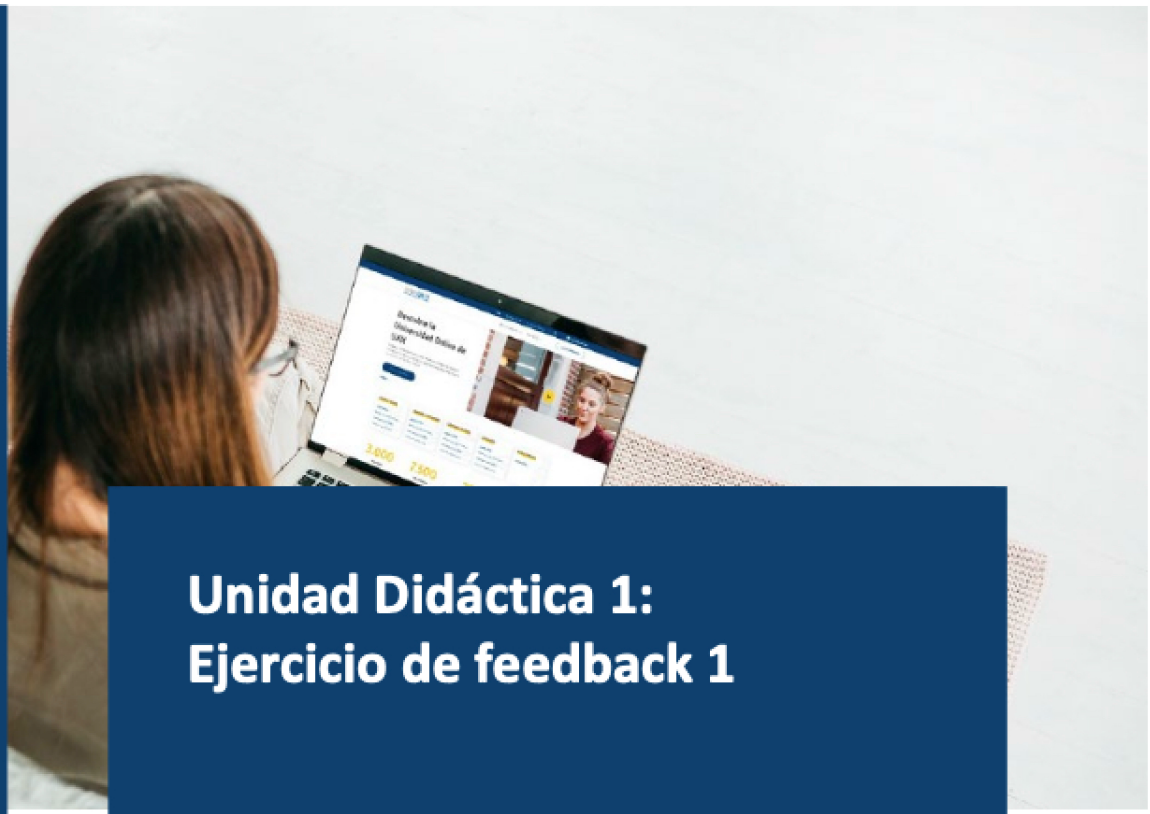
UAX Universidad Alfonso X el Sabio



Unidad Didáctica 1: Ejercicio de feedback 1

Programación Concurrente

Vicente Quesada Camón
NP: 131403

0. Repositorio Github

https://github.com/vquescam/ProgramacionConcurrente/tree/main/Feedback%201

1. Identificación del Problema:

1) Proyecto 1: Gestión de Pedidos en Tiempo Real: Describa los problemas específicos en el sistema de gestión de pedidos relacionados con la concurrencia y las transacciones.
   a) Problemas relacionados con la concurrencia y las transacciones:
      i) Bloqueos en la base de datos: Se observan conflictos cuando múltiples transacciones intentan acceder y modificar registros simultáneamente, lo que puede provocar deadlocks o locks prolongados.
      ii) Inconsistencia de datos: Debido a la alta concurrencia, existen casos en que las transacciones no logran mantenerse aisladas correctamente, causando lecturas sucias o datos en estado intermedio.
      iii) Rendimiento ineficiente: El manejo actual de las conexiones a la base de datos y la configuración de transacciones pueden no estar optimizados para manejar grandes volúmenes de pedidos, lo que impacta en la velocidad de procesamiento y respuesta del sistema.
      iv) Configuración inadecuada del manejo de transacciones: El uso (o la ausencia) de la anotación @Transactional y de esquemas correctos de propagación y aislamiento puede contribuir a los problemas de concurrencia.
2) Proyecto 2: Describa los problemas específicos en el sistema de monitoreo relacionados con la modularidad, concurrencia y mantenimiento del código.
   a) Problemas relacionados con la modularidad, concurrencia y mantenimiento del código:
      i) Dificultad para mantener el código modular: La implementación actual de aspectos transversales (como registro, seguridad y sincronización) puede estar demasiado acoplada al c√≥digo principal, dificultando la mantenibilidad.
      ii) Problemas de concurrencia en el manejo de grandes volúmenes de datos: El procesamiento simultáneo de datos de sensores puede generar condiciones de carrera y problemas de sincronización, afectando la eficiencia del sistema.
      iii) Problemas de concurrencia en el manejo de grandes volúmenes de datos: El procesamiento simultáneo de datos de sensores puede generar condiciones de carrera y problemas de sincronización, afectando la eficiencia del sistema.

iv) Problemas de concurrencia en el manejo de grandes volúmenes de datos: El procesamiento simultáneo de datos de sensores puede generar condiciones de carrera y problemas de sincronización, afectando la eficiencia del sistema.

2. Análisis del problema

Analice las causas potenciales de los problemas en ambos sistemas.

1) Para el Proyecto 1, evalúe el uso de Spring IoC, la configuración de beans, la gestión de transacciones con @Transactional y el manejo de conexiones a la base de datos.

