 2 puntos delante del nombre de la clase

Modelos de propagación .-

- **MANDATORY**; el método debe ejecutarse dentro de una transacción. Si no existe, se genera una excepción.
- **NESTED**; el método debe ejecutarse dentro de una transacción anidada, si no existe se genera una nueva.
- **NEVER**; No debe existir una transacción, su la hubiera genera excepción.
- **NOT_SUPPORTED**; No debe ejecutarse dentro de una transacción, si existiese esta se quedaría suspendida.
- **REQUIRED**; el método debe ejecutarse dentro de una transacción. Si no existe, se genera una nueva.
- **REQUIRES_NEW**; el método debe ejecutarse dentro de una transacción nueva. Si ya existe, esta se quedaría suspendida.
- **SUPPORTS**; Soporta transacciones, si existe se ejecutará en este contexto, si no , no pasa nada.

Modelos de aislamiento .-

- **DEFAULT**; Nivel de aislamiento predeterminado de la BBDD.
- **READ_UNCOMMITED**; Puede leer cambios que aun no se han aplicado. Es peligroso ya que puede efectuar lecturas sucias.
- **READ_COMMITED**; permite las lecturas de transacciones simultaneas que se han aplicado. Puede darse lectura fantasma y no repetible
- **REPEATABLE_READ**; varias lecturas del mismo campo generan el mismo resultado a menos que la tx los modifique. Se puede producir lectura fantasma
- **SERIALIZABLE**; Cumple con el principio ACID.

3. Declarativa con anotaciones .-

- Es necesario incluir los espacios de nombres:

```
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
```

- Este elemento indica a Spring que examine todos los beans o métodos anotados como **@Transactional**

```
<tx:annotation-driven />
```

Continuación .-

```
@Transactional
public void altaProductos(List<Producto> listaProductos) {
    System.out.println("altaProductos");
    Iterator<Producto> it = listaProductos.iterator();
    while (it.hasNext()) {
        Producto p = it.next();
        daoProducto.insertarProducto(p);
    }
}
```