5장 - 서비스 추상화

5.1 - 5.2

여재환

5.1 사용자 레벨 관리기능 추가

- 이전까지 만들었던 CRUD의 기능만 갖춘 UserDao를 활용하여 간단한 비즈니스 로직을 만들어 보자.
  - 사용자의 레벨을 BASIC, SILVER, GOLD 로 나눈다.
  - 처음 가입시 BASIC 레벨
  - 가입 후 50회 이상의 로그인을 한 사용자는 SILVER
  - SILVER 레벨이며 30회 이상의 추천을 받으면 GOLD 레벨

5.1.1 필드 추가

- LEVEL을 형태 그대로 DB에 저장하는 방법
  - 관리의 어려움으로 인해 적절치 못함
- 각 레벨을 코드화하여 숫자로 관리하는 것은?
  - 작은 범위의 숫자를 활용한다면 용량면에서 이득
  - 하지만 예상 범위 밖의 값 입력을 방지하지는 못함

5.1.1 필드 추가

- Enum 타입

```
public enum Level {
    BASIC(1), SILVER(2), GOLD(3);

    private final int value;

    Level(int value) { this.value = value; }

    public int intValue() { return value; }

    public static Level valueOf(int value) {
      switch (value) {
        case 1 : return BASIC;
        case 2 : return SILVER;
        case 3 : return GOLD;
        default: throw new AssertionError("Unknown value : " + value);
      }
    }
}
```

- DB에 저장하는 것은 int , 겉으로는 Level 오브젝트
  - 예측 밖의 값의 사용을 차단

5.1.2 사용자 수정기능 추가

- 사용자의 정보는 언제든지 수정될 수 있다.
  - 성능 최적화를 위해서는 상황별 적절한 update 메서드가 필요할 수 있다.
  - 하지만 지금은 규모가 그리 크지 않으므로 변경사항이 생겼을때 해당 사용자의 정보 전체를 업데이트 할 것이다.

5.1.2 사용자 수정기능 추가

- 기능 구현에 앞서 어떤 기능을 만들 것인지 테스트 코드를 작성해 보자.

```
@Test
public void update() {
  dao.deleteAll();

  dao.add(user1);

  user1.setName("오민규");
  user1.setPassword("spring06");
  user1.setLevel("LEVEL.GOLD");
  user1.setLogin(1000);
  user1.setRecommend(99);
  user1.update(user1);

  User user1update = dao.get(user1.getId());
  checkSameUser(user1, user1update);
}
```

5.1.2 사용자 수정기능 추가

- 테스트 코드를 실행하면 update가 없기 때문에 compile error 발생

```
public interface UserDao {
  public void update(User user);
}
```

```
public void update(User user) {
  this.jdbcTemplate.update(
    "update users set name = ?, password = ?, level = ?, login = ?, recommend = ?" +
    "where id = ?",
    user.getName(), user.getPassword(), user.getLevel(), user.getLogin(), user.getRecommend(), user.getId()
  )
}
```

5.1.2 사용자 수정기능 추가

- 현재의 테스트가 보완하지 못하는 부분
  - update문에서 where 절 오류
  - 수정해야 하지 않은 부분을 수정했는지 어떻게 검증할까?

- JdbcTemplate의 update()가 반환하는 리턴값 확인
- 테스트 보강을 통해 다른 사용자의 정보는 변경되지 않았는지 확인

5.1.2 사용자 수정기능 추가

```java
@Test
public void update() {
  dao.deleteAll();

  dao.add(user1);
  dao.add(user2);

  user1.setName("오민규");
  user1.setPassword("spring06");
  user1.setLevel("LEVEL.GOLD");
  user1.setLogin(1000);
  user1.setRecommend(99);
  user1.update(user1);

  User user1update = dao.get(user1.getId());
  checkSameUser(user1, user1update);
  User user2same = dao.get(user2.getId());
  checkSameUser(user2, user2same);
}
```

5.1.3 UserService.upgradeLevels()

- 사용자 관리 로직을 구현할 곳?
  - dao는 데이터를 어떻게 가져오고 조작할 지 다루는 곳
  - 비즈니스 로직을 제공한다는 의미에서 UserService 클래스를 추가하자.
  - UserService는 UserDao를 DI 받아 사용
  - UserService는 UserDao의 구현체가 바뀌어도 영향을 받으면 안된다.
  - 즉 UserService도 스프링 빈으로 등록되야 한다.

5.1.3 UserService.upgradeLevels()

```java
public class UserService {
  UserDao userDao;

  public void  setUserDao(UserDao userDao) {
    this.userDao = userDao;
  }
}
```

```xml
<bean id="userService" class="springbook.user.service.UserService">
  <property name="userDao" ref="userDao"/>
</bean>

<bean id="userDao" class="springbook.dao.UserDaoJdbc">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

5.1.3 UserService.upgradeLevels()

- 사용자의 레벨을 일괄 업그레이드 시키는 코드

```
public void upgradeLevels() {
  List<User> users = userDao.getAll();
  for(User user : users) {
    Boolean changed = null;
    if (user.getLevel == Level.BASIC && user.getLogin() >= 50) {
      user.setLevel(Level.SILVER);
      changed = true;
    }
    else if (user.getLevel == Level.SIVLER && user.getRecommend >= 30) {
      user.setLevel(Level.GOLD);
      changed = true;
    }
    else if (user.getLevel == Level.GOLD) { changed = false; }
    else { changed = false; }

    if(changed) { userDao.update(user); }
  }
}
```

5.1.4 코드 개선

- 앞선 코드의 문제점
  - for 루프 속의 가독성을 저하하는 if / else 문
    - 레벨의 변화 단계와 업그레이드 조건, 조건을 충족시켰을 때 해야할 작업이 섞여 있음
  - 플래그를 활용한 업데이트도 클린코드의 방해 요소
  - Level 이 다양해지고 업그레이드 조건이 다양해지면 관리의 복잡성 증가

5.1.4 코드 개선

- 해결방안1 - 추상화
  - 가장 추상적인 레벨에서 로직 작성

