

Hibernate

Hibernate ORM (w skrócie Hibernate) jest biblioteką dla języka Java, zainicjowaną i rozwijaną przez Gavina Kinga, dostarczającą framework zapewniający translację danych z relacyjnej bazy danych na model obiektowy i vice versa (Object/Relational Mapping). Hibernate zastępuje bezpośrednie metody dostępu do danych za pomocą wysokopoziomowych metod operujących na zmapowanych obiektach, co rozwiązuje problem różnicy pomiędzy danymi w bazie a w domenie obiektowej (object-relational impedance mismatch). Hibernate jest darmowym oprogramowaniem, dystrybuowanym pod licencją GNU Lesser General Public Licence. Jego główną funkcjonalnością jest mapowanie z klas Javy na tabele bazy danych (i z "Jawowych" typów danych na "SQLowe"). Oprócz tego wprowadza mechanizmy buforowania i minimalizacji liczby przesyłanych zapytań, zwalnia też dewelopera z konieczności ręcznej obsługi wyników zapytań i konwersji obiektów. **O**

HQL (Hibernate Query Language) jest językiem dostarczonym przez Hibernate, wzorowanym na języku SQL. Pozwala na używanie zapytań, podobnych do tych stosowanych przy użyciu SQLa, jednak zamiast działać na tabelach i kolumnach bazy danych, działa na zmapowanych przez Hibernate obiektach. Dopiero później zapytania HQL są tłumaczone przez Hibernate na zapytania w konwencjonalnym języku SQL działające bezpośrednio na bazie. Przykład:

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query      query      =          session.createQuery(hql);
List results = query.list();
```

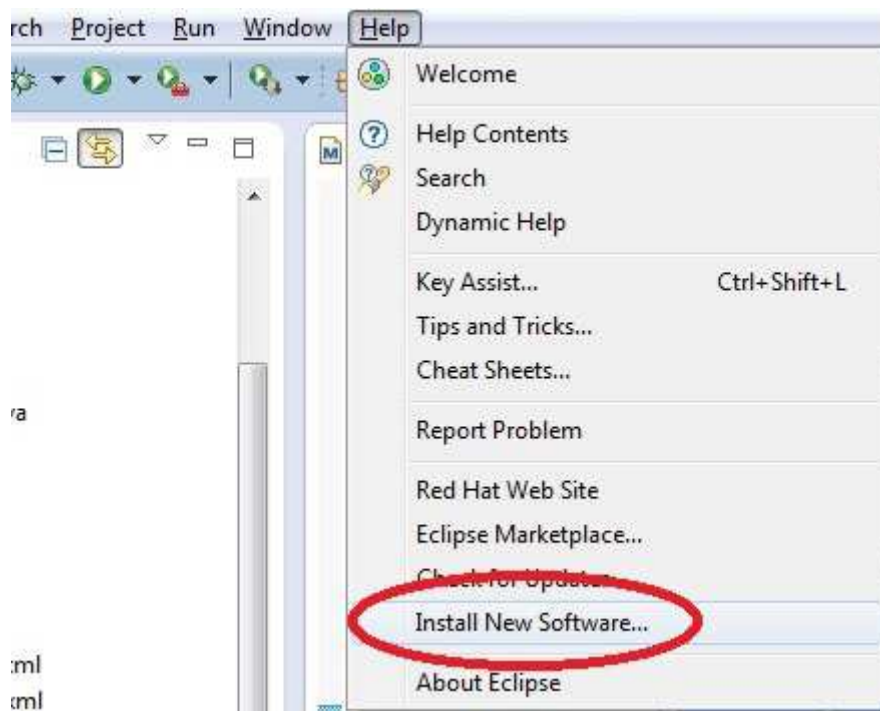
Hibernate Tools jest zestawem zintegrowanych pluginów do środowiska Eclipse. Zawiera między innymi edytor mapujących plików XML, zapewniający podświetlanie składni, podpowiedzi i automatyczne uzupełnianie; konsolę pozwalającą na konfigurowanie połączeń z bazą danych, wizualizację klas i ich relacji, wykonywanie zapytań HQL i przeglądanie ich wyników; a także narzędzia pozwalające na automatyczne wygenerowanie klas na podstawie istniejącej bazy danych (reverse engineering). Hibernate Tools jest podstawowym elementem JBoss Tools, a więc częścią JBoss Developer Studio.

2. Instalacja i konfiguracja

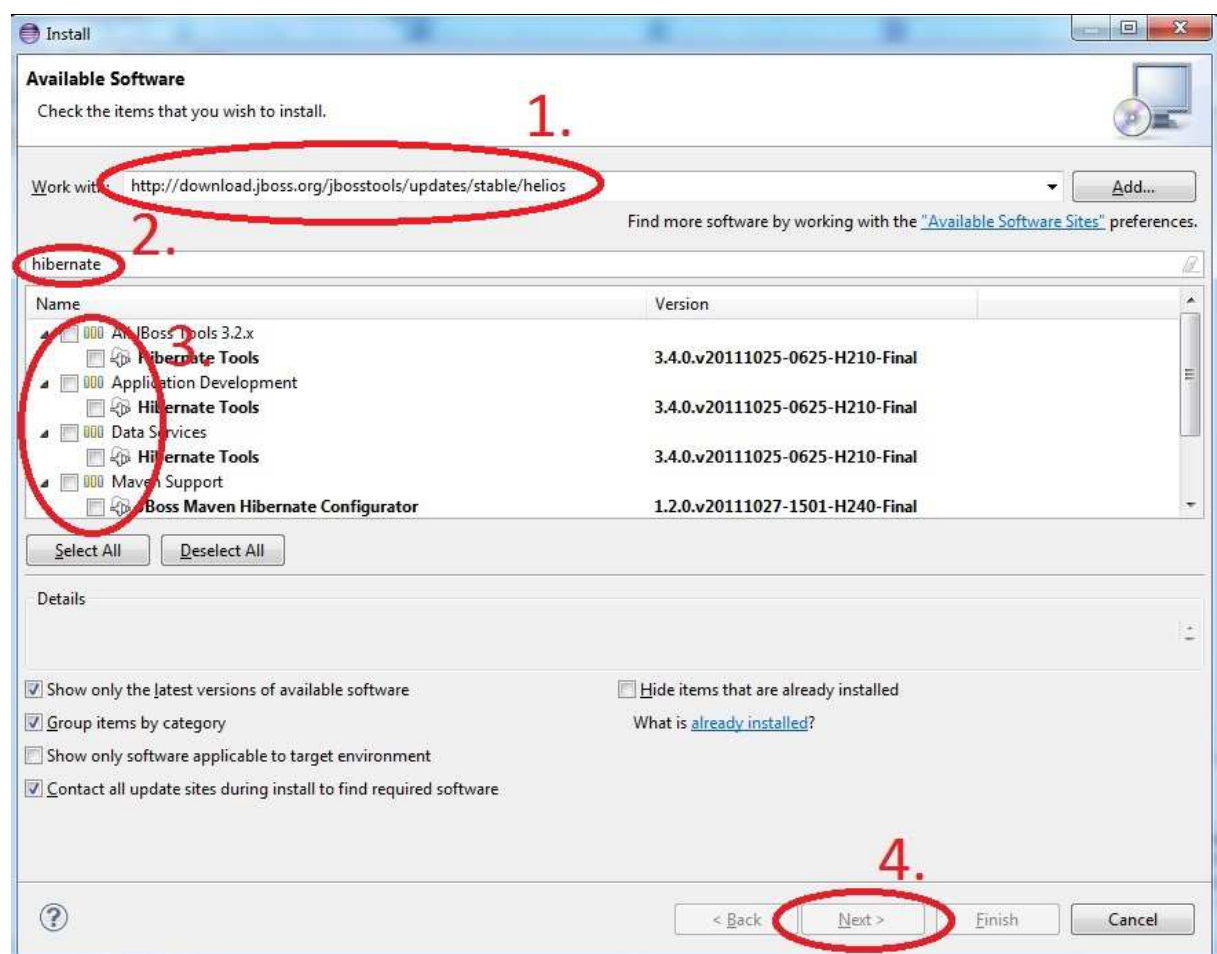
Podczas Naszej przygody z Hibernate'em będziemy korzystać z Hibernate Tools, plugina do Eclipse'a, ułatwiającego konfigurację połączenia. Przyda się on nam głównie ze względu na ułatwiony proces tworzenia plików konfiguracyjnych.

Instalacja

1. W Eclipse instalujemy plugin Hibernate Tools. W tym celu w menu Eclipse'a otwieramy Help, następnie Install New Software...



W otwartym oknie uzupełniamy Work , czekamy, aż pojawią się pluginy. W 'type filter text' wpisujemy Hibernate i zaznaczamy wszystko, co pasuje, następnie klikamy Next.



Postępujemy zgodnie z zaleceniami instalatora. Po instalacji restartujemy IDE. Aby mieć pod ręką klasy Hibernate'a, do każdego tworzonego poma dopisz: <dependency>

