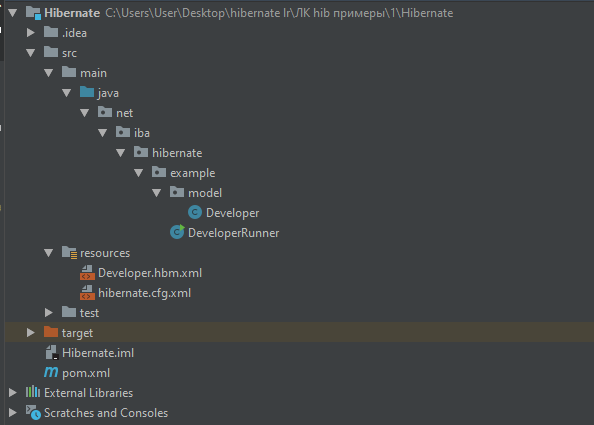
**Задание №5 Управление персистентностью на основе JPA (hibernate)**

Еще раз обсудим разницу понятий JPA, ORM, Hibernate etc.

Для начала нам необходимо подготовить структуру проекта:



Давайте еще раз обсудим структуру проекта.

**БД.**

-- --------------------------------------------------------

-- Хост: 127.0.0.1

-- Версия сервера: 8.0.15 - MySQL Community Server - GPL

-- Операционная система: Win64

-- HeidiSQL Версия: 10.1.0.5464

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!50503 SET NAMES utf8mb4 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- Дамп структуры базы данных hib1

CREATE DATABASE IF NOT EXISTS `hib1` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/;

USE `hib1`;

-- Дамп структуры для таблица hib1.hibernate\_developers

CREATE TABLE IF NOT EXISTS `hibernate\_developers` (

`ID` int(11) NOT NULL AUTO\_INCREMENT,

`FIRST\_NAME` varchar(50) DEFAULT NULL,

`LAST\_NAME` varchar(50) DEFAULT NULL,

`SPECIALTY` varchar(50) DEFAULT NULL,

`EXPERIENCE` int(11) DEFAULT NULL,

PRIMARY KEY (`ID`)

) ENGINE=InnoDB AUTO\_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

-- Экспортируемые данные не выделены.

/\*!40101 SET SQL\_MODE=IFNULL(@OLD\_SQL\_MODE, '') \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=IF(@OLD\_FOREIGN\_KEY\_CHECKS IS NULL, 1, @OLD\_FOREIGN\_KEY\_CHECKS) \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

**Класс Developer**

package net.iba.hibernate.example.model;  
  
public class Developer {  
 private int id;  
 private String firstName;  
 private String lastName;  
 private String specialty;  
 private int experience;  
  
 public Developer() {  
 }  
  
 public Developer(String firstName, String lastName, String specialty, int experience) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.specialty = specialty;  
 this.experience = experience;  
 }  
  
  
 */\*\*  
 \* Getters and Setters  
 \*/* public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getSpecialty() {  
 return specialty;  
 }  
  
 public void setSpecialty(String specialty) {  
 this.specialty = specialty;  
 }  
  
 public int getExperience() {  
 return experience;  
 }  
  
 public void setExperience(int experience) {  
 this.experience = experience;  
 }  
  
 */\*\*  
 \* toString method (optional)  
 \*/* @Override  
 public String toString() {  
 return "Developer:\n" +  
 "id: " + id +  
 "\nFirst Name: " + firstName + "\n" +  
 "Last Name: " + lastName + "\n" +  
 "Specialty: " + specialty + "\n" +  
 "Experience: " + experience + "\n";  
 }  
}

**Класс DeveloperRunner**

package net.iba.hibernate.example;  
  
import net.iba.hibernate.example.model.Developer;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import java.util.List;  
  
public class DeveloperRunner {  
 private static SessionFactory *sessionFactory*;  
  
 public static void main(String[] args) {  
 *sessionFactory* = new Configuration().configure().buildSessionFactory();  
  
 DeveloperRunner developerRunner = new DeveloperRunner();  
  
 System.*out*.println("Adding developer's records to the DB");  
 */\*\*  
 \* Adding developer's records to the database (DB)  
 \*/* developerRunner.addDeveloper("Ihor", "Developer", "Java Developer", 2);  
 developerRunner.addDeveloper("Some", "Developer", "C++ Developer", 2);  
 developerRunner.addDeveloper("Peter", "UI", "UI Developer", 4);  
  
 System.*out*.println("List of developers");  
 */\*\*  
 \* List developers  
 \*/* List<Developer> developers = developerRunner.listDevelopers();  
 for (Developer developer : developers) {  
 System.*out*.println(developer);  
 }  
 System.*out*.println("===================================");  
 System.*out*.println("Removing Some Developer and updating Ihor");  
 */\*\*  
 \* Update and Remove developers  
 \*/* developerRunner.updateDeveloper(10, 3);  
 developerRunner.removeDeveloper(11);  
  
 System.*out*.println("Final list of developers");  
 */\*\*  
 \* List developers  
 \*/* developers = developerRunner.listDevelopers();  
 for (Developer developer : developers) {  
 System.*out*.println(developer);  
 }  
 System.*out*.println("===================================");  
  
 }  
  
 public void addDeveloper(String firstName, String lastName, String specialty, int experience) {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Developer developer = new Developer(firstName, lastName, specialty, experience);  
 session.save(developer);  
 transaction.commit();  
 session.close();  
 }  
  
 public List<Developer> listDevelopers() {  
 Session session = this.*sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 List<Developer> developers = session.createQuery("FROM Developer").list();  
  
 transaction.commit();  
 session.close();  
 return developers;  
 }  
  
 public void updateDeveloper(int developerId, int experience) {  
 Session session = this.*sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Developer developer = (Developer) session.get(Developer.class, developerId);  
 developer.setExperience(experience);  
 session.update(developer);  
 transaction.commit();  
 session.close();  
 }  
  
 public void removeDeveloper(int developerId) {  
 Session session = this.*sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Developer developer = (Developer) session.get(Developer.class, developerId);  
 session.delete(developer);  
 transaction.commit();  
 session.close();  
 }  
  
}

Данный класс содержит основные классы, используемые хибернейтом.

Во-первых давайте обсудим EntityManager, которого тут нет, но тем не менее, он используется внутри нашим проектом.

Во-вторых это класс SessionFactory, который является наследником от EM.

Так же нам стоит обратится к классу Transaction.

И наконец обсудим сами методы данных классов.

Рассмотрим маппинг.

**Developer.hbm.xml**

<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC  
 "-//Hibernate/Hibernate Mapping DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
 <class name="net.iba.hibernate.example.model.Developer" table="HIBERNATE\_DEVELOPERS">  
 <meta attribute="class-description">  
 This class contains developer's details.  
 </meta>  
 <id name="id" type="int" column="ID">  
 <generator class="native"/>  
 </id>  
 <property name="firstName" column="FIRST\_NAME" type="string"/>  
 <property name="lastName" column="LAST\_NAME" type="string"/>  
 <property name="specialty" column="SPECIALTY" type="string"/>  
 <property name="experience" column="EXPERIENCE" type="int"/>  
 </class>  
</hibernate-mapping>

И подключение к БД.

**Hibernate.cfg.xml**

<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-configuration SYSTEM  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  
  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.dialect">  
 org.hibernate.dialect.MySQLDialect  
 </property>  
 <property name="hibernate.connection.driver\_class">  
 com.mysql.cj.jdbc.Driver  
 </property>  
  
 <!-- Assume test is the database name -->  
 <property name="hibernate.connection.url">  
 jdbc:mysql://localhost/hib1  
 </property>  
 <property name="hibernate.connection.username">  
 root  
 </property>  
 <property name="hibernate.connection.password">  
 root  
 </property>  
  
 <!-- List of XML mapping files -->  
 <mapping resource="Developer.hbm.xml"/>  
  
 </session-factory>  
</hibernate-configuration>

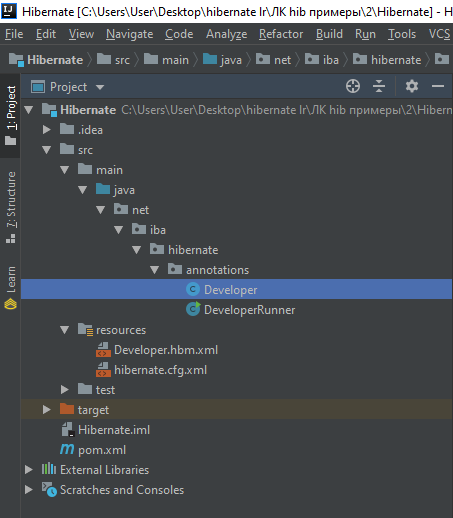
На всякий случай pom

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.springapp</groupId>  
 <artifactId>Hibernate</artifactId>  
 <packaging>war</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <name>Hibernate</name>  
  
 <properties>  
 <spring.version>4.1.1.RELEASE</spring.version>  
 </properties>  
  
 <dependencies>  
  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-core</artifactId>  
 <version>5.0.7.Final</version>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.23</version>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <finalName>Hibernate</finalName>  
 <plugins>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.6</source>  
 <target>1.6</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Можете запустить и проверить работу.

