Clase grabada

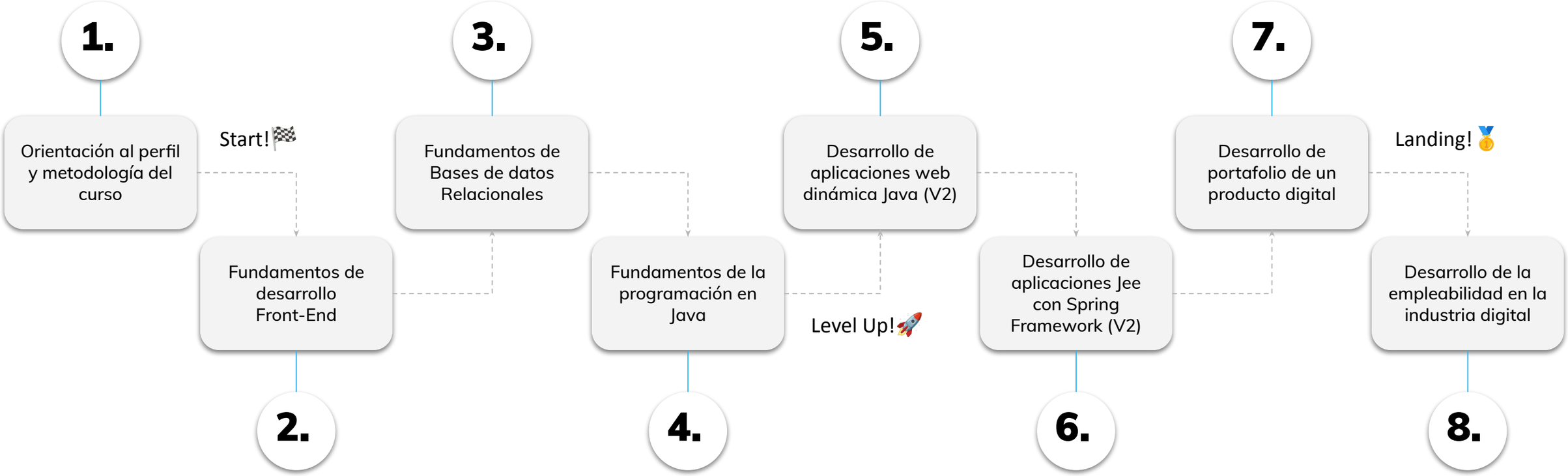
Acceso a datos en Spring Framework

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✓ *Implementar relaciones de asociación entre clase mediante JPA*
- ✓ *Invocar un repositorio desde el servicio*

English Always | alkemy

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.3 Start! 🏁

Acceso a datos en Spring Framework

Transaccionalidad en los servicios

Creando transacciones

Manejo de la transaccionalidad en los servicios
Qué es la transaccionalidad y por qué es importante
Configurando la transaccionalidad
Creando un servicio transaccional

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Comprender la configuración de la transaccionalidad en los servicios*
- *Aprender a crear un servicio transaccional*

Transaccionalidad

Transaccionalidad

La transaccionalidad es un concepto fundamental en el desarrollo de aplicaciones que involucran operaciones sobre bases de datos. Proporciona un mecanismo para garantizar la integridad y la consistencia de los datos en situaciones donde varias operaciones deben ejecutarse como una unidad atómica. En el contexto de servicios y bases de datos, la transaccionalidad juega un papel crucial para **mantener la coherencia de los datos y garantizar la ejecución exitosa o fallida de un conjunto de operaciones relacionadas**.

Transaccionalidad

Una transacción de base de datos es un conjunto de instrucciones que se ejecutan en bloque. Por ejemplo, decidimos modificar un registro “A” en la base de datos. Si en alguna de las instrucciones se produce un error **todo el proceso se echa atrás**. De esta manera, si luego consultamos la base de datos podremos ver que el registro “A” no ha sido alterado.

