**Crearea class**

Entity si Tabel sunt de la javax(JPA)!

import javax.persistence.Entity;  
import javax.persistence.Table;

import javax.persistence.Id;

import javax.persistence.Column;

@Entity  
@Table(name="student")  
public class Student {  
  
 @Id  
 @Column(name="id")  
 private int id;  
  
 @Column(name="first\_name")  
 private String firstName;  
  
 @Column(name="last\_name")  
 private String lastName;  
  
 @Column(name="email")  
 private String email;  
  
  
 public Student(){  
  
 }

}

**Mereu cream si un Constructor gol, deoarece Hibernate va avea nevoie sa creeze si obiecte goale pe care apoi sa le umple.**

**De ce folosim anotatiile de la JPA**

Folosim anotatiile de la JPA(Java Persitence API), in loc de cele de la Hibernate, deoarece Hibernate doar le implementeaza, nu le modifica in niciun fel, si echipa Hibernate recomanda folosirea anotatiilor de la JPA.

**Hibernate Code**

**Session and Session Factory**

**SessionFactory(creator de sesiuni):**

* Citeste hibernate config
* Creaza obiecte Session si deci sesiuni
* Heavy-weight object(e creat doar odata, ca Bean)
* E creata doar odata in app noastra

**Session:**

* Lucreaza cu o JDBC conexiune/sesiune
* Obiect de baza pentru a salva/scoate obiecte
* Obiect cu scurta durata de viata
* Recuperat din SessionFactory

**Creating and Saving Java Objects**

try {  
 //1 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 // 2 Session session = factory.getCurrentSession();  
}  
catch (Exception e){  
 e.printStackTrace();  
}

* La 1 primim o referinta la o SessionFactory.

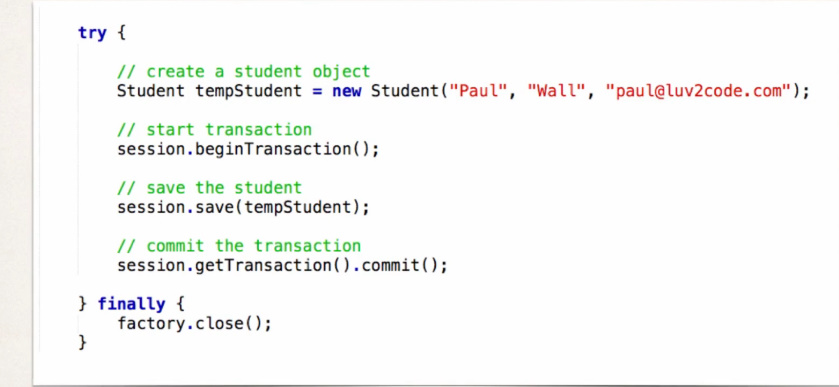
.configure specifica fisierul cfg.xml, .addAnnotatedClass specifica clasa care contine anotatiile necesare, si .buildSessionFactory ne creaza SessionFactory cu toate datele introduse de noi. **Remarca: fiecare SessionFactory obj lcureaza doar cu un tabel din baza de date, dar putem avea si mai multe addAnnotatedClass.**

* La a 2 cream o sesiune din SessionFactory
* “hibernate.cfg.xml” e optional daca numele e asa scris.
* **Aici deja importam de la hibernate!!**

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;

* Trebuie sa ne asiguram mereu ca inchidem sesiunile:
* factory.close();

**Save a Java Object**



* Cand lucram cu o sesiune, facem mereu in blocul try{}!!!!
* Trebuie mereu sa spunem ca incepe o tranzactie!
* Datorita clasei @Entity, Hibernate stie cum sa adauge fields in tabel.
* .getTransaction().commit() – salveaza in baza de date tot ce am adaugat.

public static void main(String[] args) {  
  
 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 Session session = factory.getCurrentSession();  
  
 try{  
 System.*out*.println("Creating a new student object");  
 Student student = new Student("Eduard","Eduard","edikutsu2002@mail.ru");  
 session.beginTransaction();  
 System.*out*.println("Saving the student");  
 session.save(student);  
 session.getTransaction().commit();  
 System.*out*.println("Student registered");  
 }  
 finally {  
 factory.close();  
 }  
}

**Vezi exemplu din project. ATENTIE! mysql-connector-j-8.0.31 E IN FOLDER DE BAZA! Locatia lui poate fi si schimbata, dar trebuie de specificat asta in program.**