```
<groupId>org.hibernate</groupId>  
<artifactId>hibernate-tools</artifactId>  
<version>4.3.1.CR1</version>  
</dependency>
```

Połączenie z bazą

Jako bazę danych można wykorzystać bazę Northwind. Jest to przykładowa baza danych dostarczana przez Microsoft. Do ściągnięcia np. z

<http://sqlserversamples.codeplex.com/#databases>

Lub

<http://www.microsoft.com/en-us/download/details.aspx?id=23654>

3. Generacja klas z bazy danych

Pierwszą rzeczą, jaką spróbujemy zrobić przy pomocy Hibernate'a będzie automatyczna generacja klas jadowych z istniejącej bazy danych, na którą wybraliśmy dobrze znaną bazę Northwind.

3.1 Utworzenie i konfiguracja projektu

Wybieramy File > New > Project > Maven > Maven Project. Projekt tworzymy wg wzorca quickstart. Nazywamy go wedle uznania.

Powinniśmy otrzymać szkielet projektu z przykładową klasą App, oraz plikiem pom.xml (będziemy go potrzebować później!).

Jeżeli wszystko jest w porządku, pora stworzyć pliki konfiguracyjne Hibernate'a. W tym celu użyjemy zainstalowanych przez Nas Hibernate Tools. Wchodzimy w Window > Open Perspective > Other... i wybieramy Hibernate. To powinno otworzyć zakładkę Hibernate Configurations w Project Explorerze. Jeżeli nie możesz znaleźć perspektywy Hibernate, upewnij się, że poprawnie zainstalowałeś Hibernate Tools i zrestartowałeś Eclipse.

Na razie nie przejmujemy się zakładką Hibernate Configurations. Będzie ona nam służyła później do podglądnięcia bazy danych. Lecz aby to było możliwe, musimy najpierw skonfigurować połączenie do tejże bazy. W tym celu, klikamy prawym przyciskiem myszy na Nasz projekt, wybieramy New > Hibernate Configuration File. Jak zauważyliście, oprócz niego jest tam też kilka innych Hibernate'owych pozycji - tymi zajmiemy się później, na razie skupiamy się na głównym pliku konfiguracyjnym. Po kliknięciu wspomnianej pozycji otrzymujemy okno wizarda. W pierwszym kroku możemy wybrać nazwę pliku. Najlepiej trzymać się konwencji - hibernate.cfg.xml lub Hibernate.cfg.xml. Następnie wybieramy parametry połączenia, tak jak na rysunku.

Hibernate Configuration File (cfg.xml)

This wizard creates a new configuration file to use with Hibernate.

Container: /hibernate-tutorial/src

File name: Hibernate.cfg.xml

Session factory name:

Database dialect: SQL Server

Driver class: net.sourceforge.jtds.jdbc.Driver

Connection URL: jdbc:jtds:sqlserver://localhost:11433

Default Schema:

Default Catalog:

Username: student

Password: student

☐ Create a console configuration

< Back Next > Finish Cancel

Dialekt - SQL Server, ponieważ taki jest dialekt Northwind.

Driver - tu mamy kilka opcji - możemy użyć Drivera Microsoftu, ale jeżeli używamy Mavena, wygodniej będzie wykorzystać net.sourceforge.jtds, ponieważ wtedy nie musimy ściągać jara, a jedynie wstawiamy następującą zależność do pomu:

```
<dependency>
    <groupId>net.sourceforge.jtds</groupId>
    <artifactId>jtds</artifactId>
    <version>1.3.1</version>
</dependency>
```

Maven powinien ją dla nas znaleźć i pobrać.

Jeżeli chcemy użyć drivera Microsoftu, jara dostaniemy pod <http://www.microsoft.com/en-us/download/details.aspx?id=11774> - rozpakowujemy i wkładamy wersję 4 pod Classpath.



Connection URL - wspomniane localhost:11433 z automatycznym prefiksem określającym sterownik.

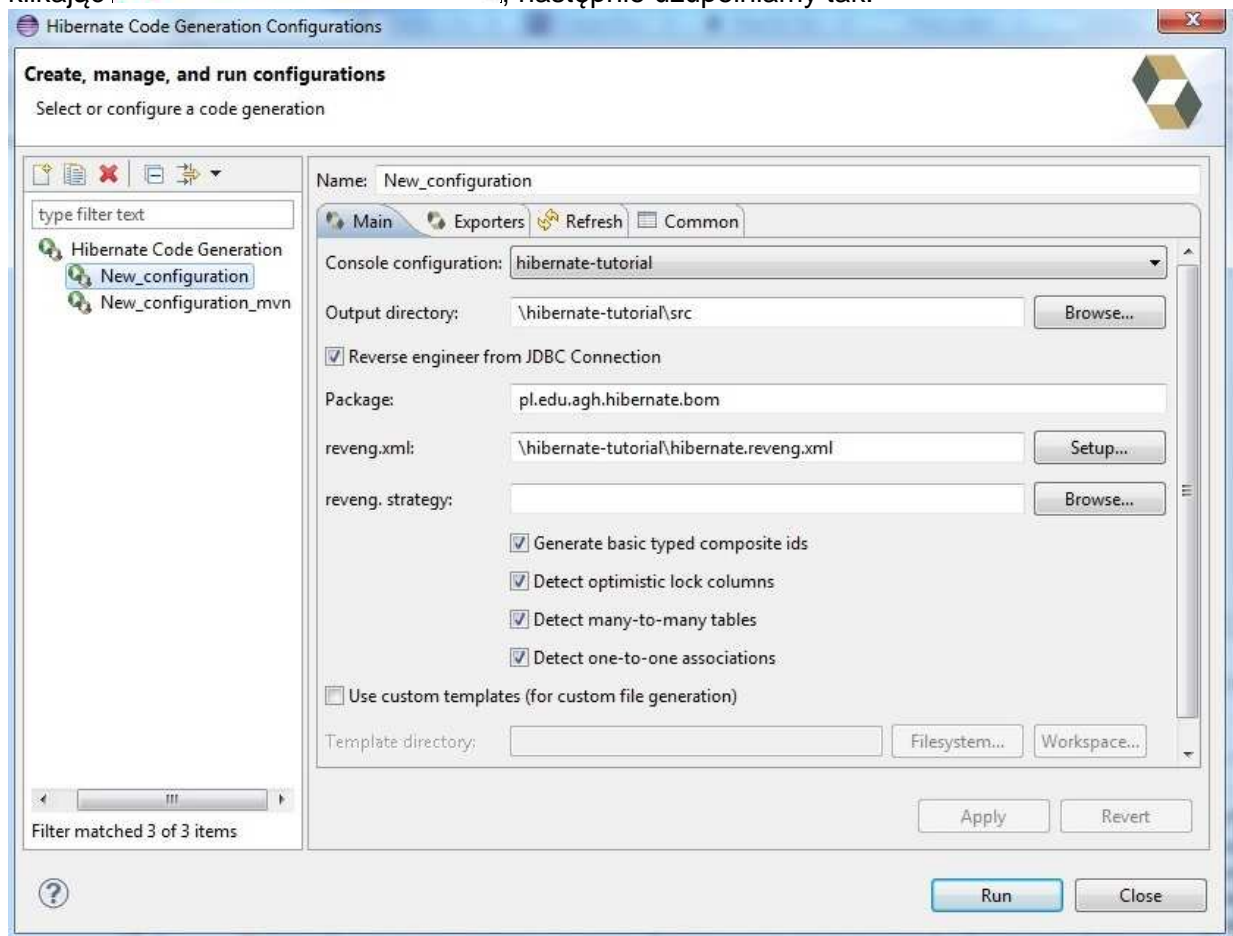
Username, Password - student, student - wystarczy nam do czytania zawartości bazy.

Zaznaczamy **Create a console configuration**, klikamy **Next**. Jedyne co specyfikujemy to nazwę konfiguracji i obecny projekt. Klikamy **Finish**.

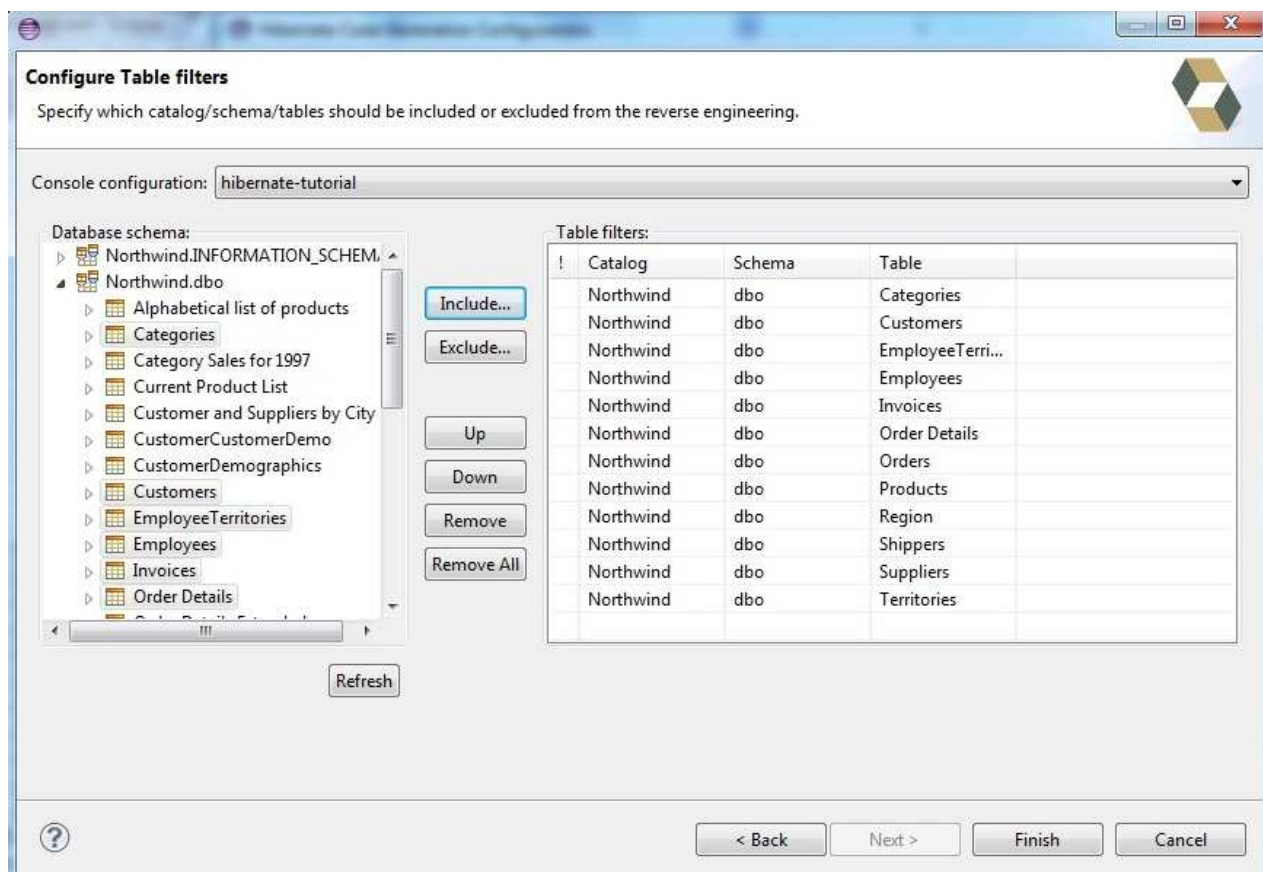
Jeżeli niczego nie spapraliśmy, powinniśmy teraz być w stanie podglądać Naszą bazę z zakładki Hibernate Configurations > <projekt> > Database - w ten sposób będziemy wiedzieć, że połączenie jest prawidłowo skonfigurowane.

3.2 Reverse engineering - generacja klas

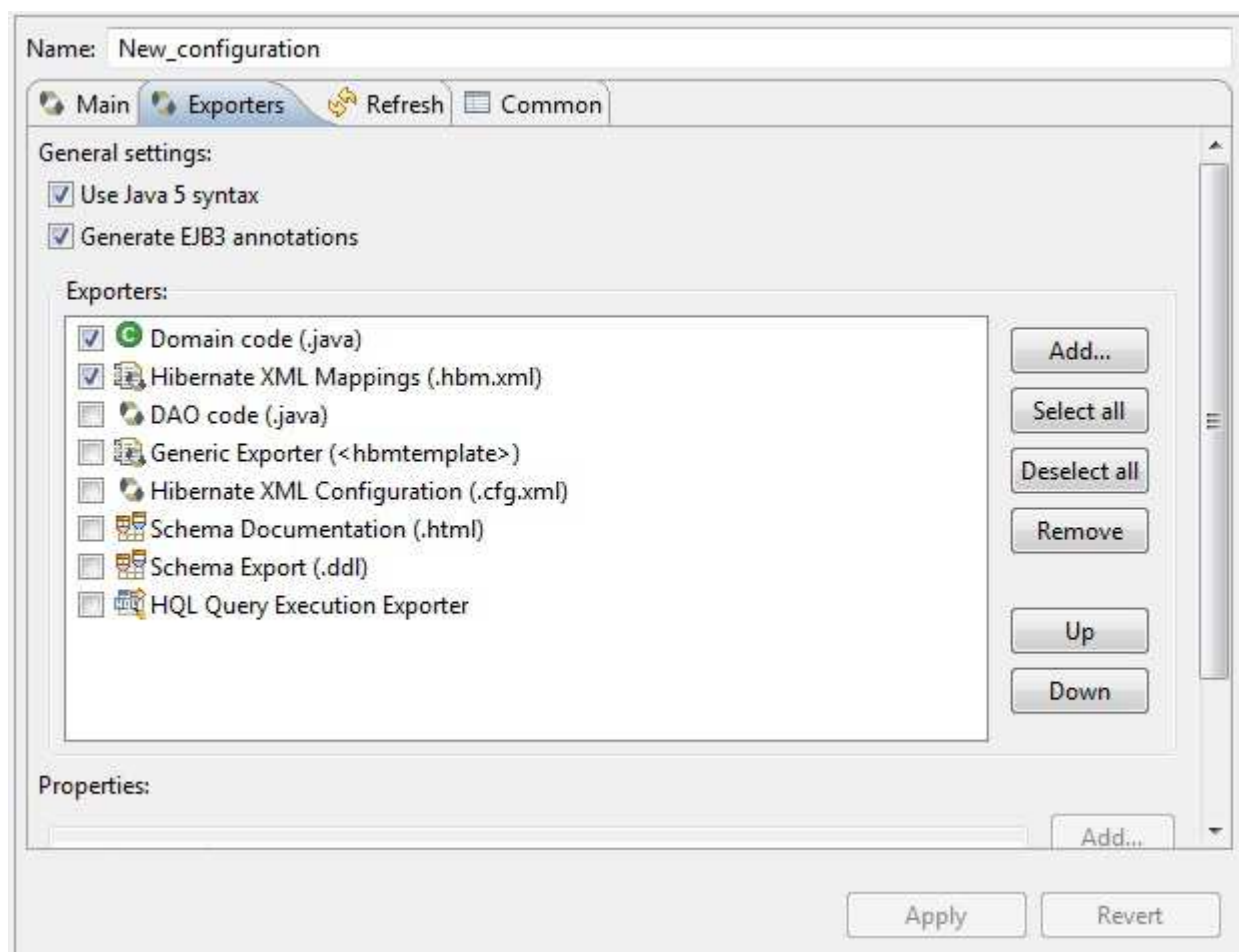
Pora wygenerować klasy. W tym celu klikamy sobie tu  i na Hibernate Code Generation Configuration. Tworzymy nową konfigurację, klikając , następnie uzupełniamy tak:



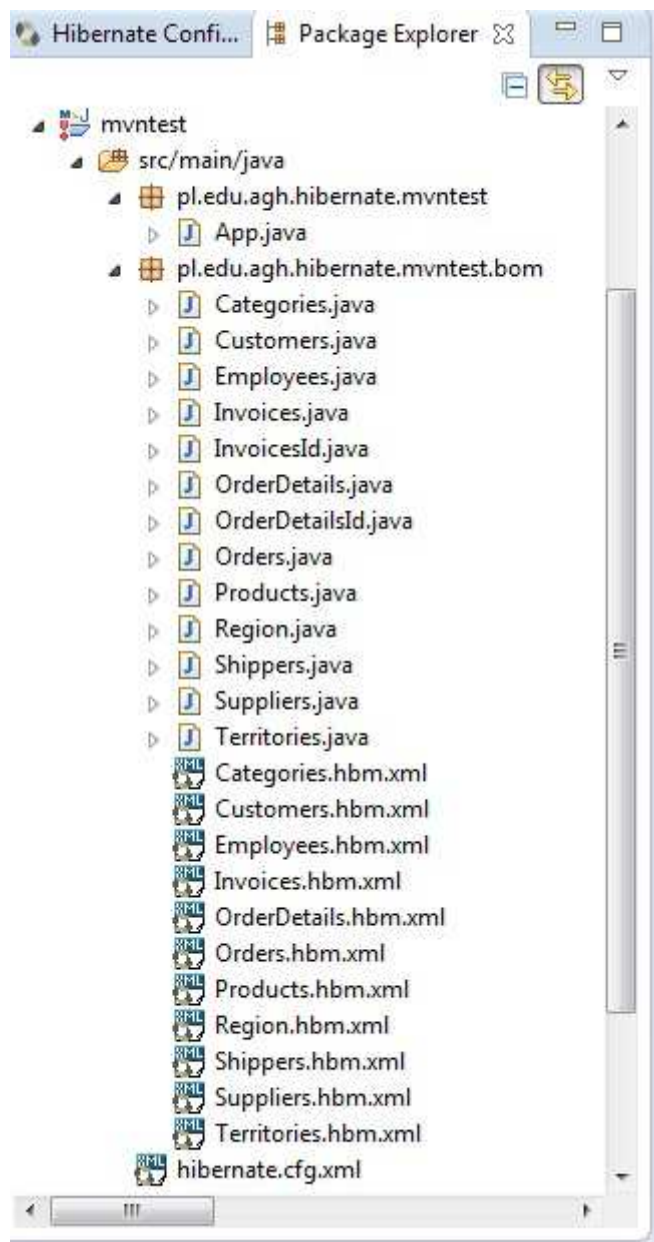
