Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Базы данных

Отчет по лабораторной работе №1

Работу выполнил:
Раскин Андрей
Группа: 43501/3
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2017

1 Цель работы

Получить практические навыки работы с БД через механизм объектно-реляционного отображения.

2 Программа работы

1. Знакомство с фреймворком Ebean:

   установка

   создание проекта

   создание приложения

2. Формирование набора моделей, соответствующих схеме БД, полученной по результатам разработки схемы БД и модификации схемы

3. Знакомство с механизмом миграций: автоматическое формирование схемы БД с помощью миграций

4. Создание команд для заполнения БД тестовыми (по несколько записей в каждой таблице)

3 Теоретическая информация

ORM - техника взаимодействия с БД из приложения, обеспечивающая двустороннее преобразование записей в БД в объекты программы. Миграция - механизм, обеспечивающий поддержание соответствия набора моделей программы и схемы БД.

ORM-решением для языка Java, является технология Hibernate, которая не только заботится о связи Java классов с таблицами базы данных (и типов данных Java в типы данных SQL), но также предоставляет средства для автоматического построения запросов и извлечения данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL и JDBC кода. Hibernate генерирует SQL вызовы и освобождает разработчика от ручной обработки результирующего набора данных и конвертации объектов, сохраняя приложение портируемым во все SQL базы данных. В последующих проектах используется opensource ORM фреймворк Ebean. Из ключевых особенностей:

1. привычный маппинг (использует аннотации java.persistence);

2. простое API;

3. легок в настройке;

4. гибкий fetching связанных сущностей;

5. partial-выборки;

6. трекинг изменений;

7. отсутствие сессий;

8. собственная поддержка транзакций;

9. асинхронная загрузка;

4 Ход выполнения работы

4.1 Окружение

При разработке использовался язык Java 8. Для описания сущностей базы данных, как объектов языка использовалась технология JPA. JPA (Java Persistence API) это спецификация Java EE и Java SE, описывающая систему управления сохранением java объектов в таблицы реляционных баз данных в удобном виде. Сама Java не содержит реализации JPA, однако есть существует много реализаций данной спецификации от разных компаний (открытых и нет). Это не единственный способ сохранения java объектов в базы данных (ORM систем), но один из самых популярных в Java мире. Для сборки проекта используется Gradle - система автоматической сборки, построенная на принципах Apache Ant и

Apache Maven, но предоставляющая DSL на языке Groovy вместо традиционной XML-образной формы представления конфигурации проекта.

Для PostgreSQL была создана база данных clinic_db, пользователь использовался стандартный - **postgres**. Далее был создан проект, а также скрипт Gradle для его сборки:

```
group 'com.xerocry'
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    compile group: 'javax.validation', name: 'validation-api', version: '1.1.0.Final'
}

dependencies {
    compile group: 'org.postgresql', name: 'postgresql', version: '9.4.1212.jre7'
    compile 'io.ebean:persistence-api:2.2.1'
    compile 'io.ebean:ebean:10.1.6'
    compile 'io.ebean:ebean-querybean:10.1.1'
    compile group: 'io.ebean', name: 'querybean-generator', version: '10.1.2'
    compile group: 'org.assertj', name: 'assertj-core', version: '3.6.2'
    compile group: 'org.slf4j', name: 'slf4j-simple', version: '1.7.23'
    compile group: 'org.testng', name: 'testng', version: '6.10'

}

buildscript {
    repositories {
        maven {
            url "https://plugins.gradle.org/m2/"
        }
    }
    dependencies {
        classpath "gradle.plugin.org.kt3k:ebean-enhance-plugin:3.0.0"
        classpath group: 'org.postgresql', name: 'postgresql', version: '9.4.1212.jre7'
    }
}

apply plugin: "com.github.kt3k.ebean.enhance"
```

Для подключения к базе данных необходимо указать хост, порт, имя базы данных и логин/пароль в специальном файле настроек для Ebean:

```
# the name of the default server
datasource.default=pg

ebean.migration.run=true

datasource.pg.username=postgres
datasource.pg.password=65225855
datasource.pg.databaseUrl=jdbc:postgresql://127.0.0.1:5432/clinic_db
datasource.pg.databaseDriver=org.postgresql.Driver
```

Затем для каждой сущности базы данных создадим класс Java. Здесь я приведу некоторые особенности описания объектов. Полный код можно найти в дополнении к данному отчёту.

Для создания таблицы необходимо указать аннотацию @Entity и унаследовать класс от базового класса *Model*. Когда класс наследуется от *Model*, он приобретает функции

1. save() - Сохранить сущность
2. supdate() - Обновить
3. sdelete() - Удалить
4. srefresh() - Обновить сущность **из** базы данных
5. ...

При объявлении переменной в таком классе она автоматически становится полем таблицы с именем переменной.

1. Аннотации для создания таблицы с простыми полями.

   (a) Для создания таблицы необходимо указать аннотацию @Entity

   (b) Для форсированного указания имени необходимо указать аннотацию @Column(name = %name %).

   (c) Для указания поля, которое будет являться Primary key существует аннотация @Id. С помощью неё генерируется автоинкрементируемое поле id. Для PostgreSQL счётчик id один для всей базы данных. Это можно изменить, но для данного проекта это не требуется.

   Пример создания простой таблицы с автоинкрементируемым полем id и и полем строкового типа.

```
@Entity
public class Departments extends Model {

    @Id
    Long depart_id;

    @Column(length=50, nullable = false)
    String depart_name;

    public Departments(String depart_name) {
        this.depart_name = depart_name;
    }

    public Long getDepart_id() {
        return depart_id;
    }

    public void setDepart_id(Long depart_id) {
        this.depart_id = depart_id;
    }

    public String getDepart_name() {
        return depart_name;
    }

    public void setDepart_name(String depart_name) {
        this.depart_name = depart_name;
    }
}
```

2. Аннотации для создания Foreign key. Для создания внешнего ключа могут быть использованы разные методы.

