

# Объектно-реляционное отображение (ORM) и Java Persistence API

...

**Hibernate, Spring**

# Jakarta EE

Детали:

[https://ru.wikipedia.org/wiki/Java\\_Platform,\\_Enterprise\\_Edition](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition)

## ЧТО ВНУТРИ?

**EJB** (Enterprise JavaBeans) спецификация технологии серверных компонентов, содержащих бизнес-логику

**JPA** (Java Persistence API) управление постоянством и объектно-реляционное отображение

**Servlet** Обслуживание запросов веб-клиентов

**JSP (JavaServer Pages)** динамическая генерация веб-страниц на стороне сервера

**JAX-WS** Java API for XML Web Services — создание веб-сервисов

**JAX-RS** Java API for RESTful Web Services — создание RESTful веб-сервисов

**JSON-P** Java API for JSON Processing — разбор и генерация JSON

**JSON-B** Java API for JSON Binding — преобразование Java объектов в/из JSON

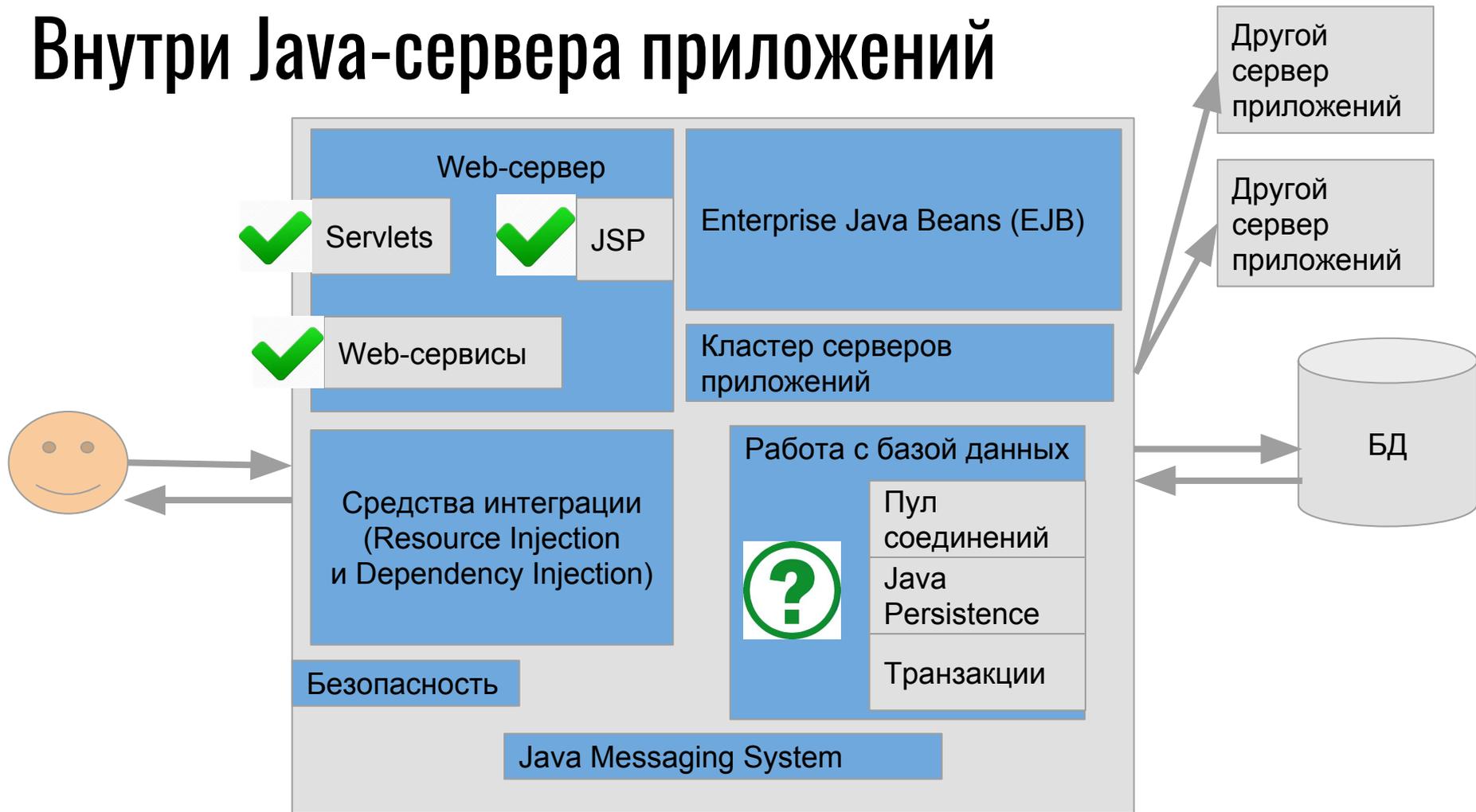
**JNDI** Java Naming and Directory Interface — служба каталогов

**JavaMail** Получение и отправка электронной почты

**JACC** Java Authorization Contract for Containers

.....