Tu podajemy nasz projekt (Console configuration), zaznaczamy **Reverse engineer from JDBC Connection**, podajemy katalog, do którego Hibernate ma wrzucić nasze zmapowane tabele jak i ich package. Warto ustawić katalog na <projekt>/src/main/java - w ten sposób Eclipse nie będzie miał problemu ze znalezieniem klas. Przy specyfikowaniu *reveng.xml* tworzymy nowy plik przez Setup -> Create new, w osobnym menu określając jego nazwę (znow, najlepiej zostawić domyślną) a następnie tabele bazodanowe, które chcemy zmapować na klasy - możemy to zrobić np. tak (pamiętajmy o kliknięciu **Refresh** - niestety może to trochę trwać):



Wracając do głównego okna konfiguracyjnego wchodzimy w zakładkę **Exporters**, gdzie zaznaczamy **Use Java 5 syntax**, **Generate EJB3 annotations**, **Domain code (.java)** oraz, opcjonalnie, **Hibernate XML Mappings** (pliki XML, pozwalające podejrzeć szczegóły mapowania).



Gotowe! Możemy kliknąć Apply, a następnie Run. Przy odrobinie szczęścia nic nie wybuchnie, ani nie poleci jakiś większy wyjątek, a w podanym przez nas pakiecie pojawią się klasy zmapowane z tabel Northwinda.



3.3 Wynik generacji

Oprócz samych klas pojawiły się także pliki XML mapowania, a także adnotacje Hibernate'owe - tak naprawdę są to dwa oblicza tego samego i z reguły używa się jednego albo drugiego, ale dla celów edukacyjnych warto podejrzeć, jak wyglądają obie formy konfiguracji.

Zadanie: Sprawdź w jaki sposób Hibernate opisuje relacje między tabelami w pliku XML, a w jaki w postaci adnotacji. Na jaki typ mapowane są relacje?

4. Mapowanie obiektów na tabele

