**GridFS 배움(MongoDB에서 파일 업,다운로드)**

* FileUp, Download

mongoDB는 마구잡이로(실시간), 중요하고 소중한 데이터면 RDBMS

예) mongoDB: 카톡 채팅창에 마구잡이로 사진 올릴때. 언젠간 사라질 데이터라면. 언젠간 삭제될 임시.

RDBMS : 영구적인 사진.

**Spring에 대하여**

* program을 짜는 방법.
* rule (IOC/ DI) : 90% / 10% 웹구성 => + AOP(Proxy 패턴)
* object관리가 중요 함.
* IOC : 필요한 것 등록.
* 사용자 정의 프레임워크 만들수 O : 스프링을 기반을 한 frame work를 또 만들어 낼 수 있음.
* EJB => Enterprise Java Bean

: java에서 제공하는 기업용 솔루션 프레임워크

단점 -> 필요 없는 부분도 같이 포함 됨. 메일링만 쓰려고해도 sms 포함.

EJB쓰려면 포함 됨.

불필요한 요소가 많다.

그래서 등장한 것이 Spring Framework.

* POJO(Plain Old Java Object) Program : 너가 원하는거 만들어서 사용해라. 간단한 자바 객체. 내가 만들어 놓은 클래스. 원하는 기능들만 추가해서 꺼내서 사용하는 방식.
* **템플릿 메소드 패턴** :

**패턴, 메서드**

- 패턴 : ~~게 하라.규칙 같은 것

|  |  |  |
| --- | --- | --- |
|  | 템플릿 메서드 패턴 | 팩토리 메서드 패턴 |
| 공통점 | 자식에서 구현해야 함. (부모클래스에서는 추상화) | |
| 차이점 | 자식에서 반환값이 없음.  부모의 흐름이 포함됨 | 자식에서 반환값 있음.  부모의 흐름에 포함되지 않음. |

- 템플릿 메서드 패턴 : 부모클래스에서 기능들의 흐름을 구현. 작업실행에 대한 흐름이 부모클래스에 있음. 자식은 일련의 템플릿 메소드를 구현. 부모의 흐름에 끼어 있어야 함. 반환값x

- 팩토리 메서드 패턴 : 돌려주는 생성 객체가 있음.(단, 자식에서). 부모에서 흐름이 있을으도 없을수도 있음.

= > 자식에 구현되는 것이 있다. 그러므로 추상화.

* hook 메소드 : 추상화클래스에서 부모에서 이미 구현이 되어 있음. 자식에서 오버라이드 해도되고 안해도 됨.

**OCP**

- Open-Closed Principle -> 개방 폐쇄 원칙

- 클래스와 모듈은 확장에는 열려있어야 하고, 변화에는 닫혀 있어야 한다.

확장 -> ConnectionMaker. UserDao는 ConnectionMaker를 통해 얼마든지 확장 가능. 인터페이스를 통해 어떠한 클래스로 확장.

변화에 닫힘 -> 확장에 따른내부의 변화가 없음(인터페이스).

→ 즉 인터페이스로 하면 변화는 낮고, 확장은 용이. 인터페이스로 여러개 확장 가능하고 인터페이스 자체는 바뀌지 않음. 반대로, 상속으로 하면 문제가 생김.

외부클래스에 대한 확장에 관한 것.

**높은 응집도, 낮은 결합도**

- 응집도가 높다 : 클래스에서 변화가 일어날 때 해당 클래스나 모듈에서 변하는 부분이 크다

-> 내부관련. 자기자신의 기능 추가나 삭제시 응집도가 높아짐. 모여있는 것. 해당하는 기능만 모여있다.. 해당 클래스에 필요한 것만 내부에 있음. 그클래스에서 사용할 기능들만 코드에 포함.

-결합도가 낮다 : 책임과 관심사가 다른 오브젝트나 모듈에 느슨하게 연결을 유지한다.

-> 다른오브젝트간의 인터페이스를 유지. ConnectinonMaker와 AConntectionnMaker와 인터페이스를 놔야. 강결합이 안됨. 의존도 낮게. 다른 클래스간의 관계를 느슨하게.

**전략패턴**

- 자신의 기능 맥락(Context)에서 필요에 따라 변경이 필요한 알고리즘을 인터페이스를

통째로 외부로 분리하여 인터페이스를 구현한 구체적인 클래스를 필요에 따라서

바꿔서 사용 할 수 있게 해주는 디자인 패턴. 결과적으로 독립적인 책임으로 분리가 가능

맥락-> 유저DAO를 만들면서 필요한 것. 생성자같은.

ex) 변경이 필요한 알고리즘 : 커넥션 메이커

구체적인 클래스 : A커넥션 메이커.

=> 즉, 생성자에 인터페이스가 있는 것은 전략패턴.(ex: UserDao 생성자에 인터페이스 있음)

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

**IOC & DI**

IOC : 제어의 역전. 제어는 객체를 만드는 것. 객체를 만드는 행위를 역전. 객체를 어디선가 만들어서 가져옴(팩토리: 객체를 만드는 곳. Object Factory). 객체를 만들어내는 곳이 팩토리로 감. new를 할 필요 없음. 제어를 스프링한테 넘김.

DI : 그것(IOC로부터 나온 것)을 필요한데다가 주입.