DAO 리팩토링 실습

—

실습 환경

https://github.com/slipp/jwp-basic 저장소를 자신의 계정으로 fork한다.

- Fork한 저장소를 https://youtu.be/xid_GG8kL_w 동영상 참고해 로컬 개발 환경을 구축한다.
- jwp-basic 저장소의 **step2-user-with-mvc-framework** 브랜치로 변경한다.
  - 브랜치를 변경하는 방법은 https://youtu.be/VeTjDYI7UVs 동영상을 참고한다.
- src/test/java 디렉토리의 next.WebServerLauncher를 실행한 후 브라우저에서 http://localhost:8080으로 접속해 질문/답변 게시판 서비스 화면이 나타나는지 확인한다.

요구사항

JDBC에 대한 공통 라이브러리를 만들어 개발자가 SQL 쿼리, 쿼리에 전달할 인자, SELECT 구문의 경우 조회한 데이터를 추출하는 3가지 구현에만 집중하도록 해야 한다.
또한 SQLException을 런타임 Exception으로 변환해 try/catch 절로 인해 소스 코드의 가독성을 헤치지 않도록 해야 한다.

공통 라이브러리와 개발자가 구현할 부분 분리

작업	공통 라이브러리	개발자가 구현할 부분
Connection 관리	O	X
SQL	X	O
Statement 관리	O	X
ResultSet 관리	O	X
Row 데이터 추출	X	O
패러미터 선언	X	O
패러미터 Setting	O	X
트랜잭션 관리	O	X

추가 구현 및 테스트

리팩토링 과정에서 테스트는 src/test/java에 있는 next.dao.UserDaoTest 클래스를 활용하면 된다. 앞의 JDBC 실습 과정에 있는 회원목록, 개인정보수정 실습을 진행하지 않으면 UserDaoTest는 실패한다. 이 테스트 코드가 성공하도록 회원목록과 개인정보수정 실습을 진행한 후 UserDao에 대한 리팩토링 실습을 진행한다.

요구사항 분리 및 힌트

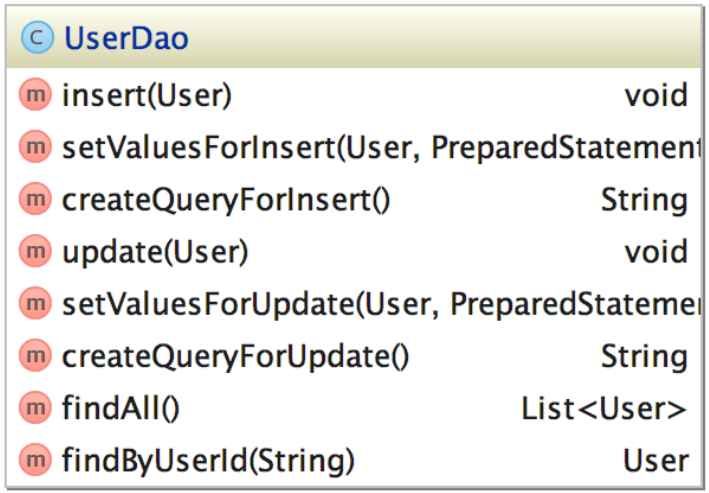
—

실습

INSERT, UPDATE 쿼리는 비슷하기 때문에 먼저 INSERT, UPDATE 쿼리를 가지는 메소드의 중복 제거 작업을 진행한다.

힌트

먼저 변하는 부분과 변하지 않는 부분을 Extract Method 리팩토링을 통해 분리한다. 리팩토링 결과는 다음과 같다.

