--종속객체주입 (DI)

DI는 애플리케이션 객체 상호 간의 결합도를 낮춰주는 방법이다. 다음의 코드를 보자

```
public class A_Knight implements Knight{
        private Go_Quest quest;
        public A_Knight(){
                quest = new Go_Quest();
        }
(생략)
```

위의 A_Knight는 생성자 안에 자신의 원정(quest)인 Go_Quest를 생성한다. 이것은 A_Knight와 Go_Quest 사이에 강한 결합도를 만들어 내는데, 이러한 강한 결합도는 A기사가 다른 Quest를 수행할 수 없게 만든다.

다음의 코드는 DI를 적용시킨 결과물이다.

```
public class A_Knight implements Knight{
        private Quest quest;
        public A_Knight(Quest quest){
                this.quest = quest;
        }
(생략)
```

이 경우 A_Knight 생성자에 Quest를 주입받는 경우인데, 여기서 Quest는 인터페이스이다. 그러므로 Quest를 상속 받는 모든 클래스를 A_Knight가 수행 가능하게 된다(A_Knight에 Quest를 상속받는 모든 클래스를 주입 가능).

여기서 요점은 A_Knight는 Quest의 특정 객체에 결합되지 않는다는것이며, DI를 사용하면 객체간 결합도를 낮출 수 있다는 것이다.

-- 와이어링

이제 어떻게 Quest에 특정 Quest를 설정하는지 알아보자. 애플리케이션 컴포넌트 간의 관계를 정하는 것을 와이어링(wiring) 이라고 하는데, 스프링에서 와이어링을 하는 가장 일반적인 방법은 XML을 이용하는 것이다. 이는 2장에서 자세히 알아본다.

--실행하기

스프링에서 application context 는 빈에 관한 정의들을 바탕으로 빈들을 엮어준다.

```
ApplicationContext context = new ClassPathXmlApplicationContext("Knights.xml");

Knight knight = (Knight) context.getBean("knight");
```

위 코드는 스프링 컨테이너가 설정 되어 있는 Knights.xml 파일을 로드하여 빈 이름이 knight인 컨테이너를 로드 하고 그것을 대입하는 것을 보여준다. 이 클래스는 knight가 어떤 Quest를 가지고 있는지 알 수 없지만 xml 파일에는 정확히 나와있다.

-- 애스펙트 적용(AOP)

AOP는 빈에(<bean><bean/>사이에) 설정 하여 빈의 특정 함수의 시작과 끝에 자동으로 수행되어야 할 기능들을 설정 할 수 있다. 대표적 예로 DB connection 이 필요한 데이터베이스 조회 관련 코드에는 DB connection을 위한 코드들이 중복되어 사용된다. 이러한 귀찮은 작업들을 AOP를 이용하여 자동 구현 시킬 수 있다.