# ■ 소스 파일 분리하기-Ⅰ

- 소스 파일을 관리하기 쉽도록 별도의 폴더로 분리하는 방법을 알아보자!

- 왜 소스 파일을 별도의 폴더로 분리하는가?

## Ⅰ. 실습코드

class Test04 {

public static void main(String[] args) {

System.out.println("안녕하세요!");

}

}

## Ⅱ. 분리

> javac -d [.class 파일을 저장할 폴더] [폴더명/.../소스파일명]

예)

$프로젝트> md src

$프로젝트> javac -d build/classes/java/main src/Test04.java

- 분리하는 이유 : 소스 파일을 보다 쉽게 관리하기 위함.

# ■ 소스 파일 분리하기-II

- Maven 프로젝트의 디렉토리 구조를 알아보자!

- 왜 Maven 프로젝트의 디렉토리 구조를 사용하는가?

## Ⅰ. 실습코드

class Test05 {

public static void main(String[] args) {

System.out.println("안녕하세요!");

}

}

## Ⅱ. Maven 프로젝트 디렉토리 구조

$프로젝트폴더

/src

/main

/java <= 자바 소스 파일을 두는 폴더

/resources <= 프로그램 설정 파일을 두는 폴더

/webapp <= 웹 애플리케이션 관련 파일을 두는 폴더

/test

/java <= 자바 테스트 소스 파일을 두는 폴더

/resources <= 자바 테스트 관련 설정 파일을 두는 폴더

## Ⅲ. 사용 이유?

- 국제 표준으로 자바 프로젝트를 관리 시 Maven 구조를 많이 사용

## Ⅳ. Gradle 빌드 도구를 활용한 Maven 프로젝트

## 구조 디렉토리 자동 생성

> gradle init --type java-application

=> Gradle 빌드 도구 관련 설정 파일 자동 생성

=> Maven 프로젝트 구조에 맞춰 폴더 자동 생성

=> 간단한 예제 파일이 자동 생성된 폴더에 저장

## Ⅴ. build.gradle

=> 자동으로 생성된 파일 중 개발자가 직접 손대는 파일

=> 이 파일에 빌드 관련 작업 내용 기술

=> 작업 명령은 groovy라는 언어를 사용하여 기술

=> groovy는 자바 호환 언어이므로 자바와 함께 사용 가능

=> build 작업 관련 문서 참조할 것

### ※ 자바 호환 언어란?

=> 컴파일 시, .class 파일을 만드는 언어

=> 바이너리 수준(bytecode 수준)에서 호환

=> 자바로 만든 모든 코드를 100% 사용할 수 있고,

자바에서도 호환 언어로 작성된 코드를 100% 사용 가능

=> ex. Scala, Groovy, Kotlin 등

## Ⅵ . Gradle을 이용한 프로젝트 빌드

> gradle build

=> 1) 의존하고 있는 외부 라이브러리를 자동으로 다운로드

=> 2) build 폴더 생성

=> 3) build 폴더에 컴파일 된 파일 저장

=> 4) 테스트 프로그램 실행

※ 라이브러리란?

- 개발자가 만든 코드들의 묶음

- 보통 .jar 파일에 묶여서 배포

- JAR(Java ARchive)의 약자

- archive : (꿀 등을 담는)항아리, 단지

즉, jar는 개발자가 만든 코드 묶음

- 압축 시, zip 알고리즘(압축공식)으로 압축

(zip 알고리즘은 특허로 등록되어 있지 않아서…)

■ **Gradle 사용법**

**Ⅰ. 자바 프로젝트 생성**

Gradle 빌드 도구를 이용하여 자바 프로젝트를 만들어보자!