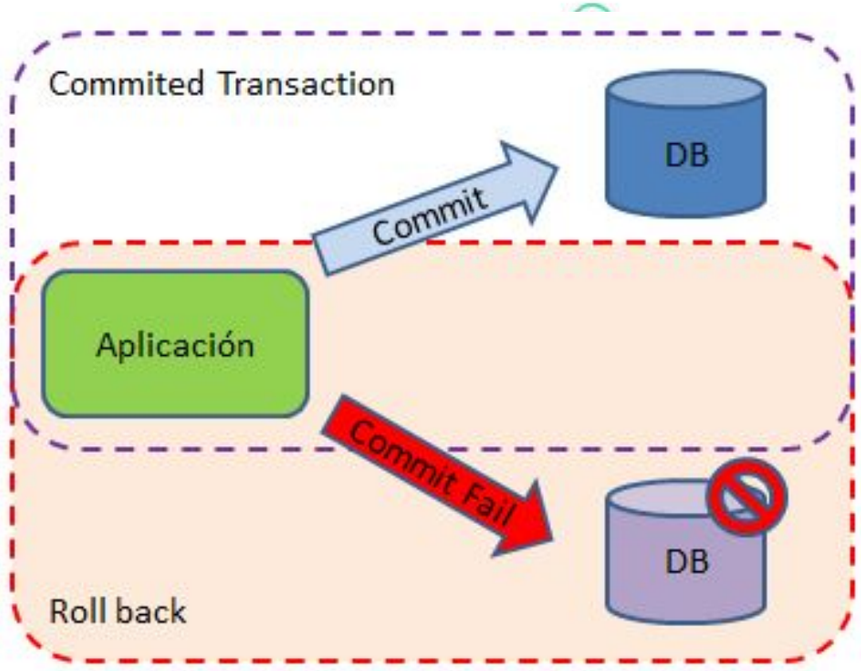
A este proceso de “**tirar atrás**” las instrucciones realizadas se le dice hacer un **rollback**, mientras que el proceso de **confirmar** todas las instrucciones en bloque una vez hemos visto que no se ha producido ningún error se le llama hacer un **commit**.

Transaccionalidad

Las transacciones cumplen con el concepto **ACID**:

Atomicidad: Garantiza que todas las operaciones de una transacción se completen con éxito o ninguna se complete en absoluto. Si alguna operación falla, la transacción se revierte a su estado original (rollback).

Consistencia: Asegura que la base de datos se mueva de un estado consistente a otro estado consistente. Cada transacción debe llevar la base de datos de un estado válido a otro.

