Работа с базой данных. JPA

Артамонов Илья

@Аннотации

Java-аннотация — в языке Java специальная форма синтаксических метаданных, которая может быть добавлена в исходный код. Аннотации используются для анализа кода, компиляции или выполнения. Аннотируемы пакеты, классы, методы, переменные и параметры.

```
@Entity
public class Order {

@Id
private long id;


@NotNull
private Float total;


@Size(min = 32, max = 512)
private String address;


@ManyToOne(fetch = FetchType.EAGER)
private Customer customer;


}
```

Аннотация выполняет следующие функции:

1) дает необходимую информацию для компилятора;

2) дает информацию различным инструментам для генерации другого кода, конфигураций и т. д.;

3) может использоваться во время работы кода;

```
@Override
public String toString(){
    return "devcolibri.com";
}
```

@Аннотации

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.simbirsoft.jpatest.entities.Order" table="order" >
        <id name="id" type="java.lang.Long">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="total" type="float">
            <column name="total" not-null="true" unique="true" />
        </property>
        <property name="address" type="string">
            <column name="ship_addr" length="512" not-null="false" unique="true" />
        </property>
    </class>
</hibernate-mapping>
```

```java
@Entity
public class Order {

@Id
private long id;


@NotNull
private Float total;


@Size(max = 512)
@Column(name = "ship_addr")
private String address;


}
```

Собственные аннотации

1. Создаём аннотацию

```
@Target(value=ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Permission {
    Boolean value();
}
```

2. Вешаем аннотацию на класс/метод/поле *

```
@Permission(true)
public class UserDeleteAction {
