Базы данных и Java

Виктор Яковлев (с) кафедра АТП МФТИ, 2017

Реляционные базы данных

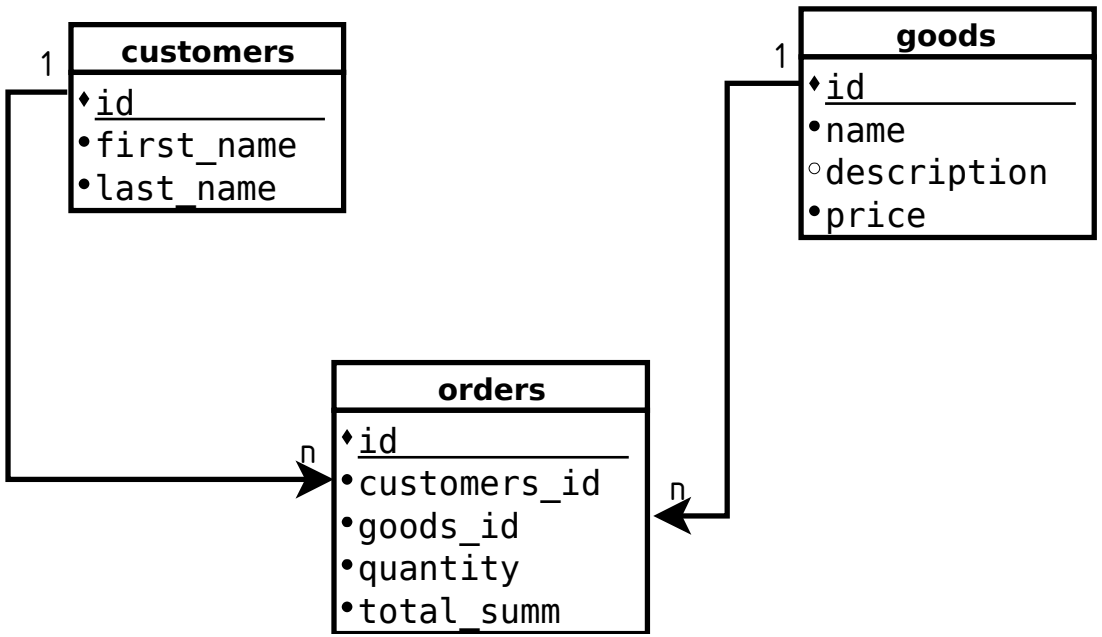
Реляционная база данных - это набор таблиц и связей между ними.

Каждая таблица, в идеале*, хранит минимальный набор аттрибутов отдельных объектов.

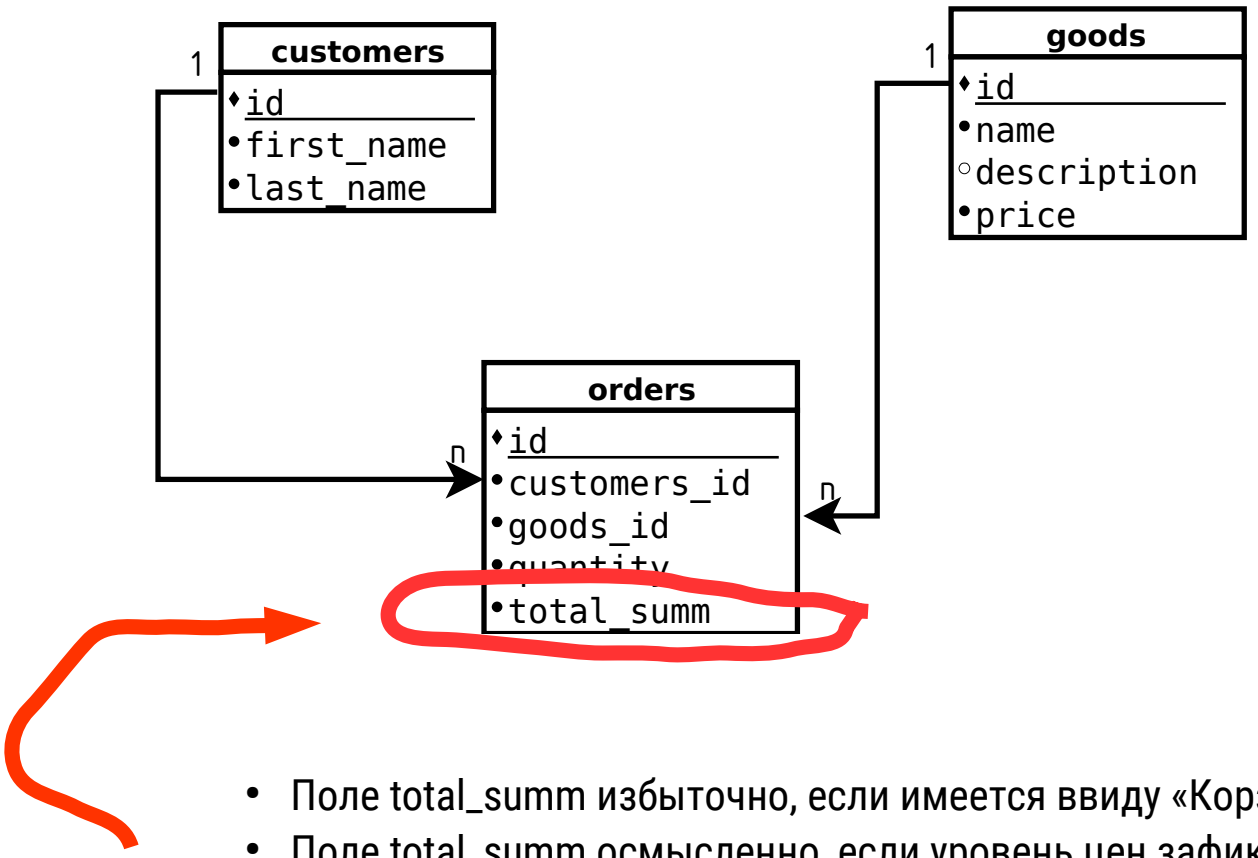
На практике, получение данных из реляционной базы данных, - это выборка строк и столбцов из нескольких таблиц, между которыми есть определенные связи.

* как мы понимаем, ничего идеального в жизни не бывает, и иногда приходится иметь дело с совершенно произвольной структурой БД

Таблицы, и связи между ними (на примере Интернет-магазина)



Таблицы, и связи между ними (на примере Интернет-магазина)

