

Bài 16

Thao tác với CSDL và ORM

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài "Views Thymleaf"

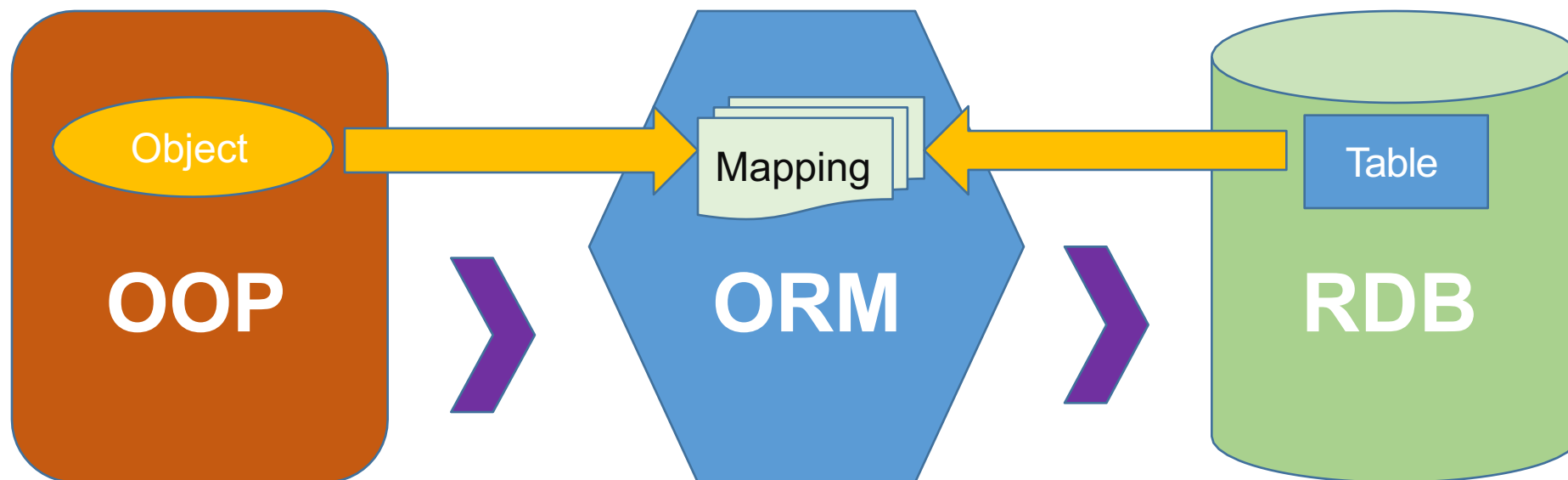
Tóm tắt lại các phần đã học từ bài "Views Thymeleaf"

- Trình bày được khái niệm ORM
- Trình bày được JPA
- Trình bày được Entity
- Trình bày được Entity Manager
- Trình bày được hoạt động của Spring Data JPA
- Triển khai được Spring Data JPA để thao tác cơ bản với CSDL

Thảo luận

ORM

- ORM (Object-Relational Mapping) là kỹ thuật liên kết giữa các đối tượng trong lập trình với các đối tượng trong CSDL
- ORM cho phép truy xuất dễ dàng đến dữ liệu thông qua các đối tượng lập trình
- ORM giúp lập trình viên tập trung thao tác với các đối tượng, không cần quá quan tâm đến CSDL thực tế đang dùng



Các góc nhìn khác nhau về dữ liệu

- Góc nhìn từ quản trị dữ liệu

```
CREATE TABLE persons (  
    id integer NOT NULL,  
    name varchar(50) NOT NULL,  
    salary float, PRIMARY KEY(id)  
);  
INSERT INTO persons (id, name) VALUES (1, 'John Doe');  
UPDATE persons SET salary=2000 WHERE id=1;
```

- Góc nhìn từ lập trình viên

```
public class Person {  
    public String name;  
    public float salary;  
    public Person(String name) { ... }  
}  
Person p = new Person("John Doe");  
PersistenceLayer.save(p);  
p.setSalary(2000);  
PersistenceLayer.update(p);
```