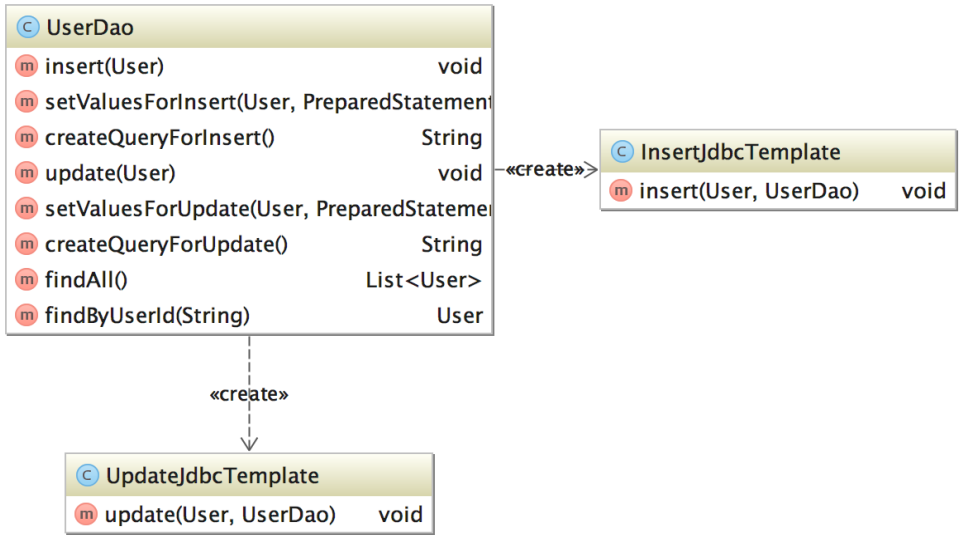


실습

분리한 메소드 중에서 변화가 발생하지 않는 부분(즉, 공통 라이브러리로 구현할 코드)을 새로운 클래스로 추가한 후 이동한다.

힌트

- insert()와 update() 메소드별로 InsertJdbcTemplate, UpdateJdbcTemplate과 같이 새로운 클래스를 추가한 후 insert()와 update() 메소드 코드를 이동한다.
- insert(), update() 메소드를 이동할 때 인자로 UserDao를 전달해야 한다. 만약 앞에서 추출한 메소드가 private 접근 제어자라면 default 접근 제어자로 리팩토링한다.
- UserDao의 insert()가 InsertJdbcTemplate의 insert(), update()가 UpdateJdbcTemplate의 update() 메소드를 호출하도록 리팩토링한다.

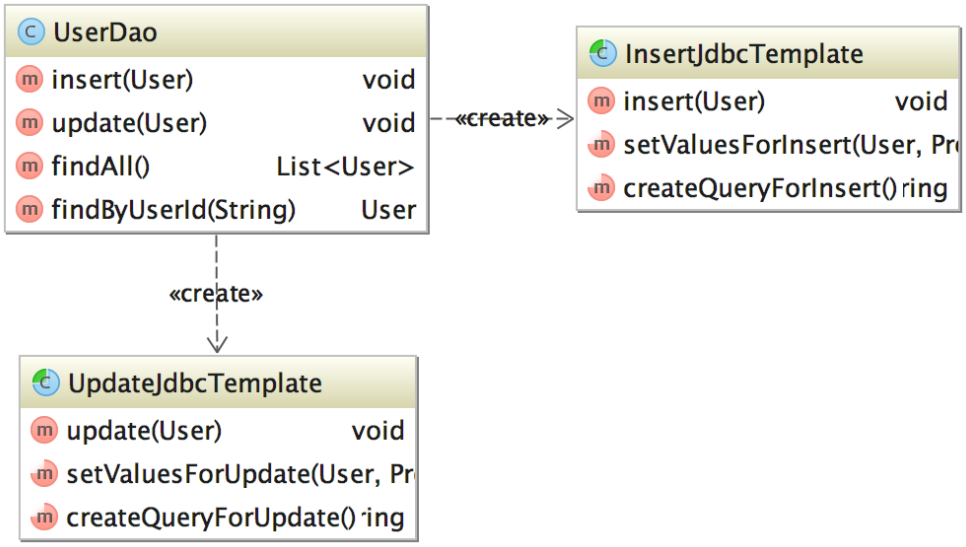


실습

InsertJdbcTemplate과 UpdateJdbcTemplate이 UserDao에 대한 의존관계를 가진다. UserDao에 대한 의존관계를 끊는다.

힌트

- InsertJdbcTemplate이 UserDao에 의존관계를 가지는 이유는 setValuesForInsert()와 createQueryForInsert() 때문이다. 이 두 개의 메소드를 추상 메소드로 구현하고 UserDao의 insert() 메소드에서 이 2개의 추상 메소드를 구현하도록 한다. 2개의 추상 메소드 구현은 insert() 메소드에서 익명 클래스로 구현한다. 익명 클래스 구현에 대해 잘 모르겠으면 추가 학습한다.
- UpdateJdbcTemplate도 같은 과정으로 리팩토링한다.

