СБЕРБАНК ТЕХНОЛОГИИ

ORM, Hibernate, Spring Data JPA

ПЛАН ЛЕКЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

- Что такое ORM, JPA , Hibernate
- Объектно-реляционное отображение
- Операции с базой данных с помощью Hibernate
- Подключение Hibernate к Spring приложению
- Hibernate как провайдер JPA в Spring
- Введение в Spring Data JPA

ORM (object relational mapping) – делегирует доступ к БД сторонним фреймворкам, которые обеспечивают объектно-ориентированное отображение реляционных данных и наоборот.

JPA (Java Persistence API) – спецификация описывающая API для управления ORM сущностями.

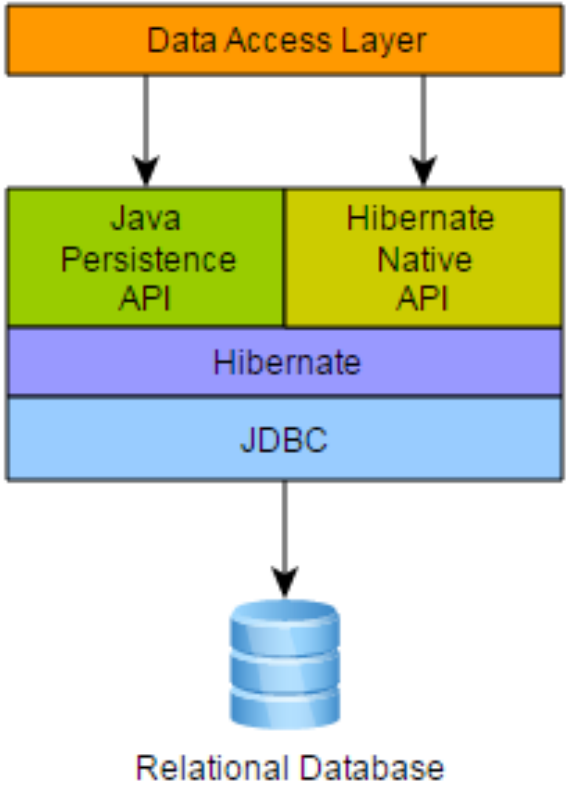
HIBERNATE – фреймворк реализующий спецификацию JPA

JPA позволяет решать следующие задачи:

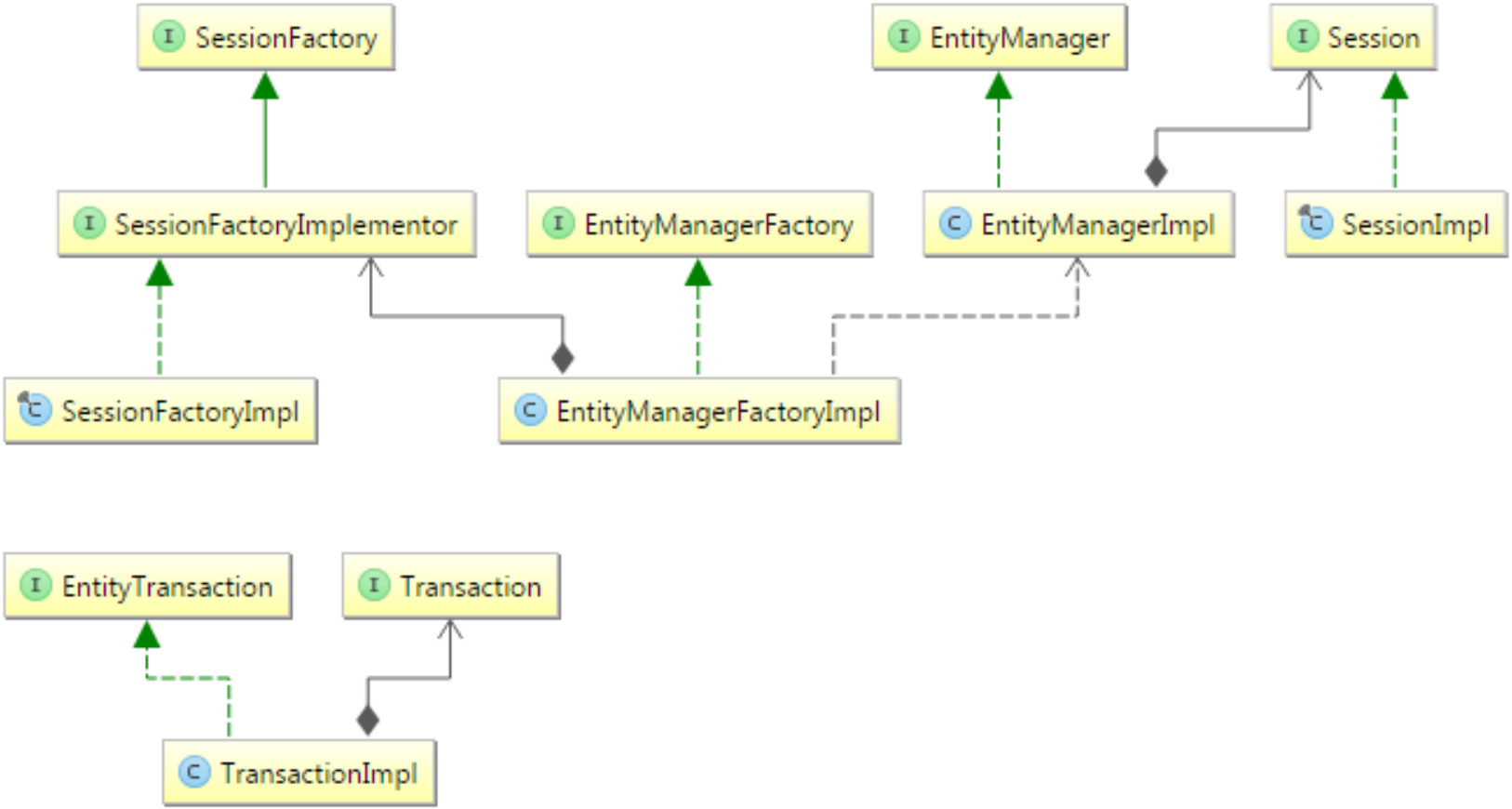
- **ORM**
- **Entity manager API** для выполнения CRUD операций с БД
- **Java Persistence Query Language (JPQL)** – SQL подобный язык, оперирующий объектами (не зависит от вендора БД)
- **Java Transaction API**
- **Механизмы блокировок**
- **Callbacks and listeners**

HIBERNATE

Находится между уровнем доступа к данным приложений Java и реляционной базой данных