Ưu điểm và nhược điểm của ORM

- Ưu điểm:
 - Quản lý dữ liệu tập trung trong code
 - Các thao tác với dữ liệu được thực hiện tự động
 - Tránh được các lỗi cú pháp SQL
 - Hỗ trợ giao dịch (transaction)
 - Có thể cache dữ liệu để truy xuất nhanh hơn
- Nhược điểm:
 - Đối với các dự án lớn, hiệu năng truy xuất dữ liệu thường hạn chế
 - Lập trình viên dễ bị rơi vào bẫy truy xuất dữ liệu quá nhiều (vì quá dễ để truy xuất), ảnh hưởng đến hiệu năng của hệ thống
 - Đối với các thao tác phức tạp, có thể cần đến việc sử dụng SQL thuần

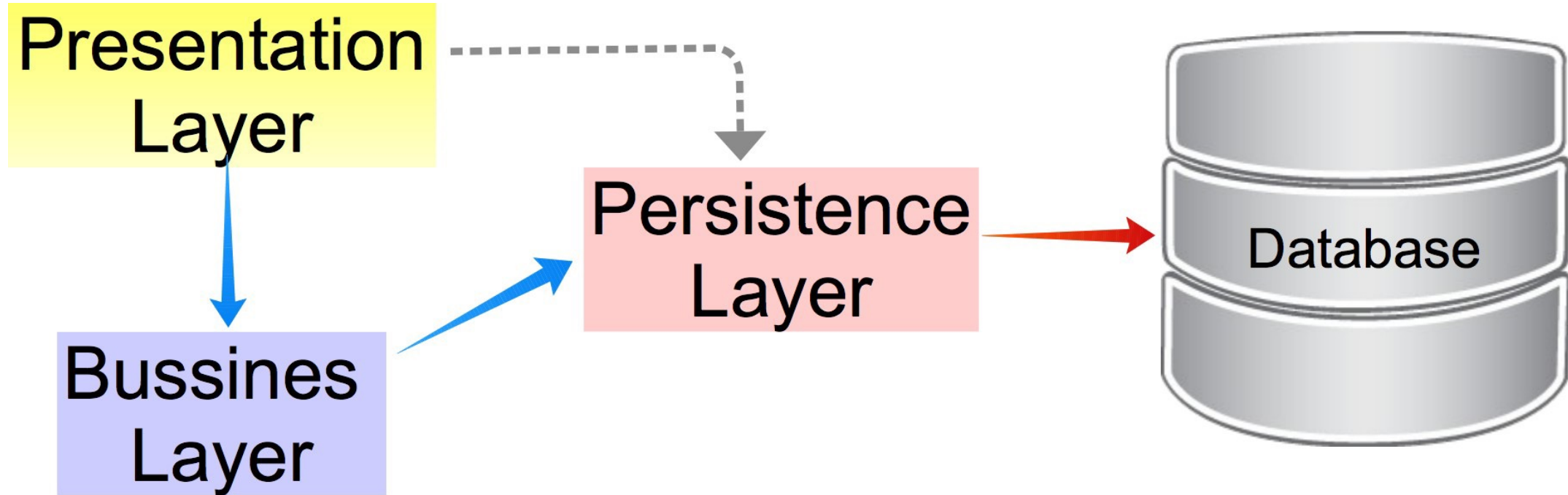
Một số ORM framework dành cho Java

- Apache Cayenne
- ActiveJDBC
- EclipseLink
- Ebean
- OpenJPA
- Hibernate
- JPA (Java Persistence API)

