**Задание.**

Реализовать от 3 до 5 запросов к вашей модели данных, используя средства HQL, JPQL.

Бонусное задание:

попробовать использовать Criteria для одного или нескольких запросов, а также попробовать реализовать сам CRUD через именованные hql-запросы.

Визуал для запросов делать не обязательно, продемонстрировать в виде тестов (но можно).

**Предыдущее задание:**

**Задание.**

Реализовать CRUD-операции по выданной теме с веб-интерфейсом на основе Hibernate JTA + СУБД MySQL согласно концепции DB First.

Требования:

* Разделение представления, контроллеров и модели
* Реализация Data Access Object – класса
* Использование интерфейса DAO или Data Source

Рекомендации:

* Реализовать сначала модель
* Протестировать модель на юнит-тестах (например)
* Реализовать сервлеты и view (рекомендуется использовать bootstrap)
* См. пример с урока от 10.10.2022

**Варианты заданий.**

1. Бабунин А. А. – Библиотечный каталог

2. Боровушкин Д. И. – Каталог виниловых пластинок

3. Гущин Л. П. – Художественная галерея

4. Дурнев А. А. – Фермерский автопарк

5. Исоков А. А. – Каталог переведенных копий книг

6. Кузнецов А. К – Учет животных в зоопарке

7. Курлыгин Р. С. – Студенческие списки поступления

8. Курлыгина В. С. – Справочник ВВП разных стран

9. Майдан М. К. – Каталог горных вершин мира

10. Найденов А. Е. – Справочник по микропроцессорам

11. Пестова О. Ю. – Учетная книга денежных операций

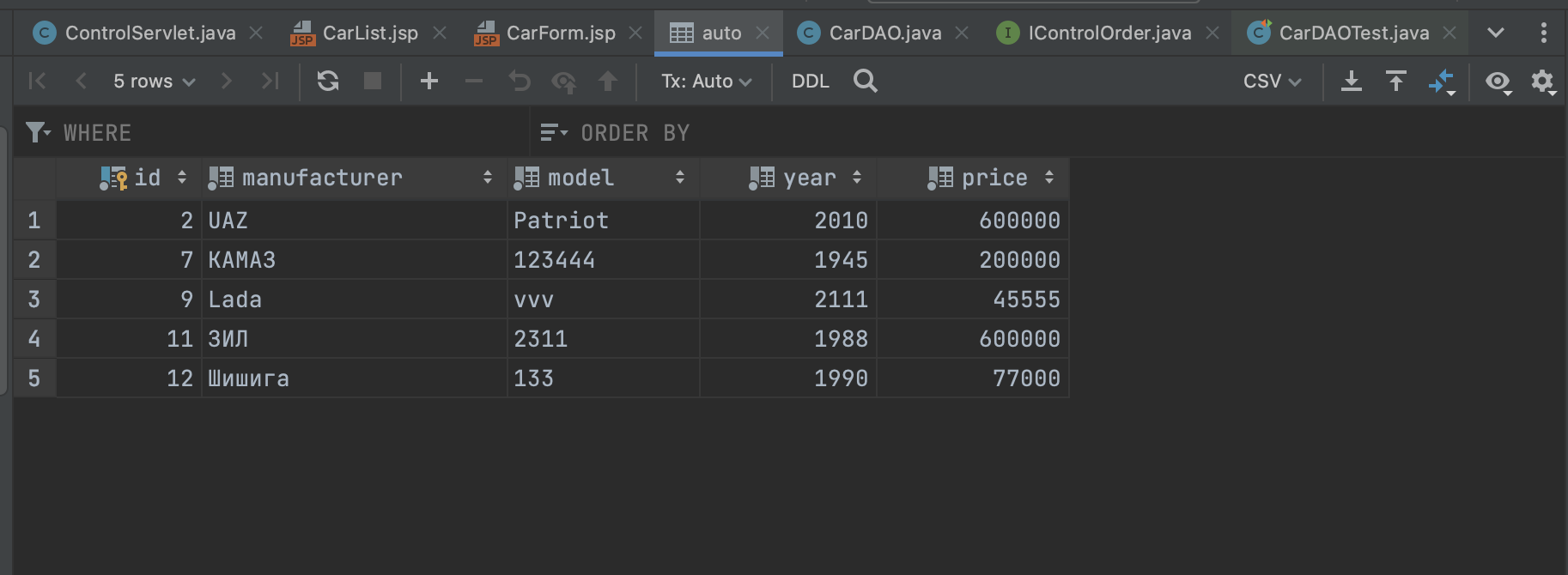
12. Поминов В. С. – Каталог автомобилей

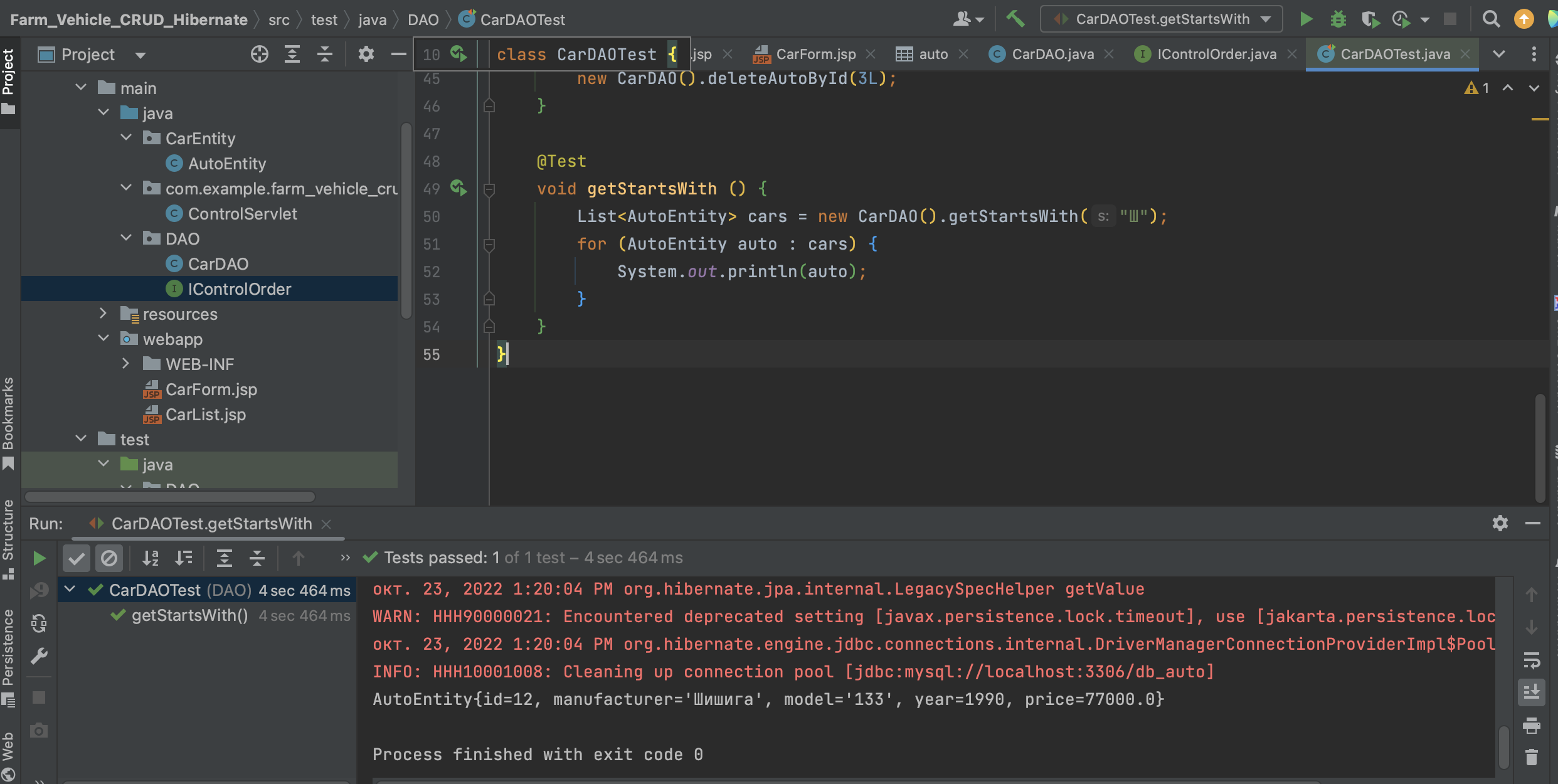
13. Скуратов Н. С. – Модельный ряд мирового авиапарка транспортной авиации