Перейдем к следующему шагу – используем аннотации.

Структура проекта будет следующая. Скопируйте проект для экономии времени.



БД будем использовать ту же.

Далее нам потребуется создать класс **Developer**

package net.iba.hibernate.annotations;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "HIBERNATE\_DEVELOPERS")  
public class Developer {  
 @Id  
 @GeneratedValue (strategy = GenerationType.*AUTO*)  
 @Column (name = "id")  
 private int id;  
 @Column (name = "FIRST\_NAME")  
 private String firstName;  
 @Column (name = "LAST\_NAME")  
 private String lastName;  
 @Column (name = "SPECIALTY")  
 private String specialty;  
 @Column (name = "EXPERIENCE")  
 private int experience;  
  
 */\*\*  
 \* Default Constructor  
 \*/* public Developer() {  
 }  
  
 */\*\*  
 \* Plain constructor  
 \*/* public Developer(String firstName, String lastName, String specialty, int experience) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.specialty = specialty;  
 this.experience = experience;  
 }  
  
 */\*\*  
 \* Getters and Setters  
 \*/* public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getSpecialty() {  
 return specialty;  
 }  
  
 public void setSpecialty(String specialty) {  
 this.specialty = specialty;  
 }  
  
 public int getExperience() {  
 return experience;  
 }  
  
 public void setExperience(int experience) {  
 this.experience = experience;  
 }  
  
 */\*\*  
 \* toString method (optional)  
 \*/* @Override  
 public String toString() {  
 return "Developer:\n" +  
 "id: " + id +  
 "\nFirst Name: " + firstName + "\n" +  
 "Last Name: " + lastName + "\n" +  
 "Specialty: " + specialty + "\n" +  
 "Experience: " + experience + "\n";  
 }  
}

И давайте разберем аннотации, которые помогают нам решить проблему маппинга.

Developer.hbm.xml можете закомментировать.

Внутри hibernate.cfg.xml изменим строку маппинга на:



Обратите внимание, что нам теперь необходимо настроить конфигурацию нашего класса отвечающего за подключение – т.е. SessionFactory.

Добавим следующее:



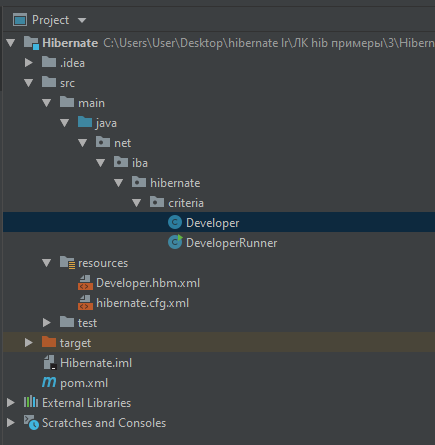
После чего пробуйте запустить.

Обратите внимание на ошибки. Как вы думаете, что в приложении вызвало ошибку?

Давайте разберем следующие значения атрибута аннотации GeneraitedValue.

Давайте перейдем к criteria.

Структура проекта будет выглядеть следующим образом.



В БД нам необходимо будет добавить еще одно поле.

Можете создать еще одну БД.

-- --------------------------------------------------------

-- Хост: 127.0.0.1

-- Версия сервера: 8.0.15 - MySQL Community Server - GPL

-- Операционная система: Win64

-- HeidiSQL Версия: 10.1.0.5464

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!50503 SET NAMES utf8mb4 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- Дамп структуры базы данных hib2

CREATE DATABASE IF NOT EXISTS `hib2` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/;

USE `hib2`;

-- Дамп структуры для таблица hib2.hibernate\_developers

CREATE TABLE IF NOT EXISTS `hibernate\_developers` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`FIRST\_NAME` varchar(50) DEFAULT NULL,

`LAST\_NAME` varchar(50) DEFAULT NULL,

`SPECIALTY` varchar(50) DEFAULT NULL,

`EXPERIENCE` int(11) DEFAULT NULL,

`SALARY` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

-- Экспортируемые данные не выделены.

/\*!40101 SET SQL\_MODE=IFNULL(@OLD\_SQL\_MODE, '') \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=IF(@OLD\_FOREIGN\_KEY\_CHECKS IS NULL, 1, @OLD\_FOREIGN\_KEY\_CHECKS) \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

Теперь создадим класс **Developer.**

package net.iba.hibernate.criteria;  
  
public class Developer {  
 private int id;  
 private String firstName;  
 private String lastName;  
 private String specialty;  
 private int experience;  
 private int salary;  
  
 */\*\*  
 \* Default Constructor  
 \*/* public Developer() {  
 }  
  
 */\*\*  
 \* Plain constructor  
 \*/* public Developer(String firstName, String lastName, String specialty, int experience, int salary) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.specialty = specialty;  
 this.experience = experience;  
 this.salary = salary;  
 }  
  
 */\*\*  
 \* Getters and Setters  
 \*/* public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getSpecialty() {  
 return specialty;  
 }  
  
 public void setSpecialty(String specialty) {  
 this.specialty = specialty;  
 }  
  
 public int getExperience() {  
 return experience;  
 }  
  
 public void setExperience(int experience) {  
 this.experience = experience;  
 }  
  
 public int getSalary() {  
 return salary;  
 }  
  
 public void setSalary(int salary) {  
 this.salary = salary;  
 }  
  
 */\*\*  
 \* toString method (optional)  
 \*/* @Override  
 public String toString() {  
 return "id: " + id +  
 "\nFirst Name: " + firstName +  
 "\nLast Name: " + lastName +  
 "\nSpecialty: " + specialty +  
 "\nExperience: " + experience +  
 "\nSalary: " + salary + "\n";  
 }  
}

Для практики – пересоберите проект на аннотациях (можете использовать как ДЗ).

Добавим класс для создания запросов к БД.

package net.iba.hibernate.criteria;  
  
import org.hibernate.Criteria;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
import org.hibernate.criterion.Projections;  
import org.hibernate.criterion.Restrictions;  
  
import java.util.List;  
  
public class DeveloperRunner {  
 private static SessionFactory *sessionFactory*;  
  
 public static void main(String[] args) {  
 *sessionFactory* = new Configuration().configure().buildSessionFactory();  
 DeveloperRunner developerRunner = new DeveloperRunner();  
  
 System.*out*.println("Adding developer's records to the database...");  
 Integer developerId1 = developerRunner.addDeveloper("Ihor", "Developer", "Java Developer", 3, 2000);  
 Integer developerId2 = developerRunner.addDeveloper("First", "Developer", "C++ Developer", 10, 2000);  
 Integer developerId3 = developerRunner.addDeveloper("Second", "Developer", "C# Developer", 5, 2000);  
 Integer developerId4 = developerRunner.addDeveloper("Third", "Developer", "PHP Developer", 1, 2000);  
  
 System.*out*.println("List of Developers with experience more than 3 years:");  
 developerRunner.listDevelopersOverThreeYears();  
  
 System.*out*.println("Total Salary of all Developers:");  
 developerRunner.totalSalary();  
 *sessionFactory*.close();  
 }  
  
 public Integer addDeveloper(String firstName, String lastName, String specialty, int experience, int salary) {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
 Integer developerId = null;  
  
 transaction = session.beginTransaction();  
 Developer developer = new Developer(firstName, lastName, specialty, experience, salary);  
 developerId = (Integer) session.save(developer);  
 transaction.commit();  
 session.close();  
 return developerId;  
 }  
  
 public void listDevelopersOverThreeYears() {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Criteria criteria = session.createCriteria(Developer.class);  
 criteria.add(Restrictions.*gt*("experience", 3));  
 List<Developer> developers = criteria.list();  
  
 for (Developer developer : developers) {  
 System.*out*.println("=======================");  
 System.*out*.println(developer);  
 System.*out*.println("=======================");  
 }  
 transaction.commit();  
 session.close();  
 }  
  
 public void totalSalary() {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Criteria criteria = session.createCriteria(Developer.class);  
 criteria.setProjection(Projections.*sum*("salary"));  
  
 List totalSalary = criteria.list();  
 System.*out*.println("Total salary of all developers: " + totalSalary.get(0));  
 transaction.commit();  
 session.close();  
 }  
  
