Konfiguracja Hibernate

Hibernate to rozwiązanie bazujące na metodzie odwzorowania obiektowej architektury systemu informatycznego na inny system o relacyjnym charakterze. W skrócie ORM (*Object-Relational Mapping*).

Pierwszym krokiem jest utworzenie pliku konfiguracyjnego dla Hibernate. Tworzymy plik o nazwie hibernate.cfg w katalogu src/main.java. Plik ten powinien mieć postać:

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5 <hibernate-configuration>
6 <session-factory>
7 <propertyname="connection.url">jdbc:mysql://localhost/Hibernate</property>
8 <property name="connection.username">root</property>
9 <property name="connection.password"></property>
10                  <propertyname="connection.driver_class">com.mysql.jdbc.
                    Driver</property>
1  <propertyname="dialect">org.hibernate.dialect.MySQL5InnoDBDialect</proper
1  ty>
12 <property name="show_sql">true</property>
13 <property name="format_sql">true</property>
14 <property name="hbm2ddl.auto">create</property>
15 <!-- JDBC connection pool (use the built-in) -->
16 <property name="connection.pool_size">1</property>
17 <propertyname="current_session_context_class">thread</property>
18 </session-factory>
19 </hibernate-configuration>
```

Komentarz do następujących linijek:

- określa adres hosta i nazwę bazy danych:

```
1 <property name="connection.url">jdbc:mysql://localhost/Osoby</property>
```

- określa nam login i hasło do bazy danych:

```
1 <property name="connection.username">root</property>
2 <property name="connection.password"></property>
```

- określa nam dialekt który odpowiada z kolei za tłumaczenie zapytań Hibernate na zapytania SQL przeznaczone dla konkretnego systemu zarządzania bazą danych:

```
1 <property
name="dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
```

tabela tworzona od nowa, automatycznie za każdym razem, nie jest to konieczne.

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

Również w katalogu src/main/java tworzymy tym razem plik log4j.properties. Zawartość pliku przedstawia się następująco:

```
1 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
2 log4j.appender.stdout.Target=System.out
3 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4 log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n
5 log4j.rootLogger=warn, stdout
6 log4j.logger.org.hibernate=info
7 log4j.logger.org.hibernate.type=info
8 log4j.logger.org.hibernate.tool.hbm2ddl=debug
```

Hibernate został skonfigurowany do pracy na bazie danych o nazwie „Hibernate”.

Mapowanie obiektów klasy ‚Klient ‚na tabelę ‚Klient’

W konfiguracji Hibernata podano nazwę bazy danych „Hibernate”. Twórzmy, więc przy pomocy Hibernata pierwszą tabelę. Tabela nosić będzie nazwę „Klient”. W tym celu tworzymy następującą klasę JAVA w stylu POJO (Plain Old Java Object):

```
1 package test.Hibernate;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Klient {
9 @Id
10  @GeneratedValue
11 private int id;
12 private String imie;
13 private String nazwisko;
14 public int getId() {
15 return id;
16 }
```

```
17 public void setId(int id) {
18 this.id = id;
19 }
20 public String getImie() {
21 return imie;
22 }
23 public void setImie(String imie) {
24 this.imie = imie;
25 }
26 public String getNazwisko() {
27 return nazwisko;
28 }
29 public void setNazwisko(String nazwisko) {
30 this.nazwisko = nazwisko;
31 }
32 }
```

Powyższa klasa została zaopatrzona w adnotacje, otóż aby klasy języka JAVA zostały prawidłowo odwzorowane należy użyć następujących oto adnotacji (nie jest to jedyna możliwość, można także użyć plików *.XML):

```
1 @Entity - Klasa reprezentuję encję.
2 @Id - Pole jest kluczem głównym.
3 @GeneratedValue - Auto numeracja danego pola.
```

W pliku App.java dodajemy w ciele metody main następujący oto kod pamiętając o zaimportowaniu potrzebnych bibliotek:

```
1 import org.hibernate.cfg.AnnotationConfiguration;
2 import org.hibernate.tool.hbm2ddl.SchemaExport;
1 AnnotationConfiguration config = newAnnotationConfiguration();
2 config.addAnnotatedClass(Klient.class);
3 config.configure("hibernate.cfg.xml");
4 new SchemaExport(config).create(true, true);
```

Uruchamiamy projekt, jeśli wszystko poszło w porządku to Hibernate utworzył nam tabelę Klient w bazie danych Hibernate. Oznacza to, że nie musimy sami już tworzyć tabel, Hibernate zrobił to za nas.

Mapowanie tabeli ‚Klient’ na obiekty klasy ‚Klient’

Dodamy kilka rekordów odwzorowując utworzoną w bazie danych tabelę Klient na klasę. W tym celu utwórzmy najpierw klasę HibernateSessionFactory w pakiecie test.Hibernate. Klasa ta ma postać:

```
1 package test.Hibernate;
2
3 import org.hibernate.SessionFactory;
4 import org.hibernate.cfg.AnnotationConfiguration;
5
6 public class HibernateSessionFactory {
7 private static final SessionFactory sessionFactory;
8
9  static {
10 try {
11 sessionFactory = new
12 AnnotationConfiguration().configure().buildSessionFactory();
13 } catch (Throwable ex) {
14 System.out.println("Błąd w inicjalizacji SessionFactory" + ex);
15 throw new ExceptionInInitializerError(ex);
16 }
17 }
18
19 public static SessionFactory getSessionFactory() {
20 return sessionFactory;
21 }
22
23 }
```

Uzupełnijmy teraz plik konfiguracyjny hibernate.cfg.xml o następujący wpis oznaczający mapowanie tabeli Klient:

```
1 <mapping class="test.Hibernate.Klient"/>
```

Samą klasę klient uzupełniamy o adnotacje @Column i @Table. Całość zamieszczona poniżej:

```
1 package test.Hibernate;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.Id;
7 import javax.persistence.Table;
8
9 @Entity
10    @Table(name = "klient")
```

```
11 public class Klient {
12 @Id
13 @Column(name = "id")
14 @GeneratedValue
15 private int id;
16 @Column(name = "imie")
17 private String imie;
18 @Column(name = "nazwisko")
19 private String nazwisko;
20 public int getId() {
21 return id;
22 }
23 public void setId(int id) {
24 this.id = id;
25 }
26 public String getImie() {
27 return imie;
28 }
29 public void setImie(String imie) {
30 this.imie = imie;
31 }
32 public String getNazwisko() {
33 return nazwisko;
34 }
35 public void setNazwisko(String nazwisko) {
36 this.nazwisko = nazwisko;
37 }
38 }
```

Ostatnim krokiem jest przetestowanie naszej aplikacji, w pliku App.java zamieszczany taki oto kod:

```
1 package test.Hibernate;
2
3 import org.hibernate.Session;
4 import org.hibernate.SessionFactory;
5 import org.hibernate.Transaction;
6
7 /**
8 * Hello world!
9 *
```

```
10 */
11 public class App
12 {
13 public static void main( String[] args )
14 {
15 /** Getting the Session Factory and session */
16 SessionFactory session = HibernateSessionFactory.getSessionFactory();
17 Session sess = session.getCurrentSession();
18 /** Starting the Transaction */
19 Transaction tx = sess.beginTransaction();
20 /** Creating Pojo */
21 Klient klient = new Klient();
22 klient.setId(new Integer(5));
23 klient.setImie("Jan");
24 klient.setNazwisko("Kowalski");
25 /** Saving POJO */
26 sess.save(klient);
27 /** Commiting the changes */
28 tx.commit();
29 System.out.println("Record Inserted");
30 /** Closing Session */
31 session.close();
32 }
33
34 }
```

W tym momencie na konsoli po uruchomieniu projektu powinniśmy uzyskać efekt informujący o poprawności dodania rekordu do utworzonej tabeli w bazie danych:

```
1 21:47:32,603  INFO Version:15 - Hibernate Annotations 3.4.0.GA
2 21:47:32,649  INFO Environment:543 - Hibernate 3.3.0.SP1
3 21:47:32,662  INFO Environment:576 - hibernate.properties not found
4 21:47:32,671  INFO Environment:709 - Bytecode provider name : javassist
5 21:47:32,682  INFO Environment:627 - using JDK 1.4 java.sql.Timestamp
handling
6 21:47:32,857  INFO Version:14 - Hibernate Commons Annotations 3.1.0.GA
7 21:47:32,868  INFO Configuration:1460 - configuring from resource:
/hibernate.cfg.xml
8 21:47:32,869  INFO Configuration:1437 - Configuration resource:
/hibernate.cfg.xml
9 21:47:33,091  INFO Configuration:1575 - Configured SessionFactory: null
            21:47:33,096  INFO HibernateSearchEventListenerRegister:53 -
10          Unable to find
            org.hibernate.search.event.FullTextIndexEventListener on the
            classpath. Hibernate Search is not enabled.
11 21:47:33,229  INFO AnnotationBinder:419 - Binding entity from annotated
class: know_how.info.pl.Hibernate.Klient
12 21:47:33,299  INFO EntityBinder:422 - Bind entity
```

```
know_how.info.pl.Hibernate.Klient on table klient
13 21:47:33,395  INFO AnnotationConfiguration:369 - Hibernate Validator not found: ignoring
14 21:47:33,521  INFO DriverManagerConnectionProvider:64 - Using Hibernate built-in connection pool (not for production use!)
15 21:47:33,521  INFO DriverManagerConnectionProvider:65 - Hibernate connection pool size: 1
16 21:47:33,522  INFO DriverManagerConnectionProvider:68 - autocommit mode: false
17 21:47:33,537  INFO DriverManagerConnectionProvider:103 - using driver: com.mysql.jdbc.Driver at URL: jdbc:mysql://localhost/Hibernate
18 21:47:33,538  INFO DriverManagerConnectionProvider:109 - connection properties: {user=root, password=****}
19 21:47:34,020  INFO SettingsFactory:116 - RDBMS: MySQL, version: 5.5.24-log
20 21:47:34,020  INFO SettingsFactory:117 - JDBC driver: MySQL-AB JDBC Driver, version: mysql-connector-java-5.1.18 ( Revision: tonci.grgin@oracle.com-20110930151701-jfj14ddfq48ifkfq )
21 21:47:34,047  INFO Dialect:175 - Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
22 21:47:34,055  INFO TransactionFactoryFactory:59 - Using default transaction strategy (direct JDBC transactions)
23 21:47:34,059  INFO TransactionManagerLookupFactory:80 - No TransactionManagerLookup configured (in JTA environment, use of read-write or transactional second-level cache is not recommended)
24 21:47:34,060  INFO SettingsFactory:170 - Automatic flush during beforeCompletion(): disabled
25 21:47:34,060  INFO SettingsFactory:174 - Automatic session close at end of transaction: disabled
26 21:47:34,060  INFO SettingsFactory:181 - JDBC batch size: 15
27 21:47:34,061  INFO SettingsFactory:184 - JDBC batch updates for versioned data: disabled
28 21:47:34,062  INFO SettingsFactory:189 - Scrollable result sets: enabled
29 21:47:34,063  INFO SettingsFactory:197 - JDBC3 getGeneratedKeys(): enabled
30 21:47:34,063  INFO SettingsFactory:205 - Connection release mode: auto
31 21:47:34,065  INFO SettingsFactory:229 - Maximum outer join fetch depth: 2
32 21:47:34,065  INFO SettingsFactory:232 - Default batch fetch size: 1
33 21:47:34,066  INFO SettingsFactory:236 - Generate SQL with comments: disabled
34 21:47:34,066  INFO SettingsFactory:240 - Order SQL updates by primary key: disabled
35 21:47:34,066  INFO SettingsFactory:244 - Order SQL inserts for batching: disabled
36 21:47:34,067  INFO SettingsFactory:420 - Query translator: org.hibernate.hql.ast.ASTQueryTranslatorFactory
37 21:47:34,072  INFO ASTQueryTranslatorFactory:47 - Using ASTQueryTranslatorFactory
38 21:47:34,073  INFO SettingsFactory:252 - Query language substitutions: {}
39 21:47:34,073  INFO SettingsFactory:257 - JPA-QL strict compliance:
```

