9월 15일 써머리(최형규)

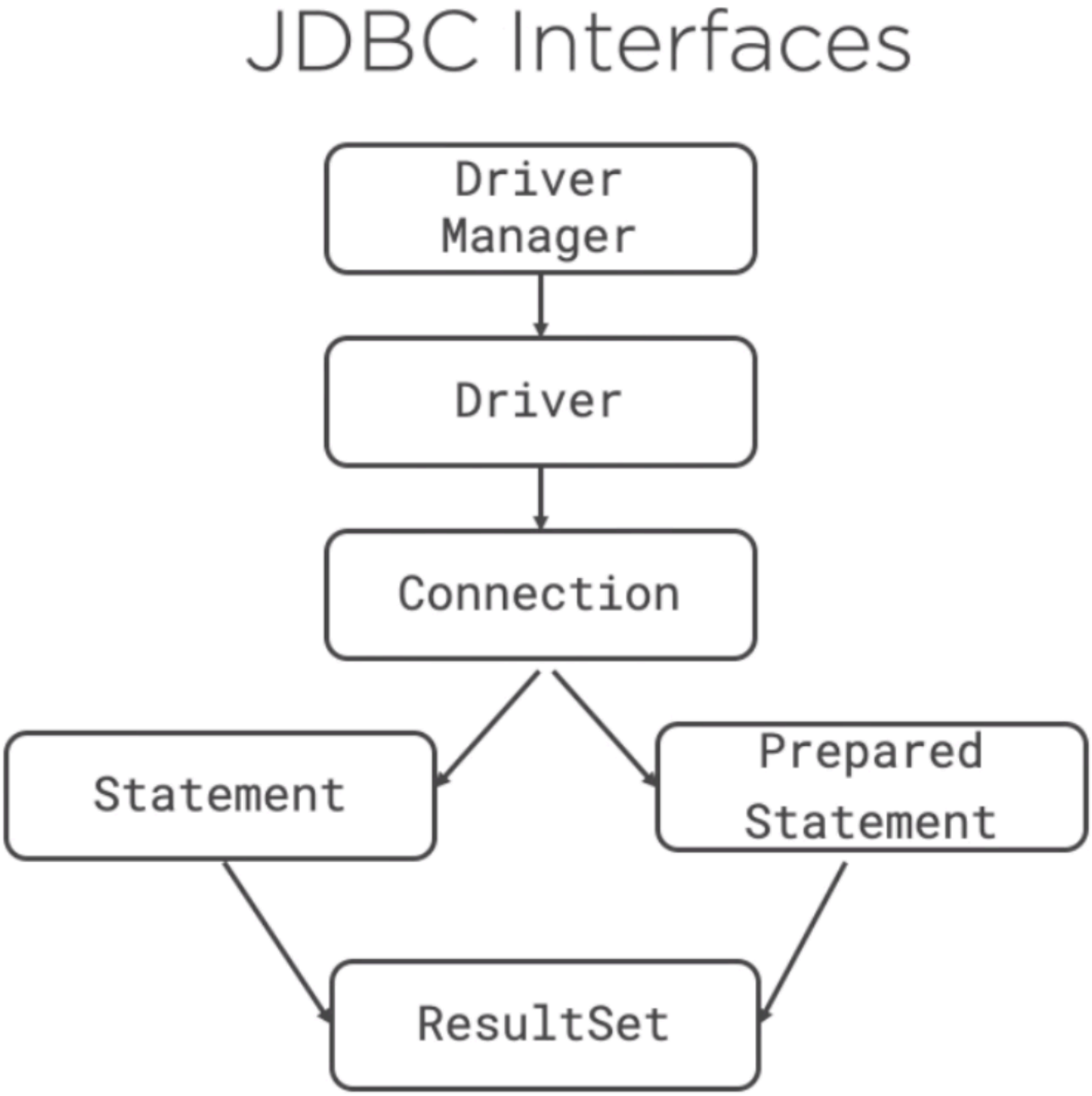
@라이브러리

- **Spring, MyBatis 같은 건 "라이브러리"**
- **Maven, Gradle은 그걸 관리하는 "빌드/의존성 관리 도구"**

정리

- **Maven**: 전통적이고 안정적, XML 기반, 학습이 쉽지만 장황함
- **Gradle**: 최신 트렌드, 빌드 속도 빠름, DSL 기반으로 간결하고 확장성 좋음

@기본 프로그래밍 순서


JDBC Interfaces
Driver Manager
Driver
Connection
Statement
Prepared Statement
ResultSet

1. **Import** java.sql 패키지

2. **Connection** 객체 생성 (DriverManager 이용)

3. **PreparedStatement** 객체 생성

4. **파라미터 바인딩** (? 값 설정)

