Implementación del Patrón de Diseño Data Mapper

StockManager

Collareda Agustín y Frey Hugo

Índice

Implementación del Patrón de Diseño Data Mapper

Punto 1

El patrón Data Mapper genera una capa entre la lógica del negocio y la lógica de la persistencia de datos. Esto se fundamenta en que a medida que crece el sistema, los modelos de clases dejan de coincidir respectivamente con el modelo de datos.

Al tener diferentes mecanismos y que debe existir una transferencia de datos entre los dos modelos, si estos saben mucho del otro al modificarse un modelo normalmente se modifica el otro lo que indica un alto acoplamiento.

Por lo que, al añadir la capa que se genera por usar este patrón de diseño desacopla los modelos de clases con el modelo de datos, lo que garantiza alta flexibilidad y mantenibilidad. Aun así este patrón es útil cuando el modelo tiene su correspondencia a la entidad de la base de datos ya que esto permite poder realizar las pruebas sin necesidad de la base de datos, garantiza el principio de la separación de responsabilidades y permite modificar el acceso de los datos sin tener que modificar toda la clase del modelo.

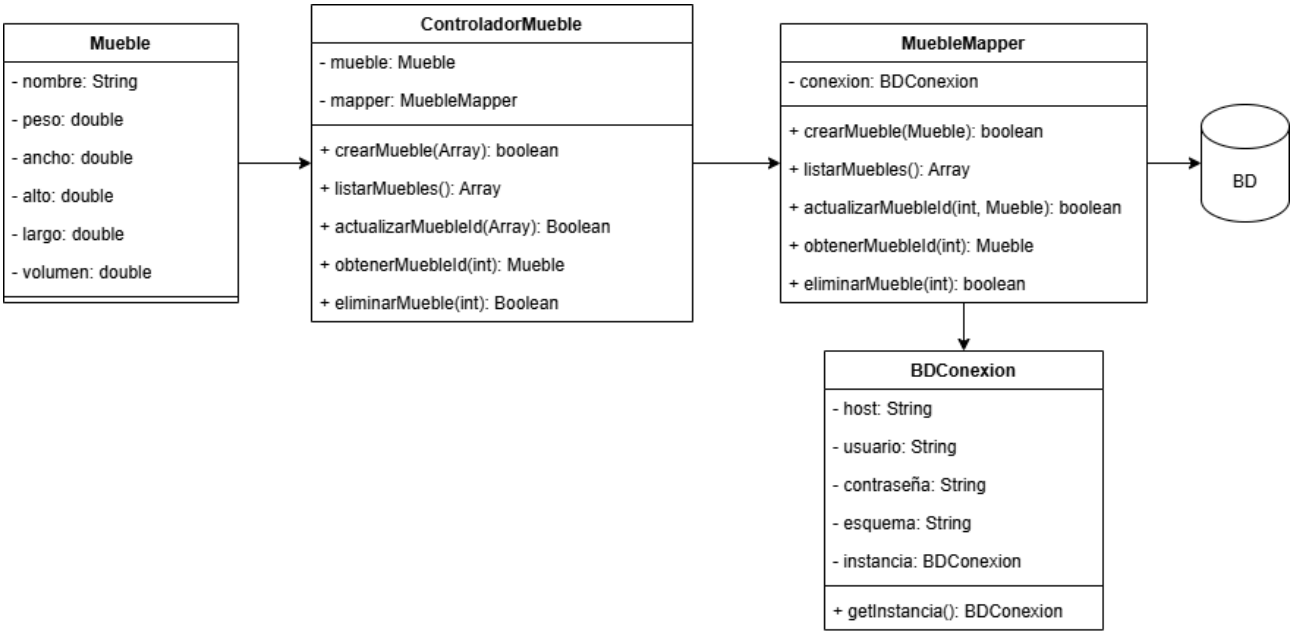
Punto 2

El patrón Data Mapper se puede relacionar con los siguientes criterios de calidad según la ISO 25010:

- **Mantenibilidad**: Al separar la lógica del negocio de la lógica de almacenamiento, se mejora la modularidad del sistema y la reusabilidad de código, y se facilita la ejecución de pruebas y la realización de modificaciones.
- **Compatibilidad**: Facilita el intercambio de información con la BD.
- **Flexibilidad**: Si se desea cambiar de DBMS, basta con modificar el data mapper sin afectar los módulos encargados de la lógica de negocio.

Punto 3

El modelo UML correspondiente a la implementación del patrón Data Mapper, según nuestro proyecto, es:



Punto 4

Implementación y pruebas:

- Se creó la nueva clase MuebleMapper, con la conexión y los métodos para interactuar con la BD. Los métodos listarMueble, obtenerMuebleId y eliminarMueble funcionan de la misma forma que lo hacían en la clase Mueble. Los métodos crearMueble y actualizarMueble ahora requieren recibir un Mueble como parámetro. crearMueble, actualizarMueble y eliminarMueble ahora devuelven un boolean (true si se concreto la acción, false si no).
- Se eliminó el atributo conexión de la clase Mueble y todos los métodos que interactuaban con la BD.
- Se agregó el atributo mapper al ControladorMueble y se eliminó el atributo de conexión con la BD. Se modificó la clase para qué utilice los métodos de interacción con la BD del mapper.

Se realizaron pruebas para probar cada una de las funcionalidades del sistema. Se creó un mueble nuevo, se editaron todos sus valores y posteriormente se eliminó.

Referencias

- Fowler, M. (2003, marzo 5). *Data Mapper*. MartinFowler.com. Recuperado de: https://martinfowler.com/eaaCatalog/dataMapper.html
- Liebler, D. (2011). *Data Mapper*. DesignPatternsPHP. Recuperado de: https://designpatternsphp.readthedocs.io/es/latest/Structural/DataMapper/README.html