    public void invoke(User user) { /* */ }
}
```

3. Работаем с аннотацией с помощью Java Reflection API

```
Class<?> someObjectClass = someObject.getClass();
Permission permission = someObjectClass.getAnnotation(Permission.class);
if (permission != null && permission.value() == true) {
        // выполнить действие
```

ORM

Object-relation mapping (объектно-реляционное отображение) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков, создавая «виртуальную объектную базу данных»

Плюсы и минусы

использование ОО-методов на всех этапах разработки приложений -> повышается скорость разработки
меньше однообразного вспомогательного кода -> меньше ошибок
позволяет абстрагироваться от источника данных -> приложение не привязано к конкретной СУБД

приложение работает медленнее и использует больше памяти
невозможно или неудобно использовать специфические особенности конкретных СУБД. Нет гарантии, что сгенерированный SQL код будет быстрым и эффективным
ORM добавляет дополнительный слой между программой и БД, у этого слоя есть собственный API, который необходимо изучить

Java Persistence API

JPA - технология, обеспечивающая объектно-реляционное отображение простых JAVA объектов и предоставляющая API для сохранения, получения и управления такими объектами.

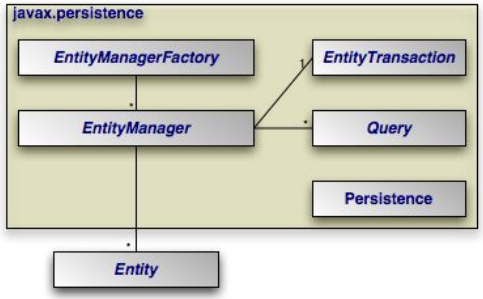
JPA - это спецификация (документ, утверждённый как стандарт, описывающий все аспекты технологии), часть EJB3-спецификации

Основные реализации:
Hibernate
Oracle TopLink
Apache OpenJPA
EclipseLink

Структура JPA

API

Интерфейсы в пакете



JPQL

Объектный язык запросов

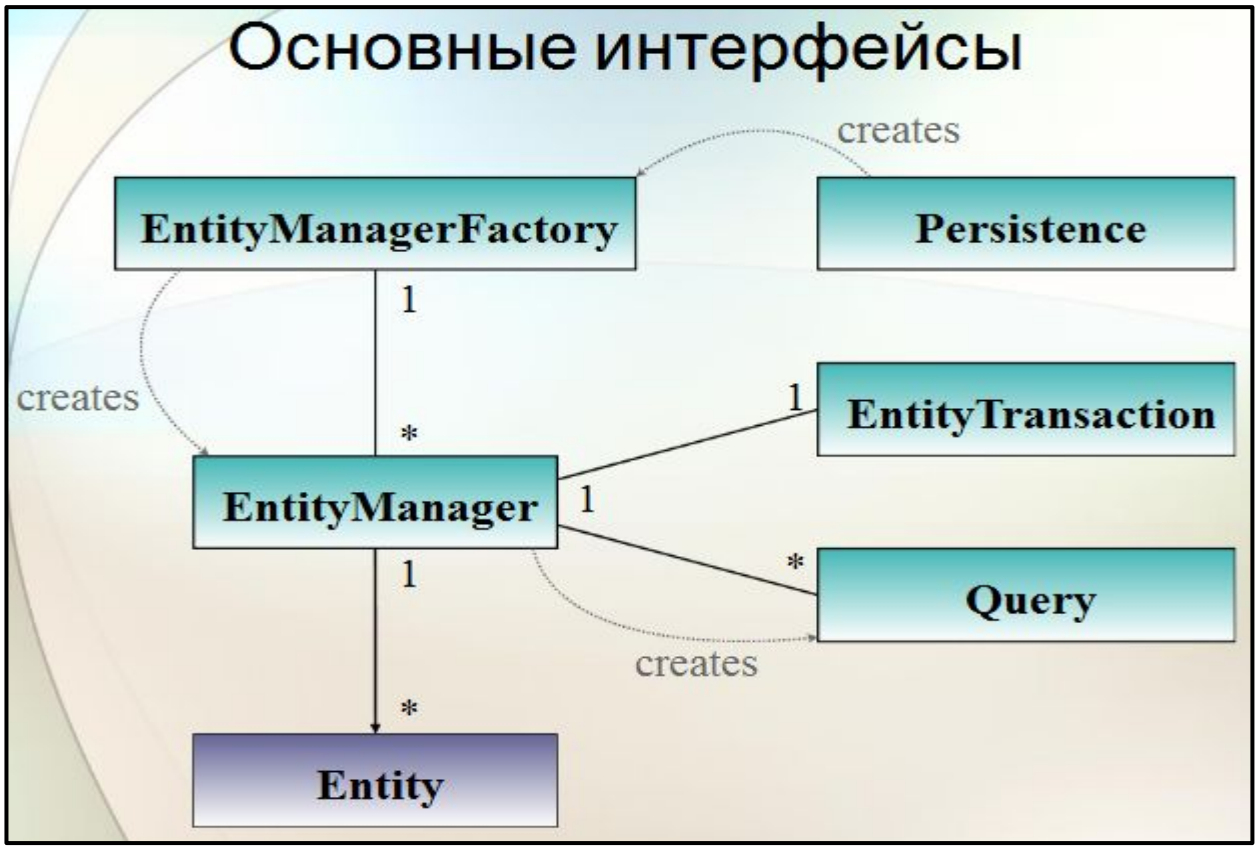
```
SELECT
User.name FROM
User WHERE
User.age = 26
AND User.id > 6
```

Metadata

Аннотации над объектами

