Hibernate Introductory Guide

Autorzy: Michał Mrowczyk, Damian Kudas

1. Założenia i cele twórców:

- Hibernate jest frameworkiem do ORM (mapowanie obiektowo-relacyjne).
- Umożliwia m.in. zapisywanie obiektów Javovych jako wierszy w tabeli relacyjnej bazy danych. (np. session.save())
- Umożliwia także łatwe uzyskiwanie obiektów Javovych z danych zapisanych w bazie (np. session.get())
- Oprogramowanie udostępnione na licencji: GNU Lesser General Public License.
- Obsługa niezliczonej ilości silników bazodanowych m.in. PostgreSQL

2. Zalety i wady Hibernate:

2.1 Zalety:

- Eliminuje ogromną ilość pracy jaką trzeba włożyć, aby obsługiwać połączenie z bazą danych przez JDBC (z punktu widzenia programisty)
- Automatyczna konwersja wielu typów (klas) Java na odpowiednie typy danych SQL (np. Date -> timestamp, String -> varchar itd.)

- Architektura warstwowa – możliwość dodawania komponentów w zależności od potrzeb aplikacji
- Mechanizmy wspierające efektywność (lazy loading, second level cache itd.)
- Łatwa konfiguracja, współpraca z adnotacjami JPA (Java Persistence API)
- Popularność, dobra i obszerna dokumentacja.

2.2 Wady:

- Duża ilość API do nauki dla programisty (a co za tym idzie długi czas nauki technologii).
- Wolniejsza technologia niż czyste JDBC (narzut związany z mapowaniem obiektowo – relacyjnym oraz obsługą mechanizmów konfiguracji ustalonej przez użytkownika).

3. Sposób instalacji:

Dokładnie opisany na stronie:

http://www.hibernate.org/downloads

W skrócie:

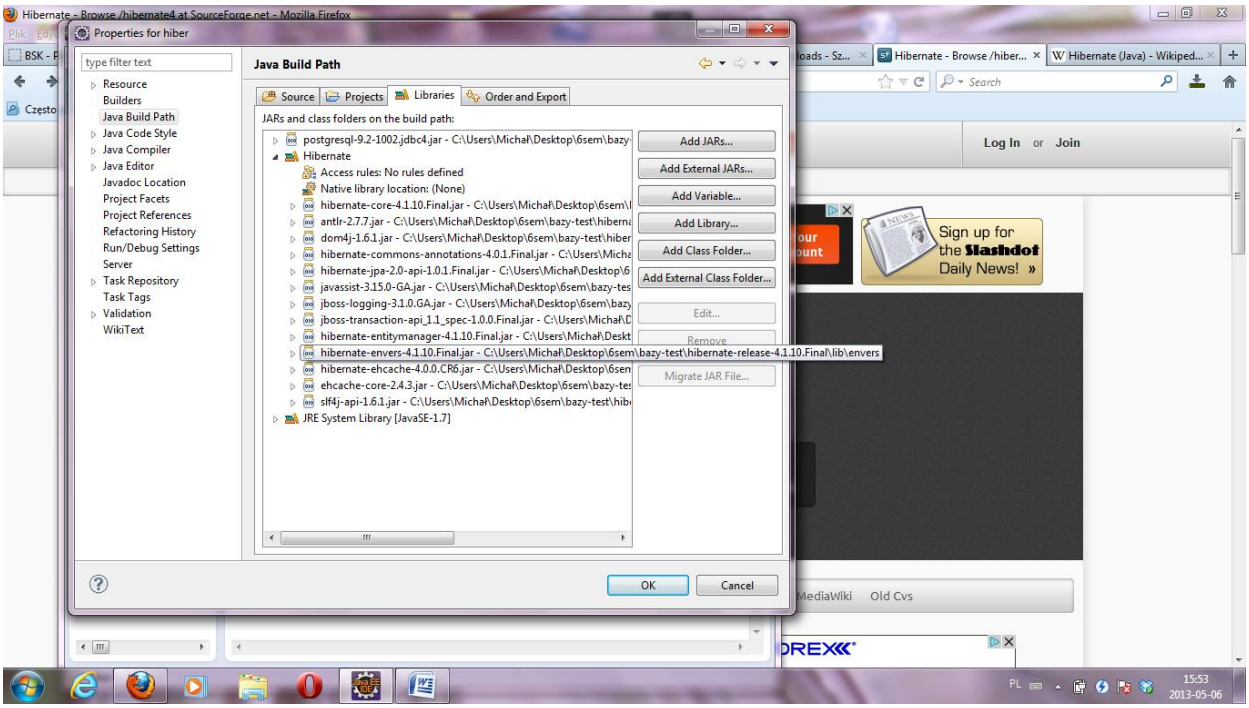
Jeśli korzystamy z Maven'a lub innego narzędzia czarnej magii, to wystarczy ustawić odpowiednie dependencje w pliku POM.xml

W przeciwnym wypadku pobieramy ze strony:

http://sourceforge.net/projects/hibernate/files/hibernate4/

najnowszą paczkę z Hibernate. Zawiera ona niezbędne jary, które musimy dodać do classpath'a naszej aplikacji w Javie.

Przykładowa konfiguracja biblioteki z jarami Hibernate została pokazana na kolejnym screenshocie:



4. Konfiguracja Hibernate:

Jeśli mamy już wszystko na właściwym miejscu tzn. jary, to możemy utworzyć sobie w naszym projekcie plik:

hibernate.cfg.xml

W pliku tym ustawiamy m.in. connection string, jak i różne informacje konfiguracyjne. Przykład:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property
name="hibernate.connection.driver_class">org.postgresql.Driver</property>
          <property
name="hibernate.connection.url">jdbc:postgresql://localhost:5432/db_test</property
>
          <property name="hibernate.connection.username">twit</property>
          <property name="hibernate.connection.password">twit</property>
          <property name="hibernate.connection.pool_size">1</property>
          <property name="hibernate.cache.use_query_cache">true</property>
          <property name="hibernate.cache.use_second_level_cache">true</property>
          <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheReg
ionFactory</property>
```

```
        <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <mapping class="edu.agh.iisg.db.dto.Person"/>
        <mapping class="edu.agh.iisg.db.dto.Customer"/>
        <mapping class="edu.agh.iisg.db.dto.Product"/>
    </session-factory>

</hibernate-configuration>
```

5. Demo:

- Plik hibernate.cfg.xml
- Klasa Person
- HelloHibernate – prosty CRUD
- Klasy Customer oraz Product – przykład OneToMany
- Przykład `RelationsCriteriaHQLazy` – Criteria API oraz HQL do uzyskiwania danych z bazy oraz Lazy vs Eager Fetching