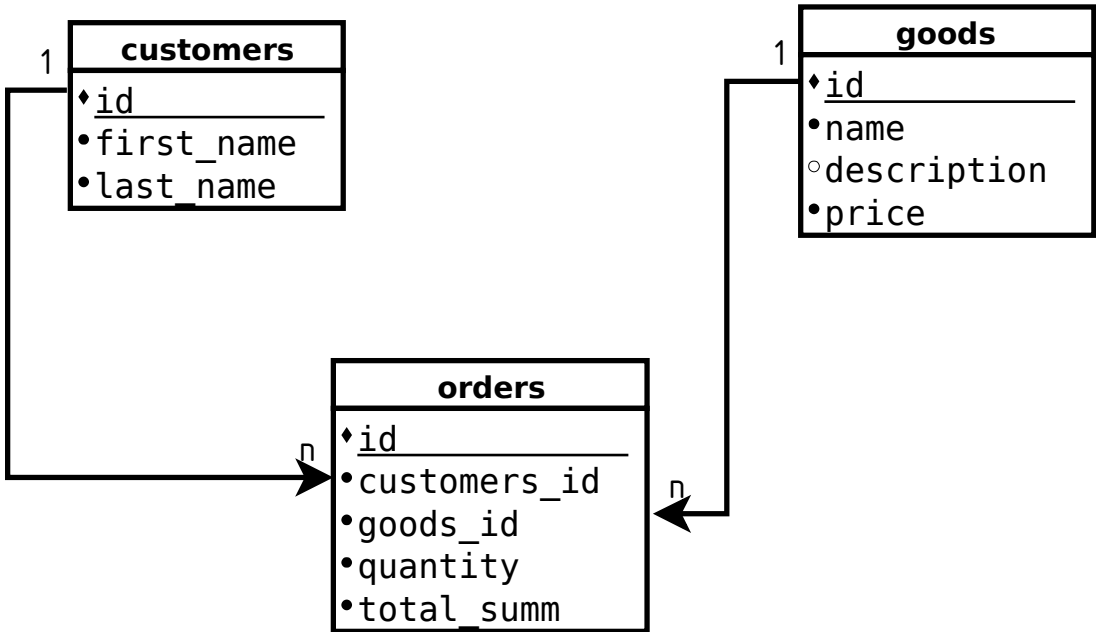


- Поле total_summ избыточно, если имеется ввиду «Корзина»
- Поле total_summ осмысленно, если уровень цен зафиксирован на момент оплаты.

SQL - операция INSERT

- Добавить нового клиента

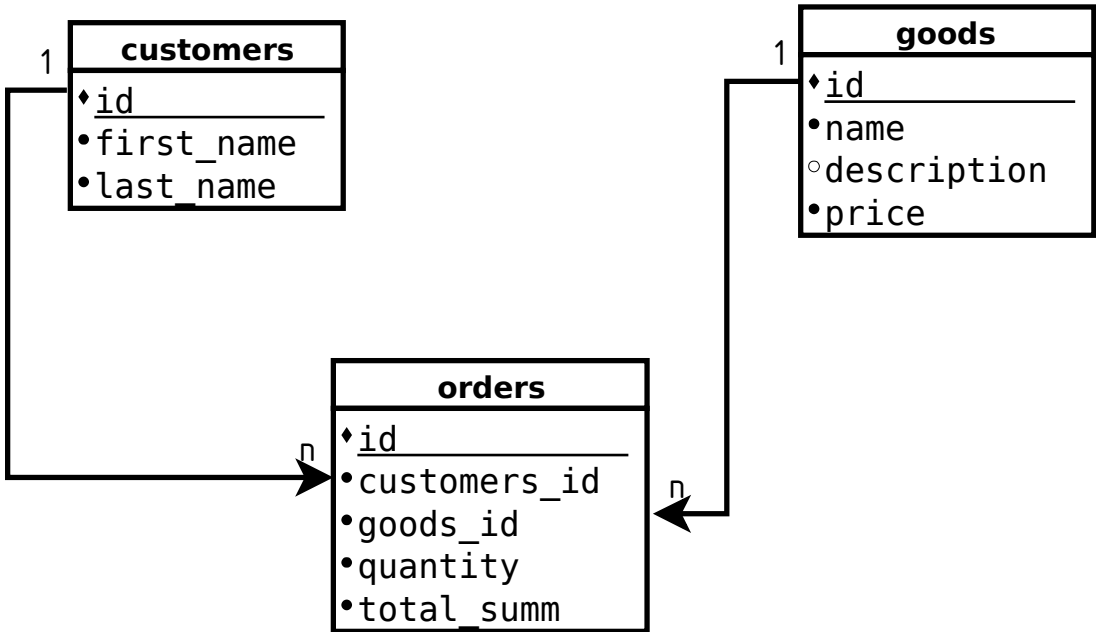
```
insert into
   customers(first_name,last_name)
values
   ("Вася", "Пупкин")
;
```