СБЕРБАНК ТЕХНОЛОГИИ

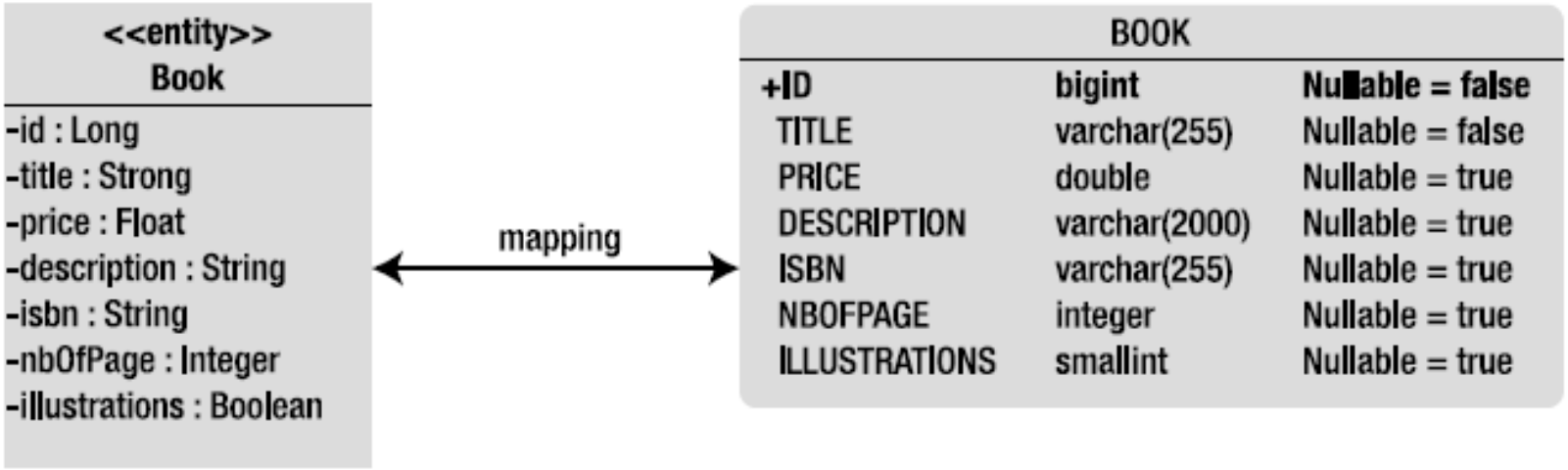


Пример класса отображаемого в базу:

```
@Entity
public class Book {
    @Id @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private String title;
    private Float price;
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    // Constructors, getters, setters
}
```

СБЕРБАНК ТЕХНОЛОГИИ

Диаграмма мапинга:



Ограничения на отображаемый объект по спеке JPA:

- Класс должен быть аннотирован @Entity
- Должен быть public или protected конструктор по-умолчанию
- Класс должен быть top-level
- Не могут быть Enum и интерфейсы
- Не может быть финальным (так же поля и методы)
- Должен имплементить Serializable

Допущения Hibernate :

- Класс не обязан быть top-level
- Допускаются final классы и методы (не рекомедуется)

TABLES

СБЕРБАНК ТЕХНОЛОГИИ

С помощью @Table можно:

- Задать имя таблицы
- Схему
- Индексы и уникальные ограничения

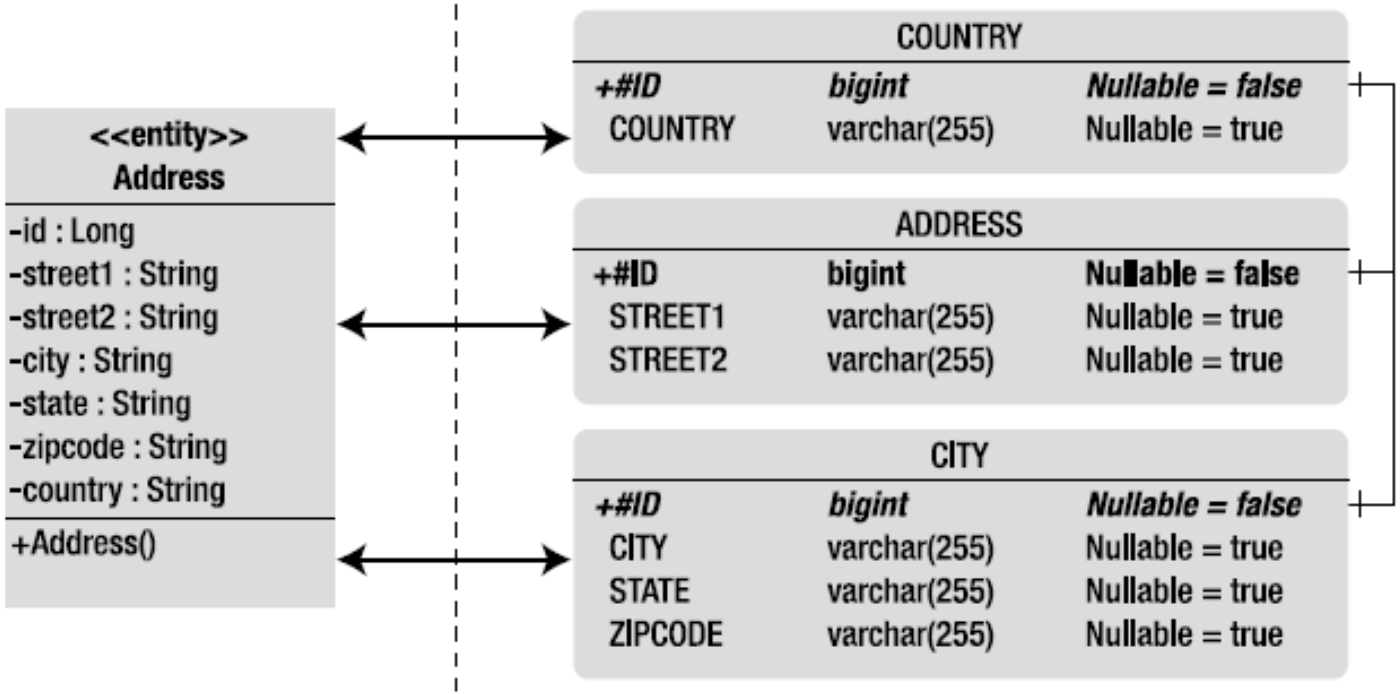
```
@Entity
@Table(name = "ITEM_TABLE")
public class Item implements Serializable{
```