Thảo luận

JPA – Java Persistence API

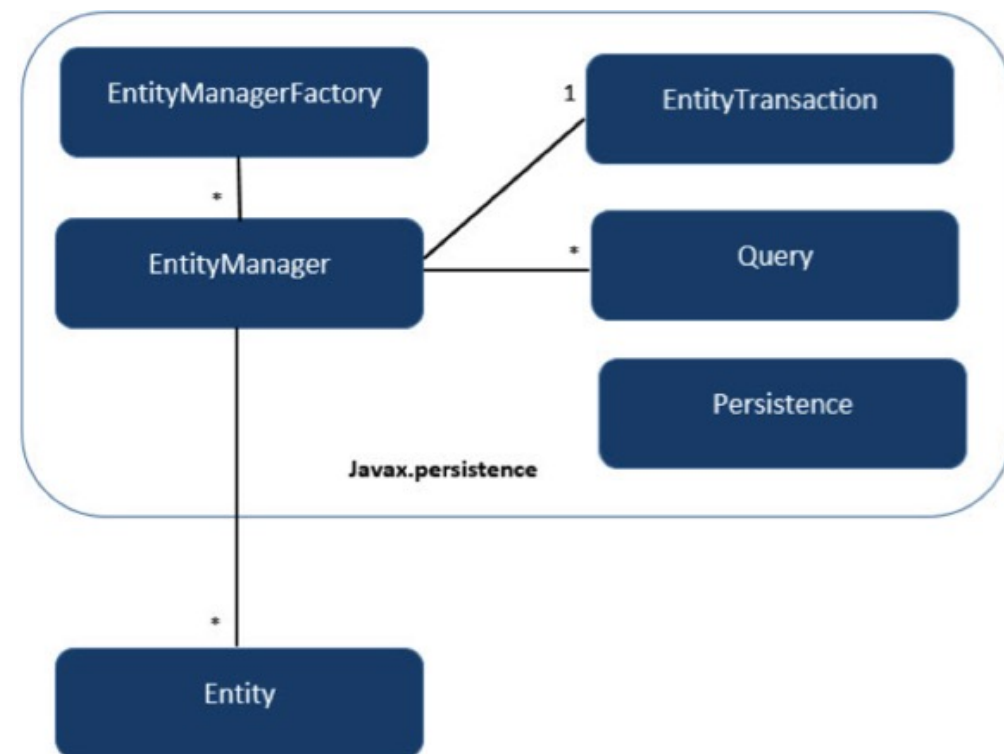
Persistence Layer



- Entity là đối tượng đại diện cho dữ liệu ở trong ứng dụng
- Entity thường là POJO (Plain Old Java Object)
- Entity sẽ được ánh xạ (mapping) tới một bảng trong CSDL
- Trong JPA, một entity cần tuân thủ:
 - Được gắn với annotation `javax.persistence.Entity`
 - Có một constructor là public và không có tham số
 - Không được khai báo final
 - Các biến đối tượng cần được khai báo là private, protected hoặc ở mức package-private

Persistence Context & Entity Manager

- Persistence Context là tập các thể hiện của entity được quản lý, tồn tại trong một kho dữ liệu
- Interface EntityManager:
 - Khai báo các phương thức để tương tác với persistence context
 - Tạo hoặc xóa các thể hiện của entity
 - Tìm kiếm entity theo khoá chính
 - Thực thi các câu lệnh truy vấn lên entity



- Container-managed Entity Manager:

```
@PersistenceContext  
EntityManager em;
```

- Application-managed Entity Manager:

```
@PersistenceUnit  
EntityManagerFactory emf;
```

```
EntityManager em = emf.createEntityManager();
```

Tìm một entity

- Ví dụ:

```
@PersistenceContext
EntityManager em;
public void enterOrder(int custID, Order newOrder) {
    Customer cust = em.find(Customer.class, custID);
    cust.getOrders().add(newOrder);
    newOrder.setCustomer(cust);
}
```

- Ví dụ:

```
@PersistenceContext  
EntityManager em;
```

```
...
```

```
public Lineltem createLineltem(Order order, Product product, int quantity) {  
    Lineltem li = new Lineltem(order, product, quantity);  
    order.getLineltems().add(li);  
    em.persist(li);  
    return li;  
}
```

Xoá entity

- Ví dụ:

```
public void removeOrder(Integer orderId) {  
    try {  
        Order order = em.find(Order.class, orderId);  
        em.remove(order);  
    }...
```


Câu lệnh truy vấn động

- Phương thức `createQuery()` của lớp `EntityManager` giúp tạo các câu truy vấn động (dynamic query)
- Ví dụ:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```

Named Query

- Phương thức `createNamedQuery()` của lớp `EntityManager` giúp tạo các câu truy vấn tĩnh (static query)
- Ví dụ, khai báo named query:

```
@NamedQuery(  
    name="findAllCustomersWithName",  
    query="SELECT c FROM Customer c WHERE c.name LIKE :custName"  
)
```

- Sử dụng named query

```
@PersistenceContext  
public EntityManager em;  
  
...  
customers = em.createNamedQuery("findAllCustomersWithName")  
    .setParameter("custName", "Smith")  
    .getResultList();
```