14. Тымкив А. С. – Список курортов мира

@Test  
void getStartsWith () {  
 List<AutoEntity> cars = new CarDAO().getStartsWith("Ш");  
 for (AutoEntity auto : cars) {  
 System.*out*.println(auto);  
 }  
}

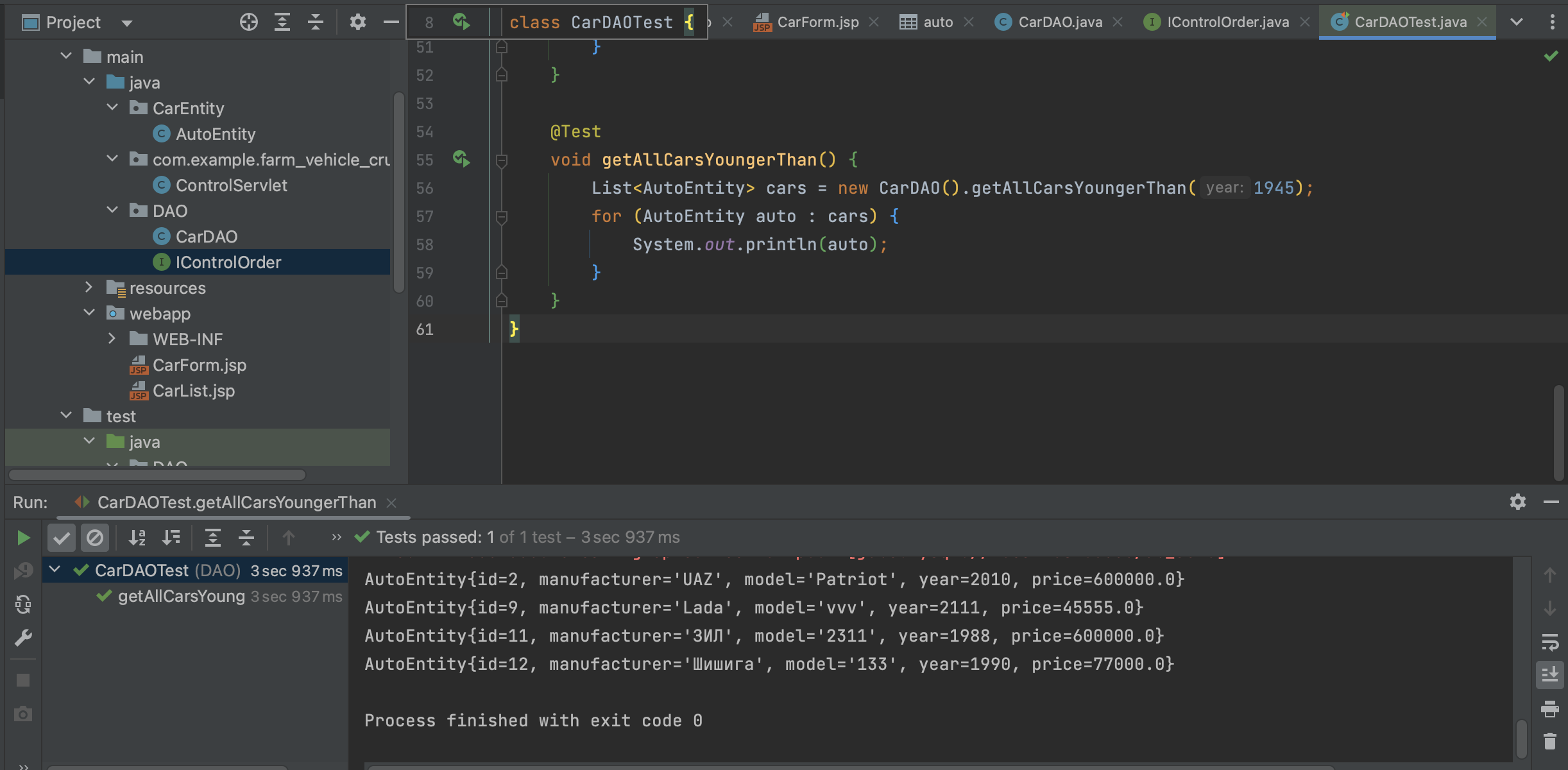
@Override  
public List<AutoEntity> getStartsWith(String s) {  
 EntityManagerFactory entityManagerFactory = Persistence.*createEntityManagerFactory*("default");  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 EntityTransaction transaction = entityManager.getTransaction();  
 // список результатов  
 List<AutoEntity> cars;  
  
