Maven 표준 프로젝트 디렉토리 구조

*Maven build tool

프로젝트
▼
App 만들기
▼

소스(src) 그림, 동영상, 사진, 소리 설정 정보 ...etc		파일
▼		▲
체계적으로 관리	▸ 정리정돈	▸ 디렉토리에 분류, 배치

Maven 표준 프로젝트 디렉토리(build tool)

- **java-lang/** ◂ 프로젝트 폴더
  - **src/** ◂ App. 제작에 관련된 파일들 두는 폴더
    - **main/** ◂ App. 관련 파일들 두는 폴더
      - **java/** ◂ 자바 소스 파일들 두는 폴더
      - **resources/** ◂ 설정 파일이나 기본 파일들 두는 폴더
    - **test/** ◂ 제작한 App. 소스를 테스트하는 파일들 두는 폴더
      - **java/** ◂ 단위 테스트 자바 소스 파일들 두는 폴더
      - **resources/** ◂ 테스트에 관련된 설정 파일 등을 두는 폴더

md = markdown -> html로 자동 변환
# 제목
README.md 대문자 - 정렬 순서상 가장 위로 올라감
polyglot 다중언어 사용자

*자바 패키지

자바 **소스 파일의 역할에 따라 관리하기 쉽게** 여러개의 폴더에 **분류**, 배치

- **src/**
  - **main/**
    - **java/**
      - **lol/**
      - **db/**
      - **ui/**

자바 소스파일을 용도에 따라 분류할 하위 폴더 = 자바 패키지

*자바 패키지 디렉토리 구조

회사 도메인명	제품명	역할
조직명	서비스명	

전세계에서 유일
다른회사와 충돌을 일어나면

예)
com/microsoft
com/bitcamp

상위조직에서 하위조직 순으로 디렉토리 계층을 구성

- **com/** ◂ 도메인(하위)
  - **microsoft** ◂ 도메인(상위)
  - **bitcamp**
    - **javaedu** ◂ 제품명/서비스명
      - **lang** ◂ 역할

예)
com/bitcamp/javaedu/lang
com/bitcamp/git/user

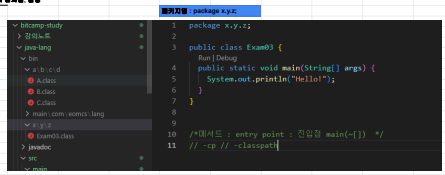
package com.eomcs.lang.ex01; ▸ **소속명 경로를 설정**

자바 소스코드를 프로젝트로 (-> ,

```
eomcs java > eomcs > java > lang > app > src > main > java > com > eomcs > lang > ex01 > Exam0110.java
1   package com.eomcs.lang.ex01;
2
```

*자바 패키지와 컴파일, 실행

패키지명 : package x.y.z;




```
1   package x.y.z;
2
3   public class Exam03 {
      Run | Debug
4     public static void main(String[] args) {
5       System.out.println("Hello!");
6     }
7   }
8
9
10  /*메서드 : entry point : 진입점 main(~[])  */
11  // -cp // -classpath
```

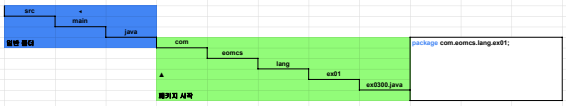
패키지에 소속되면 실행 시에도 패키지 명을 알려줘야함

```
PS D:\GIT\bitcamp-study\java-lang> java -cp bin Exam03
Error: Could not find or load main class Exam03
Caused by: java.lang.ClassNotFoundException: Exam03
```

package x.y.z -> java -cp bin(경로) + x.y.z(패키지 경로) + 실행할 클래스 파일명

```
PS D:\GIT\bitcamp-study\java-lang> javac -d bin -encoding UTF-8 src/main/java/Exam03.java
PS D:\GIT\bitcamp-study\java-lang> java -cp bin x.y.z.Exam03
Hello!
```

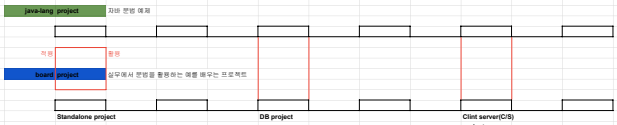
패키지에 소속된 소스코드는 컴파일하게 되면 패키지에 해당되는 폴더를 만든 후에 클래스 파일을 넣는다.
실행을 할 때는 패키지가 패키지폴더가 있는 경로까지만 지정하고, 클래스 이름을 적는다.



1 패키지에 소속될 경우, 소스파일의 첫 명령어로 package를 지정해야 한다.
2 소스 파일도 관리하기 쉽게, 패키지에 해당하는 폴더에 두어야 한다.

컴파일 $javac -d bin ---src/-/Exam0300.java
실행 $java -cp bin com.eomcs.lang.ex01.Exam0300

*수업 진행 방식



업그레이드 >

-목표
- 문법 활용 기술
- 애플리케이션 아키텍쳐 변화의 역사를 (약 20년간 진행됨) 압축해서 경험
- 실무에서 사용하는 유명 라이브러리나 프레임워크를 적절 구현
  - 라이브러리나 실제를 형상시키는 과정
- 실무에서 사용하는 라이브러리나 프레임워크 사용법 습득

*프로젝트 빌드와 빌드도구

build
Application을 만들기 위해 수행하는 개발 과정

빌드

컴파일	→	테스트	→	패키징	→	배포(배치)		프로젝트 폴더 문서
				.jar .war API 문서		deploy	→	build.gradle

↓
빌드명령어다 적절 명령 실행 = 번거로움
↓
빌드 자동화

빌드 작업흐름	load →	빌드 도구	↑ → ←	↓	빌드 작업을 순서대로 수행

↓
빌드 스크립트 파일

*.script(명령문을 작성할 때 사용하는 간단한 프로그래밍 언어)
*jar(Java Archivie)
*war(Web Archivie)

java	archive	자바를 묶어서 관리하다.
▬.jar		
▬.war Web Archice		

*☆빌드 도구☆

✚여러 라이브러리 관리 자동화 **✚빌드스크립트를 xml 대신 groovy 사용 (프로그래밍 언어)**

ANT	→	MAVEN	→	★ Gradle	→
↓		↓		↓	
build.xml		pom.xml		build.gradle	
ANT의 빌드 스크립트 파일		MAVEN의 빌드 스크립트 파일		↑ gradle의 빌드스크립트 파일	

프로그래밍 언어의 문법을 이용하여 빌드 작업을 좀 더 정확하게 제어할 ✚ 있다.
더 복잡한 빌드 작업을 수정할 수 있다.
***Android studio의 기본 빌드 도구**

XML은 태그 문법이기 때문에

다른 사람이 만든 코드 : 라이브러리 / 프레임워크

project

↓

외부 라이브러리 / Freamwork
API

dependency (의존)

태그방식으로

DSL(Domain-specific Language)

Specifis	**Domain**	**Language**	↓
Domain	-	**Specific**	Language
명		명	명

JSP	build.gradle (앱드 스크립트)
JAVA	groovy kotlin

JUit Jupiter	자바 단위 테스트

Gradle 명령어	RUN	자동 컴파일 후 **exe** 파일 실행
	-q	설명 없이 결과값만 노출
	gradle build	jar파일 + 배포파일까지 만들기

실행과정

*.com	기계어					App.com	찾아서 실행
*.exe		→	**실행**	→	**$ app**	app.exe	찾아서 실행
*.bat	스크립트 명전문					App.bat	찾아서 실행