# Внутри Java-сервера приложений



# Java Persistence API (JPA)

# Java Persistence API (JPA)

спецификация API Jakarta EE,  
предоставляет возможность сохранять в  
удобном виде **Java-объекты в базе данных**  
поддержка сохранности данных

persistence (англ.) - сохранность

# Поддержка сохранности данных

- непосредственно API, заданный в пакете `javax.persistence`;
- платформо-независимый объектно-ориентированный язык запросов `Java Persistence Query Language`;
- метаянформация, описывающая связи между объектами (аннотации вида `@Entity`, `@ManyToOne`...);
- Генерация DDL для сущностей

# Реализация JPA

Пакет

`javax.persistence.*`



**HIBERNATE**

самый навороченный



часть большой экосистемы Spring

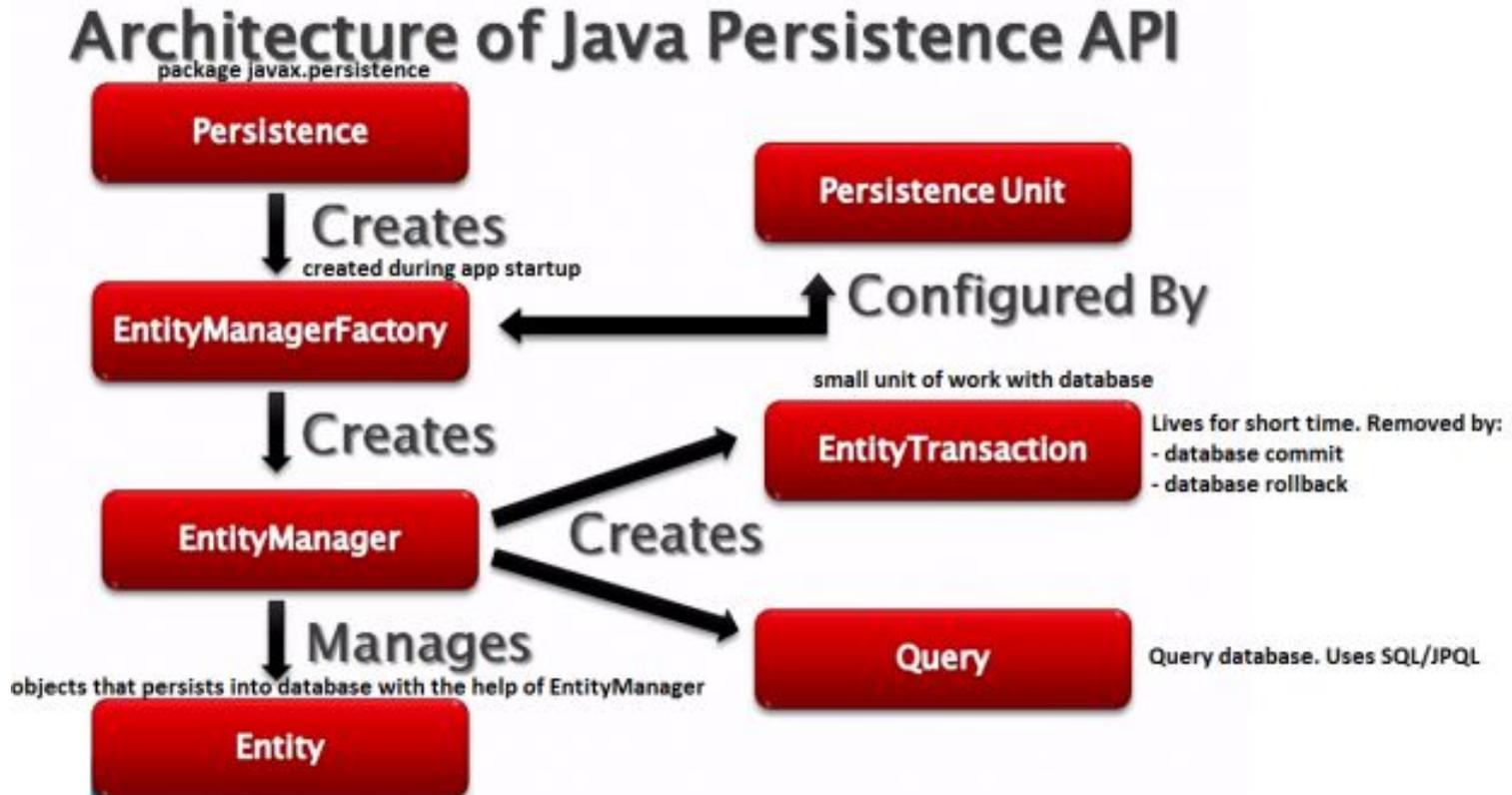
eclipse)link

Эталонная реализация  
JPA 2.0



Apache

# Архитектура JPA



Взято здесь:

<http://qaru.site/questions/59856/spring-data-jpa-versus-jpa-whats-the-difference>

# Основные JPA аннотации (1)

**@Entity** — Указывает, что данный класс является сущностью.

**@Table** — указывает на имя таблицы, которая будет отображаться в этой сущности.

**@Column** — указывает на имя колонки, которая отображается в свойство сущности.

**@Id** — id колонки

**@GeneratedValue** — указывает, что данное свойство будет создаваться согласно указанной стратегии.

**@Version** — управление версией в записи сущности. При изменении записи увеличится на 1.

**@OrderBy** — указание сортировки. В примере множество кошек будет отсортировано по имени по возрастанию.

**@Transient** — указывает, что свойство не нужно записывать. Значения под этой аннотацией не записываются в базу данных (так же не участвуют в сериализации). `static` и `final` переменные экземпляра всегда `transient`.

# Основные JPA аннотации (2)

@OneToOne — указывает на связь между таблицами «один к одному».

## Основные JPA аннотации (3)

`@OneToMany(mappedBy=»customer«, orphanRemoval=»true«, cascade=CascadeType.ALL)`

указывает на связь один ко многим. Применяется с другой стороны от сущности с `@ManyToOne`

`mappedBy` — обратная сторона связи сущности. Поле под этим атрибутом не сохраняется как часть исходной сущности в базе данных, но будет доступна по запросу.

`cascade = CascadeType.ALL` — означает, что операция, например, записи должна распространяться и на дочерние таблицы.

`orphanRemoval` — позволяет удалять объекты-сироты. При удалении родительского объекта удаляется и дочерний.

`@JoinTable` — указывает на связь с таблицей.

## Основные JPA аннотации (3)

`@OneToMany(mappedBy=»customer«, orphanRemoval=»true«, cascade=CascadeType.ALL)`

указывает на связь один ко многим. Применяется с другой стороны от сущности с `@ManyToOne`

`mappedBy` — обратная сторона связи сущности. Поле под этим атрибутом не сохраняется как часть исходной сущности в базе данных, но будет доступна по запросу.

`cascade = CascadeType.ALL` — означает, что операция, например, записи должна распространяться и на дочерние таблицы.

`orphanRemoval` — позволяет удалять объекты-сироты. При удалении родительского объекта удаляется и дочерний.

`@JoinTable` — указывает на связь с таблицей.

# Entity

Entity (Сущность) — **POJO**-класс, связанный с БД с помощью аннотации (`@Entity`) или через XML