Named Parameter

- Tên của các tham số bắt đầu bằng dấu (:)
- Ví dụ:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .getResultList();  
}
```

- Sử dụng phương thức setParameter() để truyền giá trị

Positional Parameter

- Vị trí của các tham số bắt đầu bằng dấu (?)
- Ví dụ:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE ?1")  
        .setParameter(1, name)  
        .getResultList();  
}
```

Demo

JPA – Java Persistence API

Quản lý transaction

```
@PersistenceContext
EntityManagerFactory emf;
EntityManager em;
@Resource
UserTransaction utx;

.....

    em = emf.createEntityManager();
    try {
        utx.begin();
        em.persist(SomeEntity);
        em.merge(AnotherEntity);
        em.remove(ThirdEntity);
        utx.commit();
    } catch (Exception e) {
        utx.rollback();
    }
```

Thảo luận

Spring Data JPA

Cả tiến JPA
tiêu chuẩn

Đơn giản hoá
tầng truy xuất
dữ liệu

Tự tạo
repository

Truy vấn
DSL

Ghi log,
phân trang

Có thể tùy biến
nếu cần thiết

Lựa chọn Tầng truy xuất dữ liệu

JDBC Spring JDBC

- Đơn giản
- Thuần SQL

JEE 7 Batch Spring Batch Hadoop

- Rất nhiều câu lệnh ghi SQL để thực hiện

ORM JPA/Hibernate Spring Data JPA

- Truy xuất các mối quan hệ

NoSQL MongoDB Spring Data Mongo

- Nhóm các dữ liệu có quan hệ với nhau

Spring Data Commons

- Repository
- Cross-Store persistency
- Dynamic query generation

Spring Data JPA ✓

Spring Data MongoDB

Spring Data Neo4j

Spring For Hadoop

Spring Data SOLR

Spring Data Redis

Spring Data REST

Spring Data JDBC Ext.

Spring Data Gemfire

Spring Data Cassandra

Spring Data Couchbase

Spring Data DynamoDB

Spring Data Elasticsearch

Cấu hình Spring Data JPA

build.gradle

```
dependencies {  
    compile group: 'org.springframework.data', name: 'spring-data-jpa', version: '2.0.7.RELEASE'  
    compile group: 'jstl', name: 'jstl', version: '1.2'  
    compile group: 'mysql', name: 'mysql-connector-java', version: '8.0.11'  
    testCompile group: 'org.junit.jupiter', name: 'junit-jupiter-engine', version: '5.2.0'  
}
```

Cấu hình DataSource

- Có thể cấu hình thông qua XML hoặc Annotation:

@Bean

```
public DataSource dataSource(){  
    DriverManagerDataSource dataSource = new DriverManagerDataSource();  
    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");  
    dataSource.setUrl("jdbc:mysql://localhost:3306/phone_store");  
    dataSource.setUsername( "root" );  
    dataSource.setPassword( "123456" );  
    return dataSource;  
}
```

Cấu hình Entity Manager

@Bean

@Qualifier(value = "entityManager")

```
public EntityManager entityManager(EntityManagerFactory entityManagerFactory) {  
    return entityManagerFactory.createEntityManager();  
}
```

@Bean

```
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {  
    LocalContainerEntityManagerFactoryBean em  
        = new LocalContainerEntityManagerFactoryBean();  
    em.setDataSource(dataSource());  
    em.setPackagesToScan(new String[] { "com.codegym.phonestore.model" });  
  
    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
    em.setJpaVendorAdapter(vendorAdapter);  
    em.setJpaProperties(additionalProperties());  
    return em;  
}
```

Cấu hình Model

```
@Entity
@Table(name = "phones")
public class Phone {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false)
    private String name;

    @ManyToOne(targetEntity = Manufacture.class, cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
    private Manufacture manufacture;

    //Constructors
    //Getters/Setters
}
```

Truy xuất dữ liệu qua Entity Manager

@Transactional

```
public class CountryRepositoryImpl implements CountryRepository {
```

@PersistenceContext

```
EntityManager em;
```

@Override

```
public List<Country> findAll() {  
    TypedQuery<Country> query = em.createQuery("select c from Country c", Country.class);  
    return query.getResultList();  
}
```