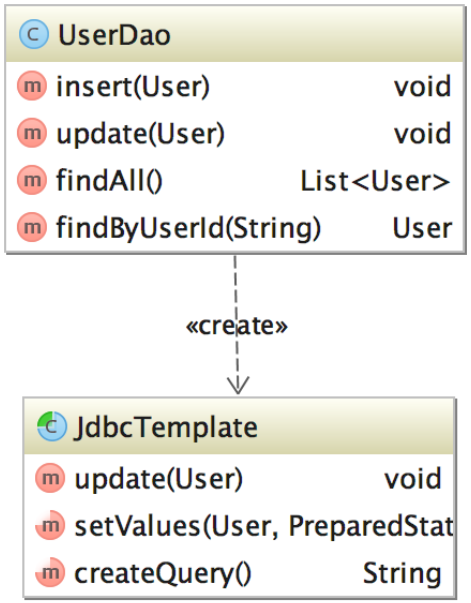


실습

InsertJdbcTemplate과 UpdateJdbcTemplate의 구현 부분이 다른 부분이 없다.
둘 중의 하나를 사용하도록 리팩토링한다.

힌트

- InsertJdbcTemplate과 UpdateJdbcTemplate의 메소드를 setValues()와 createQuery() 메소드로 Rename 리팩토링한다.
- 두 클래스의 구현 코드가 같기 때문에 둘 중 하나를 삭제하고 클래스 하나만 사용하도록 리팩토링한다.



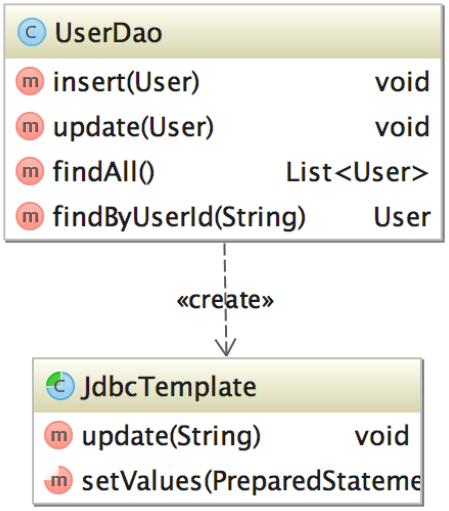
실습

JdbcTemplate는 아직도 User와 의존관계를 가지기 때문에 재사용할 수 없다.

User와 의존관계를 끊는다.

힌트

- User 값은 UserDao에서만 사용되기 때문에 JdbcTemplate의 update() 메소드에 굳이 User를 전달할 필요가 없다.
- SQL 쿼리와 같이 변경되는 부분을 추상 메소드가 아닌 메서드의 인자로 전달한다.

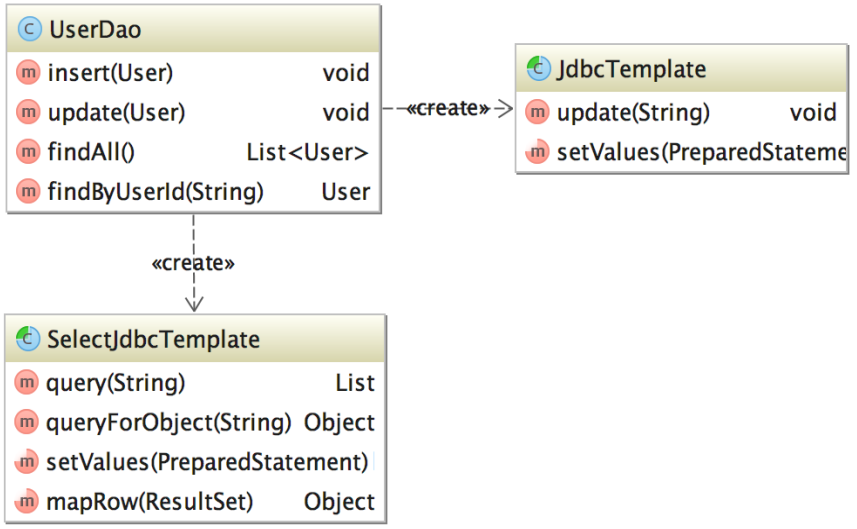


실습

더 이상 JdbcTemplate은 특정 DAO 클래스에 종속적이지 않다. 이와 똑같은 방법으로 SelectJdbcTemplate을 생성해 반복 코드를 분리한다.

