

ORM - Spring Data JPA

hannkim

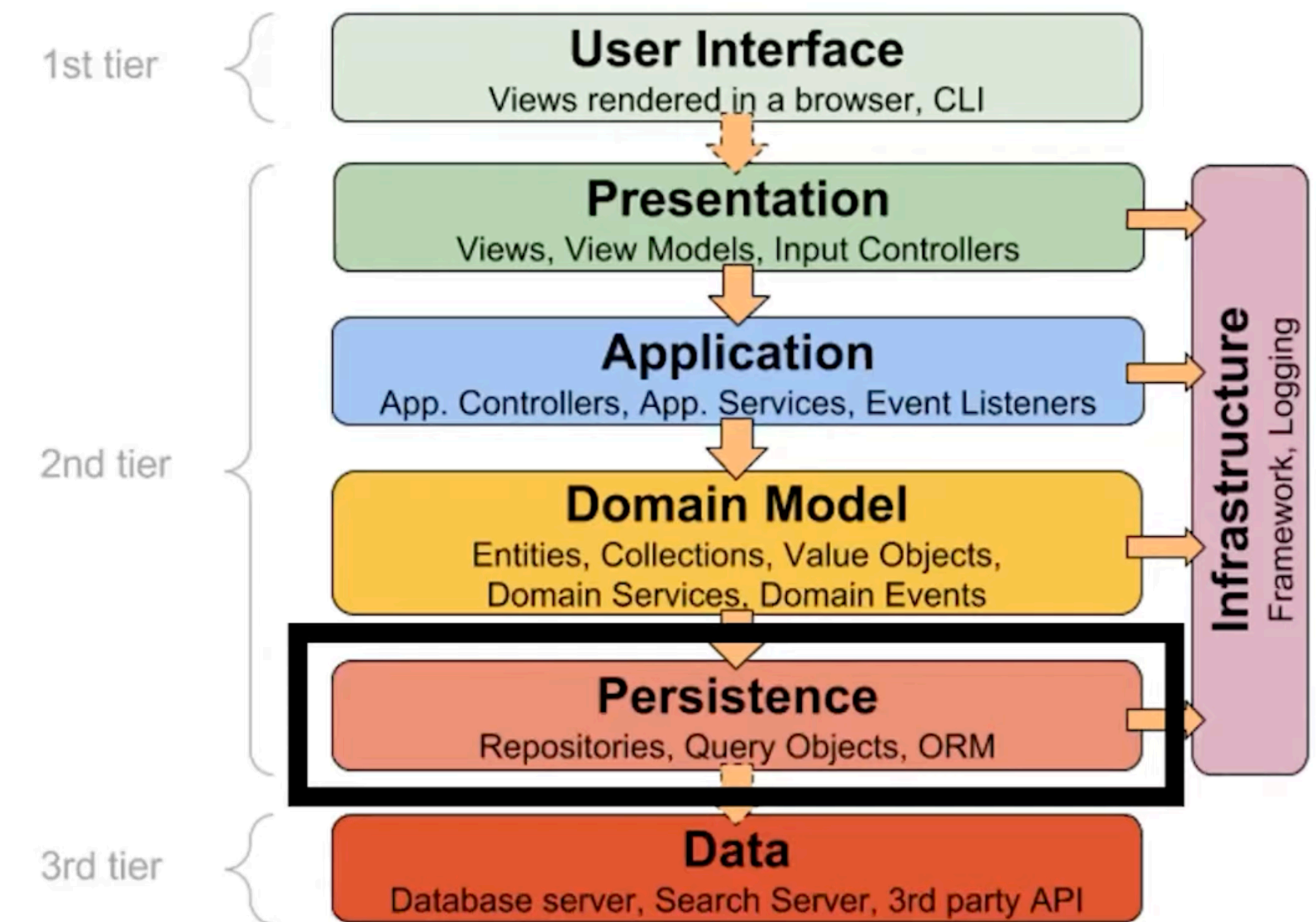
목차

- Persistence
- JDBC API
- SQL Mapper
- ORM

Persistence

영속성

- 프로그램이 종료되어도 사라지지 않는 데이터 특성
- 메모리에서만 존재하는 데이터는 프로그램이 종료되면 내용을 잃기 때문에 File, RDB, Object DB 등을 활용하여 데이터 영속성 (Persistence) 부여
- Persistence Framework
 - SQL Mapper
 - ORM