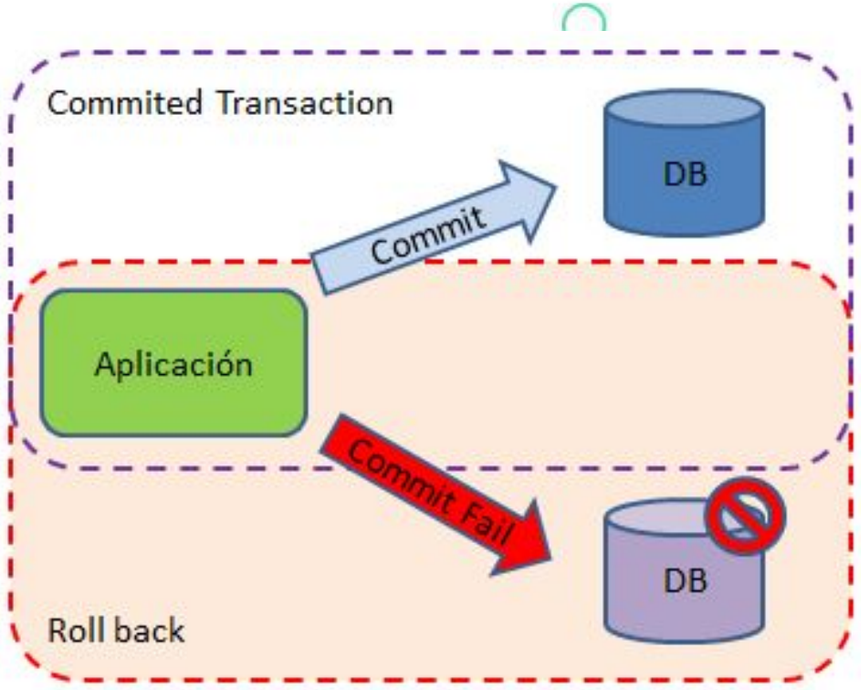


Transaccionalidad

Las transacciones cumplen con el concepto **ACID**:

Aislamiento: Garantiza que el efecto de una transacción no sea visible para otras transacciones hasta que se complete. Las transacciones deben ejecutarse de manera aislada entre sí.

Durabilidad: Una vez que una transacción se completa con éxito, sus cambios se mantienen incluso en caso de falla del sistema o reinicio. La durabilidad garantiza que los cambios sean permanentes.



Transaccionalidad

Las transacciones cumplen con el concepto **ACID**:

Aislamiento: Garantiza que el efecto de una transacción no sea visible para otras transacciones hasta que se complete. Las transacciones deben ejecutarse de manera aislada entre sí.

Durabilidad: Una vez que una transacción se completa con éxito, sus cambios se mantienen incluso en caso de falla del sistema o reinicio. La durabilidad garantiza que los cambios sean permanentes.

Configuración de la transaccionalidad

Configuración de la transaccionalidad

La manera más común de tratar las transacciones en Spring es mediante la anotación @Transactional en la cabecera del método de una clase (nunca un interfaz) gestionada por Spring.

En nuestro caso, los métodos encargados de gestionar transacciones se encontrarán en las clases de servicio o lógica de negocio de nuestra aplicación.

```
@Transactional
public void hazAlgoTransaccionalmente() {
    // Soy transaccional!
}
```

Configuración de la transaccionalidad

- **Iniciación Automática de Transacciones:** Si un método anotado con @Transactional se llama desde otro método, una nueva transacción se inicia si aún no existe. Si la transacción actual tiene éxito, se confirma al final del método; de lo contrario, se revierte.

- **Rollback Automático:** Si una excepción no controlada se lanza durante la ejecución de un método anotado con @Transactional, Spring revierte automáticamente la transacción, deshaciendo cualquier cambio realizado durante la transacción.

- **Configuración Personalizada:** Spring permite configuración personalizada, como definir la propagación de transacciones, el aislamiento y otros aspectos a través de atributos en la anotación @Transactional.

Configuración de la transaccionalidad

Propagación de Transacciones en Spring:

La propagación de transacciones en Spring se refiere a cómo se manejan las transacciones cuando un método anotado con @Transactional llama a otro método que posee la misma anotación. Algunos de los modos de propagación más comunes son:

- **REQUIRED** (Predeterminado): Si ya existe una transacción, el método se une a ella; de lo contrario, se inicia una nueva transacción. Es el comportamiento predeterminado y el más utilizado.

- **REQUIRES_NEW**: Si ya existe una transacción, se suspende y se inicia una nueva transacción. Útil cuando deseas que el método tenga su propia transacción independientemente de la existencia de una transacción anterior.

Configuración de la transaccionalidad

Propagación de Transacciones en Spring:

- **NESTED**: Si existe una transacción, crea un nuevo punto de guardado dentro de esa transacción. Si no hay transacción, se comporta como REQUIRED.
- **PROPAGATION_SUPPORTS**: Si existe transacción la aprovecha, sino no crea ninguna.
- **PROPAGATION_MANDATORY**: Si no existe una transacción abierta se lanza una excepción. Hay programadores que anotan sus DAO con esta opción.
- **PROPAGATION_NOT_SUPPORTED**: Si existe una transacción la pone en suspenso, la transacción se reactiva al salir del método.