힌트

- 리팩토링 과정은 앞의 JdbcTemplate과 같다. 다른 점이라면 ResultSet 데이터를 자바 객체로 변환하는 부분이 추가되어야 한다.
- SelectJdbcTemplate는 setValues() 메소드와 mapRow() 메소드와 같은 2개의 추상 메소드를 가져야 한다. mapRow() 메소드의 반환 값은 Object여야 한다.

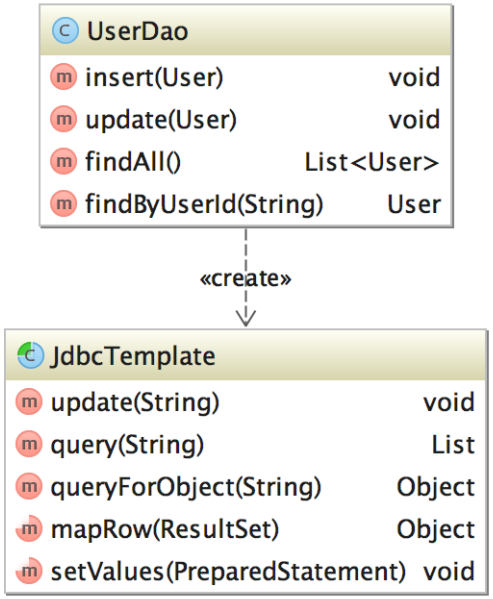


실습

JdbcTemplate과 SelectJdbcTemplate을 보니 중복 코드가 보인다. 또한 굳이 2개의 클래스를 제공하고 싶지 않다. JdbcTemplate과 같은 한 개의 클래스만을 제공하도록 리팩토링해본다.

힌트

- JdbcTemplate의 update() 메소드를 SelectJdbcTemplate로 이동해 클래스 하나에서 모든 작업을 하도록 리팩토링해본다.

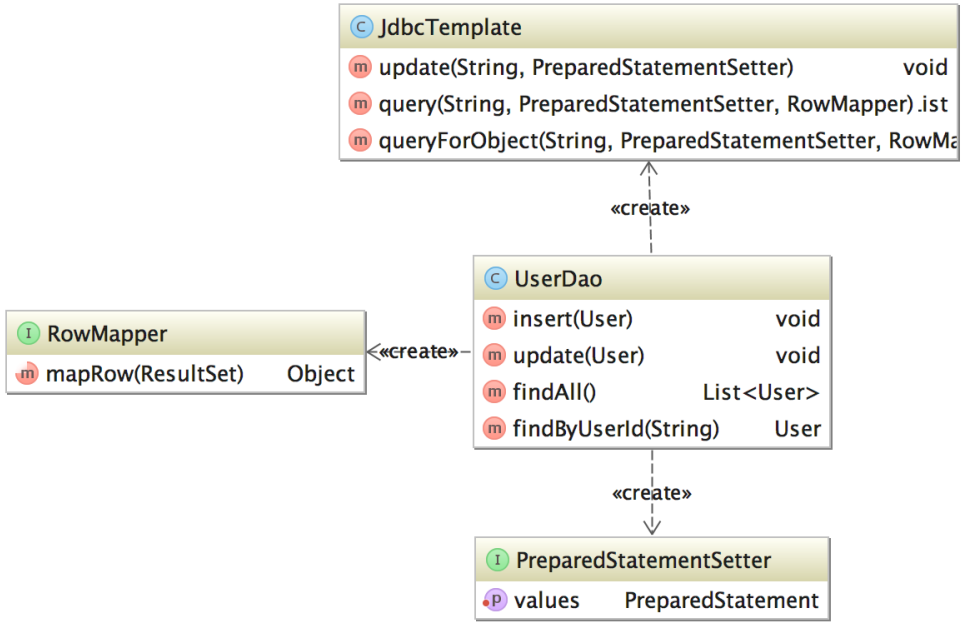


실습

위와 같이 SelectJdbcTemplate 클래스로 통합했을 때의 문제점을 찾아보고 이를 해결하기 위한 방법을 찾아본다.