Zobaczyliśmy już, jak za pomocą Hibernate'a z bazy danych otrzymać klasy javowe. Teraz będziemy dokonywać odwrotnej operacji - stworzone przez siebie klasy będziemy zapisywać w lokalnej bazie w postaci tabel. W celu zobrazowania tej funkcjonalności, najpierw postawimy naszą własną bazę danych MySQL.

4.1 Instalacja i uruchomienie MySQL

- Pobieramy instalację z <http://dev.mysql.com/downloads/mysql/> - niestety konieczne jest utworzenie darmowego konta.
- Instalujemy w wybranym przez nas folderze. Możemy dodać /bin do PATH, ale nie jest to konieczne - jeśli nie, pamiętajmy o wywoływaniu komend z katalogu bin.
- Uruchamiamy z poziomu linii komendę `mysqld --skip-grant-tables` (nie chcemy zawracać sobie głowy uprawnieniami). W ten sposób uruchomiliśmy serwer na porcie 3306.
- Okno zostawiamy otwarte. W osobnym oknie uruchamiamy `mysql` - klienta MySQL. Dla celów tutoriala tworzymy bazę HIBDB komendą `create database hibdb;` Klienta również zostawmy, w celu późniejszej weryfikacji operacji zrobionych przez Hibernate.

4.2 Mapowanie prostego obiektu

Teraz zmapujemy jakiś prosty, pojedynczy obiekt na tabelę w bazie. Przez ustawienie odpowiednich opcji będziemy w stanie podglądać, jak Hibernate wykonuje taką operację.

- W Eclipse tworzymy nowy (dla klarowności) projekt Maven;
- tworzymy klasę Event, lub jakąś inną prostą klasę, spełniającą kryteria:
 - ma bezargumentowy konstruktor
 - ma settery oraz gettery do wszystkich pól, które chcemy udostępnić Hibernate'owi
 - ma pole Long id - posłuży ono do generowania klucza głównego tabeli