5. **SQL 실행**

6. SELECT: executeQuery() → ResultSet

7. INSERT/UPDATE/DELETE: executeUpdate() → int

8. **리소스 해제** (close)

-- 교안 참조

- DriverManager
  (DriverManager.getConnection(...))
  드라이버를 관리하는 클래스 이고, 이것을 기점으로 DB연결을 시작하고, DRIVER는 실제로는 DriverManager가 내부적으로 작동(Mysql 등)
- Connection
  (Connection conn = DriverManager.getConnection(url, user, password)
  DB와 연결된 세션.
  SQL문을 실행할 수 있는 `Statement` 나 `PreparedStatement` 객체를 생성
- **Statement**
  단순 SQL 실행 객체 (문자열 그대로 실행)
  예: `Statement stmt = conn.createStatement();`
- **PreparedStatement**
  미리 컴파일된 SQL 실행 객체
  "?" 로 값이 들어올 곳을 미리 준비함
  예제:
  String sql = "insert into dept(deptno, dname, loc) values(?,?,?)";
- ResultSet
  결과조회
  rs = ps.executeQuery();

- 정보전달
  사람이 보는 것(콘솔 출력) - System.out.print()
  컴퓨터가 보려면?
  1. public int add(int a, int b) {
     return a + b; // 호출한 쪽이 이 값을 받아서 사용}
  2. 파일에 쓰거나 bufferedWriter 사용
  3. 이것 외에 많음

1. DBUtil

- **Database Utility 클래스**
- DB 연결, 자원 해제, 드라이버 로딩 같은 **공통 기능**을 모아둔 클래스.
- 보통 `static` 메서드로 만들어서, 다른 곳에서 쉽게 재사용함.

---

2. 인터페이스 (Interface)

- 클래스가 **어떤 기능을 가져야 하는지** 정의하는 **설계도**.
- 실제 구현은 없고, 메서드 시그니처만 있음.
- `DAO`, `Service` 같은 레이어에서 **표준 규격**으로 많이 사용.

---

3. DAO (Data Access Object)

- DB와 직접 통신해서 데이터를 CRUD 하는 클래스.
- SQL 실행, DB 연결, 결과를 DTO/VO로 변환하는 역할.
- 보통 인터페이스(`DAO`) + 구현체(`DAOImple`) 구조로 사용.

---

4. DTO (Data Transfer Object)

- **계층 간 데이터 전달**을 위한 객체.
- 단순히 **데이터만 담는 역할** (`getter/setter`).
- 로직이 거의 없음.
- 주로 Controller ↔ Service ↔ DAO 간 데이터 이동에 사용.

---

5. VO (Value Object)

- DTO와 비슷하게 데이터를 담는 객체.
- 차이점은 **값 자체에 의미가 있고 불변(immutable)하게 설계**하는 경우가 많음.
- `equals()`, `hashCode()` 재정의해서 **동등성 비교**에 사용됨.

📌 간단 비교:

- DTO: "데이터 전달용 가방" (값 바뀔 수 있음)
- VO: "값 자체가 중요한 객체" (값이 같으면 같은 객체로 간주)

✅ 비교 정리

구분	주요 목적	특징	예시
DBUtil	공통 DB 기능 제공	연결/종료/드라이버 관리	`DBUtil.getConnection()`
인터페이스	표준 설계 규약	구현 없음	`interface MemberDAO`
DAO	DB 접근/쿼리 실행	SQL 처리 + DTO 변환	`MemberDAOImple`
DTO	데이터 전달	단순 가방, 가변적	`MemberDTO(name, age)`
VO	값 표현	불변, equals/hashCode 재정의	`MoneyVO(1000원)`
Entity	DB 테이블 매핑	ORM에서 사용, DB와 직접 연결	`@Entity class User {...}`

비유로 설명

- **DBUtil** → 🔧 "공구상자" (DB 연결, 종료 도와주는 도구들 모음)
- **인터페이스** → 📑 "계약서" (이 기능 꼭 구현하세요!)
- **DAO** → 📬 "우체국 직원" (DB와 대화해서 데이터 가져오고 보내줌)
- **DTO** → 📦 "택배 상자" (데이터를 담아서 전달, 내용물 바뀔 수 있음)
- **VO** → 💎 "보석" (값 자체가 중요, 동일 값이면 같은 의미, 쉽게 안 바뀜)
- **Entity** → 🏠 "부동산 등기부" (DB와 직접 연결된 실체, 테이블과 매핑)

!수업 때는 DTO, DAO 구현에 대해 집중 설명

https://github.com/likelion-backend-2025/workspace/blob/main/happyjdbc/src/main/java/lion/jdbc/DeptDAO.java