힌트

- 2개의 클래스를 하나로 통합하면 INSERT, UPDATE, DELETE 쿼리의 경우 불필요한 mapRow() 메소드를 반드시 구현해야 한다. setValues(), mapRow() 2개의 메소드를 분리해 독립적으로 전달할 수 있도록 한다.
- PreparedStatementSetter(setValues() 메소드), RowMapper(mapRow() 메소드)와 같은 인터페이스를 추가해 인자로 전달하도록 리팩토링한다.



실습

SQLException을 런타임 Exception으로 변환해 throw하도록 한다. Connection, PreparedStatement 자원 반납을 close() 메소드를 사용하지 말고 try-with-resources 구문을 적용해 해결한다.

힌트

- RuntimeException을 상속하는 커스텀 Exception(DataAccessException과 같은 이름)을 추가한 후 SQLException을 새로 추가한 커스텀 Exception을 변환해 throw하도록 구현한다.

```
public class DataAccessException extends RuntimeException {
    [...]
}
```

- 자바 7 버전부터 try-with-resources 문법을 적용해 자원을 반납하는 것이 가능하다. 런타임 Exception을 적용하면서 finally 절의 복잡도가 증가하는데try-with-resources 문법을 활용해 코드를 리팩토링해본다.

실습

SELECT문의 경우 조회한 데이터를 캐스팅하는 부분이 있다. 캐스팅하지 않고 구현하도록 개선한다.

힌트

- RowMapper에 자바 제너릭을 적용해 공통 라이브러리를 개선한다.

```
public interface RowMapper<T> {
    T mapRow(ResultSet rs) throws SQLException;
}
```

실습

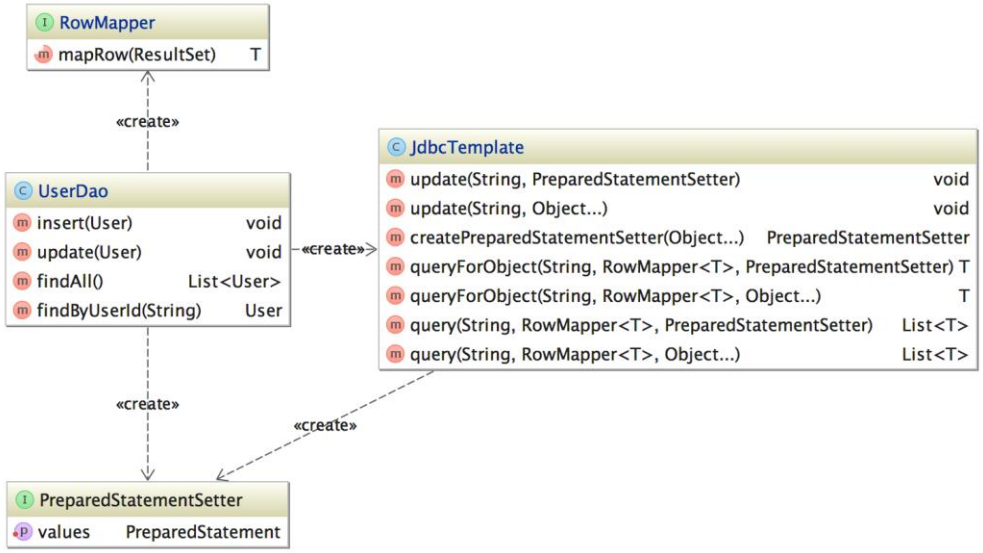
각 쿼리에 전달할 인자를PreparedStatementSetter를 통해 전달할 수도 있지만 자바의 가변인자를 통해 전달할 수 있는 메소드를 추가한다.

힌트

- 자바는 가변인자 문법으로 인자를 동적으로 전달할 수 있다.

  ```
  void update(String sql, Object... values)
  ```

- PreparedStatement에 값을 전달할 때 setObject() 메소드를 활용한다.



실습

UserDao에서PreparedStatementSetter, RowMapper 인터페이스 구현하는 부분을 JDK 8에서 추가한 람다 표현식을 활용하도록 리팩토링한다.

힌트

- 구글에서 “jdk 8 람다”로 검색해 람다 문법을 학습한 후 익명 클래스 대신 람다를 적용해 구현한다.

```
@FunctionalInterface
public interface RowMapper<T> {
    T mapRow(ResultSet rs) throws SQLException;
}
```