```
public class Event {  
    private Long id;  
    private String title;  
    private Date date;  
  
    ...(no-arg constructor, getters and setters)...  
}
```

- Tworzymy plik mapowania - podajemy utworzoną klasę w wizardzie. W ten sposób dostajemy prawie gotowy plik mapowania z informacją o polach klasy i odpowiadających kolumnach. Musimy jedynie zmienić generator class w znaczniku id na **“native”** zamiast **“assigned”** - w ten sposób Hibernate sam będzie generował wartości klucza głównego dla rekordów w tabeli.
- Tworzymy nowy plik konfiguracyjny hibernate.cfg.xml, tak jak wcześniej, tym razem podając parametry:
 - **Database dialect** - MySQL
 - **Driver class** - com.mysql.jdbc.Driver
 - **Connection URL** - jdbc:mysql://127.0.0.1:3306/HIBDB
 - **Username** - ODBC/<login>
 - **Password** - <puste>

Do wygenerowanego pliku dopisujemy kilka linii:

```
<property name="show_sql">true</property>
```

jeżeli chcemy zobaczyć komendy SQL, które Hibernate wykonuje przy operacjach na bazie.

```
<property name="hbm2ddl.auto">create-drop</property>
```

przy każdym uruchomieniu nasza baza będzie rekreowana - zapobiegniemy bałaganowi.

```
<property
```

```
name="hibernate.current_session_context_class">thread</property>
```

kontekst sesji, potrzebny do zarządzania połączeniem z bazą.