**Getteri si setterii nu sunt obligatorii.**

**Primary Key**

* Folosim @Id pentru a-i spune lui Hibernate ca acea coloana e primary key in tabel, si deci asa nu va trebui sa-i oferim nici-o valoare neaparat, ci se lasa baza de date sa creeze o primary key.
* Insa, putem sa folosim si o strategie proprie de a genera id:

**@GeneratedValue(strategy=GnerationType.TIP)**

* **GnerationType.AUTO –** foloseste cea mai apropiata strategie din baza de date
* **GnerationType.IDENTITY –** foloseste strategia bazei de date, adica lasa baza de date sa genereze singura ID, si sa foloseasca strategiile sale, ca AUTO INCREMENT. **Nu mereu e automat.Deci, obiectul cand va fi salvat, va primi un id de la baza de date, si hibernate il va pune in obiect.**
* **GnerationType.SEQUENCE –** foloseste database sequence
* **GnerationType.TABLE –** foloseste un underlying database table.

Trebuie sa vedem care din aceste strategii e suportata de baza noastra de date.

**Create our own ID**

* Putem crea propriul model de a genera ID.
* Pentru asta, trebuie sa cream o clasa care mosteneste **IdentifierGenerator,** de la **org.hibernate.id.IdentifierGenerator** si sa suprascriem metoda public Serializable **generator(...)**
* Totusi, pentru a genera id unice, trebuie multi-threading

**Modificare start ID**

* Pentru a modifica AUTO\_INCREMENT:

ALTER TABLE Tabel AUTO\_INCREMENT = 5000;

* Pentru a o reseta, truncam tabelul.

**Citirea obiectelor cu hibernate**

* Folosim session.get(Clasa.class, id);

Student retrieve = session.get(Student.class,2);

* Daca nu e gasit, se returneaza null
* In Hibernate, chiar daca citim date din baza de date, sau doar facem interogari, trebuie mereu sa dam begginTransaction(), GetTransaction.commit() si close() la session.

SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
Session session = factory.getCurrentSession();  
Student getStudent;  
  
try{  
 session.beginTransaction();  
 getStudent = session.get(Student.class,2);  
 System.*out*.println(getStudent);  
 session.getTransaction().commit();  
}  
catch (Exception e){  
 e.printStackTrace();  
}  
finally {  
 session.close();  
}

**Update Objects**

Putem sa actualizam un obiect asa:

1. Obtinem obiectul din baza de date asa:

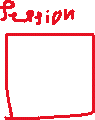
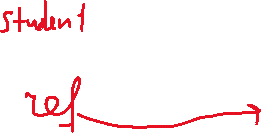
Student student = session.get(Student.class,id);

1. Modificam ce dorim in obiect:

Student.setFirstName(„New”);

1. Session.getTransaction.commit

Putem sa modificam asa direct un obiect din tabel, datorita clasei Student care e Entity, si deci are legatura cu obiectul.Sesiunea va returna o referinta la acest obiect Student,si dupa ce il modificam,ea il va trimite inapoi la tabel, in locul la vechi.



Insa, modificarea in tabel se face abea dupa Session.getTransaction.commit

, caci pana atunci modificarea se face doar in memorie, adica in obj din schema de sus, apoi deja obiectul modificat e introdus in tabel in locul la cel nou, dar asta nu e un obiect nou!

Putem si asa face, prin query:

Query query = session.createQuery("update Student set email='test@gmail.com'");  
query.executeUpdate();

executeUpdate() executa query si face update la baza de date si returneaza id.

**Deleting objects**

1. Returnam referinta la obiect din tbel:

Student student = session.get(Stuent.class, id0)

1. Folosim comanda delete() si stergem obiectul gasit:

session.delete(student)

1. Session.getTransaction.commit();

Aici, obiectul nostru, care e scos din baza de date, va fi sters de acolo doar.

Putem si asa:

Query query = session.createQuery("delete from Student where id=12586");  
query.executeUpdate();

Alta metoda este de a extrage din baza de date obiectul cu id dat si sa folosim delete(Object)

**ATENTIE! Query**

If you are using Hibernate 5.2 or higher, then the Query list() method has been deprecated.

In your code you should make the following update:

**Replace**

session.createQuery("from Student").list()

**With**

session.createQuery("from Student").getResultList()

**Hibernate Query LANGUAGE**

Hibernate are un query language care e foarte similar cu SQL. De ex:

Query query = session.createQuery("from Student");  
  
List<Student> list = query.getResultList();

Ne folosim de clasa Student si fieldurile din ea!!!

