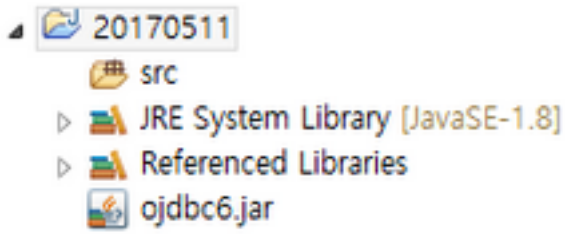
# Framework

Spring에서 program을 짜는 방법

**Framework 규칙**
IOC/DI (90%~)
web 구성(10%~)
+AOP(proxy 패턴) 객체지향을 더 객체지향 > 관점지향프로그래밍
new를 하지 않음
interface
객체지향프로그래밍의 최종정리

---

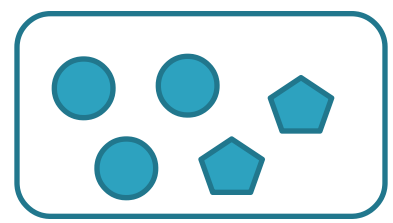
## 1. Ojdbc6.jar file build path

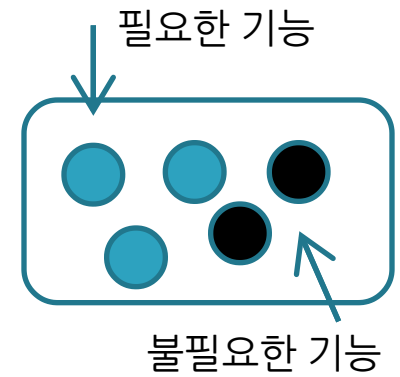



EJB > Enterprise Java Bean : java에서 **기본으로 제공하는** 기업용 solution framework (SMS, mail등)
>> 불필요한 요소가 많아짐