С помощью @SecondaryTable можно добиться распределения данных в сущности между несколькими таблицами.

```
@Entity(name = "Address")
@SecondaryTables({
        @SecondaryTable(name = "City"),
        @SecondaryTable(name = "Country")
})
public class Address {
    @Id
    private Long id;
    private String street;
    @Column(table = "city")
    private String city;
    @Column(table = "country")
    private String country;
```

СБЕРБАНК ТЕХНОЛОГИИ

Модель данных для предыдущего примера:



ПЕРВИЧНЫЙ КЛЮЧ

СБЕРБАНК ТЕХНОЛОГИИ

JPA и HIBERNATE налагает ограничения на primary key:

- Каждая сущность (@Entity) должна иметь первичный ключ
- Первичный ключ может быть составным
- Первичный ключ не может измениться

Поле первичного ключа помечается @Id и может быть типом:

- Примитивный тип
- Обёртки примитивных типов
- Массивы примитивных типов
- Строки и даты

СТРАТЕГИИ ГЕНЕРАЦИИ ПЕРВИЧНОГО КЛЮЧА

СБЕРБАНК ТЕХНОЛОГИИ

Первичный ключ может быть сгенерирован автоматически на стороне приложения или HIBERNATE с помощью **@GeneratedValue**.

Поддерживаемые стратегии:

- **AUTO** – Hibernate сам выбирает подходящую стратегию
- **IDENTITY** – будут использоваться IDENTITY колонки в БД
- **SEQUENCE** – будут использоваться sequence из БД (jpa **@SequenceGenerator**)
- **TABLE** – значение будет браться из специальной таблички (для HIBERNATE - hibernate_sequences) (jpa **@TableGenerator**)
- **UUID** – (только для HIBERNATE)

Сущность Entity может содержать атрибуты (поля класса).

Атрибуты могут быть:

- Примитивные типы и обёртки
- Массивы байт и символов
- Строки, большие цифры, даты
- Перечисления
- Коллекции базовых и embeddable типов

БАЗОВЫЕ АННОТАЦИИ НАД АТРИБУТАМИ

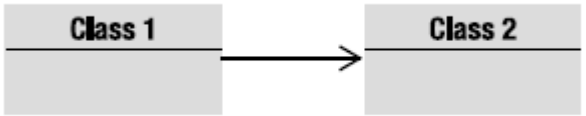
СБЕРБАНК ТЕХНОЛОГИИ

Аннотация	Назначение
@Basic	Позволяет указать nullable и fetch стратегию
@Column	Позволяет указать имя колонки в БД, размер поля, nullable, updatable или insertable
@Temporal	Позволяет преобразовывать дату и время из java в формат БД и обратно (кроме java8 new Date Time API)
@Enumerated	Позволяет указать как мапить enum значения: число или строка
@Transient	Предотвращает мапинг поля

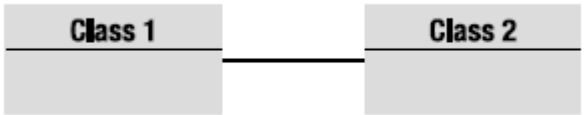
СБЕРБАНК ТЕХНОЛОГИИ

Виды:

- Однонаправленный (unidirectional)



- Двухнаправленный (bidirectional)



ВОЗМОЖНОСТЬ ОРГАНИЗОВАТЬ СВЯЗЬ В БАЗЕ

СБЕРБАНК ТЕХНОЛОГИИ

1. foreign key (join column)