  
}

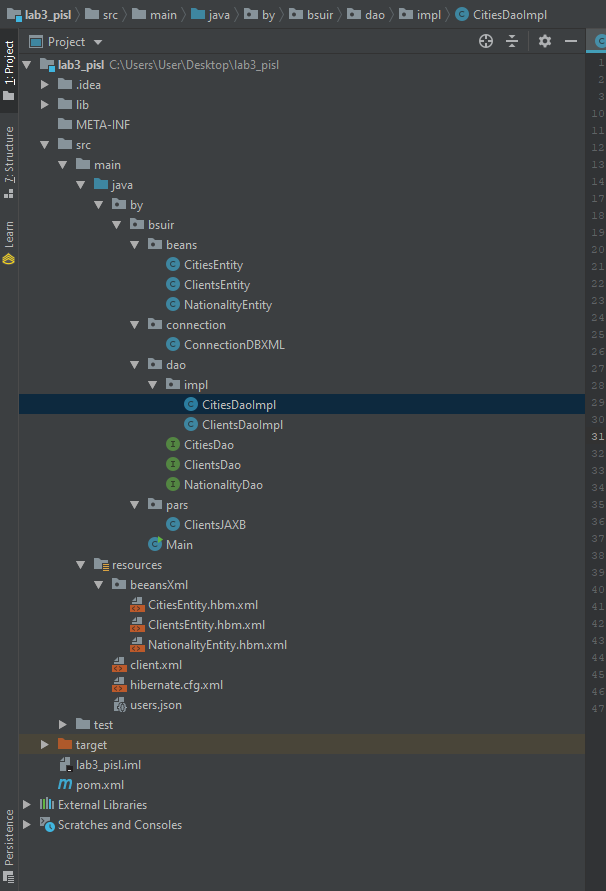
Конфигурационные файлы:

<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC  
 "-//Hibernate/Hibernate Mapping DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
  
<hibernate-mapping>  
 <class name="net.iba.hibernate.criteria.Developer" table="HIBERNATE\_DEVELOPERS">  
 <meta attribute="class-description">  
 This class contains developer details.  
 </meta>  
 <id name="id" type="int" column="id">  
 <generator class="native"/>  
 </id>  
 <property name="firstName" column="FIRST\_NAME" type="string"/>  
 <property name="lastName" column="LAST\_NAME" type="string"/>  
 <property name="specialty" column="SPECIALTY" type="string"/>  
 <property name="experience" column="EXPERIENCE" type="int"/>  
 <property name="salary" column="SALARY" type="int"/>  
 </class>  
  
</hibernate-mapping>

И крайний, типичный пример с нативными запросами:

package net.iba.hibernate.nativesql;  
  
import org.hibernate.\*;  
import org.hibernate.cfg.Configuration;  
import org.hibernate.criterion.Projections;  
  
import java.util.List;  
import java.util.Map;  
  
public class DeveloperRunner {  
 private static SessionFactory *sessionFactory*;  
  
 public static void main(String[] args) {  
 *sessionFactory* = new Configuration().configure().buildSessionFactory();  
 DeveloperRunner developerRunner = new DeveloperRunner();  
  
 System.*out*.println("Adding developer's records to the database...");  
 Integer developerId1 = developerRunner.addDeveloper("Ihor", "Developer", "Java Developer", 3, 2000);  
 Integer developerId2 = developerRunner.addDeveloper("First", "Developer", "C++ Developer", 10, 5000);  
 Integer developerId3 = developerRunner.addDeveloper("Second", "Developer", "C# Developer", 5, 4000);  
 Integer developerId4 = developerRunner.addDeveloper("Third", "Developer", "PHP Developer", 1, 1000);  
  
 System.*out*.println("List of Developers using Entity Query:");  
 developerRunner.listDevelopers();  
  
 System.*out*.println("List of Developers using Scalar Query:");  
 developerRunner.listDevelopersScalar();  
 *sessionFactory*.close();  
 }  
  
 public Integer addDeveloper(String firstName, String lastName, String specialty, int experience, int salary) {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
 Integer developerId = null;  
  
 transaction = session.beginTransaction();  
 Developer developer = new Developer(firstName, lastName, specialty, experience, salary);  
 developerId = (Integer) session.save(developer);  
 transaction.commit();  
 session.close();  
 return developerId;  
 }  
  
 public void listDevelopers() {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 SQLQuery sqlQuery = session.createSQLQuery("SELECT \* FROM HIBERNATE\_DEVELOPERS");  
 sqlQuery.addEntity(Developer.class);  
 List<Developer> developers = sqlQuery.list();  
  
 for (Developer developer : developers) {  
 System.*out*.println("=======================");  
 System.*out*.println(developer);  
 System.*out*.println("=======================");  
 }  
 transaction.commit();  
 session.close();  
 }  
  
 public void listDevelopersScalar() {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 SQLQuery sqlQuery = session.createSQLQuery("SELECT \* FROM HIBERNATE\_DEVELOPERS");  
 sqlQuery.setResultTransformer(Criteria.*ALIAS\_TO\_ENTITY\_MAP*);  
 List<Developer> developers = sqlQuery.list();  
 for (Object developer : developers) {  
 Map row = (Map) developer;  
 System.*out*.println("=======================");  
 System.*out*.println("id: " + row.get("id"));  
 System.*out*.println("First Name: " + row.get("FIRST\_NAME"));  
 System.*out*.println("Last Name: " + row.get("LAST\_NAME"));  
 System.*out*.println("Specialty: " + row.get("SPECIALTY"));  
 System.*out*.println("Experience: " + row.get("EXPERIENCE"));  
 System.*out*.println("Salary: " + row.get("SALARY"));  
 System.*out*.println("=======================");  
 }  
 transaction.commit();  
 session.close();  
 }  
  
 public void totalSalary() {  
 Session session = *sessionFactory*.openSession();  
 Transaction transaction = null;  
  
 transaction = session.beginTransaction();  
 Criteria criteria = session.createCriteria(Developer.class);  
 criteria.setProjection(Projections.*sum*("salary"));  
  
 List totalSalary = criteria.list();  
 System.*out*.println("Total salary of all developers: " + totalSalary.get(0));  
 transaction.commit();  
 session.close();  
 }  
  
  
}

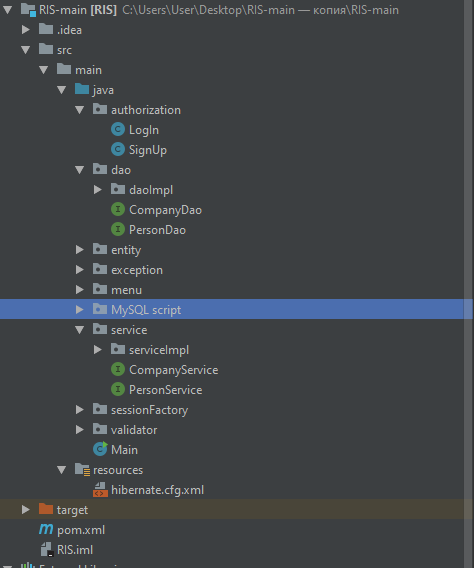
Давайте разберем еще один пример, который есть на гугл диске. Скачайте и откройте его. Так же на диске есть скрипт создания БД, выполните его.



Пример функционирования серверной части приложения.

Задача – реализовать часть бизнес логики приложения управления магазином автомобилей. Возможность добавлять марки машин, осуществлять поиск в системе и CRUD операции с компанией. Обеспечить наличие ролей.

Нам потребуется следующая структура:



Начнем с конфигурации проекта.

Создайте пустой проект с помощью maven. Далее откроем pom и воспользуемся зависимостями:

Добавить в Properties

<hibernate-version>5.0.1.Final</hibernate-version>

////

<dependencies>  
 <!--driver for connection to MYSql database -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.23</version>  
 </dependency>  
  
 <!-- Hibernate -->  
 <!-- to start need only this -->  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-core</artifactId>  
 <version>5.4.28.Final</version>  
 </dependency>  
  
 <!-- for JPA, use hibernate-entitymanager instead of hibernate-core -->  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-entitymanager</artifactId>  
 <version>${hibernate-version}</version>  
 </dependency>  
</dependencies>

После чего нам необходимо создать папку ресурсов и добавить конфигурацию подключения нашей БД к проекту.

Файл hibernate.cfg.xml

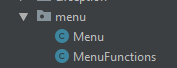
<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  
  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  
 <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/carshop</property>  
 <property name="hibernate.connection.username">root</property>  
 <property name="hibernate.connection.password">root</property>  
 <property name="hibernate.connection.pool\_size">1</property>  
 <!-- Echo all executed SQL to stdout -->  
 <property name="show\_sql">true</property>  
 <!-- SQL dialect -->  
 <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  
 <property name="hibernate.current\_session\_context\_class">thread</property>  
 <property name="hibernate.connection.CharSet">utf8</property>  
 <property name="hibernate.connection.characterEncoding">utf8</property>  
 <property name="hibernate.connection.useUnicode">true</property>  
 </session-factory>  
</hibernate-configuration>

Следующий файл – это класс Main

import menu.Menu;  
  
public class Main {  
 public static void main(String[] args) {  
 Menu menu = new Menu();  
 menu.IntroducingMenu();  
 }  
}

Тут мы сразу перенаправляем вызов на класс исполнителя. Методом хорошей практики считается не загружать класс Main кодом бизнес логики.