```
@Entity
@Table(name="users"
public class User {
@Id
Long id
```

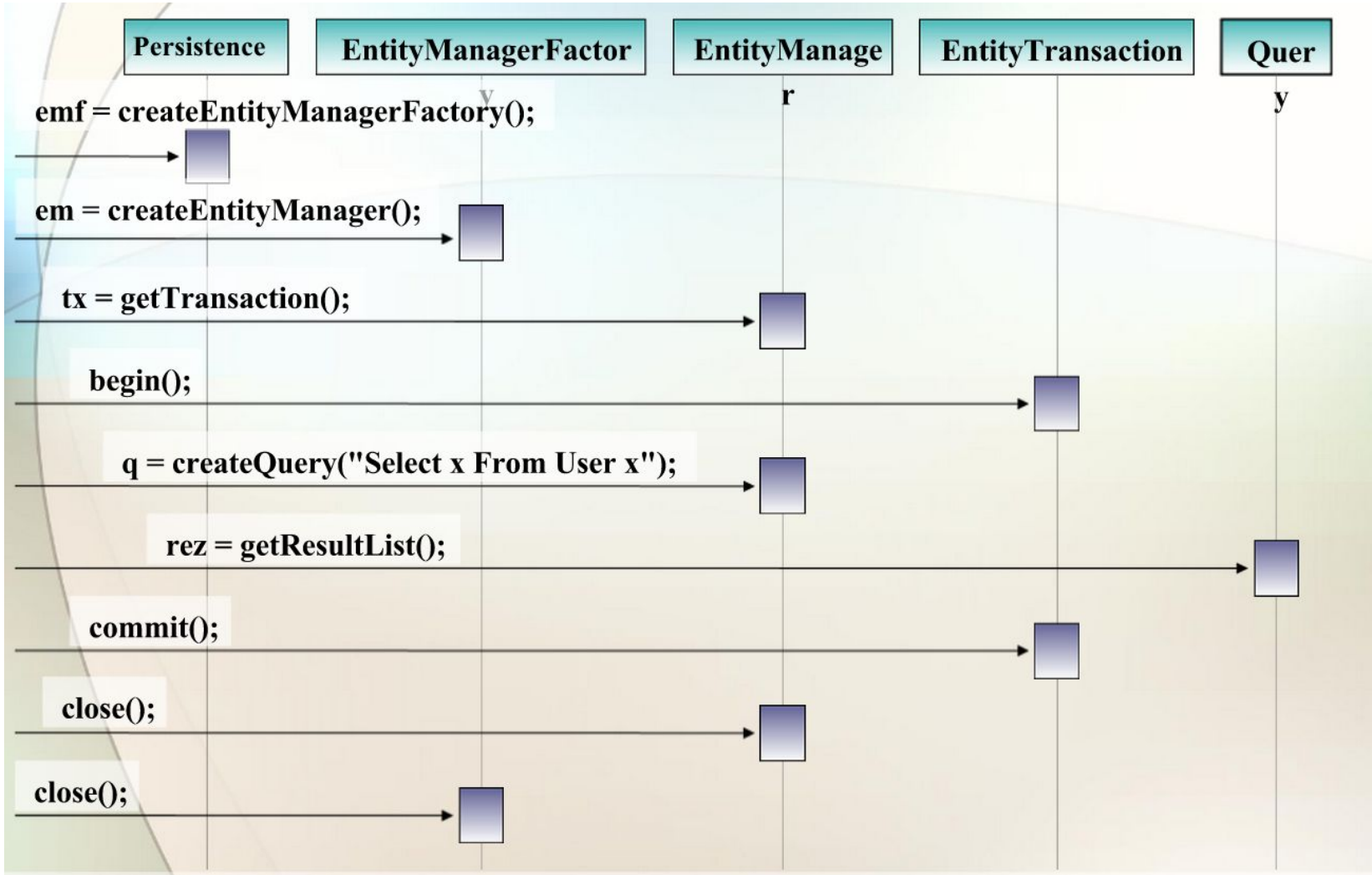
Основные интерфейсы



Последовательность вызова методов:

1. Persistence, создаем EntityManagerFactory, передавая параметры Unit. На выходе имеем фабрику либо ничего.
2. Обращаемся к фабрике и говорим “Дай мне EntityManager”
3. Потом к EntityManager “Дай мне transaction”
4. У transaction вызываем метод begin
5. Обращаемся к EntityManager. Вызываем query
6. Query. ResultList
7. Transaction. Закрываем
8. EntityManager. Закрываем
9. Фабрику. Закрываем

Последовательность взаимодействия интерфейсов



Настройка

Файл настройки: ‘src\main\resources**META-INF\persistence.xml**‘

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0">


<persistence-unit name="SSTestUnit" transaction-type="RESOURCE_LOCAL">
 <provider>org.hibernate.ejb.HibernatePersistence</ provider>
 <class>com.simbirsoft.jpatest.entities.User</ class>
 <properties>
   <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
   <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/carshop"/>
   <property name="hibernate.connection.username" value="root"/>
   <property name="hibernate.connection.password" value="root"/>
   <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
   <property name="hibernate.hbm2ddl.auto" value="update"/>
 </properties>
</persistence-unit>
</persistence>
```