Customer

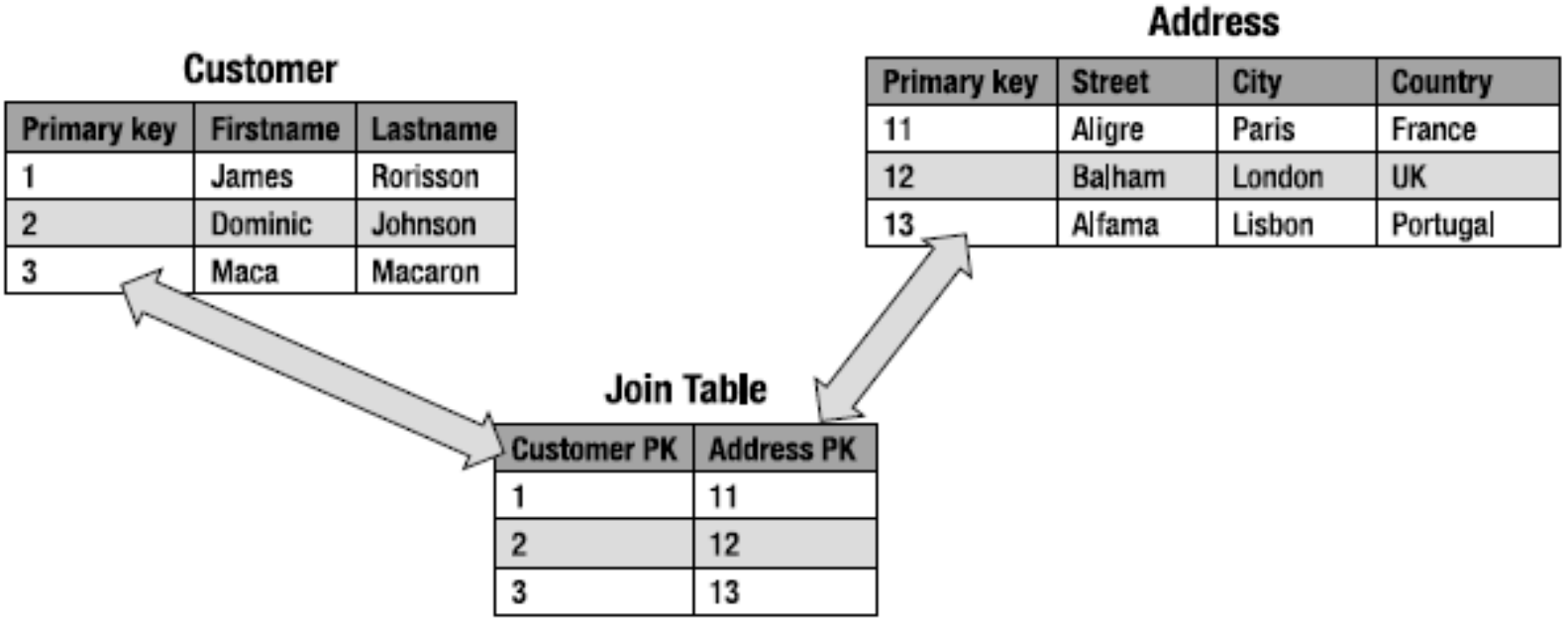
Primary key	Firstname	Lastname	Foreign key
1	James	Rorisson	11
2	Dominic	Johnson	12
3	Maca	Macaron	13

Address

Primary key	Street	City	Country
11	Allgre	Paris	France
12	Balham	London	UK
13	Alfama	Lisbon	Portugal

СБЕРБАНК ТЕХНОЛОГИИ

2. join table



Customer

Primary key	Firstname	Lastname
1	James	Rorisson
2	Dominic	Johnson
3	Maca	Macaron

Address

Primary key	Street	City	Country
11	Aligre	Paris	France
12	Balham	London	UK
13	Alfama	Lisbon	Portugal