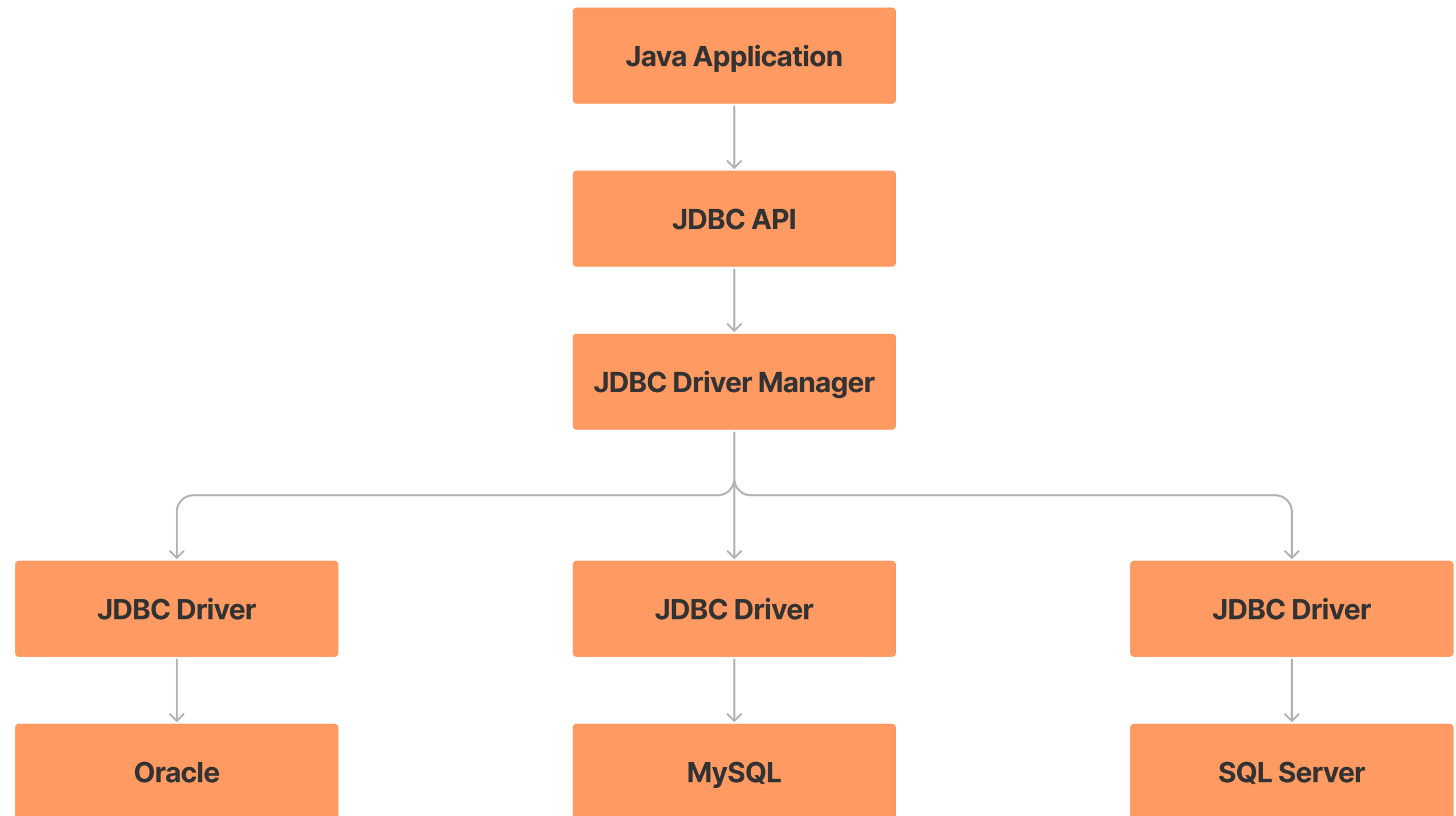
www.herbertograca.com

Mark Richards의 소프트웨어 아키텍처 패턴

JDBC : Java Database Connectivity

JDBC API

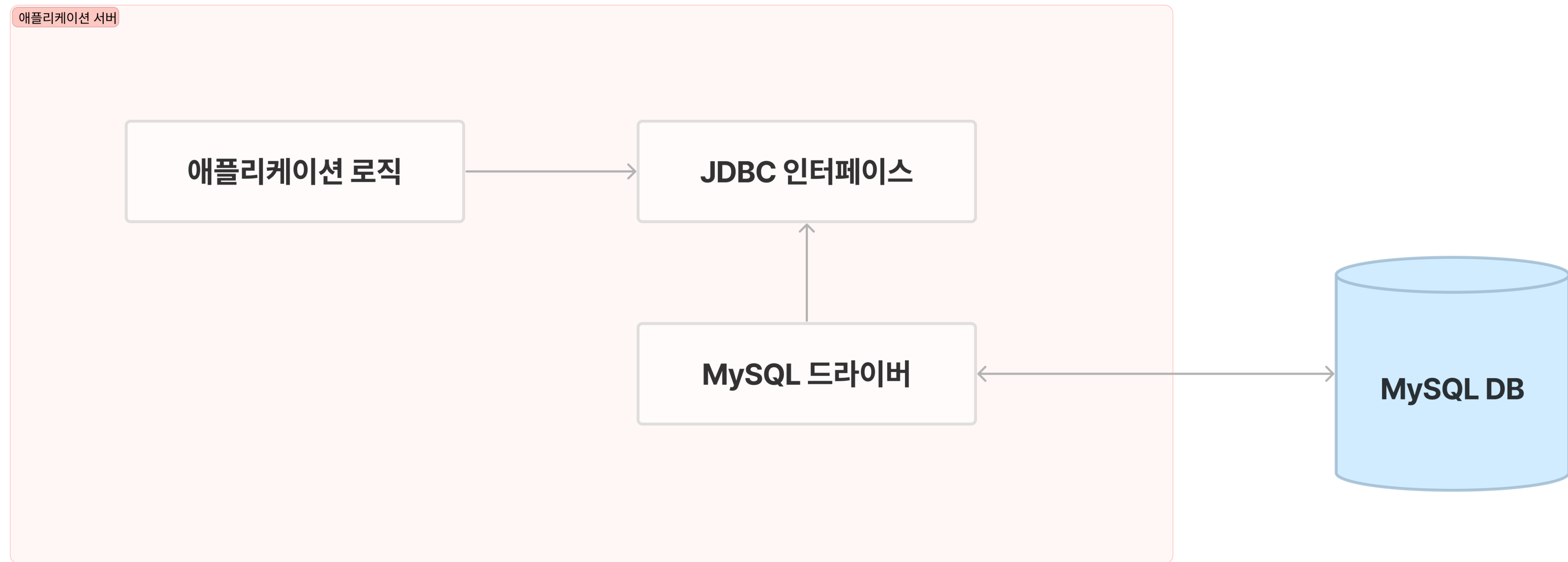
- JDBC API
 - Java 프로그램이 데이터베이스와 연결되어 데이터를 주고 받을 수 있게 해주는 Interface



Architecture of JDBC

JDBC API

Driver 예시



JDBC Flow

JDBC API

단점

- connection 관리로 sql 실행 전, 후 코드가 길어짐
- 데이터베이스 로직에 있는 코드 관리를 위한 예외 작성
- 여러 개의 데이터베이스로부터 코드 반복이라는 시간 낭비


```
public class JdbcExample {  
    public static void main(String[] args) {  
        // JDBC 연결 정보  
        String jdbcUrl = "jdbc:mysql://localhost:3306/your_database_name";  
        String username = "your_username";  
        String password = "your_password";  
  
        try {  
            // JDBC 드라이버 로드  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            // 데이터베이스 연결  
            Connection connection = DriverManager.getConnection(jdbcUrl, username, password);  
  
            // SQL 쿼리 실행  
            Statement statement = connection.createStatement();  
            String sqlQuery = "SELECT * FROM your_table_name";  
            ResultSet resultSet = statement.executeQuery(sqlQuery);  
  
            // 결과 출력  
            while (resultSet.next()) {  
                int id = resultSet.getInt("id");  
                String name = resultSet.getString("name");  
                System.out.println("ID: " + id + ", Name: " + name);  
            }  
  
            // 리소스 정리  
            resultSet.close();  
            statement.close();  
            connection.close();  
        } catch (Exception e) {  
            // ...  
        }  
    }  
}
```

SQL Mapper

Spring Data JDBC, MyBatis

- SQL을 직접 작성하고 객체의 필드를 매핑하여 데이터 객체화
- SQL <- Mapping -> Object Field
- SQL Mapper Framework
 - MyBatis
 - Spring Data Jdbc (Jdbc Template)

java

 Copy code

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public class UserDao {

    private final JdbcTemplate jdbcTemplate;