hibernate.hbm2ddl.auto — статус работы JPA:
update - база будет просто обновлять свою структуру;
validate — проверяет структуру базы но не вносит изменения;
create — создает таблицы, но уничтожает предыдущие данные;
create-drop — создает таблицы в начале сеанса и удаляет их по окончанию сеанса.

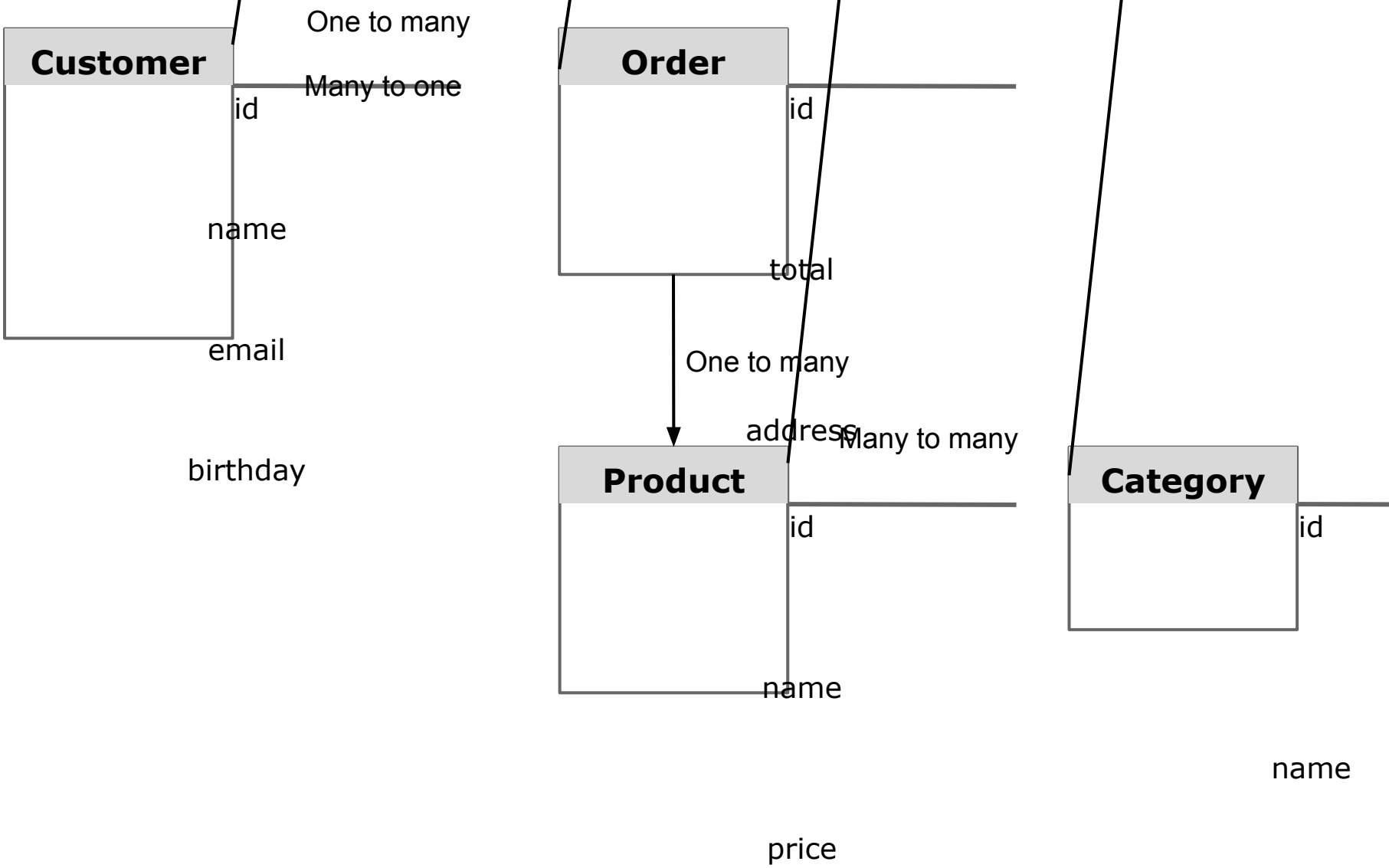
Требования к объектам сущностей

```
// * Сущность - объект, который может быть сохранён в БД
```

```
@Entity
@Table(name = "products")
public class Product {

    @Id
    private long id;

    @Column(name = "product_name")
    private String title;


    @OneToMany
    private List<Categories> categories;

    /* getters, setters, equals

}
```

- POJO или JavaBean
- Классы не final
- Наличие конструктора по умолчанию
- implements Serializable
- Наличие полей идентификации (id)
- Атрибуты-коллекции обязательно объявлены в терминах интерфейсов коллекций, а не конкретных реализаций
- В getters необходимо возвращать конкретно ссылку на коллекцию, а не на её копию

Пример работы: сущности



Пример работы: сущности

```
@Entity
public class Customer {

@Id
private long id;


private String name;


private String email;


@Temporal(TemporalType.DATE)
private Date birthday;


@OneToMany(fetch = FetchType.LAZY,
cascade = CascadeType.REMOVE)
private List<Order> orders;


    …
    /* getters and setters
…
```

```
…

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }


        Customer other = (Customer ) obj;
        if (id == null) {
            if (other.id != null) {
                return false;
            }
        } else if (!id.equals(other.id)) {
            return false;
        }
        return true;
    }


}
```