SQL - операция SELECT

- Выбрать все товары дешевле 5000Р

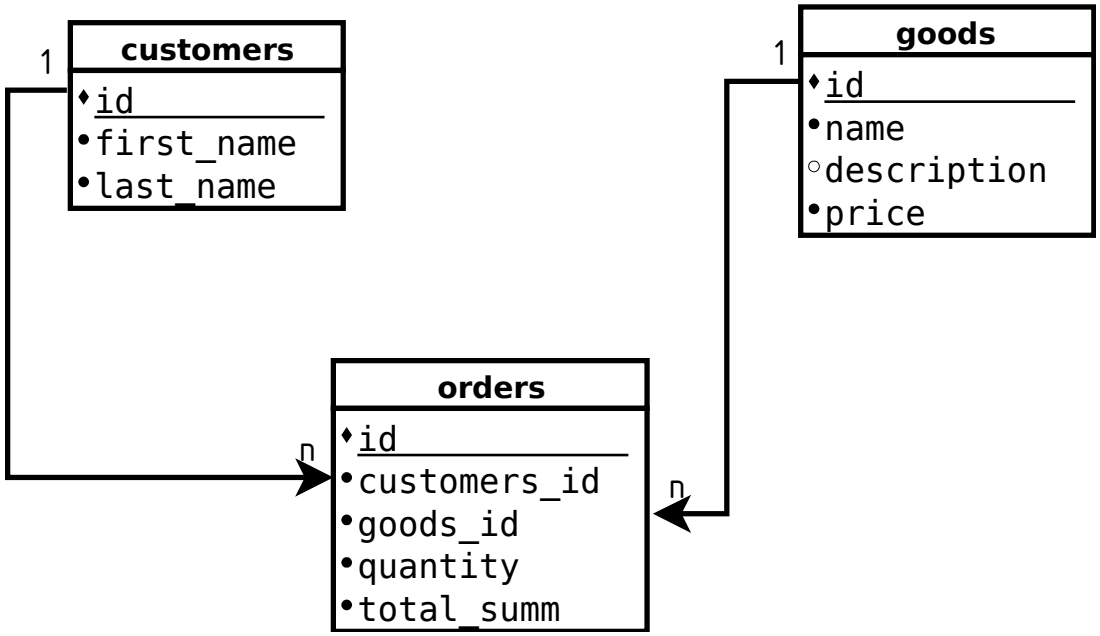
```
select id,name,description,price
from goods
where price<5000;
```



SQL - операция SELECT

- Посмотреть, что именно заказал некоторый пользователь *X*

```
select goods_id,name,price
from goods, orders
where
    goods.id=orders.goods_id and
    customers_id=X
;
```



Реляционные СУБД с поддержкой SQL

Клиент-серверные

- Oracle Database
- MySQL (MariaDB)
- PostgreSQL

Хранение БД в файле

- SQLite

Взаимодействие программы с SQL-сервером

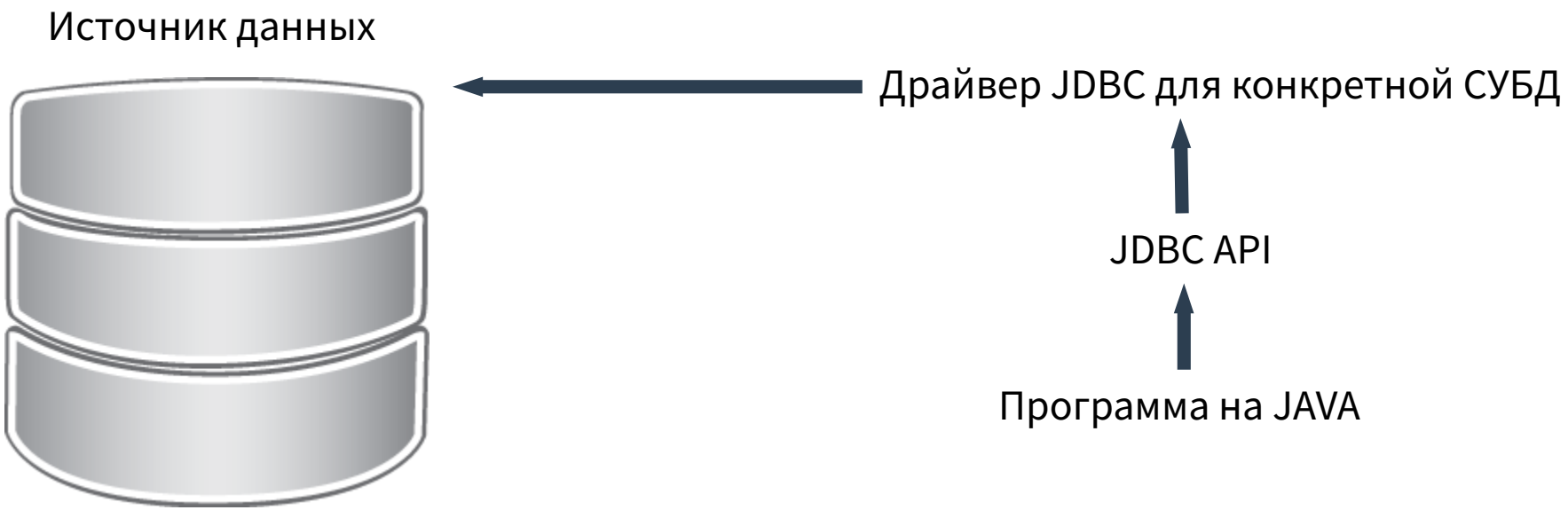
Все операции - через запросы на языке SQL.