Поработаем с частью системы в пакете Menu



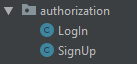
Наш класс Main перенаправил нас на метод касса Menu. Этот класс будет определять наборы меню для пользователей и вызов определенных методов – это первый слой нашей архитектуры, предназначенный для описания пользовательской информации и взаимодействия. Нас в первую очередь интересует самый первый метод, который вызывается из Main, остальное чуть позже. Давайте рассмотрим что он содержит:

package menu;  
  
import authorization.LogIn;  
import authorization.SignUp;  
import entity.User;  
  
import java.util.Scanner;  
  
public class Menu {  
  
 public Menu() { }  
  
 Scanner in = new Scanner(System.*in*);  
 MenuFunctions menuFunctions = new MenuFunctions();  
  
 public void IntroducingMenu() {  
 LogIn logIn = new LogIn();  
 SignUp signUp = new SignUp();  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 3) {  
 String s = "Меню\n" +  
 "1. Войти\n" +  
 "2. Зарегистрироваться\n" +  
 "3. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 logIn.authorization();  
 break;  
 case "2":  
 signUp.registration();  
 break;  
 case "3":  
 System.*out*.println("До свидания!");  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
 public void AdminMenu() {  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 4) {  
 String s = "Меню админа\n" +  
 "1. Работа с пользователями\n" +  
 "2. Работа с компаниями\n" +  
 "3. Работа с машинами\n" +  
 "4. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 AdminMenuWithUsers();  
 break;  
 case "2":  
 AdminMenuWithCompanies();  
 break;  
 case "3":  
 AdminMenuWithCars();  
 break;  
 case "4":  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
 public void UserMenu(User currentUser) {  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 6) {  
 String s = "Меню пользователя\n" +  
 "1. Просмотреть все компании\n" +  
 "2. Просмотреть все модели машин\n" +  
 "3. Найти компанию по названию\n" +  
 "4. Найти модель машины по названию\n" +  
 "5. Отредактировать свой профиль\n" +  
 "6. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 menuFunctions.showCompanies();  
 break;  
 case "2":  
 menuFunctions.showCars();  
 break;  
 case "3":  
 menuFunctions.showOneCompany();  
 break;  
 case "4":  
 menuFunctions.findCarByName();  
 break;  
 case "5":  
 menuFunctions.updateLoginAndPassword(currentUser);  
 break;  
 case "6":  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
 private void AdminMenuWithUsers() {  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 5) {  
 String s = "Работа с пользователями\n" +  
 "1. Добавить пользователя\n" +  
 "2. Изменить личные данные пользователя\n" +  
 "3. Удалить пользователя\n" +  
 "4. Просмотреть всех пользователей\n" +  
 "5. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 menuFunctions.addPerson();  
 break;  
 case "2":  
 menuFunctions.updatePerson();  
 break;  
 case "3":  
 menuFunctions.deletePerson();  
 break;  
 case "4":  
 menuFunctions.showPeople();  
 break;  
 case "5":  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
 public void AdminMenuWithCompanies() {  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 5) {  
 String s = "Работа с компаниями\n" +  
 "1. Добавить компанию\n" +  
 "2. Изменить компанию\n" +  
 "3. Удалить компанию\n" +  
 "4. Вывести все компании\n" +  
 "5. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 System.*out*.println(menuFunctions.addCompany());  
 break;  
 case "2":  
 menuFunctions.updateCompany();  
 break;  
 case "3":  
 menuFunctions.deleteCompany();  
 break;  
 case "4":  
 menuFunctions.showCompanies();  
 break;  
 case "5":  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
 public void AdminMenuWithCars() {  
 String choice = "0";  
 while (Integer.*parseInt*(choice) != 7) {  
 String s = "Работа с машинами\n" +  
 "1. Добавить машину\n" +  
 "2. Изменить машину\n" +  
 "3. Удалить машину\n" +  
 "4. Вывести все машины\n" +  
 "5. Вывести все машины одной компании\n" +  
 "6. Найти машину по названию\n" +  
 "7. Выход\n" +  
 "Выбор: ";  
 System.*out*.print(s);  
 choice = in.nextLine();  
 switch (choice) {  
 case "1":  
 menuFunctions.addCar();  
 break;  
 case "2":  
 menuFunctions.updateCar();  
 break;  
 case "3":  
 menuFunctions.deleteCar();  
 break;  
 case "4":  
 menuFunctions.showCars();  
 break;  
 case "5":  
 menuFunctions.showCarsFromOneCompany();  
 break;  
 case "6":  
 menuFunctions.findCarByName();  
 case "7":  
 break;  
 default:  
 System.*out*.println("Проверьте корректность ввода!");  
 choice = "0";  
 break;  
 }  
 }  
 }  
  
}

Как мы видим из данного класса есть 3 возможности. Получить доступ, зарегистрироваться и выйти.

При выборе пункта меню, нам потребуется вызвать методы класса Входа в систему и регистрации.

Рассмотрим эту часть:



Класс LogIn:

package authorization;  
  
import entity.Person;  
import entity.User;  
import menu.Menu;  
import service.PersonService;  
import service.serviceImpl.PersonServiceImpl;  
  
import java.util.List;  
import java.util.Scanner;  
  
public class LogIn {  
  
 Scanner in = new Scanner(System.*in*);  
  
 public void authorization() {  
  
 PersonService personService = new PersonServiceImpl();  
 List<Person> people = personService.showPeople();  
 System.*out*.print("Введите логин: ");  
 String login = in.nextLine();  
 System.*out*.print("Введите пароль: ");  
 String password = in.nextLine();  
 User currentUser = null;  
 for(Person p : people) {  
 if(p.getUser().getLogin().equals(login) && p.getUser().getPassword().equals(password)) {  
 currentUser = p.getUser();  
 p.setPersonId(people.size());  
 }  
 if (p.getUser().getLogin().equals(login) && !p.getUser().getPassword().equals(password)) {  
 System.*out*.println("Проверьте корректность пароля!");  
 }  
 }  
 if (currentUser != null) {  
 System.*out*.println("Авторизация пройдена успешно! Добро пожаловать " +  
 currentUser.getPerson().getSurname() + " " + currentUser.getPerson().getName());  
 Menu menu = new Menu();  
 String role = currentUser.getRole();  
 switch (role) {  
 case "Admin":  
 menu.AdminMenu();  
 break;  
 case "User":  
 menu.UserMenu(currentUser);  
 break;  
 }  
 }  
 else {  
 System.*out*.println("Такого пользователя не найдено");  
 }  
 }  
}

И класс SignUp:

package authorization;  
  
import menu.MenuFunctions;  
  
public class SignUp {  
  
 public void registration() {  
 MenuFunctions menuFunctions = new MenuFunctions();  
 menuFunctions.addPerson();  
 }  
}

Начнем с SignUp

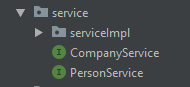
Тут мы видим создание объекта другого класса и перенаправление к нему. Вопрос – почему нельзя это было сделать в предыдущем классе? Зачем делать промежуточный класс? Ответ – во всех языках придерживаются правил написания. Существуют правила, как официальные, так и неофициальные. Например, самый распространённый стек правил, который вы могли слышать – SOLID, который устанавливает привольность или грамотность создания кода(если говорить утрированно). Также можно привести CodConventions и т.д.

Что бы закрыть раздел с Menu добавим код для классам MenuFunctions и рассмотрим его подробнее. Опять же нас будет пока что интересовать только первый метод, но остальные методы тоже быстро пробежимся. Что бы представлять назначение класса. Более подробно их рассмотрим при соответствующих вызовах.

package menu;  
  
import entity.Car;  
import entity.Company;  
import entity.Person;  
import entity.User;  
import service.CompanyService;  
import service.PersonService;  
import service.serviceImpl.CompanyServiceImpl;  
import service.serviceImpl.PersonServiceImpl;  
//import validator.Validator;  
  
import java.util.List;  
import java.util.Scanner;  
  
public class MenuFunctions {  
  
 Scanner in = new Scanner(System.*in*);  
 PersonService personService = new PersonServiceImpl();  
 CompanyService companyService = new CompanyServiceImpl();  
  
 public MenuFunctions() { }  
  
 //---PERSON AND USER---//  
  
 public void addPerson() {  
 System.*out*.println("---Добавление пользователя---");  
 Person person = getPersonInfo();  
 if (person != null) {  
 if (personService.addPerson(person)) {  
 System.*out*.println("---Добавление выполнено!---");  
 }  
 }  
 }  
  
 private Person getPersonInfo() {  
 Person person = null;  
 System.*out*.print("Введите имя пользователя: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите фамилию пользователя: ");  
 String surname = in.nextLine();  
 System.*out*.print("Введите возраст пользователя: ");  
 String age = in.nextLine();  
 System.*out*.print("Введите телефон пользователя: ");  
 String phone = in.nextLine();  
 System.*out*.print("Введите почту пользователя: ");  
 String mail = in.nextLine();  
 //if (Validator.correctPerson(name, surname, age, phone, mail)) {  
 User user = getUserInfo();  
 if (user != null) {  
 person = new Person(name, surname, Integer.*parseInt*(age), phone, mail);  
 user.setPerson(person);  
 person.setUser(user);  
 }  
 else {  
 System.*out*.println("Пароль или логин не корректны!");  
 }  
 /\* }  
 else {  
 System.out.println("Личные данные не корректны!");  
 }\*/  
 return person;  
 }  
  