 try {  
 transaction.begin();  
 // операция  
 cars = entityManager.createQuery("SELECT e FROM AutoEntity e where e.manufacturer " +  
 "like :name", AutoEntity.class)  
 .setParameter("name", s + "%")  
 .getResultList();  
 transaction.commit();  
 } finally {  
 if (transaction.isActive()) {  
 transaction.rollback();  
 }  
 entityManager.close();  
 entityManagerFactory.close();  
 }  
 return cars;  
}





List<AutoEntity> getAllCarsYoungerThan (int year);

@Override  
public List<AutoEntity> getAllCarsYoungerThan(int year) {  
 EntityManagerFactory entityManagerFactory = Persistence.*createEntityManagerFactory*("default");  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 EntityTransaction transaction = entityManager.getTransaction();  
 // список результатов  
 List<AutoEntity> cars;  
  
 try {  
 transaction.begin();  
 // операция  
 cars = entityManager.createNamedQuery("get\_all\_cars\_younger\_than", AutoEntity.class)  
 .setParameter("param", year).getResultList();  
 transaction.commit();  
 } finally {  
 if (transaction.isActive()) {  
 transaction.rollback();  
 }  
 entityManager.close();  
 entityManagerFactory.close();  
 }  
 return cars;  
}



@Override  
public List<AutoEntity> getAllCarsCheaperThan(double price) {  
 EntityManagerFactory entityManagerFactory = Persistence.*createEntityManagerFactory*("default");  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 EntityTransaction transaction = entityManager.getTransaction();  
 // список результатов  
 List<AutoEntity> cars;  
  
 try {  
 transaction.begin();  
 // операция  
 cars = entityManager.createNamedQuery("get\_all\_cars\_cheaper\_than", AutoEntity.class)  
 .setParameter("param", price).getResultList();  
 transaction.commit();  
 } finally {  
 if (transaction.isActive()) {  
 transaction.rollback();  
 }  
 entityManager.close();  
 entityManagerFactory.close();  
 }  
 return cars;  
}

List<AutoEntity> getAllCarsCheaperThan (double price);

// получаем авто дешевле какой-либо суммы  
@Test  
void getAllCarsCheaperThan() {  
 List<AutoEntity> cars = new CarDAO().getAllCarsCheaperThan(200000);  
 for (AutoEntity auto : cars) {  
 System.*out*.println(auto);  
 }  
}