**Folosirea la Query**

* Pentru a folosi query, cream un obiect de tip Query, si il egalam cu session.createQuery(“hqlQuery”), de ex:

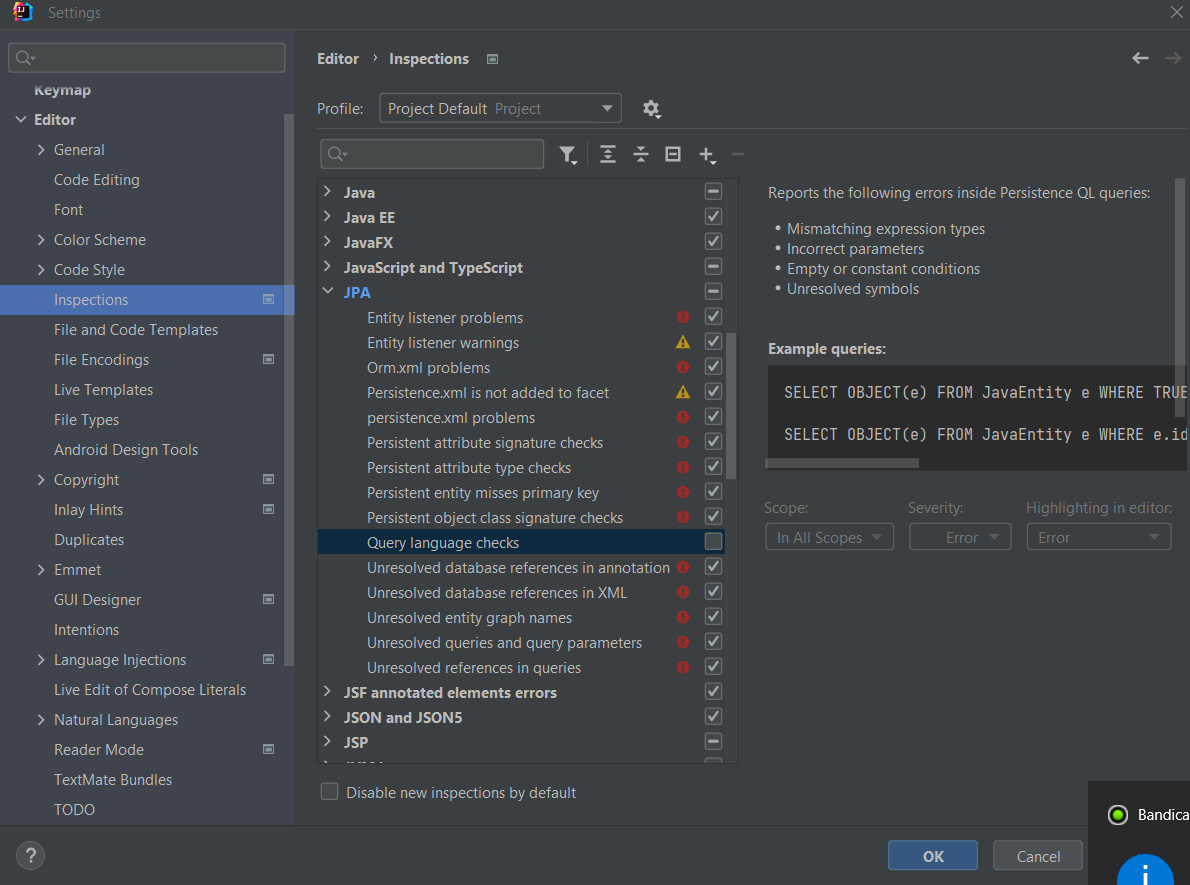
Query query = session.createQuery("from Student");

**Atentie! Student nu e numele tabelului din DBMS, ci e numele Entity clasei ce e legata de tabel! Hibernate va crea un obiect Student, cu un nume propriu, daca noi nu dam unul, si de fiecare data il va umple cu date si returna in lista.**

* Cream un obiect de tip List, si returnam lista de obiecte din baza de date cu getResultList()

List <Student> students = query.getResultList();

Uneori Intellij nu vede HQL(hibernate query language), si apar erori cu rosu in query, desi programul ruleaza. Pentru a scapa de ele, **Settings** -> **Editor** -> **Inspection** -> **JPA** -> **debifam Query language checks**

****

Si pentru a dezactiva unele warnuri enervante si inutile:

@SuppressWarnings("unchecked")

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Query;  
import java.util.List;  
  