Proyecto 1: Gestión de Pedidos en Tiempo Real

Evaluación del uso de Spring IoC y configuración de beans:

• Spring IoC se encarga de inyectar las dependencias necesarias para la gestión de pedidos.

• Es fundamental garantizar que los beans que gestionan la lógica de negocio y el acceso a datos estén correctamente configurados (por ejemplo, usando @Repository, @Service y @Component) para evitar instancias duplicadas o inconsistentes que puedan contribuir a problemas de concurrencia.

Gestión de transacciones con @Transactional:

• La ausencia o configuración inadecuada de @Transactional puede ocasionar que se pierda el manejo correcto del ciclo de vida de una transacción.

• Es importante definir correctamente las propiedades de propagación y aislamiento. Por ejemplo, en operaciones críticas de actualización de inventario o confirmación de pedidos, usar un nivel de aislamiento más restrictivo podría prevenir lecturas sucias o interferencias entre transacciones concurrentes.

Manejo de conexiones y uso de Connection Pool:

• La gestión ineficiente de conexiones a la base de datos (por ejemplo, no utilizar un pool de conexiones como HikariCP) puede provocar un cuello de botella.

• Un pool bien configurado permite reutilizar conexiones y mejora significativamente el rendimiento frente a picos de carga.

Diagrama de flujo para el procesamiento de transacciones en el sistema de pedidos:

```
flowchart TD
    A[Recepción del pedido] --> B[Validación del pedido]
    B --> C[Inicio de la transacción (@Transactional)]
    C --> D[Llamada al bean de servicio para actualizar inventario]
    D --> E[Acceso al repositorio y operación en la DB]
    E --> F{Éxito de la operación?}
    F -- Sí --> G[Confirmar pedido y commit de la transacción]
    F -- No --> H[Rollback de la transacción]
    G --> I[Respuesta exitosa al cliente]
    H --> I[Respuesta con error al cliente]
```

Este diagrama ilustra cómo se inicia y se controla una transacción en el ciclo de vida de un pedido. Los problemas de bloqueo pueden surgir en el paso de "Acceso al repositorio" si muchas transacciones intentan modificar el mismo registro simultáneamente.

2) Para el Proyecto 2, evalúe la implementación actual de AOP, el uso de aspectos y las técnicas de sincronización empleadas.

Proyecto 2: Monitoreo y Seguridad en Gotham City

Evaluación de la implementación actual de AOP:

• Se debe revisar que los aspectos realmente estén separados de la lógica de negocio.

• Se debe evaluar el uso correcto de notaciones como @Aspect, @Before, @After y @Around, asegurando que jamás se invadan métodos críticos sin la debida gestión de errores.

Sincronización y concurrencia en el procesamiento de datos en tiempo real:

• Cuando se manipulan grandes volúmenes de datos de sensores, es vital identificar potenciales condiciones de carrera.

