МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №2

з дисципліни «Екстримальне програмування»

на тему «Створення відображення між об’єктами і реляційними структурами»

Виконала
студентка групи КН-311
Мельничук М.П.

Прийняв
Щербак С.С.

Львів-2020

Мета: створення відображення між об’єктами і реляційними структурами за допомогою фреймворку Hibernate.

Теоретичні відомості

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням, яке поширюється на умовах GNU Lesser General Public License. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних. Hibernate піклується про зв'язок класів з таблицями бази даних (і типів даних мови програмування із типами даних SQL), і надає засоби автоматичної побудови SQL запитів й зчитування/запису даних, і може значно зменшити час розробки, який зазвичай витрачається на ручне написання типового SQL і JDBC коду. Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних, конвертації об'єктів і забезпечення сумісності із різними базами даних. Hibernate забезпечує прозору підтримку збереження даних, тобто їхньої персистентності (англ. persistence) для «POJO»-об'єктів, себто для звичайних Java-об'єктів; єдина сувора вимога до класу, що зберігається — конструктор за умовчанням (Для коректної поведінки у деяких застосуваннях потрібно приділити особливу увагу до методів equals() і hashCode()).

ORM (англ. Object-relational mapping, Об'єктно-реляційна проекція) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

Mapping (зіставлення, буквально — картування) Java класів з таблицями бази даних здійснюється за допомогою конфігураційних XML файлів або Java анотацій. При використанні файлу XML, Hibernate може генерувати скелет вихідного коду для класів тривалого зберігання (persistent). У цьому немає необхідності, якщо використовується анотація. Hibernate може використовувати файл XML або анотації для підтримки схеми бази даних. Забезпечуються можливості з організації відношення між класами «один-до-багатьох» і «багато-до-багатьох». На додаток до управління зв'язками між об'єктами, Hibernate також може керувати рефлексивними асоціаціями, де об'єкт має зв'язок «один-до-багатьох» з іншими

примірниками свого власного типу даних. Hibernate підтримує відображення користувацьких типів значень. Це робить можливим такі сценарії:

- Перевизначення типу за умовчанням SQL, який Hibernate вибирає при відображенні стовпчика властивості.
- Картування перераховуваного типу Java до колонок БД, так ніби вони є звичайними властивостями.
- Картування однієї властивості в декілька колонок.

Хід роботи

Для створення створення відображення між об’єктами і реляційними структурами для системи підтримки прийняття рішень щодо розміщення товарів нам необхідно розробити і описати відповідні класи а також “промапити” їх:

Клас: Category.java -> Таблиця: categories

```
@Entity
@Table(name = "categories")
public class Category implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Integer id;
   private String name;
   @OneToMany(mappedBy = "category")
   private List<Good> goods;
   @ManyToMany(mappedBy = "categories")
   private List<Shop> shops;
   @OneToMany(mappedBy = "category")
   private List<CategoryLocation> categoryLocations;
   public Category() {}
}
```

Клас: CategoryLocation.java -> Таблиця: categories_locations

```
@Entity
@Table(name = "categories_locations")
public class CategoryLocation {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Integer id;
    @Column(name = "global_location_id")
    private Integer globalLocationId;
    @Column(name="date")
    protected LocalDateTime date;
    @ManyToOne
    private Category category;
    @ManyToOne
    private Location location;
}
```

Клас: Customer.java -> Таблиця: customers

```
@Entity
@Table(name = "customers")
public class Customer implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @Column(name = "name")
    private String name;
    @Column(name = "surname")
    private String surname;
    @Column(name="date")
    protected LocalDateTime date;
    @OneToMany(mappedBy = "customer")
    private List<Purchase> purchases;
    public Customer() {}
}
```

Клас: Good.java -> Таблиця: goods

```
@Entity
@Table(name = "goods")
public class Good implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    @Column(name = "id")
    private Integer id;
    @Column(name = "name")
    private String name;
    @ManyToOne
    private Category category;
    @OneToMany(mappedBy = "good")
    private List<Purchase> purchases;
    public Good() {}
}
```

Клас: Location.java -> Таблиця: locations

```
@Entity
@Table(name = "locations")
public class Location implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @Column(name = "name")
    private String name;
    @Column(name = "distance")
    private Float distance;
    @Column(name = "side")
    private String side;
    @ManyToOne
    private Shop shop;
    @OneToMany(mappedBy = "location")
    private List<CategoryLocation> categoryLocations;
    @OneToMany(mappedBy = "location")
    private List<Purchase> purchases;
    public Location() {}
}
```

Клас: Purchase.java -> Таблиця: purchase

```
@Entity
@Table(name="purchase")
public class Purchase implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private Integer id;
   @Column(name = "purchase_id")
   private Integer purchaseId;
   @ManyToOne
   private Customer customer;
   @ManyToOne
   private Good good;
   @ManyToOne
   private Shop shop;
   @ManyToOne
   private Location location;
   @Column(name="date")
   protected LocalDateTime date;
   public Purchase() {}
}
```