На каком этапе происходит синтаксический разбор запроса?

Клиент-серверное взаимодействие с СУБД

- Программа-клиент (mysql, pg и т.д.) является посредником между сервером и источником SQL-запросов
- Данные передаются в бинарном виде, используя собственный протокол взаимодействия, напрямую не связанный с SQL

JDBC - универсальный интерфейс "клиента БД"



JDBC - это абстракция от конкретной СУБД

Работа с JDBC

1)Скачать необходимый драйвер (.jar-файл) для конкретной СУБД

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>
```

2)Установить подключение к БД

```
Connection connection =
  DriverManager
   .getConnection(
      "jdbc:mysql://ИМЯ_ХОСТА:НОМЕР_ПОРТА/ИМЯ_БАЗЫ_ДАННЫХ",
      "ИМЯ_ПОЛЬЗОВАТЕЛЯ", "ПАРОЛЬ"
   );
```

3)Отправлять запросы в формате SQL

Отправка запросов в формате SQL

```
PreparedStatement select = connection.prepareStatement(
    "select goods_id,name,price" +
    "  from goods, orders" +
    "  where " +
    "    goods.id=orders.goods_id and " +
    "    customers_id=?"
);

select.setInt(1, X)
```

```
select goods_id,name,price
from goods, orders
where
   goods.id=orders.goods_id and
   customers_id=X
;
```

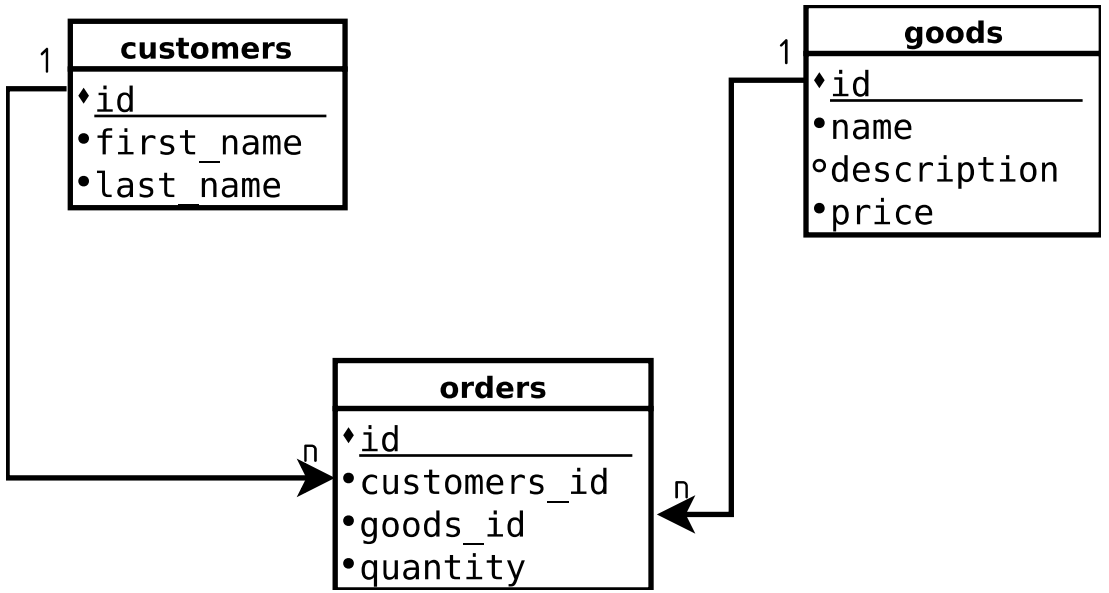
- Не контролируется корректность синтаксиса на этапе компиляции
- Получается смешанный код: JAVA+SQL
- Проблема экранирования строковых значений

Object-relational mapping (ORM)