 private User getUserInfo() {  
 User user = null;  
 System.*out*.print("Введите логин пользователя: ");  
 String login = in.nextLine();  
 System.*out*.print("Введите пароль пользователя: ");  
 String password = in.nextLine();  
 // if(Validator.correctUser(login, password)) {  
 if(checkUniqueLogin(login)) {  
 user = new User(login, password, "User");  
 }  
 else {  
 System.*out*.println("Такой логин уже занят!");  
 }  
 //}  
 return user;  
 }  
  
 private boolean checkUniqueLogin(String login) {  
 boolean isUnique = true;  
 for (Person p : getPeople()) {  
 if (p.getUser().getLogin().equals(login)) {  
 isUnique = false;  
 }  
 }  
 return isUnique;  
 }  
  
 public void updatePerson() {  
 System.*out*.println("---Изменение пользователя---");  
 showPeople();  
 System.*out*.print("Выберите ID пользователя для изменения: ");  
 String id = in.nextLine();  
 if (getPersonId(id)) {  
 Person person = findPersonById(Integer.*parseInt*(id));  
 if (person != null) {  
 changeDataFromPerson(person);  
 changeDataFromUser(person);  
 if (personService.updatePerson(person)) {  
 System.*out*.println("---Изменение выполнено!---");  
 }  
 }  
 }  
 }  
  
 private Person changeDataFromPerson(Person person) {  
 System.*out*.print("Введите имя пользователя: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите фамилию пользователя: ");  
 String surname = in.nextLine();  
 System.*out*.print("Введите возраст пользователя: ");  
 String age = in.nextLine();  
 System.*out*.print("Введите телефон пользователя: ");  
 String phone = in.nextLine();  
 System.*out*.print("Введите почту пользователя: ");  
 String mail = in.nextLine();  
 // if (Validator.correctPerson(name, surname, age, phone, mail)) {  
 person.setName(name);  
 person.setSurname(surname);  
 person.setAge(Integer.*parseInt*(age));  
 person.setPhone(phone);  
 person.setMail(mail);  
 /\* }  
 else {  
 System.out.println("Личные данные не корректны!");  
 }\*/  
 return person;  
 }  
  
 private Person changeDataFromUser(Person person) {  
  
 System.*out*.print("Введите логин пользователя: ");  
 String login = in.nextLine();  
 System.*out*.print("Введите пароль пользователя: ");  
 String password = in.nextLine();  
 // if(Validator.correctUser(login, password)) {  
 person.getUser().setLogin(login);  
 person.getUser().setPassword(password);  
 // }  
 return person;  
 }  
  
 public void updateLoginAndPassword(User user) {  
 System.*out*.println("---Изменение логина и пароля---");  
 changeDataFromUser(user.getPerson());  
 if (personService.updatePerson(user.getPerson())) {  
 System.*out*.println("---Изменение выполнено!---");  
 }  
  
 }  
  
 public void deletePerson() {  
 System.*out*.println("---Удаление пользователя---");  
 showPeople();  
 System.*out*.print("Выберите ID пользователя для изменения: ");  
 String id = in.nextLine();  
 if (getPersonId(id)) {  
 if (personService.deletePerson(Integer.*parseInt*(id))) {  
 System.*out*.println("---Удаление выполнено!---");  
 }  
 }  
 }  
  
 private boolean getPersonId(String id) {  
 boolean isAppropriateNumber = false;  
 // if (Validator.correctId(id)) {  
 if (!(Integer.*parseInt*(id) < 0) && !(Integer.*parseInt*(id) > getPeople().size())) {  
 isAppropriateNumber = true;  
 }  
 else {  
 System.*out*.println("Такого ID нет!");  
 }  
 /\* }  
 else {  
 System.out.println("ID не корректно!");  
 }\*/  
 return isAppropriateNumber;  
 }  
  
 public void showPeople() {  
 List<Person> people = getPeople();  
 if (people.size() != 0) {  
 System.*out*.format("%10s%20s%20s%10s%20s%30s%20s", "ID |", "Имя |", "Фамилия |", "Возраст |", "Телефон |", "Почта |", "Логин |");  
 for (Person p: people) {  
 System.*out*.println(" ");  
 System.*out*.format("%10s%20s%20s%10s%20s%30s%20s", p.getPersonId() + " |", p.getName() + " |",  
 p.getSurname() + " |", p.getAge() + " |",  
 p.getPhone() + " |", p.getMail() + " |", p.getUser().getLogin() + " |");  
  
 }  
 System.*out*.println(" ");  
 }  
 else {  
 System.*out*.println("Нет пользователей!");  
 }  
 }  
  
 private List<Person> getPeople() {  
 List<Person> people = personService.showPeople();  
 return people;  
 }  
  
 private Person findPersonById(int id) {  
 Person person = personService.findPersonById(id);  
 return person;  
 }  
  
 //---COMPANY---//  
  
 public String addCompany() {  
 System.*out*.println("---Добавление компании---");  
 String result = null;  
 System.*out*.print("Введите название компании: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите страну происхождения компании: ");  
 String country = in.nextLine();  
 // if (Validator.correctCompany(name, country)) {  
 Company company = new Company(name, country);  
 if (companyService.addCompany(company)) {  
 result = "---Добавление выполнено!---";  
 }  
 /\* }  
 else {  
 result = "Данные не корректны!";  
 }\*/  
 return result;  
 }  
  
 public String updateCompany() {  
 String result = null;  
 System.*out*.println("---Изменение компании---");  
 showCompanies();  
 System.*out*.print("Выберите ID компании для изменения: ");  
 String id = in.nextLine();  
 if (getCompanyId(id)) {  
 System.*out*.print("Введите название компании: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите страну происхождения компании: ");  
 String country = in.nextLine();  
 // if (Validator.correctCompany(name, country)) {  
 Company company = companyService.findCompanyById(Integer.*parseInt*(id));  
 company.setCompanyName(name);  
 company.setCompanyCountry(country);  
 if (companyService.updateCompany(company)) {  
 System.*out*.println("---Изменение выполнено!---");  
 }  
 /\* }  
 else {  
 System.out.println("Данные не корректны!");  
 }\*/  
 }  
  
 return result;  
 }  
  
 public void deleteCompany() {  
 System.*out*.println("---Удаление компании---");  
 showCompanies();  
 System.*out*.print("Введите ID компании для удаления: ");  
 String id = in.nextLine();  
 if (getCompanyId(id)) {  
 if(companyService.deleteCompany(Integer.*parseInt*(id))) {  
 System.*out*.println("---Удаление выполнено!---");  
 }  
 }  
 }  
  
 private boolean getCompanyId(String id) {  
 boolean isAppropriateNumber = false;  
 // if (Validator.correctId(id)) {  
 if (!(Integer.*parseInt*(id) < 0) && !(Integer.*parseInt*(id) > getCompanies().size())) {  
 isAppropriateNumber = true;  
 }  
 else {  
 System.*out*.println("Такого ID нет!");  
 }  
 /\* }  
 else {  
 System.out.println("ID не корректно!");  
 }\*/  
 return isAppropriateNumber;  
 }  
  
 public void showCompanies() {  
 List<Company> companies = getCompanies();  
 if (companies.size() != 0) {  
 theHeaderForCompany();  
 for (Company c: companies) {  
 theTableForCompany(c);  
 }  
 System.*out*.println(" ");  
 }  
 else {  
 System.*out*.println("Нет компаний!");  
 }  
 }  
  
 private void theTableForCompany(Company c) {  
 System.*out*.println(" ");  
 System.*out*.format("%10s%20s%30s", c.getCompanyId() + " |", c.getCompanyName() + " |", c.getCompanyCountry() + " |");  
 System.*out*.println(" ");  
 }  
  
 private void theHeaderForCompany() {  
 System.*out*.format("%10s%20s%30s"," ID |", "Название |", "Страна происхождения |");  
 }  
  
 private List<Company> getCompanies() {  
 List<Company> companies = companyService.showCompanies();  
 return companies;  
 }  
  
 public void showOneCompany() {  
 Company company = findCompanyByName();  
 if (company != null) {  
 theHeaderForCompany();  
 theTableForCompany(company);  
 }  
 }  
  
 private Company findCompanyByName() {  
 System.*out*.print("Введите название компании: ");  
 String name = in.nextLine();  
 boolean isFound = false;  
 for (Company c : getCompanies()) {  
 if (c.getCompanyName().equals(name)) {  
 isFound = true;  
 }  
 }  
 Company company = null;  
 if (isFound) {  
 company = companyService.findCompanyByName(name);  
 }  
 else {  
 System.*out*.println("Такой компании не найдено!");  
 }  
 return company;  
 }  
  
