BIBERNATE

BETTER THAN HIBERNATE ORM

Bobocode

Initial requirements. Bibernate project

Bibernate is like Hibernate, but better. It is a **simple version of an Object-Relational Mapping (ORM)** framework that provides an Object-Oriented API for working with relational databases.

The premise behind Bibernate is that a user can **store, fetch, change** and **remove objects** and the **framework will take care of the required SQL queries in order to perform corresponding operations in the database.**

Key parts

1. Configuration
2. Mapping
3. CRUD API
4. Relationship Management

Configuration

What is the minimum input a user can provide in order to start using this technology? In order to connect to the DB, it will definitely need a configured JDBC driver, so a user should be able to provide database credentials. **It's up to you if you want to use a text property file, XML, or anything else**. In order to connect to the DB, you can either

- user simple version of DataSource with a single connection
- build custom connection pool
- use third-party connection pools

Mapping

Mapping provides metadata that is used by the framework in order to **map Java classes and database tables**. The most convenient way to specify metadata is annotations, so your job is to **create a set of annotations** that will be used by framework users.

CRUD API

Once the database connection is established, and the mapping is specified, a user can start using Bibernate in order to implement some data access logic. E.g. **creating new objects**, and **storing them in the DB**, **updating the state of the object**, or even completely **removing the object**. In order to perform those operations, a user should have some form of API.

It's your decision how to name the class and its method, but **make sure you provide** an easy way to perform basics **Create, Read, Update, Remove (CRUD) operations**.

The most obvious read operation is *select by primary key*. But make sure you provide an **API for creating any other custom read operation**.

Relationship Management

Managing relationships between objects (and corresponding DB tables) is probably **the trickiest part of the ORM**. In order to implement it, you will need to answer the following questions:

- When the object is loaded, **should we load its related objects(relations)** as well?
    - If yes, how deep in the relationship graph should we go to avoid loading the whole DB?
    - If not, what should be set to the corresponding object field?
    - Is that possible to lazily load relations?
    - Should we let users configure this and how?
- When an object gets inserted or removed **should we insert/remove its relations** correspondingly?
    - Should we let users configure this and how?
- How to manage objects inheritance?

Implementation Details

Your goal is to build an ORM, and it's up to you how awesome it will be. So **feel free to add features** that you think will be helpful.

Regarding the implementation, I encourage you to take a look at the following books:

- Clean Code – try to apply its principles for this project
- Patterns of Enterprise Application Architecture – use it as a reference book for Object-Relational mapping. In particular, take a look at the following chapters:
    - **Chapter 3: Mapping to Relational Databases**
    - **Chapter 11: Object-Relational Behavioral Patterns**
    - **Chapter 12: Object-Relational Structural Patterns**
    - **Chapter 13: Object-Relational Metadata Mapping Patterns**

Checklist

- Bibernate ORM should provide **a convenient tool for configuration and object mapping**
- Once configured, it should provide **an API that allows storing, fetching, updating, or removing objects** from the database
- If something goes wrong, it should **fail gracefully** and provide **a detailed description of the problem** and/or instructions on how to fix it
- The implementation should **follow clean code rules**, it should be **covered with tests** and provide **useful documentation** (Javadoc)
- The repository should have a **nice and easy-to-understand README** file with instructions for users (maybe even a video tutorial).