   Для создания отношения Многие-ко-Многим используется аннотация ManyToMany. При этом создаётся промежуточная bridge-таблицы соотношений. Пример:

```
package com.xerocry.domain;

import io.ebean.Model;
import io.ebean.annotation.EnumValue;

import javax.persistence.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by raskia on 2/24/2017.
 */
@Entity
public class Patients extends Model {

    @Id
    @Column(name = "patient_id")
    Long patientId;

    @Column(name = "reg_date")
    LocalDate regDate;
```

```
23
24      String city;
25
26      @Column(name = "p_name", nullable = false)
27      String name;
28
29      @Column(name = "dob", nullable = false)
30      LocalDate birthDate;
31
32      public enum Gender {
33          @EnumValue("M")
34          MALE,
35          @EnumValue("F")
36          FEMALE,
37      }
38
39      @Column(nullable = false)
40      Gender gender;
41
42      @OneToMany(mappedBy = "patientId")
43      List<Treatment> treatments = new ArrayList<>();
44
45      @ManyToMany
46      @JoinTable(name = "PAYMENT_PATIENT",
47              joinColumns = @JoinColumn(name = "patient_id", referencedColumnName = "
                    ↪ patient_id"),
48              inverseJoinColumns = @JoinColumn(name = "payment_id", referencedColumnName
                    ↪ = "payment_id"))
49      List<Payments> payments;
50
51      public Patients(String name, LocalDate birthDate, Gender gender) {
52          this.name = name;
53          this.birthDate = birthDate;
54          this.gender = gender;
55      }
56
57      public LocalDate getRegDate() {
58          return regDate;
59      }
60
61      public void setRegDate(LocalDate regDate) {
62          this.regDate = regDate;
63      }
64
65      public String getCity() {
66          return city;
67      }
68
69      public void setCity(String city) {
70          this.city = city;
71      }
72
73      public String getName() {
74          return name;
75      }
76
77      public void setName(String name) {
78          this.name = name;
79      }
80
81      public LocalDate getBirthDate() {
82          return birthDate;
83      }
84
85      public void setBirthDate(LocalDate birthDate) {
86          this.birthDate = birthDate;
87      }
88
89      public Gender getGender() {
90          return gender;
91      }
92
93      public void setGender(Gender gender) {
94          this.gender = gender;
95      }
96
```

```
97      public List<Treatment> getTreatments() {
98          return treatments;
99      }
100
101     public void addTreatments(Treatment treatments) {
102         this.treatments.add(treatments);
103     }
104
105     public List<Payments> getPayments() {
106         return payments;
107     }
108
109     public void addPayments(Payments payments) {
110         this.payments.add(payments);
111     }
112 }
```

Для создания отношения Многие-к-Одному используется аннотация ManyToOne. Пример:

```
1  package com.xerocry.domain;
2
3  import io.ebean.Model;
4  import javax.persistence.Column;
5  import javax.persistence.Entity;
6  import javax.persistence.Id;
7  import javax.persistence.ManyToOne;
8
9  /**
10  * Created by raskia on 2/23/2017.
11  */
12 @Entity
13 public class Diseases extends Model {
14     @Id
15     @Column(name = "disease_id")
16     Long diseaseId;
17
18     @Column(length = 50)
19     String symptoms;
20
21     @ManyToOne(optional = false)
22     DiseasesTypes disType;
23
24     @Column(length = 50, nullable = false, name = "disease_name")
25     String disName;
26
27     public Diseases(DiseasesTypes disType, String disName) {
28         this.disType = disType;
29         this.disName = disName;
30     }
31
32     public String getSymptoms() {
33         return symptoms;
34     }
35
36     public void setSymptoms(String symptoms) {
37         this.symptoms = symptoms;
38     }
39
40     public DiseasesTypes getDisType() {
41         return disType;
42     }
43
44     public void setDisType(DiseasesTypes disType) {
45         this.disType = disType;
46     }
47
48     public String getDisName() {
49         return disName;
50     }
51
52     public void setDisName(String disName) {
53         this.disName = disName;
54     }
55 }
```

Один-ко-Многим это практически обратная связь, чтобы отобразить суть отношени объектов.

4.2 Миграция

Миграции существуют для переноса изменений в моделях (добавление поля, удаление модели и т.д.) на структуру базы данных. Сначала создадим новую миграцию.

```
/**
 * Generate the DB Migration.
 */
public class MainDbMigration {

  /**
   * Generate the next "DB schema DIFF" migration.
   * <p>
   * These migration are typically run using FlywayDB, Liquibase
   * or Ebean's own built in migration runner.
   * </p>
   */
  public static void main(String[] args) throws IOException {

    // optionally specify the version and name
    //System.setProperty("ddl.migration.version", "1.1");
    //System.setProperty("ddl.migration.name", "add bars");

    // generate a migration using drops from a prior version
    //System.setProperty("ddl.migration.pendingDropsFor", "1.2");

//    Class.forName("com.")
    DbMigration dbMigration = new DbMigration();
    dbMigration.setPlatform(Platform.POSTGRES);
    // generate the migration ddl and xml
    // ... with EbeanServer in "offline" mode
    dbMigration.generateMigration();
  }
}
```