 //---CAR---//  
  
 public void addCar() {  
 System.*out*.println("---Добавление машины---");  
 showCompanies();  
 System.*out*.print("Выберите ID компании для авто: ");  
 String id = in.nextLine();  
 Car car = getCarInfo();  
 if (car != null) {  
 Company company = companyService.findCompanyById(Integer.*parseInt*(id));  
 car.setCompany(company);  
 company.addCar(car);  
 if (companyService.updateCompany(company)) {  
 System.*out*.println("---Добавление выполнено!---");  
 }  
 }  
 }  
  
 private Car getCarInfo() {  
 System.*out*.print("Введите название: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите год создания: ");  
 String year = in.nextLine();  
 System.*out*.print("Введите пробег: ");  
 String distance = in.nextLine();  
 System.*out*.print("Введите вид топлива: ");  
 String fuel = in.nextLine();  
 System.*out*.print("Введите расход: ");  
 String fuelConsumption = in.nextLine();  
 System.*out*.print("Введите цену: ");  
 String price = in.nextLine();  
 // if (Validator.correctCar(name, year, distance, fuel, fuelConsumption, price)) {  
 // if (Validator.correctFuel(fuel)) {  
 Car car = new Car();  
 car.setName(name);  
 car.setYear(Integer.*parseInt*(year));  
 car.setDistance(Integer.*parseInt*(distance));  
 car.setFuel(fuel);  
 car.setFuelConsumption(fuelConsumption);  
 car.setPrice(Integer.*parseInt*(price));  
 return car;  
 /\* }  
 else {  
 System.out.println("Введите топливо: Бензин или Дизель!");  
 }\*/  
 /\* }  
 else {  
 System.out.println("Данные не корректны!");  
 }\*/  
 }  
  
 public void deleteCar() {  
 System.*out*.println("---Удаление машины---");  
 showCars();  
 Company company = findCompanyByName();  
 if (company != null) {  
 System.*out*.print("Введите название машины: ");  
 String name = in.nextLine();  
 Car car = findCarInList(company, name);  
 if (car != null) {  
 company.getCars().remove(car);  
 if (companyService.updateCompany(company)) {  
 System.*out*.println("---Удаление выполнено!---");  
 }  
 }  
 }  
 }  
  
 public void updateCar() {  
 System.*out*.println("---Изменение машины---");  
 showCars();  
 Company company = findCompanyByName();  
 if (company != null) {  
 System.*out*.print("Введите название машины: ");  
 String nameForChange = in.nextLine();  
 Car car = findCarInList(company, nameForChange);  
 if (car != null) {  
 System.*out*.print("Введите название: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите год создания: ");  
 String year = in.nextLine();  
 System.*out*.print("Введите пробег: ");  
 String distance = in.nextLine();  
 System.*out*.print("Введите вид топлива: ");  
 String fuel = in.nextLine();  
 System.*out*.print("Введите расход: ");  
 String fuelConsumption = in.nextLine();  
 System.*out*.print("Введите цену: ");  
 String price = in.nextLine();  
 /\* if (Validator.correctCar(name, year, distance, fuel, fuelConsumption, price)) {  
 if (Validator.correctFuel(fuel)) {\*/  
 car.setName(name);  
 car.setYear(Integer.*parseInt*(year));  
 car.setDistance(Integer.*parseInt*(distance));  
 car.setFuel(fuel);  
 car.setFuelConsumption(fuelConsumption);  
 car.setPrice(Integer.*parseInt*(price));  
 }  
 else {  
 System.*out*.println("Введите топливо: Бензин или Дизель!");  
 }  
 /\* }  
 else {  
 System.out.println("Данные не корректны!");  
 }  
 if (companyService.updateCompany(company)) {  
 System.out.println("---Изменение выполнено!---");  
 }\*/  
 // }  
 }  
 }  
  
 public void findCarByName() {  
 System.*out*.print("Введите название машины: ");  
 String name = in.nextLine();  
 Car car = null;  
 for (Company company : getCompanies()) {  
 car = findCarInList(company, name);  
 if (car != null) {  
 theHeaderForCar();  
 theTableForCar(car);  
 }  
 }  
 System.*out*.println(" ");  
 }  
  
 private Car findCarInList(Company company, String name) {  
 Car car = null;  
 if (!company.getCars().isEmpty()) {  
 for (Car c : company.getCars()) {  
 if (c.getName().equals(name)) {  
 car = c;  
 }  
 }  
 if (car == null) {  
 System.*out*.println("Такой машины не найдено в компании " + company.getCompanyName());  
 }  
  
 }  
 return car;  
 }  
  
 public void showCarsFromOneCompany() {  
 showCompanies();  
 Company company = findCompanyByName();  
 if (company != null) {  
 if (!company.getCars().isEmpty()) {  
 theHeaderForCar();  
 for (Car c : company.getCars()) {  
 theTableForCar(c);  
 }  
 System.*out*.println(" ");  
 }  
 else {  
 System.*out*.println("Компания " + company.getCompanyName() + " не имеет моделей!");  
 }  
 }  
 }  
  
 public void showCars() {  
 List<Company> companies = getCompanies();  
 if (companies.size() != 0) {  
 theHeaderForCar();  
 for (Company c: companies) {  
 List<Car> cars = c.getCars();  
 if (!cars.isEmpty()) {  
 for (Car car : cars) {  
 theTableForCar(car);  
 }  
 System.*out*.println(" ");  
 }  
  
 }  
 }  
 else {  
 System.*out*.println("Нет пользователей!");  
 }  
 }  
  
 private void theHeaderForCar() {  
 System.*out*.format("%15s%20s%10s%10s%10s%15s%10s%20s", "ID |", "Название |", "Год |", "Пробег |", "Топливо |",  
 "Расход топлива |", "Цена |", " Компания|");  
 }  
  
 private void theTableForCar(Car car) {  
 System.*out*.println(" ");  
 System.*out*.format("%15s%20s%10s%10s%10s%16s%10s%20s", car.getCarId() + " |", car.getName() + " |",  
 car.getYear() + " |", car.getDistance() + " |", car.getFuel() + " |",  
 car.getFuelConsumption() + " |", car.getPrice() + " |", car.getCompany().getCompanyName() + " |");  
 }  
}

Вернемся к классу LogIn и рассмотрим часть логики для регистрации.

Из данного класса нас направляют еще в один слой архитектуры.



Данная часть отвечает за часть бизнес-логики выполняемой системой. В частности, мы должны обработать работу с Person. Рассмотрим PersonService.

package service;  
  
import entity.Person;  
  
import java.util.List;  
  
public interface PersonService {  
 boolean addPerson(Person person);  
 boolean updatePerson(Person person);  
 boolean deletePerson(int id);  
 List<Person> showPeople();  
 Person findPersonById(int id);  
}

Если классы Menu, LogIn и т.д. отвечают за порядок предоставления данных и связи пользовательского меню с обработкой действий системы. ТО начиная с уровня Сервисов мы описываем полноценную бизнес-логику приложения, как она работает с БД или иными источниками данных, как информацию хранит и какие действия с этой информацией может выполнить.

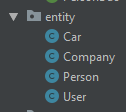
Наш интерфейс хранит методы, и мы их должны переопределить в PersonServiceImpl. Пока что так же мы рассматриваем часть – это первый метод.

package service.serviceImpl;  
  
import dao.PersonDao;  
import dao.daoImpl.PersonDaoImpl;  
import entity.Person;  
import exception.ShowException;  
import org.hibernate.HibernateError;  
import service.PersonService;  
  
import java.util.List;  
  
public class PersonServiceImpl implements PersonService {  
  
 PersonDao personDao = new PersonDaoImpl();  
  
 public PersonServiceImpl() {}  
  
 @Override  
 public boolean addPerson(Person person) {  
 boolean isAdded = false;  
 try {  
 if (personDao.addPerson(person))  
 isAdded = true;  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return isAdded;  
 }  
  
 @Override  
 public boolean updatePerson(Person person) {  
 boolean isUpdated = false;  
 try {  
 if (personDao.updatePerson(person))  
 isUpdated = true;  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return isUpdated;  
 }  
  
 @Override  
 public boolean deletePerson(int id) {  
 boolean isDeleted = false;  
 try {  
 if (personDao.deletePerson(id))  
 isDeleted = true;  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return isDeleted;  
 }  
  
 @Override  
 public List<Person> showPeople() {  
 List<Person> people = null;  
 try {  
 people = personDao.showPeople();  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return people;  
 }  
  
 @Override  
 public Person findPersonById(int id) {  
 Person person = null;  
 try {  
 person = personDao.findPersonById(id);  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return person;  
 }  
}

Разберем данный класс.

Из этого следует, что мы проваливаемся еще глубже в логику – в слой работы с БД или, иными словами, DAO (data access object). Теперь все более ярко становится ясным понятие Объектный язык программирования – начиная с простой инкапсуляции, POJO и уровнями архитектуры все завязано на этом представлении объектов и их трансформации.

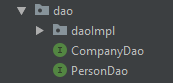
Начнем с Entity.