    public UserDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public List<User> getAllUsers() {
        String sql = "SELECT * FROM users";
        return jdbcTemplate.query(sql, new UserMapper());
    }
}
```

SQL Mapper

단점 1. 특정 DB 의존적인 개발

특정 DB에 의존적으로 개발
엔티티 필드가 수정됐을 때, SQL Mapper부터
JDBC API 코드 모두 변경해야 함

SQL Mapper

단점 2. 패러다임 불일치

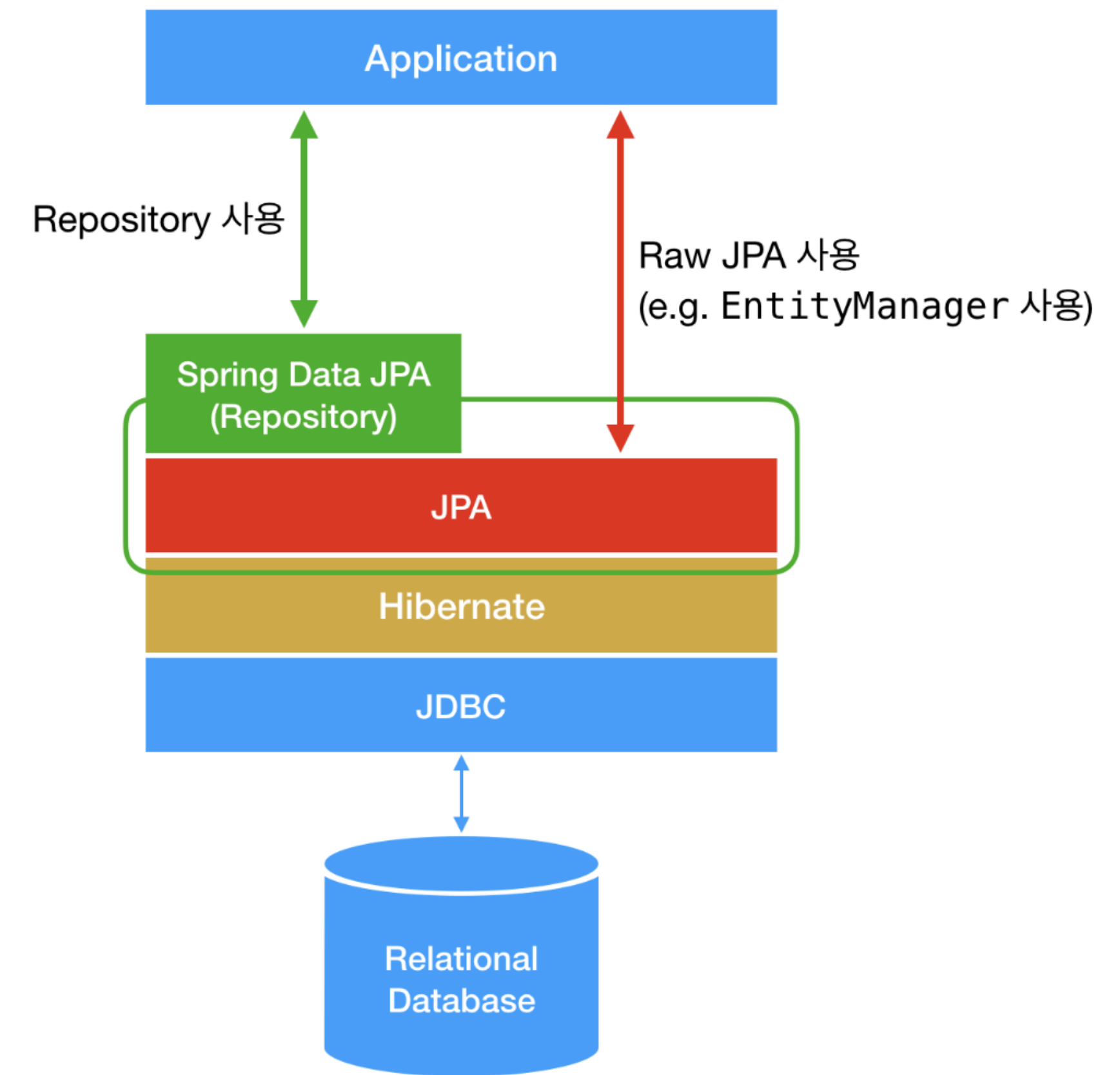
"물리적으로는 SQL과 JDBC API를 데이터 접근 계층에 숨기는 데 성공했을지 몰라도, 논리적으로는 엔티티와 아주 강한 의존 관계를 가지고 있다."

객체 지향 vs RDB의 데이터 중심 구조

ORM : Object-Relational Mapping

Spring Data JPA

- 객체와 관계형 데이터베이스의 데이터를 자동으로 mapping
- ORM Framework
 - Spring Data JPA
 - Hibernate



JPA : Java Persistence API

Spring Data JPA

- SQL Mapper의 패러다임 불일치 문제 -> "연관 관계"로 해결
- Interface (Java ORM 기술에 대한 API 표준 명세)
- JPA Interface 구현한 ORM Framework -> Hibernate!
- Entity Manager를 한 단계 더 추상화 -> Repository!

NEXT

JDBC?

Spring Data JDBC?

JDBC Template?

Spring Data JPA?

감삼다