Join Table

Customer PK	Address PK
1	11
2	12
3	13

CARDINALITY-DIRECTION COMBINATIONS

СБЕРБАНК ТЕХНОЛОГИИ

В таблице представлены возможные отношения между entities

Отношение	Направление
One-to-one	Однонаправленный
One-to-one	Двунаправленный
One-to-many	Однонаправленный
Many-to-one/one-to-many	Двунаправленный
Many-to-one	Однонаправленный
Many-to-many	Однонаправленный
Many-to-many	Двунаправленный

СПОСОБЫ ЗАГРУЗКИ АССОЦИАТИВНЫХ ДАННЫХ

СБЕРБАНК ТЕХНОЛОГИИ

В JPA существует 2 варианта загрузки данных:

- **Eagerly** – загружаются вместе с parent объектом
- **Lazily** – загружаются при первом обращении

В hibernate можно задать extra lazy загрузку (по элементную):

```
@LazyCollection(LazyCollectionOption.EXTRA)
```

Отношение	Загрузка по-умолчанию
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

HIBERNATE КОНТЕЙНЕРЫ

СБЕРБАНК ТЕХНОЛОГИИ

Hibernate так же позволяет хранить коллекции ассоциаций в собственных реализациях следующих контейнеров:

- **List<>**
- **Ordered List<>** (через @OrderBy/@OrderColumn)
- **Set<>** (через java equals/hashCode контракт)
- **SortedSet<>** (через @SortNatural/@SortComparator)
- **Map<>**
- **Массивы**

ВАРИАНТЫ МАППИНГА НАСЛЕДОВАНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

JPA и Hibernate позволяют замапить наследования 3 способами:

1. SINGLE_TABLE – одна таблица для каждой иерархии классов
2. JOINED – отдельная табличка для каждого подкласса
3. TABLE_PER_CLASS – отдельная табличка для конкретной имплементации

Настраивается через:

```
@Inheritance(strategy = InheritanceType.<стратегия>)
```

УПРАВЛЕНИЕ PERSISTENCE ОБЪЕКТАМИ

СБЕРБАНК ТЕХНОЛОГИИ

Entity Manager и Hibernate.Session – предоставляют API для управления entity объектами и управляет их жизненным циклом.

Persistence context – коллекция управляемых объектов в определённое время в рамках текущей транзакции.

ВОЗМОЖНЫЕ СОСТОЯНИЕ ENTITY ОБЪЕКТОВ

СБЕРБАНК ТЕХНОЛОГИИ

- **Transient** – только что созданный объект и пока не помещённый в persistence context
- **Managed (persistent)** – объект, добавленный в persistence context и имеющий идентификатор
- **Detached** – имеющий идентификатор, но отвязанный от контекста
- **Removed** – объект помеченный на удаление из БД

ОСНОВНЫЕ МЕТОДЫ HIBERNATE.SESSION

СБЕРБАНК ТЕХНОЛОГИИ

Метод	Назначение
save	Перевести объект в managed состояние
delete	Удалить entity
load	Ленивая загрузка entity
Find/byId().load	Полная загрузка entity
refresh	Синхронизирует entity с БД
saveOrUpdate	Снова вносит в контекст отвязанный (detach) объект
merge	Перетирает состояния объекта в БД состоянием отвязанного объекта
evict	Принудительно отвязывает объект от контекста

ТИПЫ ЗАПРОСОВ

СБЕРБАНК ТЕХНОЛОГИИ

- **Dynamic queries** – простая форма HQL/JPQL запроса
- **Named queries** – статические и не изменяемые
- **Native queries** – нативные SQL запросы
- **Criteria API** – ООП API построения запросов

ПОЛУЧЕНИЕ ОБЪЕКТОВ

СБЕРБАНК ТЕХНОЛОГИИ

JPQL и **HQL** – SQL подобные языки, манипулирующие объектами (не типо безопасный способ)

- JPQL – входит в стандарт JPA
- HQL – расширение JPQL

Пример запроса:

```
Query<Book> query = session.createQuery(
        "select b from Book b where b.title = :title"
        , Book.class)
        .setParameter("title", "Java");
```

Модель можно построить 2 способами:

- **Проектируем объектную модель, на их основе уже модель данных (hibernate. hbm2ddl. auto)**
- **Сначала модель данных потом объектная модель**