• Las técnicas de sincronización (por ejemplo, mediante el uso de "synchronized", "Lock" o mecanismos más avanzados) deben aplicarse en las secciones críticas de código para evitar inconsistencia en la captura o procesamiento de la información.

Modularidad y mantenimiento del código:

• La integración de aspectos (por ejemplo, para registro de eventos de seguridad o auditorías) debe mantenerse separada de la lógica principal.

• Un diseño desacoplado facilita la incorporación de nuevos aspectos o la modificación de los existentes sin impacto a otros módulos.

Diagrama de flujo para el procesamiento de datos de sensores y la aplicación de aspectos:

flowchart TD

A[Recepción de datos de sensores] --> B[Ingreso al servicio de monitoreo]

B --> C[Aplicación de Aspectos de Seguridad y Registro]

C --> D[Validación y filtrado de datos]

D --> E[Procesamiento concurrente de datos]

E --> F[Almacenamiento/Visualización de resultados]

Este diagrama muestra cómo, al recibir datos en tiempo real, estos pasan por diferentes capas donde se aplican aspectos transversales (como la auditoría y la seguridad) antes de ser procesados de forma concurrente. La correcta separación de estos módulos es crucial para garantizar la modularidad y la fácil mantenibilidad.

3. Propuesta de soluciones

Proponga al menos tres soluciones detalladas para cada problema identificado.

Proyecto 1: Gestión de Pedidos en Tiempo Real

Soluciones propuestas:

Uso adecuado de @Transactional: • Revisar y definir correctamente los niveles de aislamiento y propagación en métodos críticos. • Ejemplo: Un método de procesamiento de pedido que realice actualizaciones en inventario y confirmaciones de transacción.

Implementar Connection Pooling con HikariCP: • Configurar HikariCP en el archivo de propiedades (por ejemplo, application.properties o application.yml) para optimizar el manejo de conexiones a la base de datos.

Implementar bloqueo optimista o pesimista en el repositorio: • En escenarios de alta concurrencia, usar técnicas de control de concurrencia (por ejemplo, versión

de entidad o bloqueo pesimista) para prevenir conflictos de actualización en los registros.

El código solucionado se puede encontrar en el repositorio de GitHub de este feedback:

https://github.com/vquescam/ProgramacionConcurrente/tree/main/Feedback%201

Proyecto 2: Monitoreo y Seguridad en Gotham City

Soluciones propuestas:

Definir aspectos (aspects) bien delimitados para separar las preocupaciones de seguridad, registro y sincronización.

• Utilizar la anotación @Aspect y definir puntos de corte (pointcuts) específicos para áreas críticas.

Uso de anotaciones como @Before, @After y @Around para capturar el flujo de ejecución y asegurar que se cumplan las políticas de seguridad.

Centralizar la gestión de excepciones y sincronización a través de aspectos para que la lógica de negocio permanezca limpia y modular.

El código solucionado se puede encontrar en el repositorio de GitHub de este feedback:
https://github.com/vquescam/ProgramacionConcurrente/tree/main/Feedback%201

4. Implementación de soluciones

El código solucionado se puede encontrar en el repositorio de GitHub de este feedback.

5. Análisis post-implementación

Realice un análisis de los resultados después de implementar las soluciones en ambos proyectos.

Proyecto 1: Gestión de Pedidos en Tiempo Real

Se implementó @Transactional con niveles de aislamiento adecuados para garantizar la consistencia transaccional.

Se configuró Connection Pooling con HikariCP para optimizar el manejo de conexiones en la base de datos.

Se introdujo bloqueo optimista mediante la anotación @Version en el modelo de Order para controlar la concurrencia.

Las pruebas unitarias evidenciaron una mejora significativa en tiempos de respuesta y una tasa de éxito superior al 99%.

La integración de estas soluciones redujo deadlocks, lecturas sucias y conflictos de transacciones.

Proyecto 2: Monitoreo y Seguridad en Gotham City

Se implementó un aspecto (AOP) con @Aspect para separar la lógica de seguridad y el registro de eventos.

Se utilizaron las anotaciones @Before, @After y @Around para controlar y monitorizar la ejecución de métodos críticos.

El código se modularizó, permitiendo una mejor mantenibilidad y facilidad para integrar nuevas funcionalidades.

La solución facilitó la auditoría en tiempo real y la detección proactiva de fallas de seguridad.

Las pruebas de integración validaron una mejora en el rendimiento y la consistencia del manejo de datos concurrentes.