А именно с Person

package entity;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "people")  
public class Person {  
  
 @Id  
 @GeneratedValue (strategy = GenerationType.*IDENTITY*)  
 @Column (name = "person\_id")  
 private int personId;  
  
 @Column  
 private String surname;  
  
 @Column  
 private String name;  
  
 @Column  
 private int age;  
  
 @Column  
 private String phone;  
  
 @Column  
 private String mail;  
  
 @OneToOne(mappedBy = "person", cascade = CascadeType.*ALL*, orphanRemoval = true)  
 private User user;  
  
 public Person() {  
 }  
  
 public Person(String surname, String name, int age, String phone, String mail) {  
 this.surname = surname;  
 this.name = name;  
 this.age = age;  
 this.phone = phone;  
 this.mail = mail;  
 user = new User();  
 }  
  
 public int getPersonId() {  
 return personId;  
 }  
  
 public void setPersonId(int personId) {  
 this.personId = personId;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public String getPhone() {  
 return phone;  
 }  
  
 public void setPhone(String phone) {  
 this.phone = phone;  
 }  
  
 public String getMail() {  
 return mail;  
 }  
  
 public void setMail(String mail) {  
 this.mail = mail;  
 }  
  
 public User getUser() {  
 return user;  
 }  
  
 public void setUser(User user) {  
 this.user = user;  
 }  
  
 public void addUser(User newUser) {  
 newUser.setPerson(this);  
 user = newUser;  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "surname='" + surname + '\'' +  
 ", name='" + name + '\'' +  
 ", age=" + age +  
 ", phone='" + phone + '\'' +  
 ", mail='" + mail + '\'' +  
 '}';  
 }  
}

И после этого перейдем уже в DAO.



В интерфйсе PersonDao

package dao;  
  
import entity.Person;  
import entity.User;  
  
import java.util.List;  
  
public interface PersonDao {  
 boolean addPerson(Person person);  
 boolean updatePerson(Person person);  
 boolean deletePerson(int id);  
 List<Person> showPeople();  
 Person findPersonById(int id);  
}

И реализации Impl

package dao.daoImpl;  
  
import dao.PersonDao;  
import entity.Person;  
import org.hibernate.Query;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
import sessionFactory.SessionFactoryImpl;  
  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import java.util.List;  
  
public class PersonDaoImpl implements PersonDao {  
  
 @Override  
 public boolean addPerson(Person person) {  
 boolean isAdded = false;  
 try {  
 Session session = SessionFactoryImpl.*getSessionFactory*().openSession();  
 Transaction tx = session.beginTransaction();  
 session.save(person);  
 tx.commit();  
 session.close();  
 isAdded = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isAdded;  
 }  
  
 @Override  
 public boolean updatePerson(Person person) {  
 boolean isUpdated = false;  
 try {  
 Session session = SessionFactoryImpl.*getSessionFactory*().openSession();  
 Transaction tx = session.beginTransaction();  
 session.update(person);  
 tx.commit();  
 session.close();  
 isUpdated = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isUpdated;  
 }  
  
 @Override  
 public boolean deletePerson(int id) {  
 boolean isDeleted = false;  
 try {  
 Session session = SessionFactoryImpl.*getSessionFactory*().openSession();  
 Person person = session.load(Person.class, id);  
 Transaction tx = session.beginTransaction();  
 session.delete(person);  
 tx.commit();  
 session.close();  
 isDeleted = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isDeleted;  
 }  
  
 @Override  
 public List<Person> showPeople() {  
 List<Person> people = (List<Person>)SessionFactoryImpl.*getSessionFactory*().openSession().createQuery("FROM Person").list();  
 return people;  
 }  
  
 @Override  
 public Person findPersonById(int id) {  
 Person person = null;  
 try {  
 Session session = SessionFactoryImpl.*getSessionFactory*().openSession();  
 Transaction tx = session.beginTransaction();  
 CriteriaBuilder cb = session.getCriteriaBuilder();  
 CriteriaQuery<Person> cr = cb.createQuery(Person.class);  
 Root<Person> root = cr.from(Person.class);  
 cr.select(root).where(cb.equal(root.get("personId"), id));  
 person = session.createQuery(cr).getSingleResult();  
 tx.commit();  
 session.close();  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return person;  
 }  
}

Ну и наконец крайний класс:



package sessionFactory;  
  
  
import entity.Car;  
import entity.Company;  
import entity.Person;  
import entity.User;  
import org.hibernate.SessionFactory;  
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;  
import org.hibernate.cfg.Configuration;  
//import org.hibernate.cfg.Configuration;  
  
public class SessionFactoryImpl {  
 private static SessionFactory *sessionFactory*;  
  
 private SessionFactoryImpl() {}  
  
 public static SessionFactory getSessionFactory() {  
 if (*sessionFactory* == null) {  
 try {  
 Configuration configuration = new Configuration().configure();  
 configuration.addAnnotatedClass(Company.class);  
 configuration.addAnnotatedClass(Car.class);  
 configuration.addAnnotatedClass(Person.class);  
 configuration.addAnnotatedClass(User.class);  
 StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties());  
 *sessionFactory* = configuration.buildSessionFactory(builder.build());  
  
 } catch (Exception e) {  
 System.*out*.println("Исключение!" + e);  
 }  
 }  
 return *sessionFactory*;  
 }  
}

Давайте теперь добавим БД.

DROP SCHEMA IF EXISTS carshop;  
CREATE SCHEMA carshop;  
  
USE carshop;  
  
 CREATE TABLE `companies` (  
 `company\_id` int NOT NULL AUTO\_INCREMENT primary key,  
 `company\_name` varchar(45) NOT NULL,  
 `company\_country` varchar(45) NOT NULL);  
  
 CREATE TABLE `cars` (  
 `car\_id` int NOT NULL AUTO\_INCREMENT primary key,  
 `name` varchar(45) NOT NULL,  
 `year` int NOT NULL,  
 `distance` int DEFAULT NULL,  
 `fuel` varchar(45) NOT NULL DEFAULT 'Бензин',  
 `fuel\_consumption` varchar(45) NOT NULL,  
 `price` int NOT NULL,  
 `company\_id` int NOT NULL,  
 FOREIGN KEY (`company\_id`) REFERENCES companies(`company\_id`)  
 ON delete cascade  
 );  
  
CREATE TABLE `people` (  
 `person\_id` int NOT NULL primary key auto\_increment,  
 `surname` varchar(45) NOT NULL,  
 `name` varchar(45) NOT NULL,  
 `age` int NOT NULL,  
 `phone` varchar(45) NOT NULL,  
 `mail` varchar(45) NOT NULL  
);  
  
CREATE TABLE `users` (  
 `user\_id` int NOT NULL primary key auto\_increment,  
 `login` varchar(45) NOT NULL,  
 `password` varchar(45) NOT NULL,  
 `role` varchar(45) not null DEFAULT 'User',  
 `person\_id` int,  
  
 constraint `fk\_user\_person` foreign key (`person\_id`) references `people` (`person\_id`)  
 on delete cascade  
);

Рассмотрим БД и добавим Entity в проект:

Класс Car

package entity;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "cars")  
public class Car {  
  
 @Id  
 @GeneratedValue (strategy = GenerationType.*IDENTITY*)  
 @Column(name = "car\_id")  
 private int carId;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "year")  
 private int year;  
  
 @Column(name = "distance")  
 private int distance;  
  
 @Column(name = "fuel")  
 private String fuel;  
  
 @Column(name = "fuel\_consumption")  
 private String fuelConsumption;  
  
 @Column(name = "price")  
 private int price;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "company\_id")  
 private Company company;  
  
 public Car() {  
 }  
  
 public Car(String name, int year, int distance, String fuel, String fuelConsumption, int price) {  
 this.name = name;  
 this.year = year;  
 this.distance = distance;  
 this.fuel = fuel;  
 this.fuelConsumption = fuelConsumption;  
 this.price = price;  
 }  
  
 public int getCarId() {  
 return carId;  
 }  
  
 public void setCarId(int carId) {  
 this.carId = carId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getYear() {  
 return year;  
 }  
  
 public void setYear(int year) {  
 this.year = year;  
 }  
  
 public int getDistance() {  
 return distance;  
 }  
  
 public void setDistance(int distance) {  
 this.distance = distance;  
 }  
  
 public String getFuel() {  
 return fuel;  
 }  
  
 public void setFuel(String fuel) {  
 this.fuel = fuel;  
 }  
  
 public String getFuelConsumption() {  
 return fuelConsumption;  
 }  
  
 public void setFuelConsumption(String fuelConsumption) {  
 this.fuelConsumption = fuelConsumption;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public Company getCompany() {  
 return company;  
 }  
  
 public void setCompany(Company company) {  
 this.company = company;  
 }  
  
 @Override  
 public String toString() {  
 return "Car{" +  
 "carId=" + carId +  
 ", name='" + name + '\'' +  
 ", year=" + year +  
 ", distance=" + distance +  
 ", fuel='" + fuel + '\'' +  
 ", fuelConsumption='" + fuelConsumption + '\'' +  
 ", price=" + price +  
 ", company=" + company +  
 '}';  
 }  
}

Класс Company

package entity;  
  
import javax.persistence.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
@Entity  
@Table (name = "companies")  
public class Company {  
  
 @Id  
 @GeneratedValue (strategy = GenerationType.*IDENTITY*)  
 @Column(name = "company\_id")  
 private int companyId;  
  
