1. POJO(Plain Old Java Object)

: 특정 [자바](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%B0%94_(%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D_%EC%96%B8%EC%96%B4)) 모델이나 기능, [프레임워크](https://ko.wikipedia.org/wiki/%ED%94%84%EB%A0%88%EC%9E%84%EC%9B%8C%ED%81%AC) 등을 따르지 않은 자바 오브젝트를 지칭하는 말로 사용되었다. [스프링 프레임워크](https://ko.wikipedia.org/wiki/%EC%8A%A4%ED%94%84%EB%A7%81_%ED%94%84%EB%A0%88%EC%9E%84%EC%9B%8C%ED%81%AC)는 POJO 방식의 프레임워크이다.

1. DI(Dependency Injection)

: [프로그래밍](https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D)에서 구성요소간의 의존 관계가 [소스코드](https://ko.wikipedia.org/wiki/%EC%86%8C%EC%8A%A4%EC%BD%94%EB%93%9C) 내부가 아닌 외부의 설정파일 등을 통해 정의되게 하는 [디자인 패턴](https://ko.wikipedia.org/wiki/%EB%94%94%EC%9E%90%EC%9D%B8_%ED%8C%A8%ED%84%B4) 중의 하나이다.

* 1. DI의 장점
* 의존 관계 설정이 [컴파일](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%8C%8C%EC%9D%BC)시가 아닌 [실행](https://ko.wikipedia.org/wiki/%EC%8B%A4%ED%96%89)시에 이루어져 모듈들간의 [결합도](https://ko.wikipedia.org/wiki/%EA%B2%B0%ED%95%A9%EB%8F%84)를 낮출 수 있다.
* [코드 재사용](https://ko.wikipedia.org/wiki/%EC%BD%94%EB%93%9C_%EC%9E%AC%EC%82%AC%EC%9A%A9)을 높여서 작성된 모듈을 여러 곳에서 소스코드의 수정 없이 사용할 수 있다.
* [모의 객체](https://ko.wikipedia.org/wiki/%EB%AA%A8%EC%9D%98_%EA%B0%9D%EC%B2%B4) 등을 이용한 [단위 테스트](https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%9B_%ED%85%8C%EC%8A%A4%ED%8A%B8)의 편의성을 높여준다.
  1. DI 주입
* Context 를 통한 주입 (datasource 예)

|  |
| --- |
| <bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>  <!-- JDBC Driver 클래스 명 설정 -->  <property name=*"driverClassName"* value=*"Altibase.jdbc.driver.AltibaseDriver"* />  <!-- connection url -->  <property name=*"url"* value=*"jdbc:Altibase://ip:port/mydb"* />  <!-- DB 사용자 계정 설정 -->  <property name=*"username"* value=*"user"* />  <!-- DB 사용자 패스워드 설정 -->  <property name=*"password"* value=*"pass"* />  </bean> |

* Setter를 통한 주입

UserManager의 User정보를 MakeUser class에서 생성하여 주입한다.

User.java

|  |
| --- |
| **public** **class** User {  **private** String userId;  **private** String userName;  **public** String getUserId() {  **return** userId;  }  **public** **void** setUserId(String userId) {  **this**.userId = userId;  }  **public** String getUserName() {  **return** userName;  }  **public** **void** setUserName(String userName) {  **this**.userName = userName;  }  } |

UserManage.java

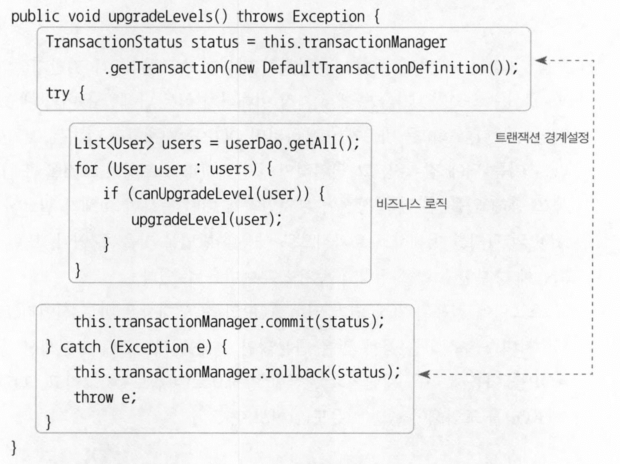
|  |
| --- |
| **public** **class** UserManager {  **private** User user;  **public** **void** setUser(User user) {  **this**.user = user;  }  **public** User getUser() {  **return** user;  }  } |

MakeUser.java

|  |
| --- |
| **public** **class** MakeUser {  **public** **static** **void** main(String[] args) {  UserManager um = **new** UserManager();    User user = **new** User();  user.setUserId("ajh3109");  user.setUserName("ajh");  //setter를 통한 user 삽입  um.setUser(user);    }  } |

1. AOP(Aspect Oriented Programming)

* 그림의 비즈니스 로직과 트랜잭션 경계처럼 원하는 처리는 비즈니스 로직 처리이나, 비즈니스 로직을 처리하기 위한 전/후 처리에 대한 로깅, 시큐리티, 트랜젝션 로직 등을 분리한다.



위 예제는 토비의 스프링 책에서 발췌하였으며, 해당 부분의 설명은 차후 심화 학습 시 열거하고자 한다.