Пример работы: сущности

```
@Entity
public class Order {

@Id
private long id;

@NotNull /* Валидация на null
private Float total;

@Size(min = 32, max = 512) /* Валидация
private String address;

@ManyToOne(fetch = FetchType.EAGER)
private Customer customer;

@OneToMany(fetch = FetchType.EAGER)
private List<Product> products;

      …
      /* getters, setters, equals
…
}
```

```
@Entity
public class Product {

@Id
private long id;
private String name;
private Float price;

@ManyToMany(fetch = FetchType.EAGER)
private List<Category> categories;

      …
      /* getters, setters, equals
…
}
```

```
@Entity
public class Category{

@Id
private long id;

@Pattern(regexp = "^[A-Za-zA-Яа-яёЁ ]{0,}$")
private String name;

@ManyToMany(fetch = FetchType.Lazy)
private List<Product> products;

…
      /* getters, setters, equals
```

Пример использования

```
public class CustomerService {

    private EntityManager em = Persistence.createEntityManagerFactory("SSTestUnit").createEntityManager();

    public Customer add(Customer customer){ // Добавление клиента
        em.getTransaction().begin();
        Customer customerFromDB = em.merge(customer);
        em.getTransaction().commit();
        return customerFromDB ;
    }


    public Customer get(long id){ // Выборка клиента по id
        return em.find(Customer.class, id);
    }

    public void delete(long id){ // Удаление клиента
        em.remove(get(id));
    }

    public void update(Customer car){ // Сохранение клиента
        em.getTransaction().begin();
        em.merge(car);
        em.getTransaction().commit();
    }

    public List<Customer> getAll(){ //* Список всех клиентов
        TypedQuery<Customer> namedQuery = em.createNamedQuery("Customer.getAll", Customer.class);
        return namedQuery.getResultList();
    }
}
```

Работа с сущностями

```
public class TestJPA {

CustomerService service = new CustomerService ();

public void workWithEntities(){
//Создание нового клиента
Customer customer1 = new Customer();
customer1.setName("Вася Иванов");
customer1.setEmail("vasia@ivanov.ru");
customer1.setBirthday(new Date(12314234233));

//Записали в БД
Customer customer1 = service.add(customer);


//Достали клиента по id
Customer customer2 = service.get(2);

//Вывели записанную в БД запись
System.out.println(customer2.getOrders());


//Достали всех клиентов из базы
List<Customer> allCustomers = new ArrayList<>();
allCustomers = service.getAll();


//Удалил клиента №1 из базы
service.delete(customer1.getId());
}
}
```

Именные запросы: @NamedQuery

```
@Entity
@NamedQuery(name="Country.findAll", query="SELECT c FROM Country c") // Если одна
public class Country {
   ...
}
```

```
@Entity
@NamedQueries({ // Если несколько
    @NamedQuery(name="Country.findAll", query="SELECT c FROM Country c"),
    @NamedQuery(name="Country.findByName", query="SELECT c FROM Country c WHERE c.name = :name"),
})
public class Country {
   ...
}
```

```
List<Country> countries = em.createNamedQuery("Country.findAll", Country.class).getResultList();
```

```
Country country = em.createNamedQuery("Country.findByName", Country.class)
.setParameter("name", "Russia") // Задаём параметр
.getSingleResult(); // Достаём один результат
```

JPQL объектно-ориентированный язык запросов

```
@Entity
@NamedQueries({
    @NamedQuery(name="Customer.findByTotalOrders",
    query="SELECT c FROM Customer c, Order o WHERE o.customer = c AND o.total >= :minTotal GROUP BY c.id"),

    @NamedQuery(name="Customer.findCustomersByOrders",
    query="SELECT c FROM Customer c WHERE c.orders IN :orders"),

    @NamedQuery(name="Customer.findByOrderedProduct",
    query="SELECT c FROM Customer c, Order o WHERE o.customer = c AND :product MEMBER OF o.products
ORDER BY c.name GROUP BY c.id"),
})
public class Customer {
  ...
}
 List<Order> ordersToFind = new ArrayList<>();
 ordersToFind.add(order1);
 ordersToFind.add(order2);

 List<Customer> customers = em.createNamedQuery("Customer.findCustomersByOrders",
 Customer.class).setParameter("orders", ordersToFind ).getResultList();
```

Основные аннотации

Над классами	Над полями
@Entity	@Basic(fetch= LAZY)
@Table(name ="", schema="")	@Column(na me = "", unique = true, nullable = false)
@NamedQueri es	@OneToMany(fetch = "", cascade = "")
@NamedQuer y(name = "", query = "")	@OneToOne(f etch = "", cascade = "")
	@ManyToOne(fetch = "", cascade = "")
	@ManyToMan y(fetch = "", cascade = "")
	@JoinColumn(name = "CUSTOMER_I D")
	@OrderBy("na

Валидация: javax.validation.constraints

@Digits
@Future
@Past
@Max(value="")
@Min(value="")
@NotNull
@Pattern(regexp="")
@Size(min = 1, max = 128)