```
   disabled
40 21:47:34,073  INFO SettingsFactory:262 - Second-level cache: enabled
41 21:47:34,074  INFO SettingsFactory:266 - Query cache: disabled
42 21:47:34,074  INFO SettingsFactory:405 - Cache region factory :
   org.hibernate.cache.impl.NoCachingRegionFactory
43 21:47:34,075  INFO SettingsFactory:276 - Optimize cache for minimal puts:
   disabled
44 21:47:34,075  INFO SettingsFactory:285 - Structured second-level cache
   entries: disabled
45 21:47:34,083  INFO SettingsFactory:305 - Echoing all SQL to stdout
46 21:47:34,084  INFO SettingsFactory:314 - Statistics: disabled
47 21:47:34,085  INFO SettingsFactory:318 - Deleted entity synthetic
   identifier rollback: disabled
48 21:47:34,086  INFO SettingsFactory:333 - Default entity-mode: pojo
49 21:47:34,087  INFO SettingsFactory:337 - Named query checking : enabled
50 21:47:34,181  INFO SessionFactoryImpl:187 - building session factory
51 21:47:34,581  INFO SessionFactoryObjectFactory:105 - Not binding factory
   to JNDI, no JNDI name configured
52 21:47:34,595  INFO SchemaExport:226 - Running hbm2ddl schema export
53 21:47:34,596 DEBUG SchemaExport:242 - import file not found: /import.sql
54 21:47:34,596  INFO SchemaExport:251 - exporting generated schema to
   database
55 21:47:34,598 DEBUG SchemaExport:377 -
56 drop table if exists klient
57 21:47:34,738 DEBUG SchemaExport:377 -
58 create table klient (
59 id inte
60 ger not null auto_increment,
61 imie varchar(255),
62 nazwisko varchar(255),
63 primary key (id)
64 ) ENGINE=InnoDB
65 21:47:34,750  INFO SchemaExport:268 - schema export complete
66 Hibernate:
67 insert
68 into
69 klient
70 (imie, nazwisko)
71 values
72 (?, ?)
73 Record Inserted
74 21:47:35,041  INFO SessionFactoryImpl:805 - closing
75 21:47:35,041  INFO DriverManagerConnectionProvider:170 - cleaning up
   connection pool: jdbc:mysql://localhost/Hibernate
```

Odczyt danych zawartych w tabeli ‚Klient’

Odczytanie danych zawartych w tabeli Klient. W ciele metody main pliku App.java zamieszczamy taki oto kod:

```
1 SessionFactory session = HibernateSessionFactory.getSessionFactory();
2 Session sess = session.openSession();
3 ArrayList <Klient>listaKlientow = new ArrayList<Klient>();
4
5 Klient klient = new Klient();
6
7 try {
8
9  Query query = sess.createQuery("from Klient");
10
11 for(int i=0;i<query.list().size();i++){
12
13 listaKlientow.add((Klient)query.list().get(i));
14
15 }
16
17 } finally {
18 session.close();
19 }
20 session.close();
21
22 for(int i=0;i<listaKlientow.size();i++){
23 System.out.println(listaKlientow.get(i).getImie() +"
"+listaKlientow.get(i).getNazwisko());
24 }
```

Została utworzona lista klientów do której to poprzez zapytanie języka HQL (Hibernate Query Language) zostali wpisany wszyscy klienci znajdujący się w tabeli Klient. Wyświetlono razem w zestawieniu Imię i Nazwisko wszystkich klientów.