```
<mapping resource="pl/edu/agh/hibernate/mapping/Event.hbm.xml"/>
```

plik klasy, którą będziemy mapować.

Ponadto, w celu połączenia z MySQL potrzebujemy klasy Drivera - dopisujemy do poma:

```
<dependency>
```

```
<groupId>mysql</groupId>
```

```
<artifactId>mysql-connector-java</artifactId>
```

```
<version>5.1.9</version>
```

```
</dependency>
```

Tworzymy klasę EventManager, której zadaniem będzie zapisać do bazy przykładowy obiekt klasy Event:

```
public class EventManager {
    private static SessionFactory getSessionFactory() {
        Configuration conf = new Configuration()
            .configure();
        ServiceRegistry registry = new
StandardServiceRegistryBuilder()
            .applySettings(conf.getProperties()).build(
);
        SessionFactory factory =
conf.buildSessionFactory(registry);
        return factory;
    }
    public static void main(String[] args) {
        Event event = createAndStoreEvent("Test", new Date());
    }
    private static Event createAndStoreEvent(String title, Date date) {
        Session session = getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Event theEvent = new Event();
        theEvent.setTitle(title);
        theEvent.setDate(date);
        session.save(theEvent);
        session.getTransaction().commit();
        return theEvent;
    }
}
```

Metoda getSessionFactory() odpowiada za utworzenie fabryki sesji. Sesja odpowiada za zarządzanie transakcją, wewnątrz której będziemy wykonywać nasze operacje na bazie. Odpalamy klasę EventManager (lub Twoją własną) i obserwujemy logi. Jeżeli poleciał jakiś wyjątek, sprawdź, czy pliki XML zostały na pewno poprawnie skonfigurowane. Jeśli tak, powinieneś zobaczyć komendy SQL, które Hibernate wywołuje, by wykonać nasze operacje. Powinno to wyglądać mniej więcej tak:

```
Hibernate: drop table if exists EVENT
```

```
Hibernate: create table EVENT (ID bigint not null auto_increment,
TITLE varchar(255), DATE datetime, primary key (ID))
```

```
Hibernate: insert into EVENT (TITLE, DATE) values (?, ?)
```

Aby zweryfikować, że faktycznie zapisaliśmy coś do bazy, wykonujemy komendy w uruchomionym w tle kliencie MySQL:

```
mysql> use hibdb
```

```
mysql> show tables; // powinno pokazać m.in. tabelę event
```

```
mysql> select * from event; // powinno pokazać nowo wpisany wiersz
odpowiadający naszemu obiektowi
```

4.3 Mapowanie powiązań między obiektami

Teraz spróbujemy mapowania dwóch powiązanych ze sobą obiektów - stworzymy relację "wiele do wielu" i zobaczymy, jak łatwo ją modyfikować.

4.3.1 Tworzenie relacji

W tym celu dokładamy klasę Person:

```
public class Person {
    private Long id;
    private int age;
    private String firstName, lastName;
    private Set events = new HashSet();
    ...(no-arg constructor, getters and setters for the fields)... }
```

Dla klasy Person generujemy plik mapowania. Dodajemy go do głównego pliku

hibernate.cfg.xml <mapping

resource="pl/edu/agh/hibernate/mapping/Person.hbm.xml" />

Niestety musimy ten plik troszkę zmodyfikować, aby dało się uzyskać relację między tabelami.

```
<id name="id" type="java.lang.Long">
    <column name="PERSON_ID" />
    <generator class="native" />
</id>
...
<set name="events" table="PERSON_EVENT">
    <key column="PERSON_ID" />
    <many-to-many column="EVENT_ID"
class="pl.edu.agh.hibernate.mapping.Event" />
</set>
```

ponadto, w Event.hbm.xml:

```
<id name="id" type="java.lang.Long">
    <column name="EVENT_ID" />
    <generator class="native" />
</id>
```

Co się stało?

- zmieniliśmy nazwy kolumn identyfikatorów, tak, by miały inne nazwy (dla tabeli łącznikowej);
- zmieniliśmy pole events tak, by odwzorowywane było na relację many-to-many z klasą Event z użyciem tabeli łącznikowej PERSON_EVENT;

Do klasy EventManager dopisujemy:

```
private static Person createAndStorePerson(String firstName, String
lastName, int age, Event event) {
    Session session = sessionFactory().getCurrentSession();
    session.beginTransaction();

    Person person = new Person();
    person.setFirstName(firstName);
    person.setLastName(lastName);
    person.setAge(age);
    Set<Event> events = new HashSet<Event>();
    events.add(event);
    person.setEvents(events);
    session.save(person);
}
```

```

        session.getTransaction().commit();
        return person;
    }

    , main:
    public static void main(String[] args) {
        Event event = createAndStoreEvent("Tworzenie tutoriala", new
Date());
        Person person = createAndStorePerson("Jan", "Kowalski", 20, event);
    }

```

Zmieniamy również `hbm2dll.auto` na `update` (mamy tu 2 sesje i nie chcemy, żeby po zapisaniu eventu, został on usunięty).

Po uruchomieniu w bazie powinny pojawić się trzy tabele: **Person**, **Event** oraz **Person_Event**, zawierająca powiązanie między dwoma stworzonymi instancjami.

4.3.2 Przechodzenie po relacji w obiektach

Stworzyliśmy relację wiele do wielu. Załóżmy, że chcemy teraz do istniejącego Eventu dołożyć nową osobę - jak to zrobić?

Okazuje się to bardzo proste (`personId` i `eventId` to klucze główne wierszy w tabelach **Person** i **Event**):

