PREGUNTA 1

¿Cuáles son las ventajas de separar el código que accede a la base de datos del código que implementa la lógica de negocio, en este caso, solo la clase principal?

La separación del código que interactúa con la base de datos y el código que ejecuta la lógica de negocio ofrece múltiples beneficios:

Desacoplamiento Mejorado: Al separar estas responsabilidades, la lógica de negocio no está directamente vinculada a los detalles específicos de cómo se accede a los datos. Esto simplifica el mantenimiento y la evolución del sistema, permitiendo cambios en una capa sin afectar a la otra. Asimismo, simplifica las pruebas unitarias al posibilitar la evaluación independiente de cada capa. Esta separación también mejora la legibilidad y la mantenibilidad del código al asignar responsabilidades claras a cada componente. Además, fomenta la reutilización del código, ya que las clases de acceso a datos pueden ser empleadas en diferentes partes del proyecto o incluso en otros proyectos.

Flexibilidad Incrementada: La separación facilita la adopción de distintas tecnologías de acceso a datos (por ejemplo, SQL, NoSQL) sin impactar en la lógica de negocio. Esto hace que el sistema sea más adaptable a cambios en los requisitos o en el entorno tecnológico. También simplifica la escalabilidad del sistema, permitiendo la incorporación de nuevas capas de acceso a datos o la modificación de la tecnología subyacente sin afectar al resto del código.

Robustez Mejorada: Al aislar la lógica de negocio de los detalles de implementación del acceso a datos, se reduce la probabilidad de errores. Además, la identificación rápida de la capa donde se origina un error facilita la depuración del código, contribuyendo así a la robustez del sistema.

PREGUNTA 2

¿Qué propuesta sugieres para evitar la creación del objeto Connection en la clase principal?

Una propuesta para evitar la creación del objeto Connection en la clase principal es emplear el patrón Singleton para gestionar la conexión a la base de datos. Este enfoque presenta diversas ventajas:

Reducción del Acoplamiento: Al centralizar la gestión de la conexión en una clase Singleton, se disminuye el nivel de dependencia en la clase principal y entre otras clases. Esto mejora la modularidad y la flexibilidad del código.

Simplificación del Código: Al tener un único punto de acceso a la conexión, el código se vuelve más claro y fácil de entender y mantener.

Mejor Control del Ciclo de Vida: El patrón Singleton permite un control centralizado del ciclo de vida de la conexión, lo que ayuda a prevenir fugas de recursos y otros problemas relacionados con la gestión de conexiones.

Reutilización de Conexiones: Al mantener una única instancia de conexión, se reduce el riesgo de crear conexiones adicionales innecesarias, optimizando así el rendimiento y el uso de recursos.

En conclusión, la implementación del patrón Singleton para gestionar la conexión a la base de datos es una solución efectiva para evitar la creación del objeto Connection en la clase principal, lo que conduce a un código más limpio, desacoplado y fácil de mantener.

FUNTES CONSULTADAS

https://es.linkedin.com/pulse/aplicaciones-n-capas-estructurando-el-desarrollo-de-ojeda-montoya

https://refactoring.guru/es/design-patterns/singleton

REPOSITORIO

https://github.com/alvarowau/PROG11_tarea