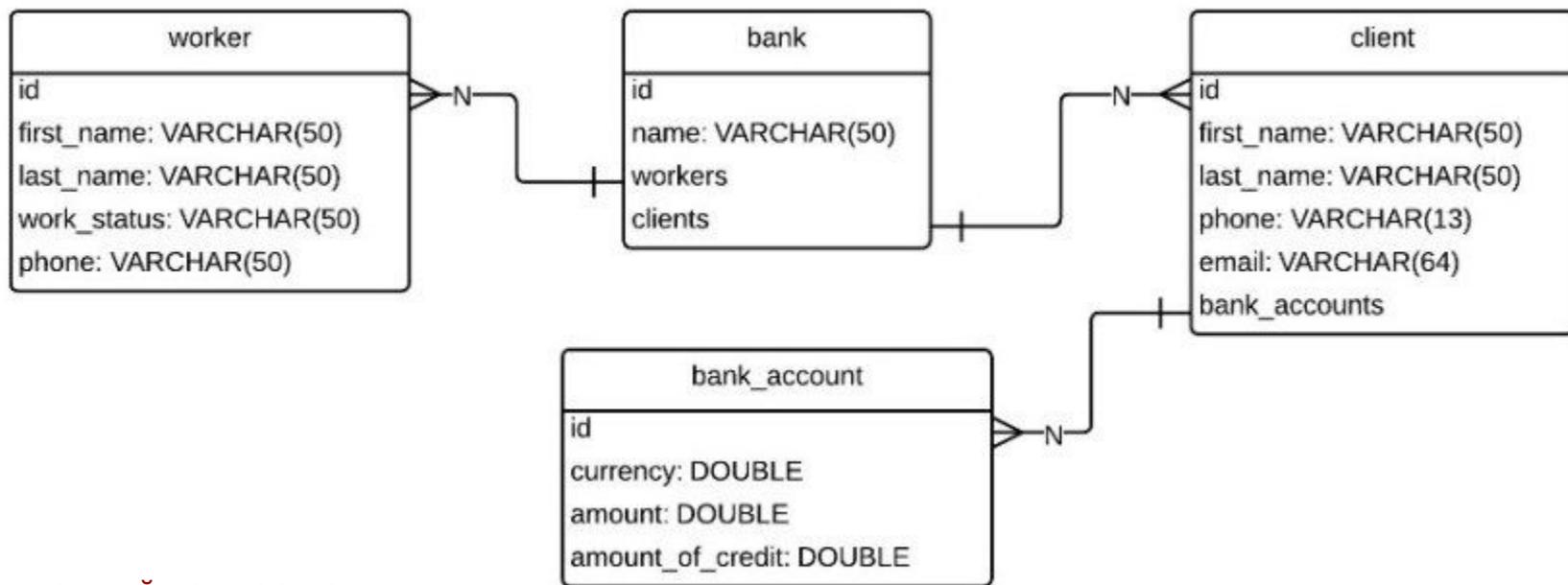
POJO - plain old Java object

# Работы по созданию Java Persistence

# Работы по созданию Java Persistence

1. Выбрать фреймворк, реализующий JPA API
2. Создать Java классы для сущностей (entities), которые будут сохраняться в базе данных
3. Описать маппинг (mapping, отображение) сущностей в таблицы базы данных

# Учебный пример: структура базы данных, в которую нужно сохранить Java объекты



См. полный пример на

<https://devcolibri.com/spring-data-jpa-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D0%B1%D0%B4-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1/>

# Пример Entity

```
package  
4325ORM;
```

```
import org.hibernate.annotations.GenericGenerator;  
import javax.persistence.*;  
@Entity  
@Table(name = "bank")  
public class Bank {  
    @Id  
    @GeneratedValue(generator = "increment")  
    @GenericGenerator(name= "increment", strategy= "increment")  
    @Column(name = "id", length = 6, nullable = false)  
    private long id;  
    @Column(name = "name")  
    private String name;  
    public Bank() {}  
    public Bank(String name) { this.name = name; }  
    public long getId() {return id; }  
    public void setId(long id) { this.id = id; }  
    public String getName() {return name; }  
    public void setName(String name) {this.name = name; } }
```

# Пример Entity со связью: M:1

```
@Entity
@Table(name = "bank_account")
public class BankAccount {
    @Id
    @GeneratedValue(generator = "increment")
    @GenericGenerator(name = "increment", strategy = "increment")
    @Column(name = "id", length = 6, nullable = false)
    private long id;
    @Column(name = "currency")
    private double currency;
    @Column(name = "amount")
    private double amount;
    @Column(name = "amount_of_credit")
    private double amountOfCredit;
    @ManyToOne(fetch = FetchType.LAZY, cascade = {CascadeType.MERGE,
    CascadeType.PERSIST})
    @JoinColumn(name = "client_id", nullable = false)
    private Client client;...
```

Client может иметь  
много банковских  
счетов в одном банке



См. полный пример на

<https://devcolibri.com/spring-data-jpa-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D0%81%D0%B1-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1/>

# hbm.xml- файл для Entity

```
<hibernate-mapping>  
  <class name=«4325ORM.Bank» table=«bank»>  
    <id column=«id» name=«id» type=«java.lang.Long»>  
      <generator class=«increment»/>  
    </id>  
    <property column=«name» name=«name» type=«java.lang.String»/>  
  </class>  
</hibernate-mapping>
```

# hbm.xml- файл для Entity со связью M:1

```
<hibernate-mapping>
<class name=«4325ORM.BankAccount» table=bank_account>
<id column=«id» name=«id» type=«java.lang.Long»>
  <generator class=«increment»/>
</id>
<property column=«currency» name=«currency» type=«java.lang.Double»/>
<property column=«amount» name=«amount» type=«java.lang.Double»/>
<property column=«amount_of_credit» name=«amount» type=«java.lang.Double»/>

  <set name=«bank_account» table=«client» lazy=«false»>
    <key column=«client_id»/>
    <many-to-one column=«client_id» class=«4325ORM.Client»/>
  </set>

</class>
</hibernate-mapping>
```