```

private static void addPersonToEvent(Long personId, Long eventId) {
    Session session = getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Person aPerson = (Person) session.load(Person.class, personId);
    Event anEvent = (Event) session.load(Event.class, eventId);
    aPerson.getEvents().add(anEvent);
    session.getTransaction().commit();
}

```

Po prostu po wczytaniu obiektu **Person**, wywołujemy na nim gettera do kolekcji, która mapuje się na relację, po czym dokładamy do niej nowy obiekt **Event**.

Po zweryfikowaniu, w tabeli łącznikowej powinien pojawić się nowy wiersz.

4.4 Mapowanie hierarchii dziedziczenia

Hibernate wspiera trzy rozdzielne strategie mapowania hierarchii dziedziczenia. Te strategie to:

- `table per class hierarchy` (jedna tabela bazodanowa na całą hierarchię dziedziczenia);
- `table per subclass` (jedna tabela dla nad-klasy, posiadająca klucze obce do osobnych tabel dla każdej pod-klasy);
- `table per concrete class` (jedna tabela dla każdej konkretnej klasy).
- `strategie mieszane` - możemy mieszać powyższe trzy rozwiązania w obrębie drzewa dziedziczenia. Nie będziemy się tu zajmować tą opcją, po szczegóły zapraszamy do odnośników.

Najlepiej pokazać to na przykładzie. Stwórzmy przykładową klasę **Shape**:

```

public class Shape {
    protected long id;
    protected String name;
}

```

oraz dziedziczące po nim dwie klasy: **Triangle** oraz **Rectangle**:

```

public class Rectangle extends Shape {
    private int sideALength;
    private int sideBLength;
}

```

```

}
public class Triangle extends Shape {
    private int baseLength;
    private int height;
}

```

Nie zapomnij o dodaniu bezargumentowych konstruktorów oraz getterów i setterów! Zmień również hbm2dll.auto na create-drop.

Metoda main prezentuje się bez niespodzianek - tworzymy po jednym obiekcie z każdej klasy i zapisujemy je do bazy.

4.4.1 Table per subclass

Różnica pojawi się w plikach mapowania. Domyślnie Hibernate Tools generuje dla Nas pliki realizujące politykę **table per subclass**. Strategia jest identyfikowana przez znacznik `joined-subclass` w implementacjach.

```

<joined-subclass name="pl.edu.agh.hibernate.hierarchy.Triangle"
extends="pl.edu.agh.hibernate.hierarchy.Shape" lazy="false">

```

Zamieniamy standardowo w Shape.hbm.xml generator `class` na `native`.

Dodajemy również wygenerowane pliki mapowania do hibernate.cfg.xml:

```

<mapping resource="pl/edu/agh/hibernate/hierarchy/Shape.hbm.xml" />
<mapping resource="pl/edu/agh/hibernate/hierarchy/Triangle.hbm.xml"
/>
<mapping resource="pl/edu/agh/hibernate/hierarchy/Rectangle.hbm.xml"
/>

```

Po zrobieniu tego i odpaleniu maina powinniśmy w bazie otrzymać trzy tabele. Tabela Shape zawiera część pól obiektów podklas, które są odziedziczone z nad-klasy, natomiast pozostałe tabele zawierają pola 'dodane' w podklasach. Jest to częste rozwiązanie w przypadku mapowania hierarchii dziedziczenia.

4.4.2 Table per class hierarchy

Aby zmienić strategię na **table per class hierarchy**, zmieniamy, co następuje:

- W mapowaniu interfejsu dodajemy pole dyskryminatora: `<discriminator column="SHAPE_TYPE" type="string"/>`;
- W mapowaniu podklas zmieniamy `joined-subclass` na `subclass`, zmieniamy `table` na `discriminator-value`, usuwamy `<key column="ID"/>`.

Po odpaleniu (najpierw najlepiej wyczyścić albo zmienić bazę) dostajemy tym razem tylko tabelę **Shape**. Zawiera ona wszystkie pola, które mogą się pojawić w obiekcie klasy Shape bądź potomnych. Stąd często pojawiają się NULLE.

4.4.4 Table per concrete class

Wreszcie, aby otrzymać strategię **table per concrete class**, zmieniamy:

- W Shape.hbm.xml zmieniamy generator na `hilo` (generuje on globalnie unikatowe klucze dla rozłącznych tabel);
- W mappingach podklas zmieniamy `joined-subclass` na `union-subclass`, usuwamy `discriminator-value`.

Po uruchomieniu maina powinniśmy otrzymać tym razem 3 tabele, każda zawierająca daną klasę w całości.

4.5 Adnotacje

Strategię możemy również określać w adnotacjach:

```
@Entity @Inheritance(strategy=InheritanceType.SINGLE_TABLE) @DiscriminatorColumn( name="planetype" , discriminatorType=DiscriminatorType.STRING ) @DiscriminatorValue( "Plane" )
```

Ogólnie, adnotacje dają nam kontrolę na niemal takim samym poziomie, co pliki XML. Wiele osób jednak stoi po stronie XML, argumentując, że pozwala on nam na oddzielenie właściwego kodu aplikacji od jej konfiguracji, co wzmacnia czytelność. Wybór jednego bądź drugiego jest kwestią naszych preferencji (bądź polityki firmy). W tym tutorialu skupiamy się na XML-u, ponieważ wspiera go Hibernate Tools.

7. Referencje

Warto choć zerknąć na następujące materiały:

- <http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/> - oficjalny tutorial do Hibernate'a, z którego po części korzystaliśmy przy tworzeniu niniejszego. Można go traktować jako kolejny krok na Naszej drodze ku mistrzowskiemu opanowaniu tego frameworka - w dogłębny sposób opisuje poruszone przez nas aspekty, jak i wiele innych. Niestety w niektórych miejscach może okazać się trochę przestarzały.
- <http://www.wikihow.com/Generate-Hibernate-Pojo-Classes-from-DB-Tables> - jak wygenerować klasy z bazy przy pomocy Hibernate Tools.
- <http://stackoverflow.com/> - w razie problemów, które niestety lubią się pojawić