```
public void upgradeLevels() {
  List<User> users = userDao.getAll();

  for(User user : users) {
    if(canUpgradeLevel(user)) {
      upgradeLevel(user);
    }
  }
}
```

```
private boolean canUpgradeLevel(User user) {
  Level currentLevel = user.getLevel();
  switch(currentLevel) {
    case BASIC : return (user.getLogin() >= 50);
    case SILVER : return (uesr.getRecommend >= 30);
    case GOLD : return false;
    default : throw new IllegalArgumentException("Unkonwn level "+ currentLevel);
  }
}
```

5.1.4 코드 개선

```
private void upgradeLevel(User user) {
  if(user.getLevel() == Level.BASIC) { user.setLevel(Level.SILVER); }
  else if(user.getLevel() == Level.SILVER) { user.setLevel(Level.GOLD); }
  userDao.update(user);
}
```

- 레벨 변경의 목적은 달성
  - 예외 처리의 부재
  - 레벨의 확장성에 불리

5.1.4 코드 개선

```
public enum Level {
    GOLD(3, null), SILVER(2, GOLD), BASIC(1, SILVER) ;

    private final int value;
    private final Level next;

    Level(int value, Level next) {
      this.value = value;
      this.next = next;
    }

    public int intValue() { return value; }
    public Level nextLevel() { return this.next; }

    public static Level valueOf(int value) {
      switch (value) {
        case 1 : return BASIC;
        case 2 : return SILVER;
        case 3 : return GOLD;
        default: throw new AssertionError("Unknown value : " + value);
      }
    }
}
```

- Level enum에 next 필드 추가
  - 다음 레벨의 정보 지정
  - 레벨 업그레이드에 대한 것은 Level enum이 담당
  - 복잡한 if 문 제거 가능

5.1.4 코드 개선

```
public void upgradeLevel() {
  Level nextLevel = this.level.nextLevel();
  if(nextLevel == null) {
    throw new IllegalStateException(this.level + "은 업그레이드가 불가능합니다.")
  } else {
    this.level = nextLevel;
  }
}
```

- 사용자의 정보가 바뀌는 부분도 UserService가 아닌 User에서 담당하도록 변경
- User도 자바 오브젝트이기 때문에 내부 정보를 다루는 기능 추가 가능
- UserSevice가 User에게 업그레이드 요청

5.1.4 코드 개선

```
public void upgradeLevel(User user) {
 user.upgradeLevel();
 userDao.upadate(user);
}
```

- 각 오브젝트의 책임을 깔끔하게 분리
- 객체지향의 기본 원리
  - 다른 오브젝트에게 데이터를 요구하지 말고 작업을 요청하라.
  - 코드의 이해하기 쉽게 만들고 변화에 대응하기 쉽게 만드는 기법

5.1.4 코드 개선

- User Test 추가

```
public class UserTest {
  User user;

  @Before
  public void setUp() { user = new User();}

  @Test
  public void upgradeLevel() {
    Level[] levels = Level.values();
    for(Level level : levels) {
      if(level.nextLevel() == null) continue;
      user.setLevel(level);
      user.upgradeLevel();
      assertThat(user.getLevel(), is(level.nextLevel()));
    }
  }

  @Test(expected=IllegalStateException.class)
  public vodi cannotUpgradeLevel {
    Level[] levels = Level.values();
    for(Level level : levels) {
      if(level.nextLevel() != null) continue;
      user.setLevel(level);
      user.upgradeLevel();
    }
  }
}
```

5.1.4 코드 개선

- UserService 리팩토링 & UserServiceTest 리팩토링

```
public static final int MIN_LOGOUT_FOR_SILVER = 50;
public static final int MIN_RECOMMEND_FOR_GOLD = 30;

private boolean canUpgradeLevel(User user) {
  Level currentLevel = user.getLevel();
  switch(currentLevel) {
    case BASIC : return (user.getLogin() >= MIN_LOGOUT_FOR_SILVER);
    case SILVER : return (uesr.getRecommend >= MIN_RECOMMEND_FOR_GOLD);
    case GOLD : return false;
    default : throw new IllegalArgumentException("Unkonwn level "+ currentLevel);
  }
}
```

```
@Before
public void setUp() {
  users = Arrays.asList(
    new User("bumjin", "박범진", "p1", Level.BASIC, MIN_LOGOUT_FOR_SILVER-1, 0);
    new User("joytouch", "강명성", "p2", Level.BASIC, MIN_LOGOUT_FOR_SILVER, 0);
    new User("erwins", "신승한", "p3", Level.SILVER, 60, MIN_RECOMMEND_FOR_GOLD-1);
    new User("madnite1", "이상호", "p4", Level.SILVER, 60, MIN_RECOMMEND_FOR_GOLD);
    new User("green", "오민규", "p5", Level.GOLD, 100, Integer.MAX_VALUE);
  )
}
```

- 상수 지정을 통해 숫자의 중복을 피하고 변경의 이유가 생겼을 때 쉽게 대응 가능
- 테스트는 경계값 위주의 테스트를 통해 원하는 결과를 직관적으로 확인