Configuración de la transaccionalidad

Propagación de Transacciones en Spring:

La configuración de la propagación de las transacciones se realizan mediante la escritura de estas órdenes en el atributo “propagation” de la anotación @Transactional.

```
@Transactional(propagation = Propagation.MANDATORY)
public void eliminarUsuario(Long id) {
    usuarioRepositorio.deleteById(id);
}
```

Configuración de la transaccionalidad

Aislamiento de Transacciones en Spring:

El aislamiento de transacciones en Spring controla cómo las transacciones concurrentes interactúan entre sí. Algunos niveles de aislamiento comunes son:

- **DEFAULT** (Predeterminado): Utiliza el nivel de aislamiento predeterminado de la base de datos.
- **READ_COMMITTED**: Evita la lectura de datos no comprometidos (datos modificados pero aún no confirmados) por otras transacciones.
- **READ_UNCOMMITTED**: Permite la lectura de datos no comprometidos por otras transacciones.

```
@Transactional(isolation = Isolation.READ_COMMITTED)
public void metodoConTransaccionReadCommitted() {
    // Operaciones con la base de datos
}
```

Configuración de la transaccionalidad

También podemos configurar tanto la propagación como el aislamiento para el mismo método:

```
@Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.READ_COMMITTED)
public void metodoConTransaccionPersonalizada() {
    // Operaciones con la base de datos
}
```

Configuración de la transaccionalidad

Para poder manejar bien las transacciones, debemos tener en cuenta que **todo método que pueda generar un cambio persistente en la base de datos, debe ser marcado como @Transactional.**

En los casos de los **métodos de consulta**, podemos encontrarnos que también están anotados como transaccionales, pero en estos casos **debemos tener en cuenta su lectura**.

Configuración de la transaccionalidad

La autorización se configura con la propiedad readOnly (solo lectura) de @Transactional. Su valor predeterminado es false, lo que significa que no está en modo de solo lectura, ergo se permite la escritura. Dicho con otras palabras, @Transactional siempre otorga permiso de escritura a la transacción, salvo que se indique lo contrario de manera explícita. La forma de declararlo es la siguiente:

```
@Transactional(readOnly = true)
public Usuario obtenerUsuariosPorNombre(String nombre) {
    return usuarioRepositorio.buscarPorNombre(nombre);
}
```

Momento:

Time-out!

⏰5 min.

Creando un servicio transaccional

Creando un servicio transaccional

Crear un servicio transaccional en Spring implica utilizar la anotación @Transactional en los métodos del servicio que necesitan una gestión transaccional. En esta clase vamos a anotar los métodos que realizan transacciones en la base de datos.

Recordemos que en caso de que el método no vaya a realizar escrituras sobre la base de datos, podemos marcarlo como “solo lectura”, lo que le dará mayor beneficio de rendimiento. Cuando un método es anotado como readOnly = true, si se intenta realizar alguna operación de escritura sobre la BD, se generará una excepción.

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

Transacciones en la AlkeWallet

- *Vamos a agregar la anotación @Transactional a todos los métodos de los servicios de Cuenta y Usuario.*
- *En caso de que el método sea de "sólo lectura" (no intenta persistir en la base de datos), deberemos anotarlo.*
- *Además, podemos probar las transacciones viendo el impacto en la base de datos.*

Tiempo: 20 minutos

Ejercicio

Creando transacciones

Creando transacciones

Contexto: 🙌

Vamos a continuar con el ejercicio de la clase anterior. En este caso vamos agregar la anotación @Transactional a todos los métodos de cada servicio en donde se interactúe con la base de datos.

Consigna: ✍️

Agregar la anotación @Transactional e indicar con el atributo readOnly=true en caso de que el método no genere cambios en la base de datos.

Tiempo🕐: *15 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Comprender los procesos de transaccionalidad y su importancia*
- ✓ *Aprender a configurar la transaccionalidad desde un servicio*

English Always | alkemy

#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Modulo 6, Lección 3: páginas 23 - 28*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento:

Time-out!

⏰5 min.