@Override

```
public Country findById(Long id) {  
    return em.find(Country.class, id);  
}
```

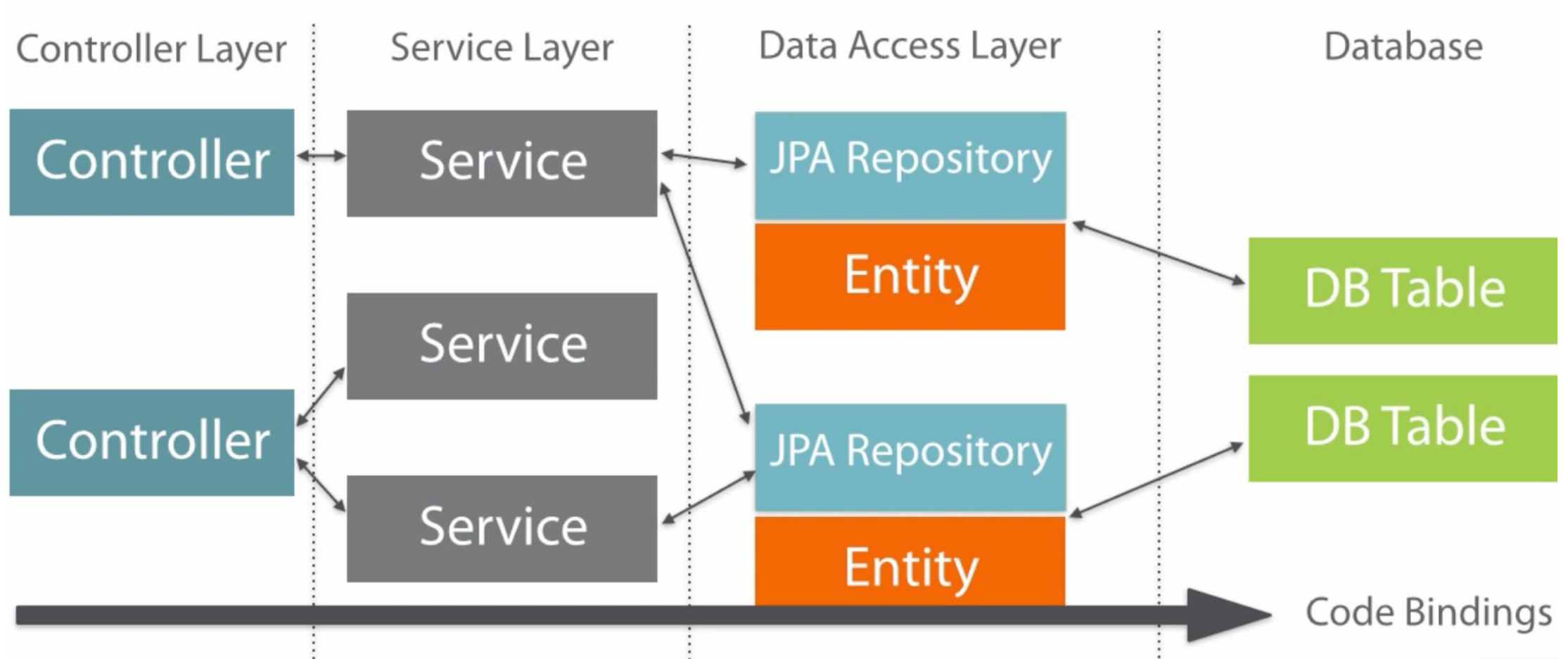
```
.....
```

```
}
```

Các thao tác cơ bản

```
public interface Repository <T> {  
    List<T> findAll();  
  
    T findById(Long id);  
  
    T save(T model);  
  
    void remove(Long id);  
}
```


Kiến trúc Repository



Demo

Spring Data JPA

- ORM là cơ chế ánh xạ dữ liệu giữa môi trường thực thi và CSDL
- JPA là đặc tả của Java dành cho các thao tác với dữ liệu
- Entity là các đối tượng đại diện cho dữ liệu
- Entity Manager là đối tượng quản lý các entity
- Spring Data JPA là framework của Spring hỗ trợ JPA, giúp cho việc triển khai JPA trở nên thuận tiện hơn

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Spring Data Repository*