В процессе создания генерируется SQL-скрипт на языке DDL на основе созданных классов и аннотаций:

```
-- apply changes
create table departments (
  depart_id                     bigserial not null,
  depart_name                   varchar(50) not null,
  constraint pk_departments primary key (depart_id)
);

create table diseases (
  disease_id                    bigserial not null,
  symptoms                      varchar(50),
  dis_type_type_id              bigint not null,
  disease_name                  varchar(50) not null,
  constraint pk_diseases primary key (disease_id)
);

create table diseases_types (
  type_id                       bigserial not null,
  dis_type                      varchar(50),
  constraint pk_diseases_types primary key (type_id)
);

create table doctors (
  doctor_id                     bigserial not null,
  years_of_expirience           integer,
  skill_desc                    varchar(50),
  hire_date                     date not null,
  depart_id_depart_id           bigint not null,
  constraint pk_doctors primary key (doctor_id)
);

create table drugs (
  drug_id                       bigserial not null,
```

```
  type_id_type_id               bigint not null,
  drug_name                     varchar(50),
  price                         integer,
  constraint pk_drugs primary key (drug_id)
);

create table grants (
  grant_id                      bigserial not null,
  grant_sum                     bigint,
  grant_date                    date not null,
  paid_up                       boolean,
  service_service_id            bigint,
  drug_drug_id                  bigint,
  doctor_doctor_id              bigint,
  patient_patient_id            bigint,
  constraint pk_grants primary key (grant_id)
);

create table patients (
  patient_id                    bigserial not null,
  reg_date                      date,
  city                          varchar(255),
  p_name                        varchar(255) not null,
  dob                           date not null,
  gender                        varchar(1) not null,
  constraint ck_patients_gender check ( gender in ('M','F')),
  constraint pk_patients primary key (patient_id)
);

create table services (
  service_id                    bigserial not null,
  service_name                  varchar(50) not null,
  price                         integer,
  constraint pk_services primary key (service_id)
);

create table treatment (
  treatment_id                  bigserial not null,
  patient_id_patient_id         bigint not null,
  doctor_id_doctor_id           bigint not null,
  disease_id_disease_id         bigint,
  start_date                    date not null,
  end_date                      date,
  treatment                     varchar(255),
  constraint pk_treatment primary key (treatment_id)
);

create table treatment_drugs (
  treatment_id                  bigint not null,
  drug_id                       bigint not null,
  constraint pk_treatment_drugs primary key (treatment_id,drug_id)
);

create table treatment_services (
  treatment_id                  bigint not null,
  service_id                    bigint not null,
  constraint pk_treatment_services primary key (treatment_id,service_id)
);

alter table diseases add constraint fk_diseases_dis_type_type_id foreign
    ↪ key (dis_type_type_id) references diseases_types (type_id) on delete
```

```
    ↪  restrict on update restrict;
93 create index ix_diseases_dis_type_type_id on diseases (dis_type_type_id);
94
95 alter table doctors add constraint fk_doctors_depart_id_depart_id foreign
    ↪ key (depart_id_depart_id) references departments (depart_id) on
    ↪ delete restrict on update restrict;
96 create index ix_doctors_depart_id_depart_id on doctors (
    ↪ depart_id_depart_id);
97
98 alter table drugs add constraint fk_drugs_type_id_type_id foreign key (
    ↪ type_id_type_id) references diseases_types (type_id) on delete
    ↪ restrict on update restrict;
99 create index ix_drugs_type_id_type_id on drugs (type_id_type_id);
100
101 alter table grants add constraint fk_grants_service_service_id foreign key
    ↪  (service_service_id) references services (service_id) on delete
    ↪ restrict on update restrict;
102 create index ix_grants_service_service_id on grants (service_service_id);
103
104 alter table grants add constraint fk_grants_drug_drug_id foreign key (
    ↪ drug_drug_id) references drugs (drug_id) on delete restrict on
    ↪ update restrict;
105 create index ix_grants_drug_drug_id on grants (drug_drug_id);
106
107 alter table grants add constraint fk_grants_doctor_doctor_id foreign key (
    ↪ doctor_doctor_id) references doctors (doctor_id) on delete restrict
    ↪ on update restrict;
108 create index ix_grants_doctor_doctor_id on grants (doctor_doctor_id);
109
110 alter table grants add constraint fk_grants_patient_patient_id foreign key
    ↪  (patient_patient_id) references patients (patient_id) on delete
    ↪ restrict on update restrict;
111 create index ix_grants_patient_patient_id on grants (patient_patient_id);
112
113 alter table treatment add constraint fk_treatment_patient_id_patient_id
    ↪ foreign key (patient_id_patient_id) references patients (patient_id)
    ↪  on delete restrict on update restrict;
114 create index ix_treatment_patient_id_patient_id on treatment (
    ↪ patient_id_patient_id);
115
116 alter table treatment add constraint fk_treatment_doctor_id_doctor_id
    ↪ foreign key (doctor_id_doctor_id) references doctors (doctor_id) on
    ↪ delete restrict on update restrict;
117 create index ix_treatment_doctor_id_doctor_id on treatment (
    ↪ doctor_id_doctor_id);
118
119 alter table treatment add constraint fk_treatment_disease_id_disease_id
    ↪ foreign key (disease_id_disease_id) references diseases (disease_id)
    ↪  on delete restrict on update restrict;
120 create index ix_treatment_disease_id_disease_id on treatment (
    ↪ disease_id_disease_id);
121
122 alter table treatment_drugs add constraint fk_treatment_drugs_treatment
    ↪ foreign key (treatment_id) references treatment (treatment_id) on
    ↪ delete restrict on update restrict;
123 create index ix_treatment_drugs_treatment on treatment_drugs (treatment_id
    ↪ );
124
125 alter table treatment_drugs add constraint fk_treatment_drugs_drugs
    ↪ foreign key (drug_id) references drugs (drug_id) on delete restrict
    ↪ on update restrict;
```

```
create index ix_treatment_drugs_drugs on treatment_drugs (drug_id);

alter table treatment_services add constraint
    ↪ fk_treatment_services_treatment foreign key (treatment_id)
    ↪ references treatment (treatment_id) on delete restrict on update
    ↪ restrict;
create index ix_treatment_services_treatment on treatment_services (
    ↪ treatment_id);

alter table treatment_services add constraint
    ↪ fk_treatment_services_services foreign key (service_id) references
    ↪ services (service_id) on delete restrict on update restrict;
create index ix_treatment_services_services on treatment_services (
    ↪ service_id);
```

Затем применим текущую миграцию:

```
public class ApplyDbMigration {


  public static void main(String[] args) {

    // ignore test-ebean.properties
    System.setProperty("disableTestProperties", "true");

    // starting EbeanServer triggers the apply of migrations
    // ... when ebean.migration.run=true
    Ebean.getDefaultServer();

    System.out.println("done");
  }

}
```

Увидим, что есть одна локальная и одна успешно применённая миграция.

```
[main] INFO io.ebeaninternal.server.core.bootup.BootupClassPathSearch - Classpath search
    ↪ entities[10] searchTime[660] in packages[[]]
[main] INFO org.avaje.datasource.pool.ConnectionPool - DataSourcePool [pg] autoCommit[false]
    ↪  transIsolation[READ_COMMITTED] min[2] max[100]
[main] INFO io.ebean.internal.DefaultContainer - DatabasePlatform name:pg platform:postgres
[main] INFO io.ebean.dbmigration.MigrationRunner - local migrations:1  existing migrations:1
done
```

По завершению миграции, база данных содержит все таблицы из схемы, а также таблицу db_migrations, которая необходима для работы системы миграции.