Клас: Shop.java -> Таблиця: shops

```
@Entity
@Table(name="shops")
public class Shop implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private Integer id;
   @Column(name="name")
   private String name;
   @Column(name="country")
   private String country;
```

```
    @Column(name="city")
    private String city;
    @ManyToOne
    @JoinColumn(name = "owner")
    private User user;
    @ManyToMany
    @JoinTable(name = "shops_categories",
          joinColumns = @JoinColumn(name = "shop_id"),
          inverseJoinColumns = @JoinColumn(name = "category_id")
    )
    private List<Category> categories;
    @OneToMany(mappedBy = "shop")
    private List<Location> locations;
    @OneToMany(mappedBy = "shop")
    private List<Purchase> purchases;
    public Shop() {}
}
```

Клас: User.java -> Таблиця: users

```
@Entity
@Table(name = "users")
public class User implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @Column(name = "name")
    private String name;
    @Column(name="surname")
    private String surname;
    @Column(name = "email")
    private String email;
    @Column(name = "password")
    private String password;
    @OneToMany(mappedBy = "user")
```

```
    private List<Shop> shops;

    public User() {}
}
```

У результаті можемо побачити вигляд відображення реляційної бази даних у розробленій (“промапленої”) об’єктно-орієнтованій моделі даних:

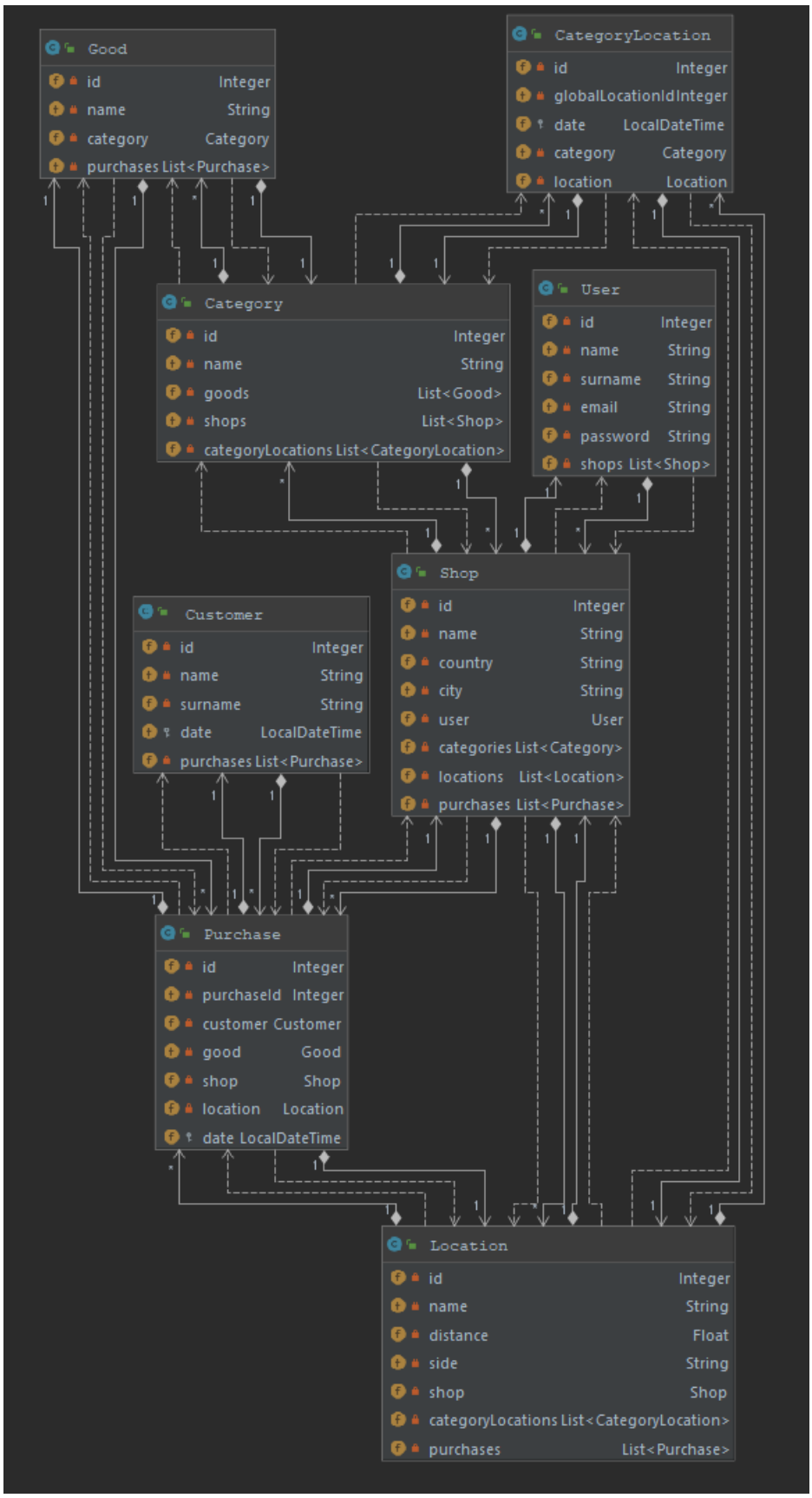

Рис.2.1 Вигляд об'єктно-орієнтованої моделі даних

Висновок: у результаті виконання лабораторної роботи було розроблено і описано відображення між об'єктами і реляційними структурами за допомогою фреймворку Hibernate.