↓ **Spring Framework**

POJO > Plane Old Java Object :만들어 놓은 class
>> 필요한 기능만 등록/사용





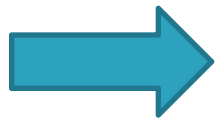

★Container

객체를 만들고 안만들고의 차이
getConnection은 둘 다 포함

return 값이 void면 팩토리 메소드 아님
근데, 작업의 흐름에 영향을 주면 템플릿 메소드

**UserDao**

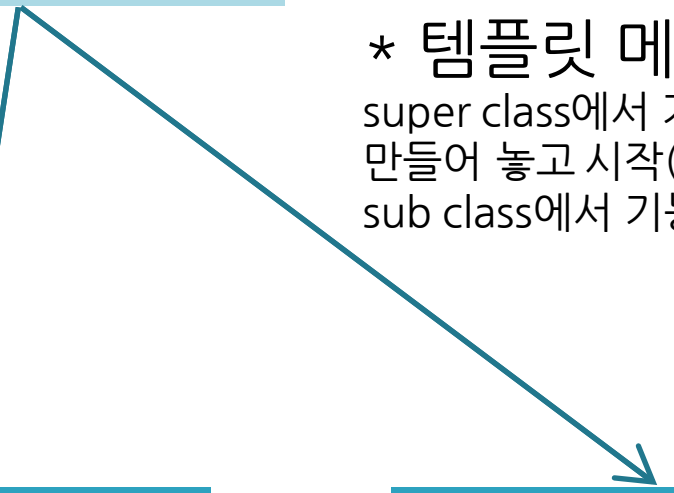
add
get
getConnection



**UserDao**

add
get
getConnection

Super Class



**AUserDao**

getConnection

**BUserDao**

getConnection

Sub Class

### * 템플릿 메소드 패턴

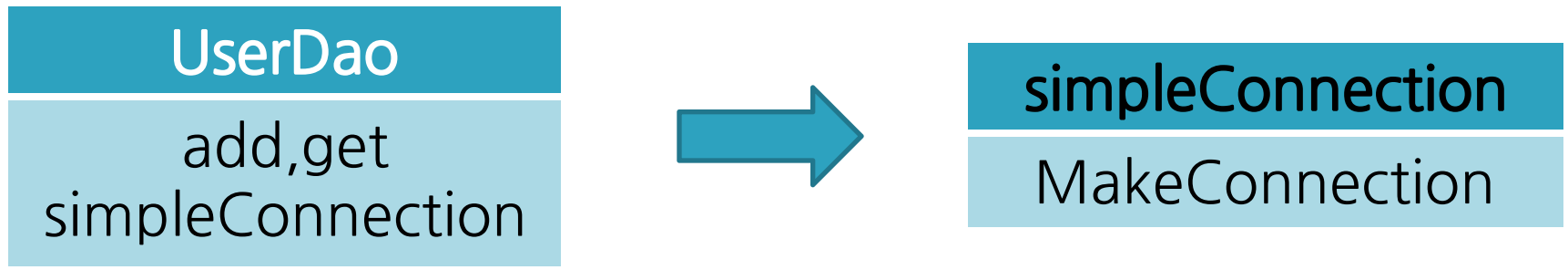
super class에서 거의 모든 기능에 대한 흐름을 만들어 놓고 시작(작업의 선택권)
sub class에서 기능을 구현(Override)한다.

```
//부모가 구현한 메소드의 기능에 포함되지 않는 것은 템플릿 메소드 패턴이라고 할 수 없다.
protected abstract void foo();
//부모가 필요로 하는 객체를 서브클래스에서 만들게 하는 패턴 :  팩토리 메소드 패턴
//서브클래스에서 구체적인 오브젝트의 생성 방법을 결정지어야 하기 때문에 팩토리메소드 패턴에 포함되는 팩토리 메소드
protected abstract Connection getConnection() throws SQLException, ClassNotFoundException;
//자식 클래스에서 오브젝트를 만드는 방법을 결정지을 수 없기 때문에 팩토리메소드패턴에포함되지않는 팩토리 메소드
public User getUser(){
    return new User();
}
```

### * 팩토리 메소드 패턴

부모가 필요로 하는 객체를 서브클래스에서 만들게 함
서브클래스에서 구체적인 오브젝트의 생성 방법 결정

# exam03



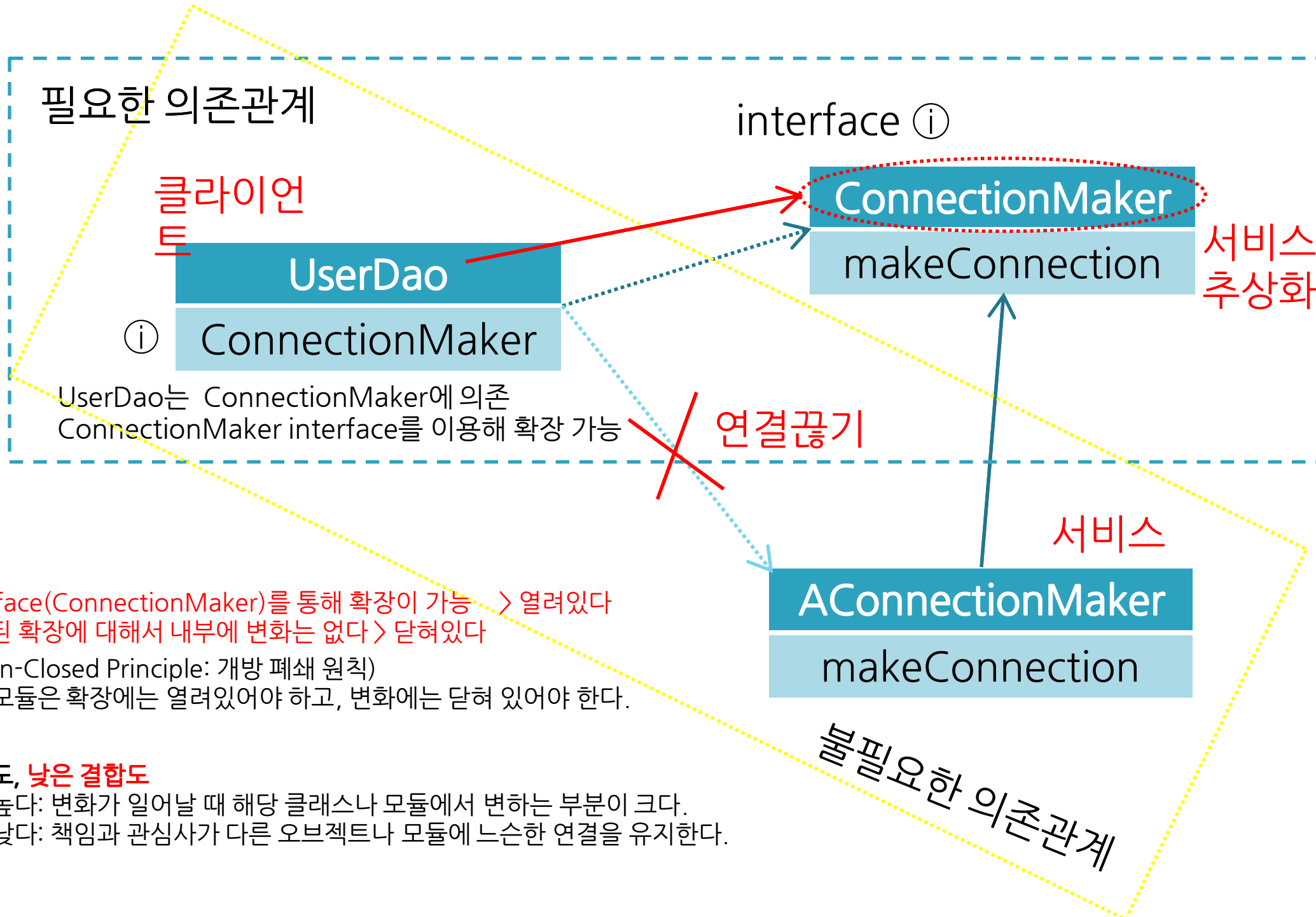

관심사 분리(Connection 분리)
UserDao class에서 어떠한 연결 클래스가 만들어지는지 알고 있다. >> 실패한 패턴
따라서 인터페이스를 이용해 어떤한 클래스가 만들어지는지 몰라야한다.

결론적으로 명시적인 클래스를 가지고 있으면 안된다. >> 클래스의 이름을 알고 있으면 안된다.

```java
public abstract class UserDao {

    private SimpleConnectionMaker simpleConnectionMaker;

    public UserDao() {
        simpleConnectionMaker = new SimpleConnectionMaker();
    }
    //관심사의 분리는 되었지만
```

```java
public class SimpleConnectionMaker {
    public Connection makeConnection() throws ClassNotFoundException, SQLException {
        final String DB_URL = "jdbc:oracle:thin:@localhost:1521:orcl";
        final String DB_USER = "hsj";
        final String DB_PASSWORD = "hsj";

        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection c = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
        return c;
    }
}
```

# exam04
# 관계설정책임의 분리




interface(ConnectionMaker)를 통해 확장이 가능 > 열려있다
개방된 확장에 대해서 내부에 변화는 없다 > 닫혀있다

**OCP**(Open-Closed Principle: 개방 폐쇄 원칙)
클래스와 모듈은 확장에는 열려있어야 하고, 변화에는 닫혀 있어야 한다.

**높은 응집도, 낮은 결합도**
응집도가 높다: 변화가 일어날 때 해당 클래스나 모듈에서 변하는 부분이 크다.
결합도가 낮다: 책임과 관심사가 다른 오브젝트나 모듈에 느슨한 연결을 유지한다.

**전략패턴**
자신의 기능 맥락(Context)에서 필요에 따라 변경이 필요한 알고리즘 인터페이스를 통째로 외부로 분리하여 인터페이스를 구현한 구체적인 클래스를 필요에 따라서 바꿔서 사용할 수 있게 해주는 디자인 패턴 결과적으로, 독립적인 책임으로 분리가 가능하다.

스프링은 OCP, 높은 응집도, 낮은 결합도, 전략패턴에 나타난 장점을 개발자들이 활용