4.3 Заполнение данными

Для заполнения тестовыми данными таблиц создадим отдельный класс:

```
public class LoadExampleData {

    private static boolean runOnce;

    private static EbeanServer server = Ebean.getServer(null);

    public static synchronized void load() {

        if (runOnce) {
            return;
        }

        final LoadExampleData me = new LoadExampleData();

        server.execute(() -> {
            me.deleteAll();
            me.insertPatients();
            me.createTreatment("Treat1", LocalDate.now(), LocalDate.of(2015, 12, 02));
            me.createTreatment("Treat2", LocalDate.now(), LocalDate.of(2016, 11, 02));
        });
        runOnce = true;
```

```
22      }
23
24      public void deleteAll() {
25          Ebean.execute(() -> {
26              // orm update use bean name and bean properties
27              server.createUpdate(Departments.class, "delete from departments").execute();
28              server.createUpdate(Diseases.class, "delete from diseases").execute();
29              server.createUpdate(Doctors.class, "delete from doctors").execute();
30              server.createUpdate(Drugs.class, "delete from drugs").execute();
31              server.createUpdate(DiseasesTypes.class, "delete from diseasesTypes").execute();
32              server.createUpdate(Grants.class, "delete from grants").execute();
33              server.createUpdate(Patients.class, "delete from patients").execute();
34              server.createUpdate(Services.class, "delete from services").execute();
35              server.createUpdate(Treatment.class, "delete from treatment").execute();
36          });
37      }
38
39
40      public void insertPatients(){
41          server.execute(()->{
42             new Patients( "Andrey", LocalDate.now(), Patients.Gender.MALE).save();
43             new Patients( "Marina", LocalDate.now(), Patients.Gender.FEMALE).save();
44             new Patients( "Derek", LocalDate.now(), Patients.Gender.MALE).save();
45          });
46      }
47
48
49
50      public void insertDoctors(){
51          server.execute(()->{
52              new Doctors(5,"Can heal", LocalDate.of(1995, 03, 12)).save();
53          });
54      }
55
56      private static Departments insertDepartment(String name) {
57          Departments department = new Departments(name);
58          Ebean.save(department);
59          return department;
60      }
61
62      public Doctors createDoctor(String skills, int exp, LocalDate hiredDate) {
63          Departments department = insertDepartment("Depart" + UUID.randomUUID().toString());
64          Doctors doctor = new Doctors(exp, skills, hiredDate);
65          doctor.setDepartId(department);
66          Ebean.save(doctor);
67          return doctor;
68      }
69
70      public static DiseasesTypes insertType(String type) {
71          DiseasesTypes disType = new DiseasesTypes(type);
72          Ebean.save(disType);
73          return disType;
74      }
75
76      public Diseases createDisease(String name) {
77          Diseases dis = new Diseases(insertType("type"+UUID.randomUUID().toString()), name);
78          Ebean.save(dis);
79          return dis;
80      }
81
82
83      public static Patients createPatient(LocalDate regDate, String city, String name,
            ↪ LocalDate birthDate, Patients.Gender gender) {
84          Patients patient = new Patients(name, birthDate, gender);
85          if(regDate != null){
86              patient.setRegDate(regDate);
87          }
88          if (city != null) {
89              patient.setCity(city);
90          }
91          Ebean.save(patient);
92          return patient;
93      }
94
95      public void createTreatment(String treatment, LocalDate endDate, LocalDate startDate) {
96          Treatment treatment1 = new Treatment(startDate);
```

```
97          treatment1.setDoctorId(createDoctor(UUID.randomUUID().toString(), 10, LocalDate.of
                ↪ (1995, 10, 1)));
98          treatment1.setPatientId(createPatient(LocalDate.now(), "Piter", "Andrey", LocalDate.
                ↪ now(), Patients.Gender.MALE));
99          treatment1.setDiseaseId(createDisease("dis" + UUID.randomUUID().toString()));
100         if (treatment != null) {
101             treatment1.setTreatment(treatment);
102         }
103         if (endDate != null) {
104             treatment1.setEndDate(endDate);
105         }
106         Ebean.save(treatment1);
107     }
108 }
```

5 Выводы

В данной работы было проведено знакомство с фреймворком Ebean для Java, позволяющим создавать ORM представление базы данных, миграциями моделей, а также с manage-командами для наполнения базы данных. Из достоинств фреймворка можно выделить

1. Ускорение процесса изменения схемы базы данных;
2. Возможность отслеживания схемы базы данных;
3. Поддержка многими бэкендами(PostgreSQL, MySQL, SQLite);
4. Возможность отката.

ORM как раз и предназначен для инкапсуляции бизнес логики, но не на уровне СУБД, а на уровне приложения. ORM дает много других приемуществ: валидация, кеширование, разделение прав доступа, миграции и много других готовых вещей, которые не нужно изобретать заново. Использование ORM в проекте избавляет разработчика от необходимости работы с SQL и написания большого количества кода, часто однообразного и подверженного ошибкам. Весь генерируемый ORM код предположительно хорошо проверен и оптимизирован, поэтому не нужно в целом задумывается о его тестировании. Однако при больших и тяжёлых запросах всё таки эффективнее использовать прямые SQL запросы.

6 Дополнения

```
1  @Entity
2  public class Departments extends Model {
3
4      @Id
5      Long depart_id;
6
7      @Column(length=50, nullable = false)
8      String depart_name;
9
10     /*@OneToMany(mappedBy = "DOCTORS")
11     List<Doctors> doctors;*/
12
13     public Departments(String depart_name) {
14         this.depart_name = depart_name;
15     }
16
17     public Long getDepart_id() {
18         return depart_id;
19     }
20
21     public void setDepart_id(Long depart_id) {
22         this.depart_id = depart_id;
23     }
24
25     public String getDepart_name() {
26         return depart_name;
27     }
28
29     public void setDepart_name(String depart_name) {
30         this.depart_name = depart_name;
```

```
    }

    /*public List<Doctors> getDoctors() {
        return doctors;
    }

    public void setDoctors(List<Doctors> doctors) {
        this.doctors = doctors;
    }*/
}
```

```
package com.xerocry.domain;

import io.ebean.Model;

import javax.persistence.*;
import java.time.LocalDate;

/**
 * Created by raskia on 2/23/2017.
 */
@Entity
@Table(name = "DOCTORS")
public class Doctors extends Model {

    @Id
    @Column(name = "doctor_id")
    Long id;

/*    @OneToMany(mappedBy = "Doctors")
    @Column(name = "treatment_id")
    List<Treatment> treatmentId = new ArrayList<>();*/

    @Column(name = "years_of_expirience")
    Integer experience;

    @Column(length=50, name = "skill_desc")
    String skills;

    @Column(nullable = false, name = "hire_date")
    LocalDate hireDate;

    @ManyToOne(optional = false)
    @Column(name = "depart_id")
    Departments departId;

    public Doctors(Integer experience, String skills, LocalDate hireDate) {
        this.experience = experience;
        this.skills = skills;
        this.hireDate = hireDate;
    }

    /* public List<Treatment> getTreatmentId() {
        return treatmentId;
    }

    public void setTreatmentId(List<Treatment> treatmentId) {
        this.treatmentId = treatmentId;
    }*/

    public Integer getExperience() {
        return experience;
    }

    public void setExperience(Integer experience) {
        this.experience = experience;
    }

    public String getSkills() {
        return skills;
    }

    public void setSkills(String skills) {
        this.skills = skills;
    }

```

```
    public LocalDate getHireDate() {
        return hireDate;
    }

    public void setHireDate(LocalDate hireDate) {
        this.hireDate = hireDate;
    }

    public Departments getDepartId() {
        return departId;
    }

    public void setDepartId(Departments departId) {
        this.departId = departId;
    }
}
```

```
@Entity
public class Patients extends Model {
        ...
    @ManyToMany
    @JoinTable(name = "PAYMENT_PATIENT",
            joinColumns = @JoinColumn(name = "patient_id", referencedColumnName = "
                ↪ patient_id"),
            inverseJoinColumns = @JoinColumn(name = "payment_id", referencedColumnName = "
                ↪ payment_id"))
    List<Payments> payments;
         ...
}

@Entity
public class Payments extends Model {
        ...
    @ManyToMany(mappedBy = "payments")
    List<Patients> patients = new ArrayList<>();
        ...
}
```

```
package com.xerocry.domain;

import io.ebean.Model;

import javax.persistence.*;
import java.time.LocalDate;

/**
 * Created by raskia on 2/24/2017.
 */
@Entity
public class Grants extends Model {

    @Id
    @Column(name = "grant_id")
    Long grantId;

    @Column(name = "grant_sum")
    Long sum;

    @Column(name = "grant_date", nullable = false)
    LocalDate date;

    @Column(name = "paid_up")
    Boolean paidUp;

    @ManyToOne
    @Column(name = "service_id")
    Services service;

    @ManyToOne
    @Column(name = "drug_id")
    Drugs drug;

    @ManyToOne
    @Column(name = "doctor_id")
    Doctors doctor;
```

```
38
39      @ManyToOne
40      @Column(name = "patient_id")
41      Patients patient;
42
43      public Grants(LocalDate date, Boolean paidUp, Doctors doctor, Patients patient) {
44          this.date = date;
45          this.paidUp = paidUp;
46          this.doctor = doctor;
47          this.patient = patient;
48      }
49
50      public Long getSum()      {
51          return sum;
52      }
53
54      public void setSum(Long sum) {
55          this.sum = sum;
56      }
57
58      public LocalDate getDate() {
59          return date;
60      }
61
62      public void setDate(LocalDate date) {
63          this.date = date;
64      }
65
66      public Boolean getPaidUp() {
67          return paidUp;
68      }
69
70      public void setPaidUp(Boolean paidUp) {
71          this.paidUp = paidUp;
72      }
73
74      public Services getService() {
75          return service;
76      }
77
78      public void setService(Services service) {
79          this.service = service;
80      }
81
82      public Drugs getDrug() {
83          return drug;
84      }
85
86      public void setDrug(Drugs drug) {
87          this.drug = drug;
88      }
89
90      public Doctors getDoctor() {
91          return doctor;
92      }
93
94      public void setDoctor(Doctors doctor) {
95          this.doctor = doctor;
96      }
97
98      public Patients getPatient() {
99          return patient;
100     }
101
102     public void setPatient(Patients patient) {
103         this.patient = patient;
104     }
105 }
```

```
1  package com.xerocry.domain;
2
3  import io.ebean.Model;
4
5  import javax.persistence.*;
6  import java.util.ArrayList;
7  import java.util.List;
```

```

/**
 * Created by raskia on 2/23/2017.
 */
@Entity
public class Drugs extends Model {

    @Id
    @Column(name = "drug_id")
    Long drugId;

    @ManyToOne(optional = false)
    @Column(name = "type_id")
    DiseasesTypes typeId;

    @Column(length = 50, name = "drug_name")
    String drugName;

    Integer price;

    @ManyToMany(mappedBy = "drugs")
    List<Treatment> treatments = new ArrayList<>();

    public Drugs(DiseasesTypes typeId, String drugName, Integer price) {
        this.typeId = typeId;
        this.drugName = drugName;
        this.price = price;
    }

    public DiseasesTypes getTypeId() {
        return typeId;
    }

    public void setTypeId(DiseasesTypes typeId) {
        this.typeId = typeId;
    }

    public String getDrugName() {
        return drugName;
    }

    public void setDrugName(String drugName) {
        this.drugName = drugName;
    }

    public Integer getPrice() {
        return price;
    }

    public void setPrice(Integer price) {
        this.price = price;
    }

    public List<Treatment> getTreatments() {
        return treatments;
    }

    public void setTreatments(List<Treatment> treatments) {
        this.treatments = treatments;
    }
}
```

```
package com.xerocry.domain;

import io.ebean.Model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

/**
 * Created by raskia on 2/23/2017.
 */
@Entity
public class Diseases extends Model {
    @Id
```

```
    @Column(name = "disease_id")
    Long diseaseId;

    @Column(length = 50)
    String symptoms;

    @ManyToOne(optional = false)
    DiseasesTypes disType;

    @Column(length = 50, nullable = false, name = "disease_name")
    String disName;

    public Diseases(DiseasesTypes disType, String disName) {
        this.disType = disType;
        this.disName = disName;
    }

    public String getSymptoms() {
        return symptoms;
    }

    public void setSymptoms(String symptoms) {
        this.symptoms = symptoms;
    }

    public DiseasesTypes getDisType() {
        return disType;
    }

    public void setDisType(DiseasesTypes disType) {
        this.disType = disType;
    }

    public String getDisName() {
        return disName;
    }

    public void setDisName(String disName) {
        this.disName = disName;
    }
}
```

```
package com.xerocry.domain;

import io.ebean.Model;

import javax.persistence.*;
import java.time.LocalDate;
import java.util.List;

/**
 * Created by raskia on 2/23/2017.
 */
@Entity
@Table(name = "Treatment")
public class Treatment extends Model {

    @Id
    @Column(name = "treatment_id")
    Long id;

    @ManyToOne(optional = false)
    @Column(name = "patient_id")
    Patients patientId;

    @ManyToOne(optional = false)
    @Column(name = "doctor_id")
    Doctors doctorId;

    @ManyToOne
    @Column(name = "disease_id")
    Diseases diseaseId;

    @Column(name = "start_date", nullable = false)
    LocalDate startDate;

```

```
35      @Column(name = "end_date")
36      LocalDate endDate;
37
38      @Column
39      String treatment;
40
41      @ManyToMany
42      @JoinTable(name = "TREATMENT_DRUGS",
43          joinColumns = @JoinColumn(name = "treatment_id", referencedColumnName = "
                ↪ treatment_id"),
44          inverseJoinColumns = @JoinColumn(name = "drug_id", referencedColumnName = "drug_id")
                ↪ )
45      List<Drugs> drugs;
46
47      @ManyToMany
48      @JoinTable(name = "TREATMENT_SERVICES",
49              joinColumns = @JoinColumn(name = "treatment_id", referencedColumnName = "
                    ↪ treatment_id"),
50              inverseJoinColumns = @JoinColumn(name = "service_id", referencedColumnName = "
                    ↪ service_id"))
51      List<Services> services;
52
53      public Treatment(LocalDate startDate) {
54  //          this.patientId = patientId;
55  //          this.doctorId = doctorId;
56  //          this.diseaseId = diseaseId;
57          this.startDate = startDate;
58      }
59
60      public Patients getPatientId() {
61          return patientId;
62      }
63
64      public void setPatientId(Patients patientId) {
65          this.patientId = patientId;
66      }
67
68      public LocalDate getEndDate() {
69          return endDate;
70      }
71
72      public void setEndDate(LocalDate endDate) {
73          this.endDate = endDate;
74      }
75
76      public String getTreatment() {
77          return treatment;
78      }
79
80      public void setTreatment(String treatment) {
81          this.treatment = treatment;
82      }
83
84      public List<Drugs> getDrugs() {
85          return drugs;
86      }
87
88      public void addDrugs(Drugs drug) {
89          this.drugs.add(drug);
90      }
91
92      public Doctors getDoctorId() {
93          return doctorId;
94      }
95
96      public void setDoctorId(Doctors doctorId) {
97          this.doctorId = doctorId;
98      }
99
100     public Diseases getDiseaseId() {
101         return diseaseId;
102     }
103
104     public void setDiseaseId(Diseases diseaseId) {
105         this.diseaseId = diseaseId;
106     }
```

```
107
108     public LocalDate getStartDate() {
109         return startDate;
110     }
111
112     public void setStartDate(LocalDate startDate) {
113         this.startDate = startDate;
114     }
115
116     public List<Services> getServices() {
117         return services;
118     }
119
120     public void addServices(Services service) {
121         this.services.add(service);
122     }
123 }
```

```
1  package com.xerocry.domain;
2
3  import io.ebean.Model;
4
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.Id;
8  import javax.persistence.ManyToMany;
9  import java.util.List;
10
11 /**
12  * Created by raskia on 2/23/2017.
13  */
14 @Entity
15 public class Services extends Model {
16
17     @Id
18     @Column(name = "service_id")
19     Long serviceId;
20
21     @Column(length = 50, nullable = false, name = "service_name")
22     String serviceName;
23
24     Integer price;
25
26     @ManyToMany(mappedBy = "services")
27     List<Treatment> treatments;
28
29     public Services(String serviceName, Integer price) {
30         this.serviceName = serviceName;
31         this.price = price;
32     }
33
34     public String getServiceName() {
35         return serviceName;
36     }
37
38     public void setServiceName(String serviceName) {
39         this.serviceName = serviceName;
40     }
41
42     public Integer getPrice() {
43         return price;
44     }
45
46     public void setPrice(Integer price) {
47         this.price = price;
48     }
49
50     public List<Treatment> getTreatments() {
51         return treatments;
52     }
53
54     public void setTreatments(List<Treatment> treatments) {
55         this.treatments = treatments;
56     }
57 }
```

```java
package com.xerocry.domain;

import io.ebean.Model;
import io.ebean.annotation.EnumValue;

import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by raskia on 2/27/2017.
 */
public class Payments extends Model {

    @Id
    @Column(name = "payment_id")
    Long paymentId;

    Double discount;

    @Column(nullable = false)
    Double balance;

    public enum State {
        @EnumValue("P")
        PAID,
        @EnumValue("N")
        NOT_PAID,
    }

    @ManyToMany(mappedBy = "payments")
    List<Patients> patients = new ArrayList<>();


    public Double getDiscount() {
        return discount;
    }

    public void setDiscount(Double discount) {
        this.discount = discount;
    }

    public Double getBalance() {
        return balance;
    }

    public void setBalance(Double balance) {
        this.balance = balance;
    }

    public List<Patients> getPatients() {
        return patients;
    }

    public void addPatients(Patients patients) {
        this.patients.add(patients);
    }
}
```

```java
package com.xerocry.domain;

import io.ebean.Model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

/**
 * Created by raskia on 2/23/2017.
 */
@Entity
public class DiseasesTypes extends Model {

```

```
    @Id
    Long type_id;

    @Column(length = 50, name = "dis_type")
    String disType;

    /*@OneToMany(mappedBy = "Diseases")
    List<Diseases> diseases;*/

    public DiseasesTypes(String disType) {
        this.disType = disType;
    }

    public String getDisType() {
        return disType;
    }

    public void setDisType(String disType) {
        this.disType = disType;
    }

    /*public List<Diseases> getDiseases() {
        return diseases;
    }

    public void setDiseases(List<Diseases> diseases) {
        this.diseases = diseases;
    }*/
}
```

```
package com.xerocry.service;

/**
 * Created by raskia on 2/23/2017.
 */
import com.xerocry.domain.*;
import io.ebean.Ebean;
import io.ebean.EbeanServer;
import javafx.scene.media.EqualizerBand;
import org.assertj.core.internal.cglib.core.Local;
//import com.xerocry.domain.Address;
//import com.xerocry.domain.Contact;
//import com.xerocry.domain.Country;
//import com.xerocry.domain.Customer;
//import com.xerocry.domain.Order;
//import com.xerocry.domain.Order.Status;
//import com.xerocry.domain.OrderDetail;
//import com.xerocry.domain.Product;

import javax.print.Doc;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.UUID;

public class LoadExampleData {

    private static boolean runOnce;

    private static EbeanServer server = Ebean.getServer(null);

    public static synchronized void load() {

        if (runOnce) {
            return;
        }

        final LoadExampleData me = new LoadExampleData();

        server.execute(() -> {
//            if (Country.find.query().findCount() > 0) {
//                return;
//            }
            me.deleteAll();
            me.insertPatients();
```

```
47              me.createTreatment("Treat1", LocalDate.now(), LocalDate.of(2015, 12, 02));
48              me.createTreatment("Treat2", LocalDate.now(), LocalDate.of(2016, 11, 02));
49  //            me.insertCountries();
50  //            me.insertProducts();
51  //            me.insertTestCustAndOrders();
52          });
53          runOnce = true;
54      }
55
56      public void deleteAll() {
57          Ebean.execute(() -> {
58
59              // Ebean.currentTransaction().setBatchMode(false);
60
61              // orm update use bean name and bean properties
62              server.createUpdate(Departments.class, "delete from departments").execute();
63              server.createUpdate(Diseases.class, "delete from diseases").execute();
64              server.createUpdate(Doctors.class, "delete from doctors").execute();
65              server.createUpdate(Drugs.class, "delete from drugs").execute();
66              server.createUpdate(DiseasesTypes.class, "delete from diseasesTypes").execute();
67              server.createUpdate(Grants.class, "delete from grants").execute();
68              server.createUpdate(Patients.class, "delete from patients").execute();
69              server.createUpdate(Services.class, "delete from services").execute();
70              server.createUpdate(Treatment.class, "delete from treatment").execute();
71
72  //            // sql update uses table and column names
73  //            server.createSqlUpdate("delete from o_country").execute();
74  //            server.createSqlUpdate("delete from o_product").execute();
75          });
76      }
77
78
79      public void insertPatients(){
80          server.execute(()->{
81             new Patients( "Andrey", LocalDate.now(), Patients.Gender.MALE).save();
82             new Patients( "Marina", LocalDate.now(), Patients.Gender.FEMALE).save();
83             new Patients( "Derek", LocalDate.now(), Patients.Gender.MALE).save();
84          });
85      }
86
87
88
89      public void insertDoctors(){
90          server.execute(()->{
91              new Doctors(5,"Can heal", LocalDate.of(1995, 03, 12)).save();
92          });
93      }
94
95      private static Departments insertDepartment(String name) {
96          Departments department = new Departments(name);
97          Ebean.save(department);
98          return department;
99      }
100
101     public Doctors createDoctor(String skills, int exp, LocalDate hiredDate) {
102         Departments department = insertDepartment("Depart" + UUID.randomUUID().toString());
103         Doctors doctor = new Doctors(exp, skills, hiredDate);
104         doctor.setDepartId(department);
105         Ebean.save(doctor);
106         return doctor;
107     }
108
109     public static DiseasesTypes insertType(String type) {
110         DiseasesTypes disType = new DiseasesTypes(type);
111         Ebean.save(disType);
112         return disType;
113     }
114
115     public Diseases createDisease(String name) {
116         Diseases dis = new Diseases(insertType("type"+UUID.randomUUID().toString()), name);
117         Ebean.save(dis);
118         return dis;
119     }
120
121
122     public static Patients createPatient(LocalDate regDate, String city, String name,
```

```
        ↪ LocalDate birthDate, Patients.Gender gender) {
123         Patients patient = new Patients(name, birthDate, gender);
124 //        Contact contact = new Contact();
125         if(regDate != null){
126             patient.setRegDate(regDate);
127         }
128         if (city != null) {
129             patient.setCity(city);
130         }
131         Ebean.save(patient);
132 
133 //        contact.setFirstName(firstName);
134 //        contact.setLastName(lastName);
135 //        String email = contact.getLastName() + (contactEmailNum++) + "@test.com";
136 //        contact.setEmail(email.toLowerCase());
137         return patient;
138     }
139 
140     public void createTreatment(String treatment, LocalDate endDate, LocalDate startDate) {
141         Treatment treatment1 = new Treatment(startDate);
142         treatment1.setDoctorId(createDoctor(UUID.randomUUID().toString(), 10, LocalDate.of
            ↪ (1995, 10, 1)));
143         treatment1.setPatientId(createPatient(LocalDate.now(), "Piter", "Andrey", LocalDate.
            ↪ now(), Patients.Gender.MALE));
144         treatment1.setDiseaseId(createDisease("dis" + UUID.randomUUID().toString()));
145         if (treatment != null) {
146             treatment1.setTreatment(treatment);
147         }
148         if (endDate != null) {
149             treatment1.setEndDate(endDate);
150         }
151         Ebean.save(treatment1);
152     }
153 
154 
155 //    public void insertCountries() {
156 //
157 //        server.execute(() ->  {
158 //            new Country("NZ", "New Zealand").save();
159 //            new Country("AU", "Australia").save();
160 //        });
161 //    }
162 
163 //    public void insertProducts() {
164 //
165 //        server.execute(() -> {
166 //            Product p = new Product("C001", "Chair");
167 //            server.save(p);