 @Column(name = "company\_name")  
 private String companyName;  
  
 @Column(name = "company\_country")  
 private String companyCountry;  
  
 @OneToMany(mappedBy = "company", cascade = CascadeType.*ALL*, orphanRemoval = true, fetch = FetchType.*EAGER*)  
 private List<Car> cars;  
  
 public Company() {  
 }  
  
 public Company(String companyName, String companyCountry) {  
 this.companyName = companyName;  
 this.companyCountry = companyCountry;  
 cars = new ArrayList<>();  
 }  
  
 public int getCompanyId() {  
 return companyId;  
 }  
  
 public void setCompanyId(int companyId) {  
 this.companyId = companyId;  
 }  
  
 public String getCompanyName() {  
 return companyName;  
 }  
  
 public void setCompanyName(String companyName) {  
 this.companyName = companyName;  
 }  
  
 public String getCompanyCountry() {  
 return companyCountry;  
 }  
  
 public void setCompanyCountry(String companyCountry) {  
 this.companyCountry = companyCountry;  
 }  
  
 public List<Car> getCars() {  
 return cars;  
 }  
  
 public void setCars(List<Car> cars) {  
 this.cars = cars;  
 }  
  
 public void addCar(Car car) {  
 car.setCompany(this);  
 this.cars.add(car);  
 }  
  
 @Override  
 public String toString() {  
 return "Company{" +  
 "companyId=" + companyId +  
 ", companyName='" + companyName + '\'' +  
 ", companyCountry='" + companyCountry + '\'' +  
 '}';  
 }  
}

Класс User

package entity;  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "users")  
public class User {  
  
 @Id  
 @GeneratedValue (strategy = GenerationType.*IDENTITY*)  
 @Column(name = "user\_id")  
 private int userId;  
  
 @Column  
 private String login;  
  
 @Column  
 private String password;  
  
 @Column  
 private String role;  
  
 @OneToOne  
 @JoinColumn(name = "person\_id")  
 private Person person;  
  
 public User() {  
 }  
  
 public User(String login, String password) {  
 this.login = login;  
 this.password = password;  
 }  
  
 public User(String login, String password, String role) {  
 this.login = login;  
 this.password = password;  
 this.role = role;  
 }  
  
 public int getUserId() {  
 return userId;  
 }  
  
 public void setUserId(int userId) {  
 this.userId = userId;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getRole() {  
 return role;  
 }  
  
 public void setRole(String role) {  
 this.role = role;  
 }  
  
 public Person getPerson() {  
 return person;  
 }  
  
 public void setPerson(Person person) {  
 this.person = person;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "userId=" + userId +  
 ", login='" + login + '\'' +  
 ", password='" + password + '\'' +  
 ", role=" + role +  
 '}';  
 }  
}

Создадим оставшиеся интрфейсы:

CompanyService

package service;  
  
import entity.Company;  
import java.util.List;  
  
public interface CompanyService {  
 boolean addCompany(Company company);  
 boolean updateCompany(Company company);  
 boolean deleteCompany(int id);  
 List<Company> showCompanies();  
 Company findCompanyById(int id);  
 Company findCompanyByName(String name);  
}

CompanyDao

package dao;  
  
import entity.Company;  
  
import java.util.List;  
  
public interface CompanyDao {  
 boolean addCompany(Company company);  
 boolean updateCompany(Company company);  
 boolean deleteCompany(int id);  
 List<Company> showCompanies();  
 Company findCompanyById(int id);  
 Company findCompanyByName(String name);  
}

ServiceCompanyImpl

package service.serviceImpl;  
  
import dao.CompanyDao;  
import dao.daoImpl.CompanyDaoImpl;  
import entity.Company;  
import exception.ShowException;  
import org.hibernate.HibernateError;  
import service.CompanyService;  
  
import java.util.List;  
  
public class CompanyServiceImpl implements CompanyService {  
  
 CompanyDao companyDao = new CompanyDaoImpl();  
  
 public CompanyServiceImpl() {}  
  
 @Override  
 public boolean addCompany(Company company) {  
 boolean isAdded = false;  
 try {  
 companyDao.addCompany(company);  
 isAdded = true;  
 }  
 catch (HibernateError e) {  
 ShowException.*showNotice*(e);  
 }  
 return isAdded;  
 }  
  
 @Override  
 public boolean updateCompany(Company company) {  
 boolean isUpdated = false;  
 System.*out*.println("Это ДЗ");  
 return isUpdated;  
 }  
  
 @Override  
 public boolean deleteCompany(int id) {  
 System.*out*.println("Это ДЗ");  
 // return isDeleted;  
 return false;  
 }  
  
 @Override  
 public List<Company> showCompanies() {  
 System.*out*.println("Это ДЗ");  
 // return companies;  
 return null;  
 }  
  
 @Override  
 public Company findCompanyById(int id) {  
 System.*out*.println("Это ДЗ");  
 //return company;  
 return null;  
 }  
  
 @Override  
 public Company findCompanyByName(String name) {  
 System.*out*.println("Это ДЗ");  
 // return company;  
 return null;  
 }  
}

CompanyDaoImpl

package dao.daoImpl;  
  
import dao.CompanyDao;  
import entity.Company;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
import sessionFactory.SessionFactoryImpl;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import java.util.List;  
  
public class CompanyDaoImpl implements CompanyDao {  
  
 public CompanyDaoImpl() {  
 }  
  
 @Override  
 public boolean addCompany(Company company) {  
 boolean isAdded = false;  
 try {  
 Session session = SessionFactoryImpl.*getSessionFactory*().openSession();  
 Transaction tx = session.beginTransaction();  
 session.save(company);  
 tx.commit();  
 session.close();  
 isAdded = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isAdded;  
 }  
  
 @Override  
 public boolean updateCompany(Company company) {  
 boolean isUpdated = false;  
 try {  
 // Тут нужно обновление  
 isUpdated = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isUpdated;  
 }  
  
 @Override  
 public boolean deleteCompany(int id) {  
 boolean isDeleted = false;  
 try {  
 // Тут нужно удаление  
 isDeleted = true;  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return isDeleted;  
 }  
  
 @Override  
 public Company findCompanyById(int id) {  
 Company company = null;  
 try {  
 //Тут нужен поиск  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return company;  
 }  
  
 @Override  
 public Company findCompanyByName(String name) {  
 Company company = null;  
 try {  
 // Тут нужен поиск по имени  
 }  
 catch (NoClassDefFoundError e) {  
 System.*out*.println("Exception: " + e);  
 }  
 return company;  
 }  
  
 @Override  
 public List<Company> showCompanies() {  
 List<Company> companies = (List<Company>)SessionFactoryImpl.*getSessionFactory*().openSession().createQuery("From Com1pany").list();  
 return companies;  
 }  
}

Дополним наш пример JDK 11

Создайте класс Validator

package validator;  
  
import java.util.regex.Pattern;  
  
public class Validator {  
  
 private static final Pattern *STRING\_PATTERN* = Pattern.*compile*("^[\\p{L}]+$");  
 private static final Pattern *NUMBER\_PATTERN* = Pattern.*compile*("\\d+");  
 private static final Pattern *FUEL\_CONSUMPTION\_PATTERN* = Pattern.*compile*("\\-?\\d+(\\.\\d{0,})?");  
  
 public static boolean correctCompany(String name, String country) {  
 boolean isCorrect = true;  
 if (name.isBlank() || country.isBlank() || !*STRING\_PATTERN*.matcher(country).matches()) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctPerson(String name, String surname, String age, String phone, String mail) {  
 boolean isCorrect = true;  
 if (name.isBlank() || surname.isBlank() || age.isBlank() || !*NUMBER\_PATTERN*.matcher(age).matches() ||  
 phone.isBlank() || mail.isBlank()) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctUser(String login, String password) {  
 boolean isCorrect = true;  
 if (login.isBlank() || password.isBlank()) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctRole(String role) {  
 boolean isCorrect = true;  
 if (!role.equals("User") || !role.equals("Admin")) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctId(String id) {  
 boolean isCorrect = true;  
 if (id.isBlank() || !*NUMBER\_PATTERN*.matcher(id).matches()) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctCar(String name, String year, String distance,  
 String fuel, String fuelConsumption, String price) {  
 boolean isCorrect = true;  
 if (name.isBlank() || year.isBlank() || !*NUMBER\_PATTERN*.matcher(year).matches() || distance.isBlank() ||  
 !*NUMBER\_PATTERN*.matcher(distance).matches() || fuel.isBlank() || fuelConsumption.isBlank() ||  
 !*FUEL\_CONSUMPTION\_PATTERN*.matcher(fuelConsumption).matches() || price.isBlank() ||  
 !*NUMBER\_PATTERN*.matcher(price).matches()) {  
 isCorrect = false;  
 }  
 return isCorrect;  
 }  
  
 public static boolean correctFuel(String fuel) {  
 boolean isCorrect = false;  
 if (fuel.equals("Бензин") || fuel.equals("Дизель")) {  
 isCorrect = true;  
 }  
 return isCorrect;  
 }  
}