```
class Customer {
    long id;
    String firstName;
    String lastName;
}

class Good {
    long id;
    String name;
    String description;
    double price;
}

class Order {
    long id;
    long customerId;
    long goodId;
    double quantity;
}
```

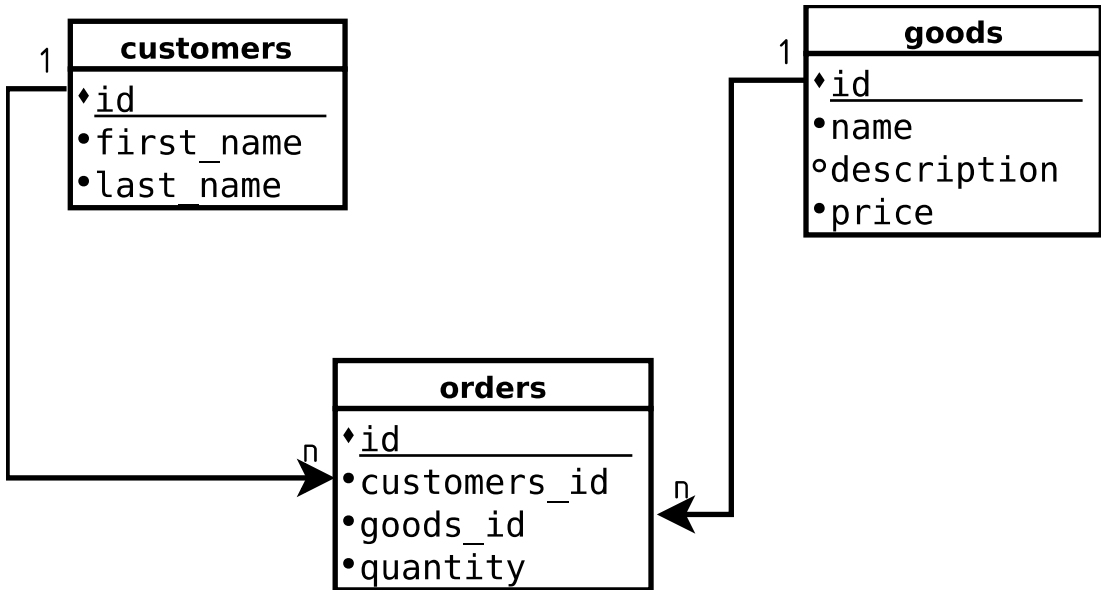


Object-relational mapping (ORM)

```
class Customer {
    long id;
    String firstName;
    String lastName;
}

class Good {
    long id;
    String name;
    String description;
    double price;
}

class Order {
    long id;
    long customerId;
    long goodId;
    Customer customer;
    Good good;
    double quantity;
}
```



ORM-фреймворки

Python:

- SQLAlchemy
- Django

Java Persistence API:

- EclipseLink
- Hibernate ORM

C++ - требуется препроцессор кода!!!

- ODB

Hibernate ORM [http://hibernate.org/orm/]

- Использует внутренний механизм JDBC
- Препроцессинг на этапе компиляции для повышения производительности

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.12.Final</version>
</dependency>
```

Hibernate: аннотирование классов

```
@Entity
@Table(name="customers")
public class Customer {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private long id;

    @Column(name="first_name", length=100)
    private String firstName;

    @Column(name="last_name", length=100)
    private String lastName;

}
```

Hibernate: внешнее описание соответствия

```java
public class Customer {
    private long id;
    private String firstName;
    private String lastName;
}
```

```xml
<class name="Customer" table="customers">
  <id name="id" type="long" column="id">
    <generator class="identity"/>
  </id>
  <property name="firstName" type="string">
    <column name="first_name" length="100"/>
  </property>
  <property name="lastName" type="string">
    <column name="last_name" length="100"/>
  </property>
</class>
```