168 //
169 //            p = new Product("DSK1","Desk");
170 //            server.save(p);
171 //
172 //            p = new Product("C002", "Computer");
173 //
174 //            server.save(p);
175 //
176 //            p = new Product("C003", "Printer");
177 //            server.save(p);
178 //        });
179 //    }
180 
181 //    public void insertTestCustAndOrders() {
182 //
183 //
184 //        Ebean.execute( () -> {
185 //                    Customer cust1 = insertCustomer("Rob");
186 //                    Customer cust2 = insertCustomerNoAddress();
187 //                    insertCustomerFiona();
188 //                    insertCustomerNoContacts("NocCust");
189 //
190 //                    createOrder1(cust1);
191 //                    createOrder2(cust2);
192 //                    createOrder3(cust1);
193 //                    createOrder4(cust1);
194 //                }
195 //        );
```

```
196 //    }
197 
198 //    public static Customer createCustAndOrder(String custName) {
199 //
200 //        LoadExampleData me = new LoadExampleData();
201 //        Customer cust1 = insertCustomer(custName);
202 //        me.createOrder1(cust1);
203 //        return cust1;
204 //    }
205 //
206 //    public static Order createOrderCustAndOrder(String custName) {
207 //
208 //        LoadExampleData me = new LoadExampleData();
209 //        Customer cust1 = insertCustomer(custName);
210 //        Order o = me.createOrder1(cust1);
211 //        return o;
212 //    }
213 
214     private static int contactEmailNum = 1;
215 
216 //    private Customer insertCustomerFiona() {
217 //
218 //        Customer c = createCustomer("Fiona", "12 Apple St", "West Coast Rd", 1);
219 //
220 //        c.addContact(createContact("Fiona", "Black"));
221 //        c.addContact(createContact("Tracy", "Red"));
222 //
223 //        Ebean.save(c);
224 //        return c;
225 //    }
226 //
227 //    public static Contact createContact(String firstName, String lastName) {
228 //        Contact contact = new Contact();
229 //        contact.setFirstName(firstName);
230 //        contact.setLastName(lastName);
231 //        String email = contact.getLastName() + (contactEmailNum++) + "@test.com";
232 //        contact.setEmail(email.toLowerCase());
233 //        return contact;
234 //    }
235 
236 //    private Customer insertCustomerNoContacts(String name) {
237 //
238 //        Customer c = createCustomer(name, "15 Kumera Way", "Bos town", 1);
239 //
240 //        Ebean.save(c);
241 //        return c;
242 //    }
243 //
244 //    private Customer insertCustomerNoAddress() {
245 //
246 //        Customer c = new Customer("Jack Hill");
247 //        c.addContact(createContact("Jack", "Black"));
248 //        c.addContact(createContact("Jill", "Hill"));
249 //        c.addContact(createContact("Mac", "Hill"));
250 //
251 //        Ebean.save(c);
252 //        return c;
253 //    }
254 //
255 //    private static Customer insertCustomer(String name) {
256 //        Customer c = createCustomer(name, "1 Banana St", "P.O.Box 1234", 1);
257 //        Ebean.save(c);
258 //        return c;
259 //    }
260 //
261 //    private static Customer createCustomer(String name, String shippingStreet, String
    ↪ billingStreet, int contactSuffix) {
262 //
263 //        Customer c = new Customer(name);
264 //        if (contactSuffix > 0) {
265 //            c.addContact(new Contact("Jim" + contactSuffix, "Cricket"));
266 //            c.addContact(new Contact("Fred" + contactSuffix, "Blue"));
267 //            c.addContact(new Contact("Bugs" + contactSuffix, "Bunny"));
268 //        }
269 //
270 //        if (shippingStreet != null) {
```

```
271 //             Address shippingAddr = new Address();
272 //             shippingAddr.setLine1(shippingStreet);
273 //             shippingAddr.setLine2("Sandringham");
274 //             shippingAddr.setCity("Auckland");
275 //             shippingAddr.setCountry(Country.find.ref("NZ"));
276 //
277 //             c.setShippingAddress(shippingAddr);
278 //         }
279 //
280 //         if (billingStreet != null) {
281 //             Address billingAddr = new Address();
282 //             billingAddr.setLine1(billingStreet);
283 //             billingAddr.setLine2("St Lukes");
284 //             billingAddr.setCity("Auckland");
285 //             billingAddr.setCountry(Ebean.getReference(Country.class, "NZ"));
286 //
287 //             c.setBillingAddress(billingAddr);
288 //         }
289 //
290 //         return c;
291 //     }
292 //
293 //     private Order createOrder1(Customer customer) {
294 //
295 //         Product product1 = Product.find.ref(1L);
296 //         Product product2 = Product.find.ref(2L);
297 //         Product product3 = Product.find.ref(3L);
298 //
299 //         Order order = new Order(customer);
300 //
301 //         List<OrderDetail> details = new ArrayList<>();
302 //         details.add(new OrderDetail(product1, 5, 10.50));
303 //         details.add(new OrderDetail(product2, 3, 1.10));
304 //         details.add(new OrderDetail(product3, 1, 2.00));
305 //         order.setDetails(details);
306 //
307 //         //order.addShipment(new OrderShipment());
308 //
309 //         Ebean.save(order);
310 //         return order;
311 //     }
312 //
313 //     private void createOrder2(Customer customer) {
314 //
315 //         Product product1 = Ebean.getReference(Product.class, 1);
316 //
317 //         Order order = new Order(customer);
318 //         order.setStatus(Status.SHIPPED);
319 //         order.setShipDate(LocalDate.now().plusDays(1));
320 //
321 //         List<OrderDetail> details = new ArrayList<>();
322 //         details.add(new OrderDetail(product1, 4, 10.50));
323 //         order.setDetails(details);
324 //
325 //         //order.addShipment(new OrderShipment());
326 //
327 //         Ebean.save(order);
328 //     }
329 //
330 //     private void createOrder3(Customer customer) {
331 //
332 //         Product product1 = Product.find.ref(1L);
333 //         Product product3 = Product.find.ref(3L);
334 //
335 //         Order order = new Order(customer);
336 //         order.setStatus(Status.COMPLETE);
337 //         order.setShipDate(LocalDate.now().plusDays(2));
338 //
339 //         List<OrderDetail> details = new ArrayList<>();
340 //         details.add(new OrderDetail(product1, 3, 10.50));
341 //         details.add(new OrderDetail(product3, 40, 2.10));
342 //         order.setDetails(details);
343 //
344 //         //order.addShipment(new OrderShipment());
345 //
346 //         Ebean.save(order);
```

```
347 //     }
348 //
349 //     private void createOrder4(Customer customer) {
350 //
351 //         Order order = new Order(customer);
352 //         Ebean.save(order);
353 //     }
354 }
```