@SuppressWarnings("unchecked")  
public class Data {  
 public static void main(String[] args){  
 SessionFactory sessionFactory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 Session session = sessionFactory.getCurrentSession();  
  
 try{  
 session.beginTransaction();  
 Query query = session.createQuery("from Student");  
 List<Student> lista = query.getResultList();  
  
 for(Student student : lista){  
 System.*out*.println(student);  
 }  
 session.getTransaction().commit();  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 finally {  
 session.close();  
 }  
  
 }  
}

Other queries

* Putem folosi orice clauze, ca WHERE, HAVING etc., de ex:

Query query = session.createQuery("FROM Student test WHERE test.firstName='Eduard'");  
List<Student> lista = query.getResultList();

**Student test** – aici indicam ca vom lucra cu un obiect de tip Student, clasa Student fiind un entity, insa avem nevoie ca dupa cumva sa ne referim la atributul firstName, si o putem face doar prin crearea unui obiect. Acest obiect test va fi de fiecare data creat si umplut cu datele gasite. E ceva ca AS.

In exemplul de mai sus, am fi putut pune si

Query query = session.createQuery("from Student s");

**Oricum Hibernate creaza un obiect e tip Student ca dam noi un nume sau nu.**

**firtName e evident din clasa Student, adica e fieldul ce corespunde coloanei din tabel, dar in clasa.**

**Putem insa face si asa:**

Query query = session.createQuery("from Student");  
  
List<Student> list = query.getResultList();

Query query = session.createQuery("from Student s where s.firstName='Eduard' or s.lastName='Alone'");

Query query = session.createQuery("FROM Student s WHERE s.lastName LIKE 'A%'");

**Problema cu ? ? ?**

Cand Hibernate executa query, de ex de introducere a datelor in DBMS, valorile noastre sunt inlocuite de ?, si deci nu le putem vedea in consola. De ex:

Hibernate: insert into student (email, first\_name, last\_name, id) values (?, ?, ?, ?)

Totusi, unica solutie de a le vedea este prin loguri. Vezi LogBack din acest Folder

**Parameter Order Query**

Putem introduce variabile deja existente in query. De ex:

String username = user.getUsername();  
String password = user.getPassword();

Query query = *session*.createQuery("FROM User WHERE username = ?0 AND password = ?1").setParameter(0,username).setParameter(1,password);

* setParameter(pozitie,variabila) – seteaza variabila la pozitia data.
* ?0 inseamna ca in locul lui ? va fi valoarea unei variabile deja definite de noi in program, si 0 e pozitia.

Sau

* cu :nume

Query query = *session*.createQuery("FROM User WHERE username = :id1 AND password = :id2").setParameter(“id1”,username).setParameter(“id2”,password);

**Config cu Java Code in loc de .xml**

@Bean  
public DataSource dataSource(){  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
 dataSource.setUsername("testuser");  
 dataSource.setPassword("Frb2eshox!");  
 dataSource.setUrl("jdbc:mysql://localhost:3306/test?useSSL=false");  
 dataSource.setDriverClassName(com.mysql.jdbc.Driver.class.getName());  
  
 return dataSource;  
  
}  
@Bean  
public LocalSessionFactoryBean sessionFactoryBean(){  
 LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setPackagesToScan("org.java.app");  
 Properties properties = new Properties();  
 properties.setProperty("show\_sql","true");  
 properties.setProperty("dialect",org.hibernate.dialect.MySQL8Dialect.class.getName());  
 sessionFactoryBean.setDataSource(dataSource());  
 sessionFactoryBean.setHibernateProperties(properties);  
  
 return sessionFactoryBean;  
}

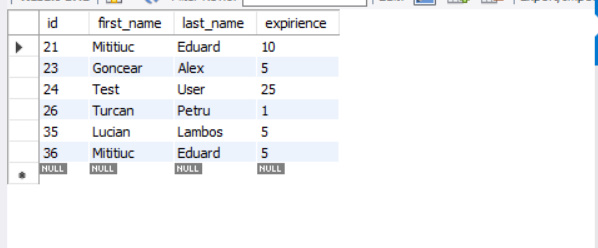
Deci, anume clasa **DriverManagerDataSource** ne ajuta sa cream o Data Source

**Inserarea unui ID deja existent**

* Daca am pus:
* @GeneratedValue(strategy = GenerationType.*IDENTITY*)

Hibernate mereu va ignora ID dat de noi, si va lasa BD sa puna un ID.

* Teacher teacher = new Teacher("Mititiuc","Eduard",5);  
  teacher.setId(21);  
    
  session.save(teacher);





Desi am pus id 21, el a fost oricum ignorat.

* Daca nu punem GeneratedValue, pai atunci noi va trebuie sa alegem neaparat ID, si in cazul de sus am primi eroare:

