# 자바 리플랙션 잘 정리한 블로그

https://medium.com/msolo021015/%EC%9E%90%EB%B0%94-reflection%EC%9D%B4%EB%9E%80-ee71caf7eec5

인터페이스를 상속받은 새로운 클래스를 만들었을 때, 소스 코드 수정 없이 새로운 클래스를 적용할 수 있는 방법 : 동적로딩(Reflection)

자바의 Reflection은 JVM에서 실행되는 애플리케이션의 런타임 동작을 검사하거나 수정할 수 있는 기능이 필요한 프로그램에서 사용된다.

쉽게 말하면, 클래스의 구조를 개발자가 확인할 수 있고, 값을 가져오거나 메소드를 호출하는데 사용된다.

Reflection을 사용하는 기술을 나열하자면, 스프링 프레임워크, 대표적 ORM 기술인 하이버네이트, jackson라이브러리 등에서 사용된다.

Reflection을 사용해서 스프링에서는 런타임 시에 개발자가 등록한 빈을 애플리케이션에서 가져와 사용할 수 있게 된다.

장점 : 확장성이 좋다. 소스코드 수정 없이 런타임 동작 시에 다른 클래스를 생성하여 적용할 수 있다.

단점 : 성능 저하 - Reflection에는 동적으로 해석되는 유형이 포함되므로, 특정 JVM 최적화를 수행할 수 없다. 따라서 Reflection 작업이 비 Reflection 작업보다 성능이 떨어진다.

# 스프링 DI

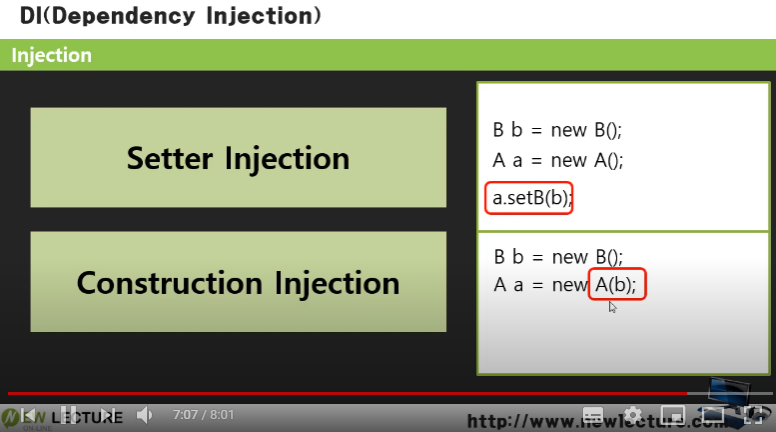
객체를 생성해주고 객체들을 조립해주는 능력

DI를 수행하기 위하여 IOC 컨테이너가 필요함

사용되는 용어

1. Di, 2. IoC 컨테이너

## 클래스의 Setter에 부품(다른 인스턴스)을 꽂는다 == dependency를 주입한다. 처럼 생각할 수 있음



스프링이 조립해주는 역할(dependency injection)을 수행해 준다.

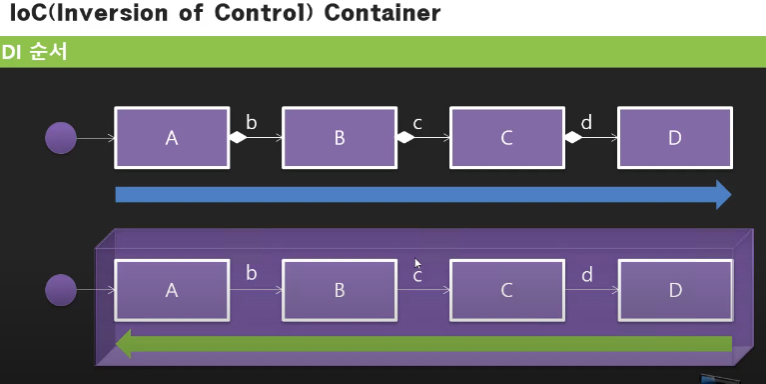
원하는 부품들이 어느 것들이 있는지, 어떤 부품 결합을 하기를 원하는 지만 설정하면 알아서 해준다.

단 이 객체를 조립하는데 IOC 컨테이너가 필요하다.

# .IoC 컨테이너는 객체를 생성 조립해주는 역할이 포함된 컨테이너입니다.

주문서에 입력되어 있는 내용대로 객체를 생성해서 담을 수 있는 그릇이 필요.

소프트웨어에서 그런 그릇들을 일반적으로 컨테이너 라고 한다.



결합 형태는 A>B>C>D 순으로 객체가 만들어지는데, IOC에서는 역순으로 D>C>B>A 순으로 객체를 생성하는 컨테이너 라고 해서 ioc 컨테이너라고 부른다.

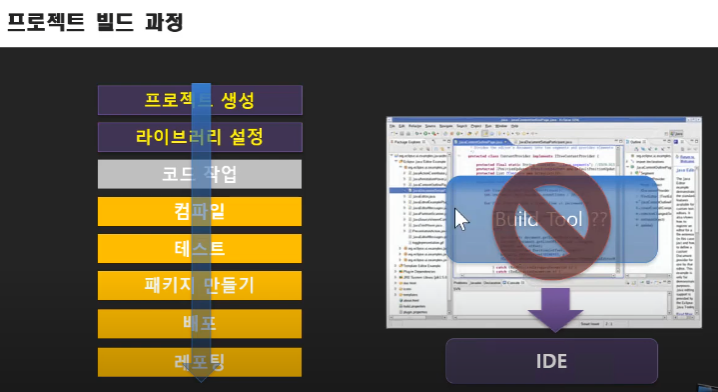
# 메이븐(maven)

<https://www.youtube.com/watch?v=VAp0n9DmeEA>

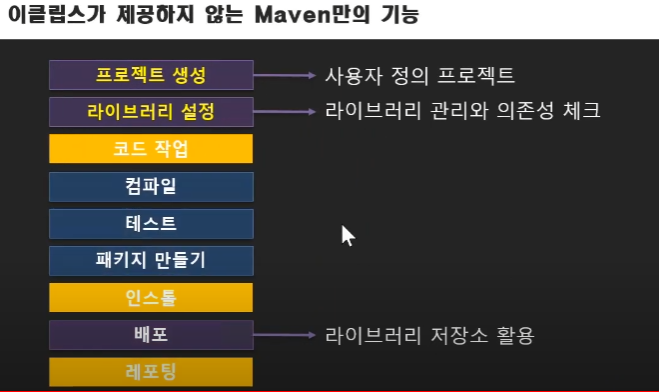
## 1강

Maven이란? 빌드할 때 사용하는 도구

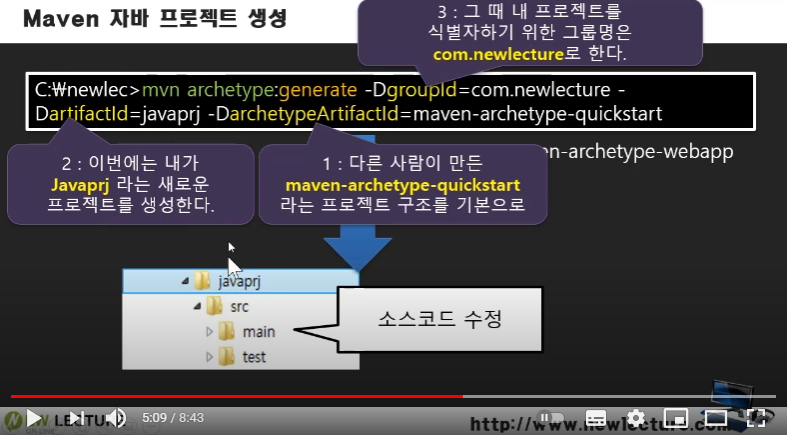
작업과정에서 반복되는 내용들을 효율적으로 개발할 수 있도록 도와주는 빌드도구(빌드 도와주는 역할 – 프로젝트 설정, 라이브러리 설정)





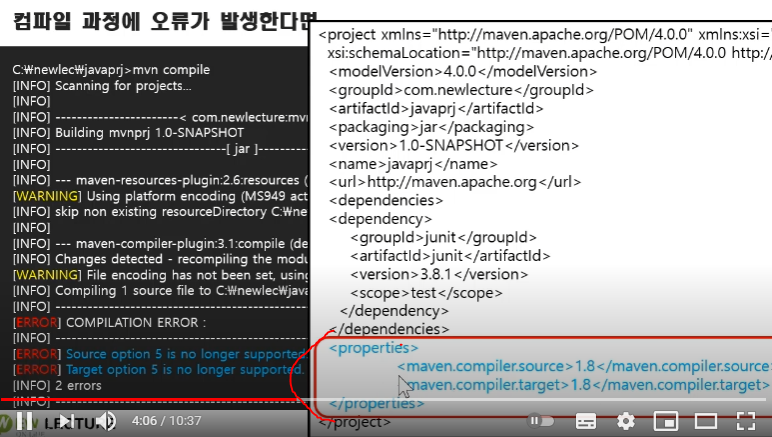


## 3강



## 4강 – 컴파일과 실행하기

Maven 프로젝트는 root 위치에 항상 pom.xml이 있는데, 프로젝트를 컴파일 하거나 실행을 할 때 항상 pom.xml이 있는 경로에서 mvn 명령어 사용해야 한다.

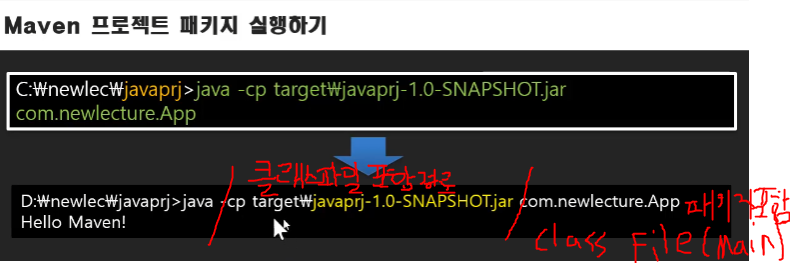


컴파일 & 실행할 JAVA 버전 <properties>에 저장 필요

Mvn compile 명령어로 컴파일 가능

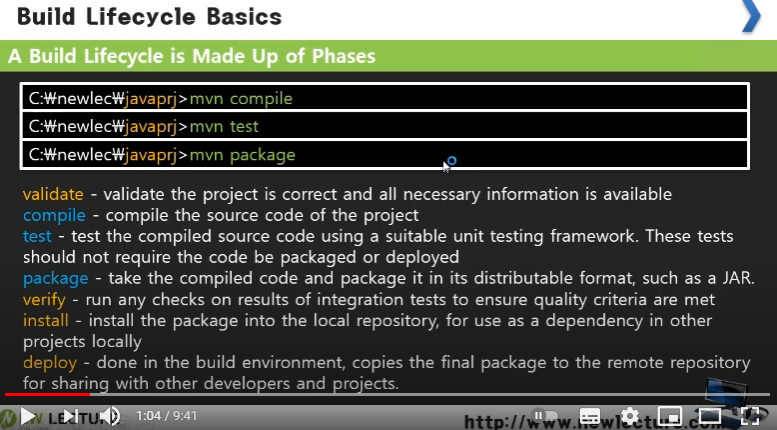
Mvn package 명령어로 Jar 파일 생성 가능

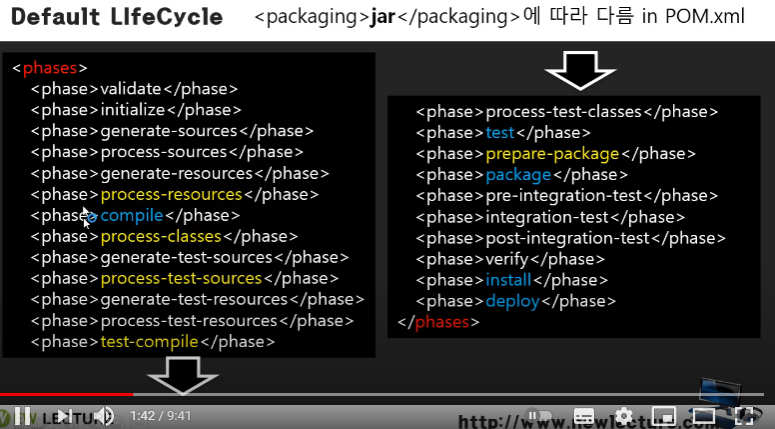




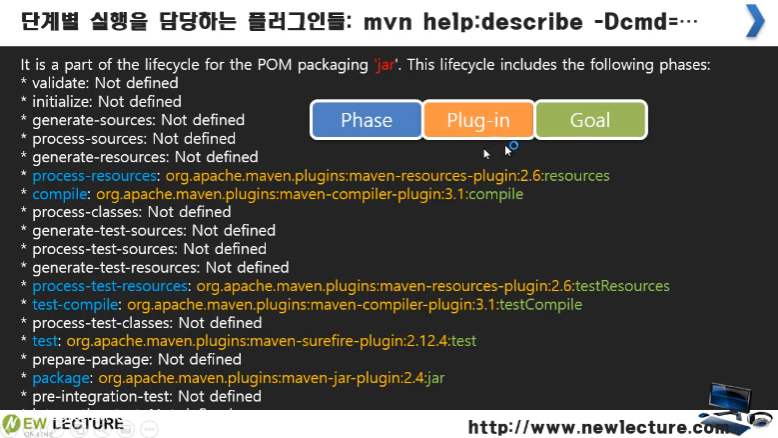
## 5강 – Build LifeCycle과 Phase 들

<https://www.youtube.com/watch?v=fQsTKKkZ6d8>



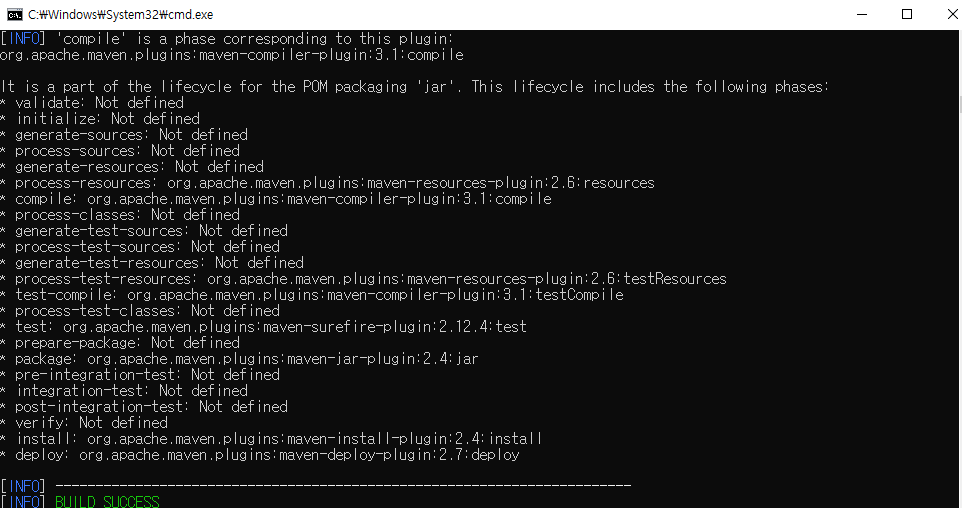


Phase는 수행 단계를 의미하고, ‘mvn 수행단계’ 으로 Maven 명령 실행할 수 있으며, 입력한 수행 단계 이전까지의 모든 명령 또한 일괄 실행된다.



특정 phase를 실행할 때 사용되는 프로그램을 플러그인이라 한다.

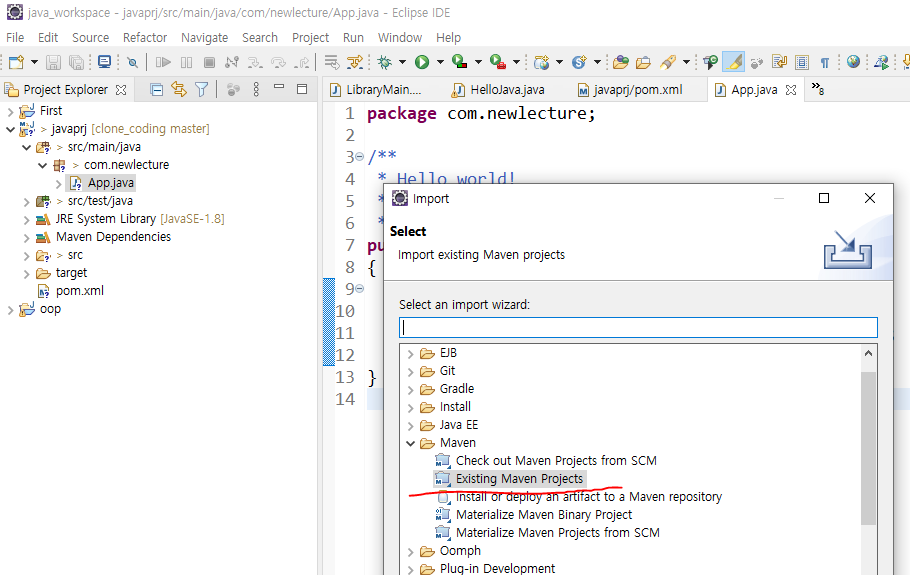
위 이미지에 지정된 플러그인들은 default으로 설정된 것이며, Not defined(플러그인이 지정 안된 단계)는 실행되지 않는다.



Mvn 명령 실행할 때, ‘mvn **help:describe** -Dcmd=compile’

처럼 입력하면 각 단계별 어떤 플러그인들이 설정되어 있는지 알 수 있다.

## 6강-메이븐 프로젝트 이클립스에서 로드하기

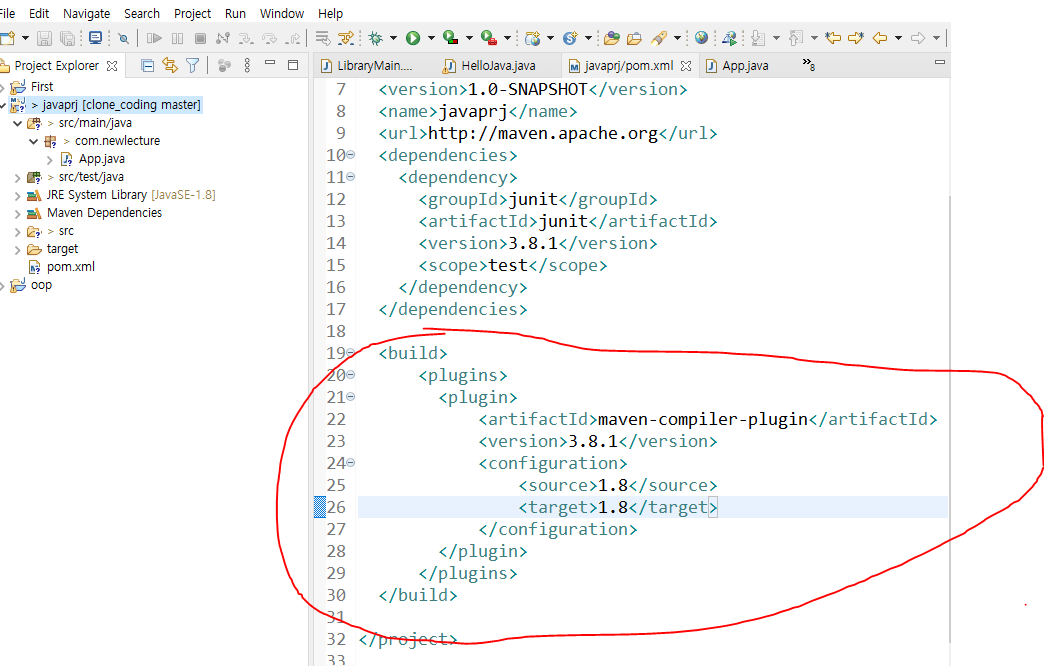


Import/maven 탭에서 기존 maven 프로젝트를 임포트하여 이클립스에서 사용할 수 있음.

## 7강 – 컴파일 플러그인으로 JDK 버전 변경하기

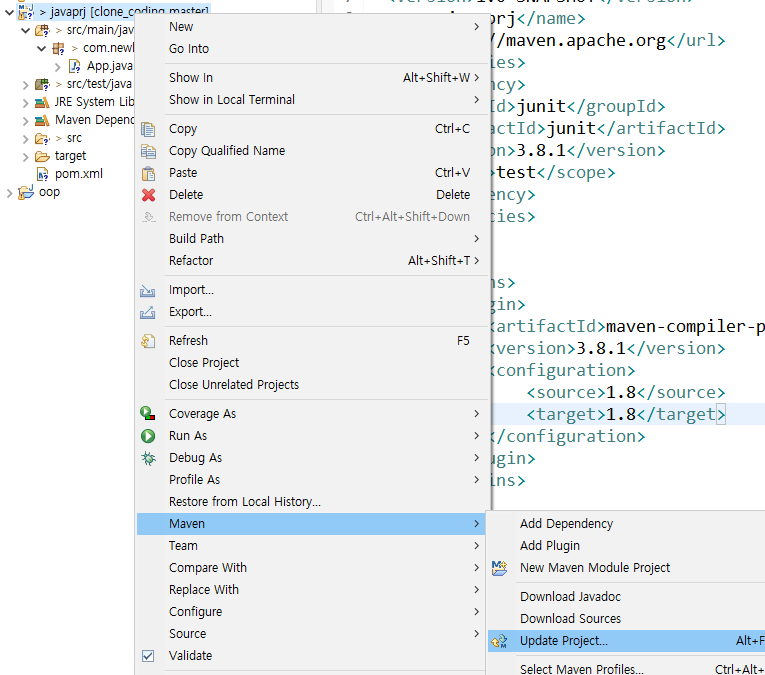
https://www.youtube.com/watch?v=\_6H0E49UGoM

기존 maven 프로젝트 생성할 때 ‘maven-archetype-quickstart’로 만들었는데, 저 버전의 기본 JDK가 1.5로 만들어져 있어서 아무런 설정을 하지 않으면 JDK 1.5로 설정된다.



<build></build> 태그 쓰고 build 과정에서 사용하는 프로그램의 정보를 입력할 수 있다.

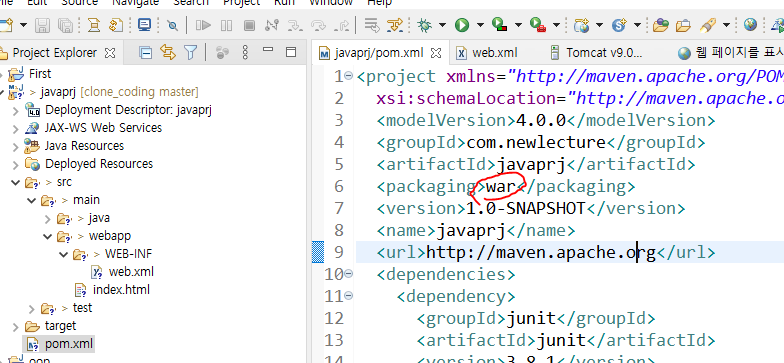
위에서 버전정보 입력하여 plugin을 재정의할 수 있다.



Pom.xml 파일 변경시키면 프로젝트우클릭/maven/update project 눌러서 항상 변경사항 업데이트 해야 함

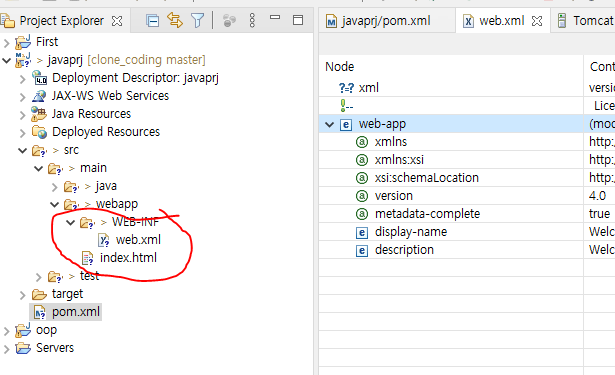
## 8강 – 웹프로젝트로 변경하기

Maven의 장점이 project 모델을 jar에서 war로 바꾸면 자바프로젝트에서 웹프로젝트가 만들어 진다. 자바->웹 프로젝트로 바꾸기 위한 작업을 maven이 알아서 해준다.



처럼 ‘war’로 바꾸면 된다.

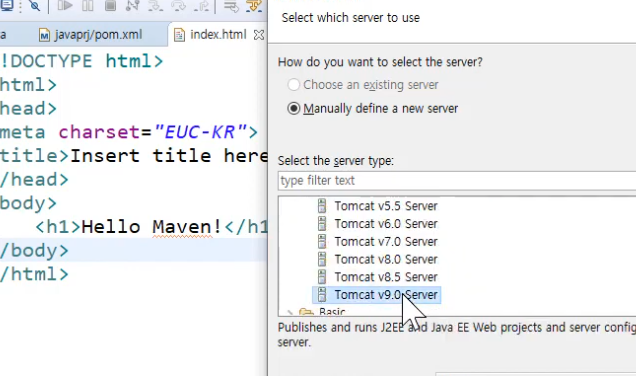
그리고 프로젝트 우클릭 후 ‘maven/project update’ 누르면 웹 프로젝트 형태로 변화한 것 확인 가능하다.

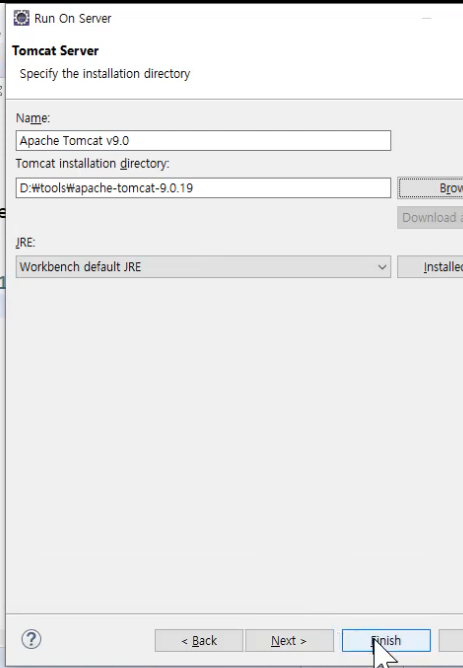


webapp폴더 하위에 WEB-INF 루트 폴더 만들고, 톰캣 WEB-INF/ 하위에 있는 web.xml 파일 복사해서 이클립스 WEB-INF 폴더 아래에 복붙한다.

그리고 샘플 index.html 만든다.

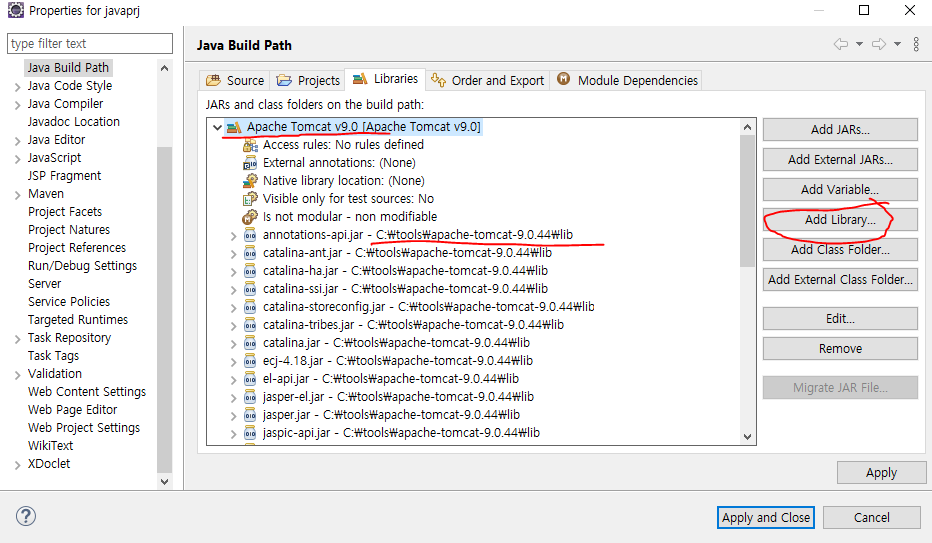
Ctrl+F11을 누르면 실행하기 위한 was를 설치하라고 나온다.





톰캣 설치한 홈 디렉토리 설정 후 finish 누르면 서버 설치되고 실행된다.

## 9강 – 서블릿/JSP 라이브러리 설정하기

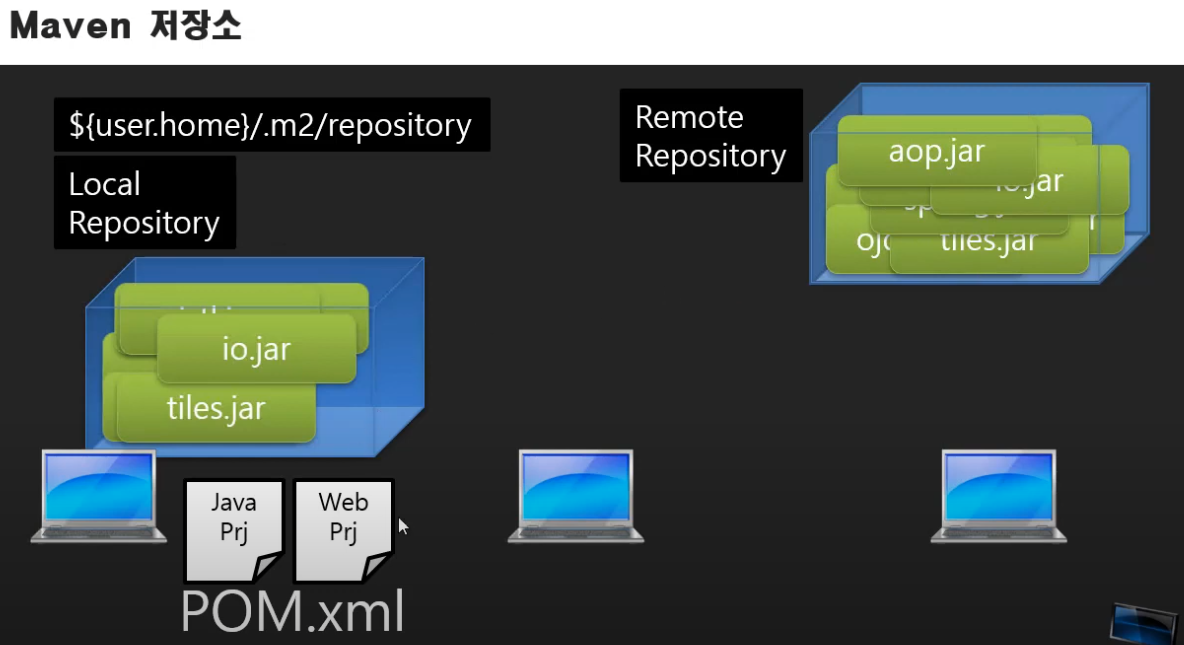


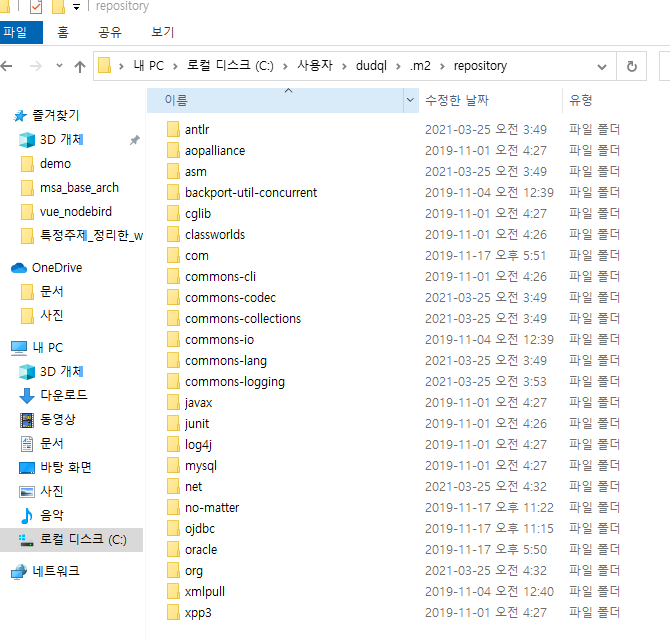
저렇게 로컬PC에 설치되어 있는 라이브러리를 import하면 라이브러리들의 경로가 특정 PC의 절대주소로 되어 있기 때문에 다른 PC에서 프로젝트를 받아서 실행하려고 하면 라이브러리가 없거나 경로가 맞지 않아 실행할 수 없게 되는 불편함이 있다.

### Maven을 사용하면 아래처럼 원격 Remote Repository에서 필요한 라이브러리가 있으면 설정하면 알아서 개별 ${user.home}/.m2/repository에 다운로드 받아준다.

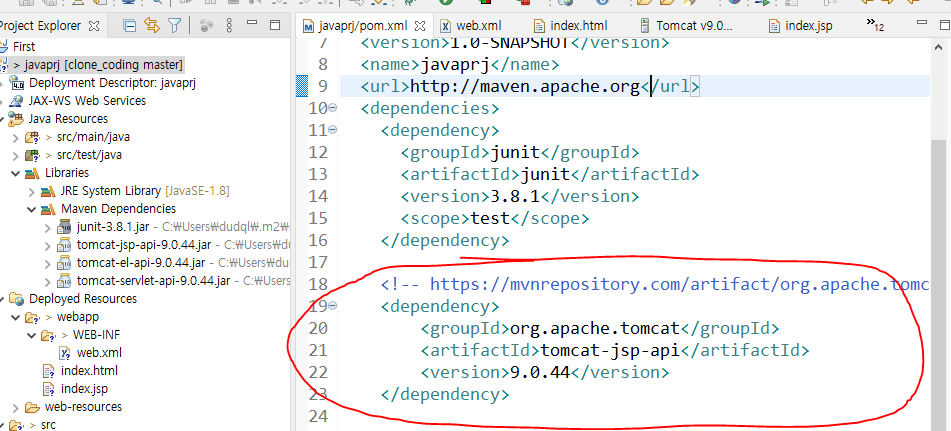
그래서 프로그램 빌드 과정 중 프로젝트 설정, 라이브러리 설정, 배포(라이브러리 저장) 부분에서 maven을 사용하면 아주 편리하다.

또한 그 라이브러리를 사용하기 위하여 필요한 라이브러리들까지 한번에 추가를 해준다.

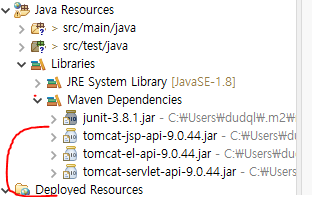




‘${user.home}/.m2/repository’ 경로에 Maven Remote Repository에서 다운받은 jar들을 저장하여 사용한다.



Pom.xml에 라이브러리 정보를 입력하여 설정할 수 있다.

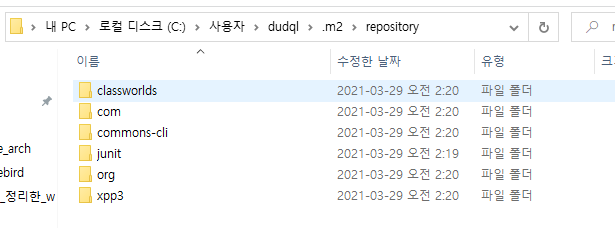


MAVEN dependencies 밑에 설정한 라이브러리와, 그 라이브러리를 사용하기 위하여 필요한 라이브러리들까지 한번에 추가된 것을 볼 수 있다.

## 10강 – 라이브러리 오류 문제

라이브러리 오류가 생기는 경우

1. 최초 다운로드 받다가 끊겨서 전부 못 받는 경우
2. M2파일 안의 라이브러리들이 손상된 경우

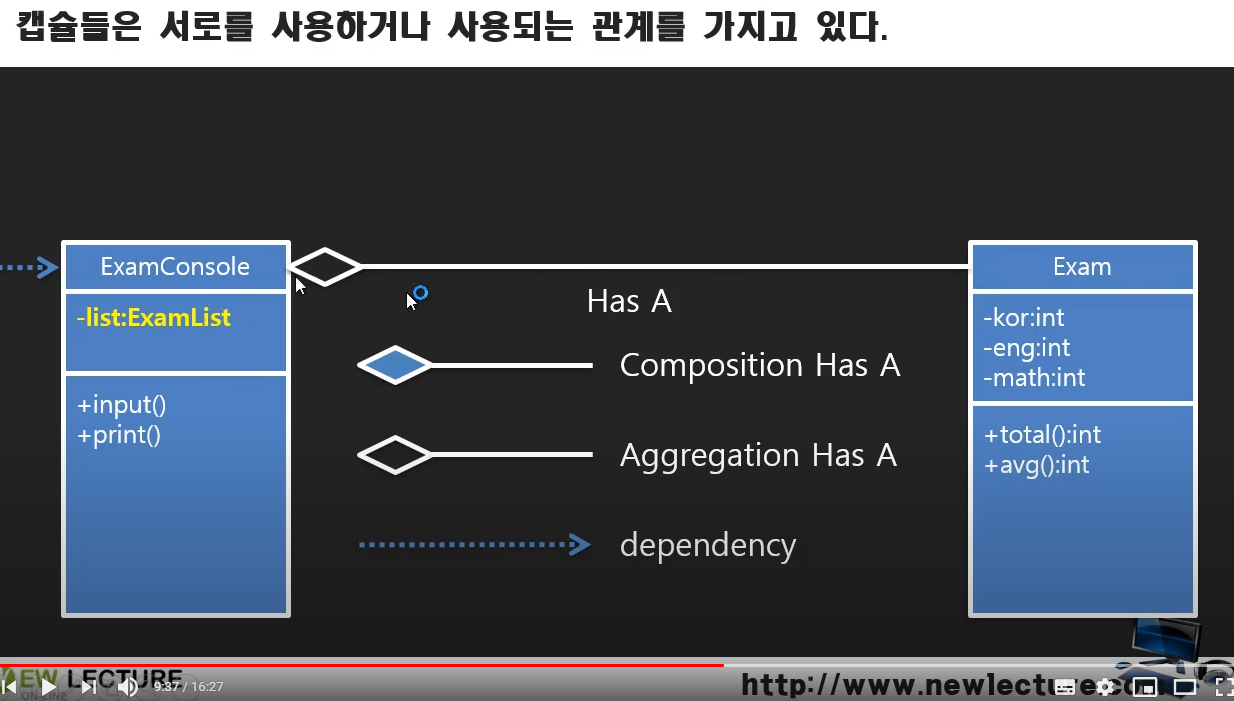


그럴경우 이클립스 종료하고 repository 들어와서 어떤 라이브러리에 문제가 있는지 모르니 전부 삭제하고 이클립스 다시 켜면 라이브러리 전부 새로 다운받게되어 문제 해결할 수 있다.

# 서블릿/JSP 강의

# 자바 객체지향 강의

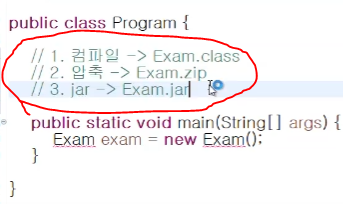
## 12강 Has A 상속



Composition – 일체형이라 생각하면 됨. 부품에 해당하는 객체가 그것을 소유한 객체와 동일한 시점에 생성되고 동일한 시점에 사라지는 관계를 composition 관계라고 한다.

Aggregation – composition과 차이점은 동일한 생명주기를 가지지 않는다는 점이다.

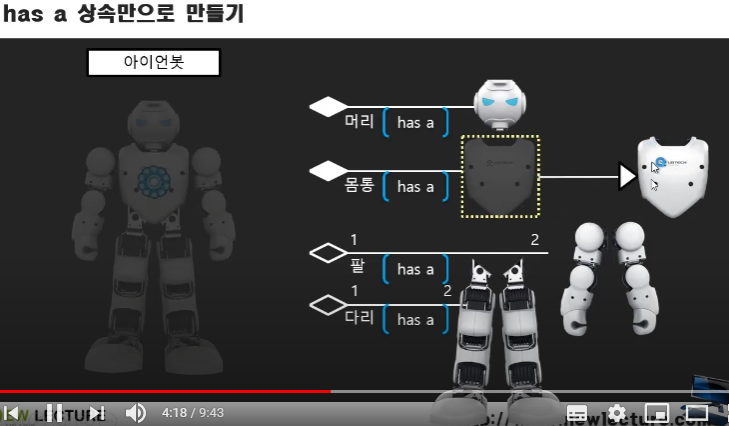
## 13강 코드 재사용이란?



외부프로젝트에 import하여 사용하기 위한 배포파일 만드는 과정

1. 컴파일 하여 class 파일 생성 -> exam.class
2. 압축 -> exam.zip
3. Jar -> exam.jar

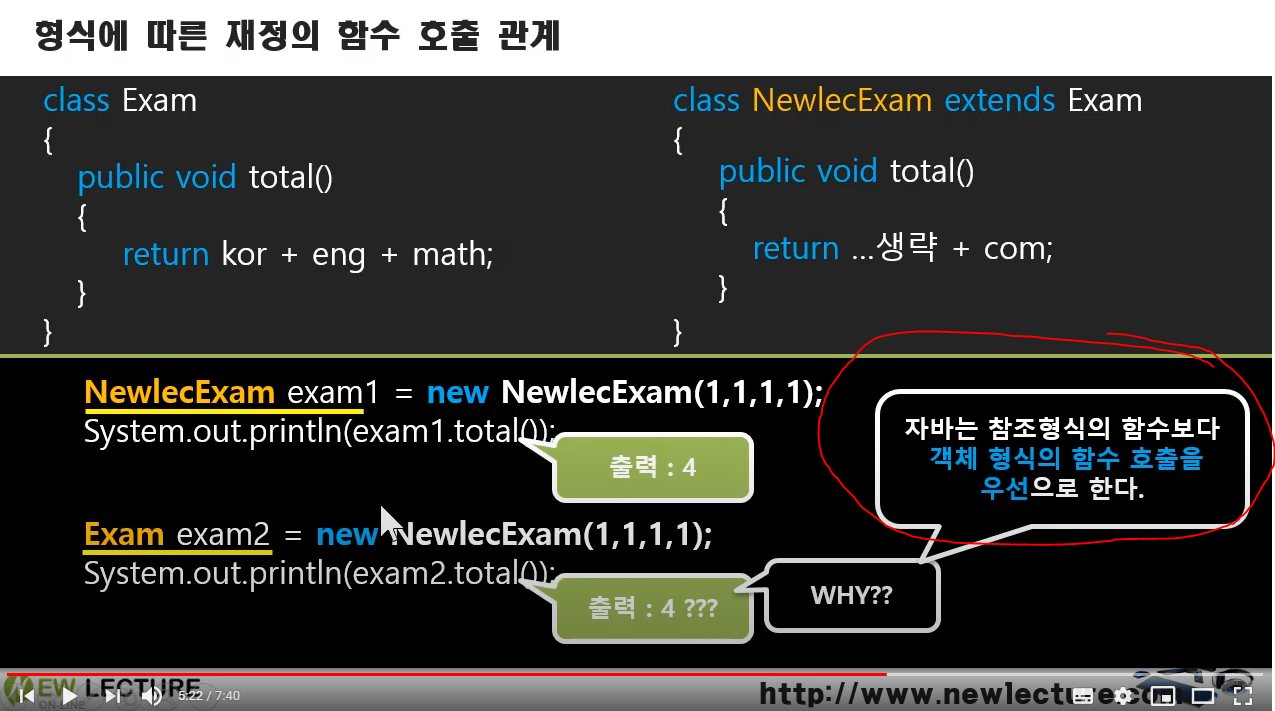
## 14강 IS A 상속



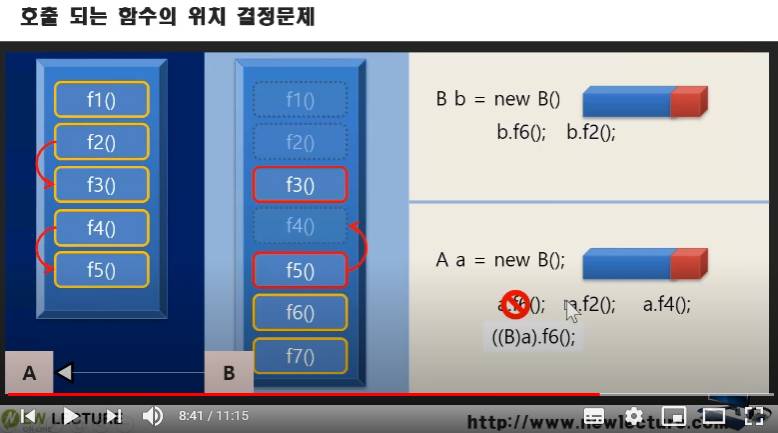
부품으로 가져다 쓰는 상속 -> has a 상속

틀로써 상속 받아서 사용 -> is a 상속

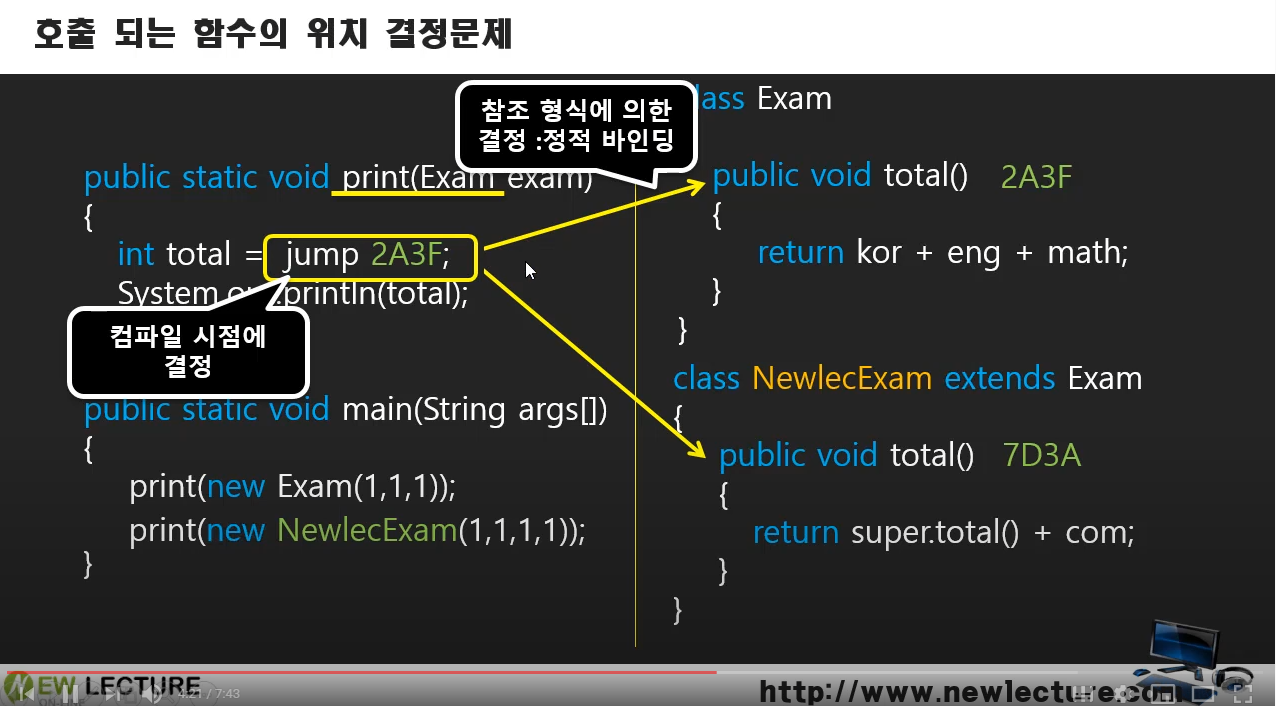
## 18강 참조형식과 호출되는 메소드의 관계



## 19강 메소드의 관계 연습하기



## 21 – 메소드 동적 바인딩



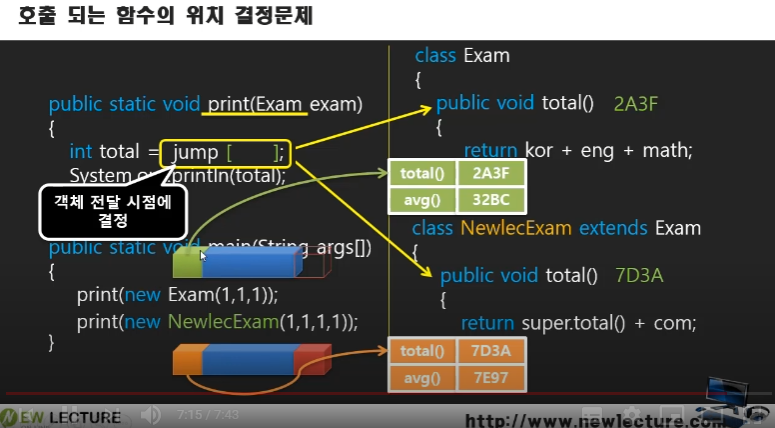
C++은 컴파일시에 어떤 메서드를 참조할 것인지 결정이 된다. (=정적바인딩)

### 하지만 java에서는 어떤 메서드를 참조할 지 컴파일시에 결정되지 않는다.

전달되는 객체에 따라서 실행되는 메서드가 달라진다. (=동적바인딩)

### 참조형식이 가지고 있는 형식에 한해서 전달받은 객체의 메서드를 호출할 수 있다.

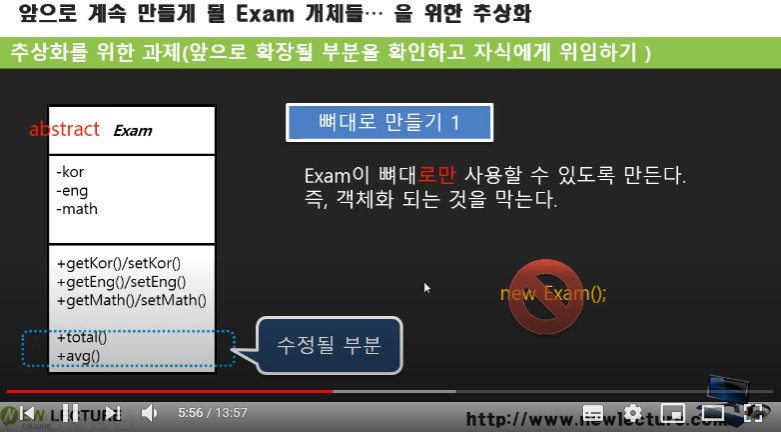
동적 바인딩을 지원하는 객체(자바는 모든 객체가 동적 바인딩을 지원한다.)들은 생성 시 본인 메서드의 테이블을 정보를(데이터 외 4byte) 가지고, 함수 테이블 정보가 함께 전달되어 전달되는 객체에 따라 호출되는 함수가 달라질 수 있다.

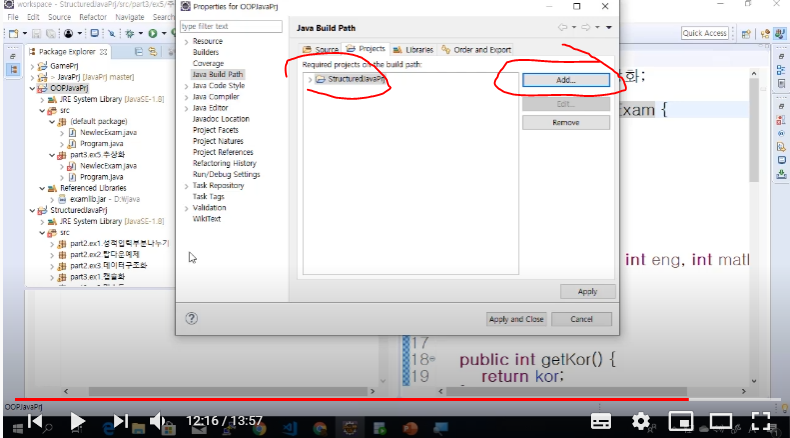


## 23강 – 추상클래스 만들기

### 추상 클래스는 공통 자료형이다,

공통기능을 제공하는 클래스여서 실체화(new) 되면 안되는 경우에는 추상클래스로 생성한다. Abstract class

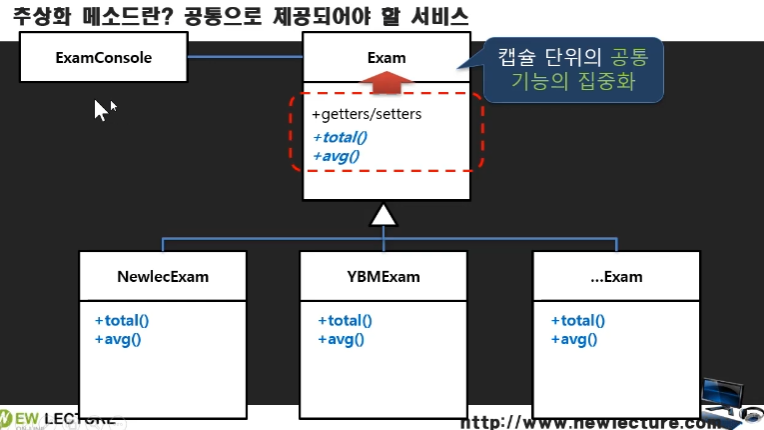




### 다른 프로젝트의 클래스 사용이 필요할 때, java build path/project 탭에서 현재 내가 가지고 있는 프로젝트 중에서 내가 사용할 프로젝트를 참고할 수 있다.

## 24강 – 추상메소드

### 추상 메소드란? 공통으로 제공되어야 하는 서비스

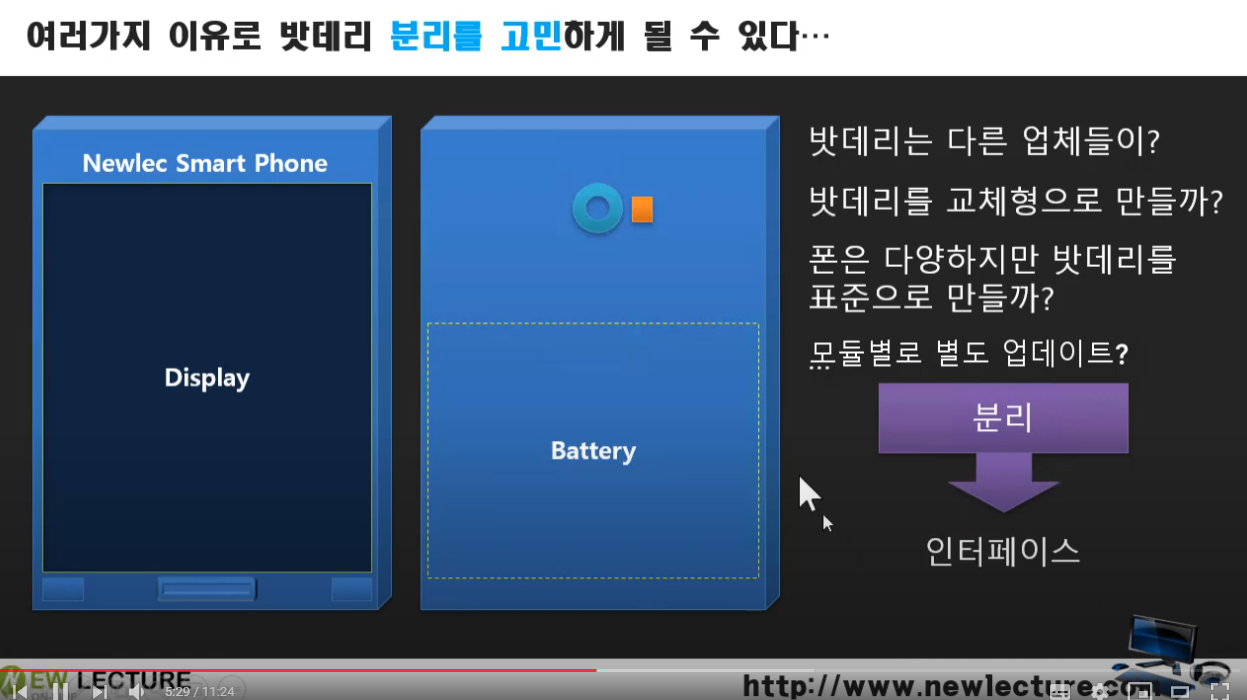


## 29장 – 코드 분리와 인터페이스

추상화 라는 것은 집중화를 의미하는 것이다. (코드가 집중화 되거나 데이터가 집중화가 되거나)

인터페이스는 집중화가 아니다.

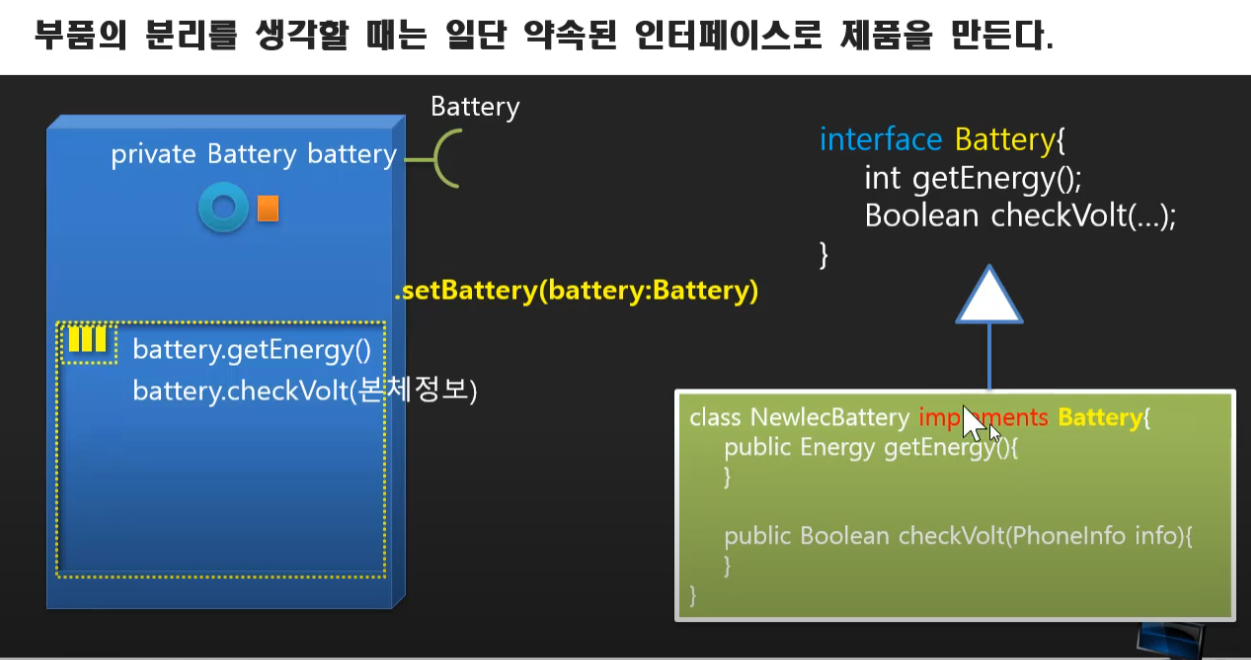
### 인터페이스는 코드 분리와 관련이 있다.



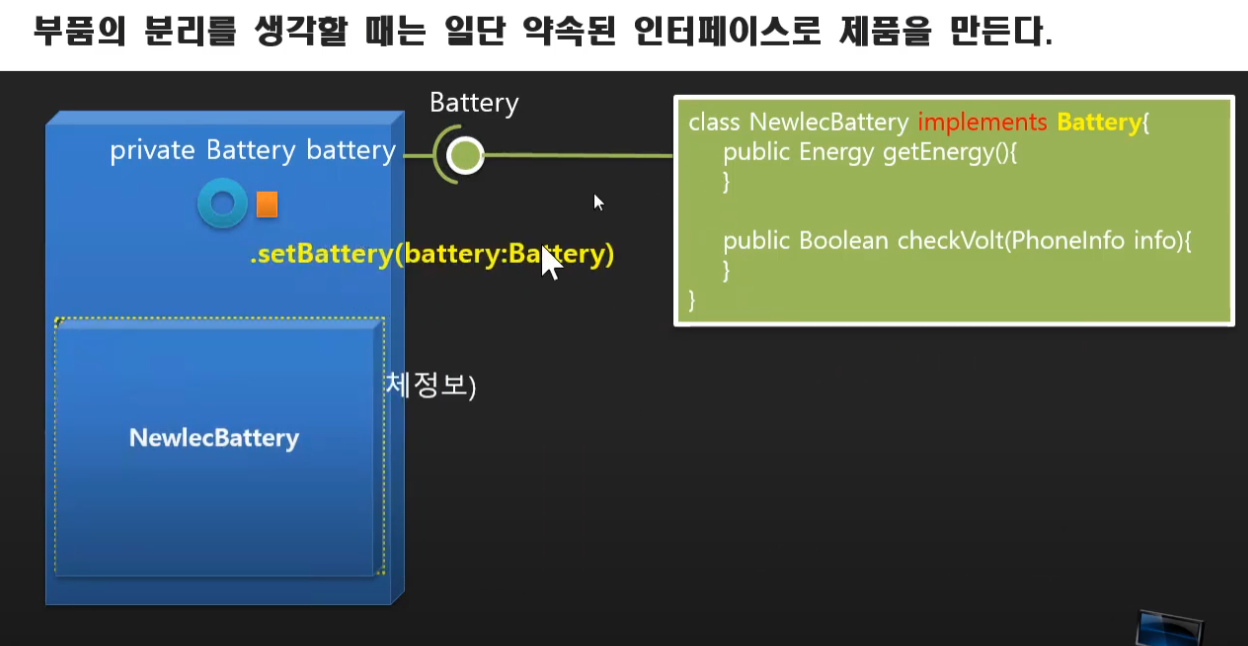
위 영상 예시에서 배터리를 만들어서 아무런 약속(규약)도 없이 분리했다 연결하면 되는 것인가? 당연히 안된다.

### 따라서 부품의 연결을 위한 규칙이나 규약이 있어야 하는데 이것이 바로 인터페이스이다.

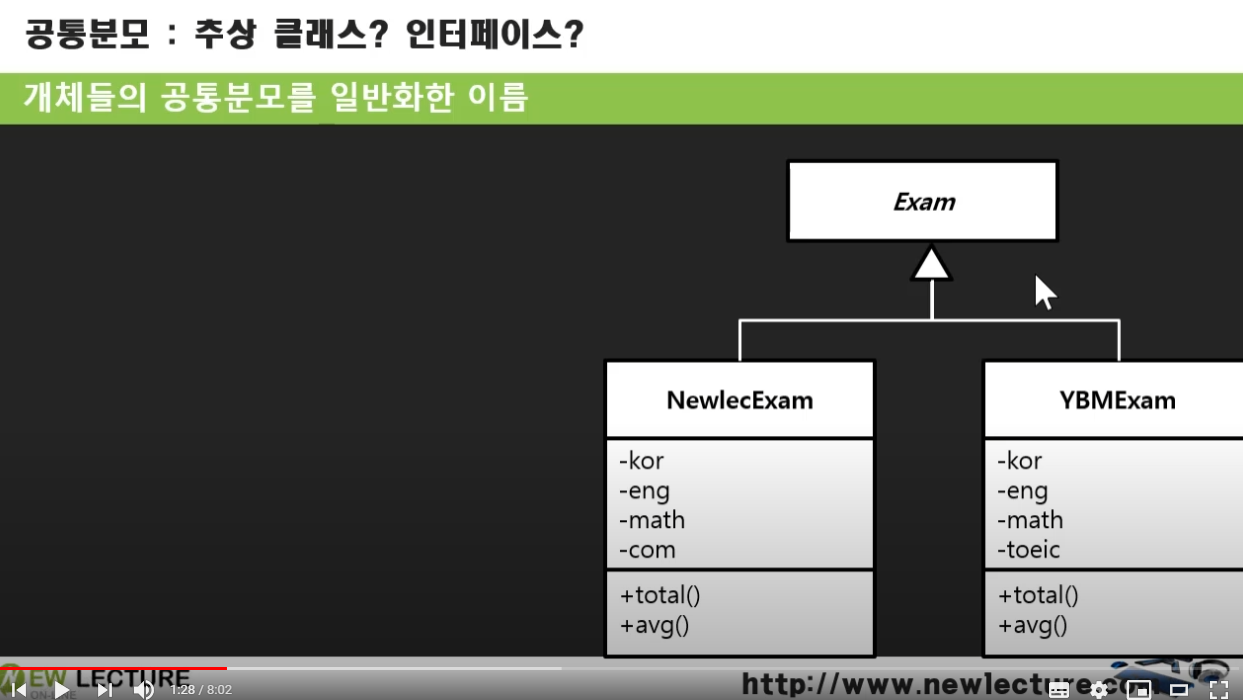
### 부품의 분리를 생각할 때는 일단 약속된 인터페이스로 제품을 만든다.

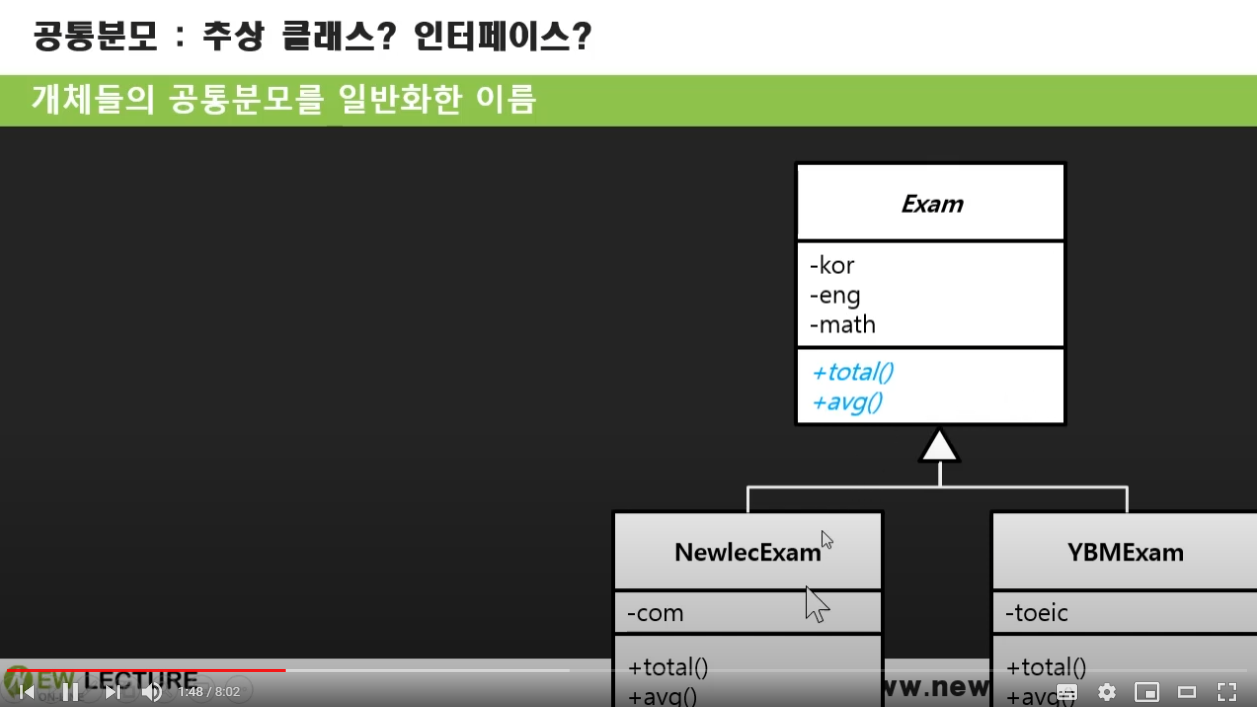


인터페이스 상속을 받은 battery 구현체를 구현하여, 아래처럼 battery 인터페이스를 이용해서 다른 객체에서 사용할 수 있게 된다.



## 30장 – 추상클래스와 인터페이스의 차이

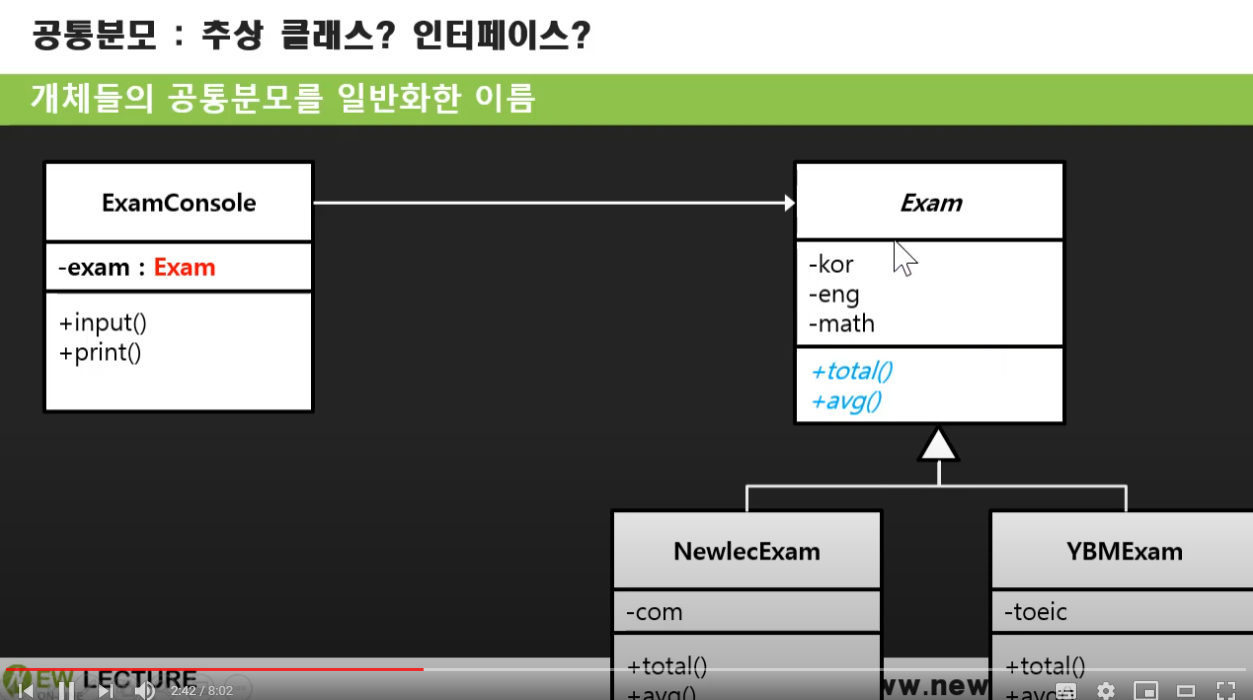


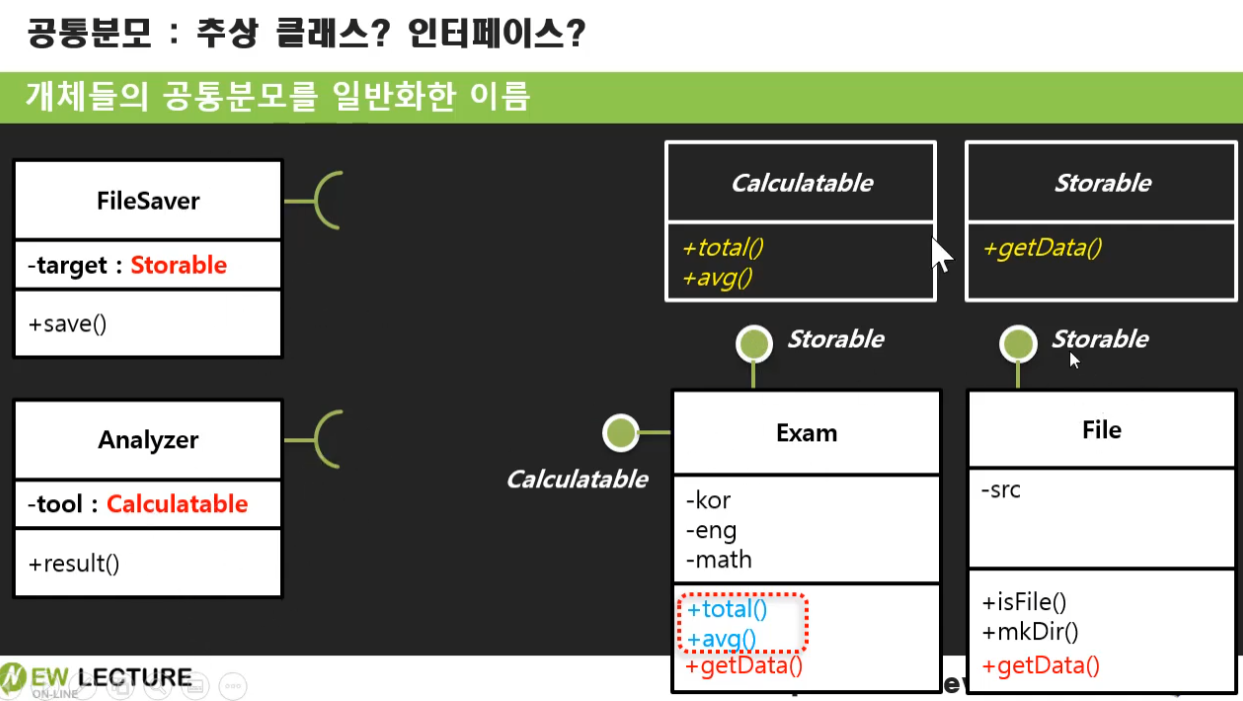


추상화는 기능적으로 반드시 같이 가지고 있어야 할 공통 분모의 어떤 기능/서비스가 있다고 하면 그것을 집중화 하거나 강제화 할 수 있다.

추상클래스 만드는 경우에 이 추상클래스를 만들어 사용하는 것이 누군지는 관심이 없다.

### 중요한거는 이 녀석이 가지고 있는 공통사항을 집중화 하겠다는 것이 목표이다.





인터페이스, 추상 클래스를 사용하는 것의 장점은, 이 클래스들을 사용하는 다른 클래스에서 공통 분모에 해당하는 참조변수만으로 인터페이스를 상속받은 다양한 구현체들을 받아서 사용할 수 있다는 점이다. (다형성)

Ex) 인터페이스나 추상클래스가 없다면 동일 기능을 하는 다른 클래스들의 구현체를 받을 때(ex- OO배터리, XX배터리 등) OO배터리, XX배터리 의 참조변수로만 구현체를 받을 수 있어서 코드 개발이나 유지보수에 번거로움이 큰데, 다형성을 활용하면 ‘배터리’ 참조변수로 OO배터리, XX배터리 구현체를 다 받을 수 있고, 그 구현체가 가진 기능을 사용할 수 있다.