**1) 프로젝트 폴더 생성**

자바 프로젝트 폴더를 만들기 위해 빈 폴더 생성

c:\workspace> md test

**2) Maven 프로젝트 구조로 만든다.**

Maven 프로젝트를 위한 기본 파일들과 디렉토리 자동 생성

c:\workspace> cd test

c:\workspace\test> gradle init --type java-application

* .gradle/ : gradle 관련 설정 파일이 들어 있는 폴더
* gradle/ : gradle wrapper 관련 파일
* src/main/java : 자바 기본 소스 폴더
* src/test/java : 자바 테스트 소스 폴더
* 기타 gradle 설명 파일들

**3) 프로젝트 빌드**

프로젝트의 소스를 컴파일하고 배포 파일을 생성한다.

c:\workspace\test> gradle build

* build : 컴파일된 .class 파일과 배포 파일(.jar) 등 생성

**4) 프로젝트 실행**

build.gradle 파일에서 mainClassName에 지정된 클래스 실행

윈도우에서만 가능

c:\workspace\test> gradle run

리눅스에 설치된 gradle은 run task가 없어 다음과 같이 build.gradle 파일에 task를 정의해야 함

//build.gradle 파일에 선언된 mainClassName 속성을 제거한다.

//mainClassName = 'Hello'

//build.gradle 파일에 다음 코드 추가

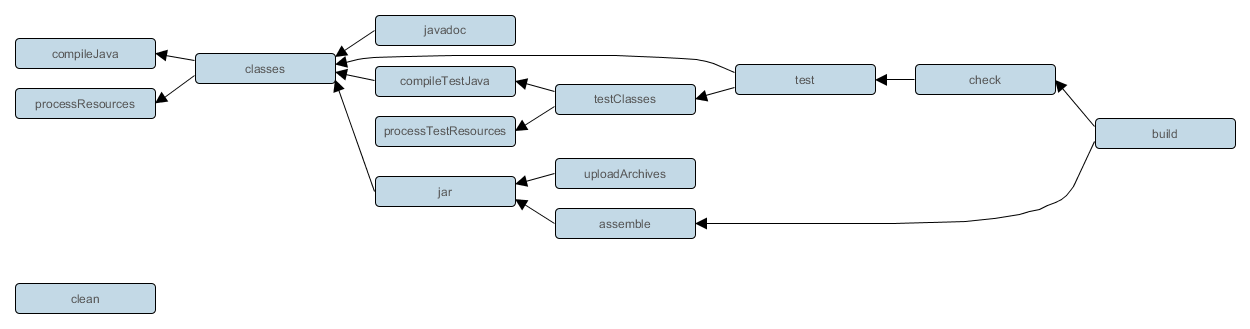
task run(type:JavaExec) {

main = 'Hello'

classpath = sourceSets.main.runtimeClasspath

}

**※ 참고 : Java plugin - tasks 구조도**

[](https://camo.githubusercontent.com/cb2dc9e96fb632ee3404ee3a16a6133e20b42d9b/68747470733a2f2f646f63732e677261646c652e6f72672f342e322f7573657267756964652f696d672f6a617661506c7567696e5461736b732e706e67)

**Ⅱ. Gradle 기본 명령어**

1. **clean**

프로젝트 폴더에서 build 폴더를 제거한다.

c:\workspace\test> gradle clean

1. **compileJava**

자바 소스(src/main/java)를 컴파일하는 명령어이다. 소스 폴더의 파일을 컴파일하여 build/classes/java/main 폴더에 둔다.

c:\workspace\test> gradle compileJava

1. **processResources**

src/main/resources 폴더의 모든 파일을 build/resources/main/ 폴더에 복사한다.

c:\workspace\test> gradle processResources

1. **classes**

포함작업

* compileJava
* processResources

c:\workspace\test> gradle classes

1. **compileTestJava**

포함작업

* classes

자바 테스트 소스(src/test/java)를 컴파일하는 명령어이다. 소스 폴더의 파일을 컴파일하여 build/classes/java/test 폴더에 둔다.

c:\workspace\test> gradle compileTestJava

1. **processTestResources**

src/test/resources 폴더의 모든 파일을 build/resources/test/ 폴더에 복사한다.

c:\workspace\test> gradle processTestResources

1. **jar**

포함작업

* classes

.jar 배포 파일을 생성하여 build/libs 폴더에 둔다.

c:\workspace\test> gradle jar

1. **build**

모든 작업을 수행한다. 따라서 마지막으로 테스트 및 배포 파일 생성 시 실행

c:\workspace\test> gradle build

1. **tasks**

Gradle 명령으로 수행할 수 있는 작업 정보 표시

일부 작업 목록 출력

c:\workspace\test> gradle tasks

전체 작업 목록 출력

c:\workspace\test> gradle tasks --all

**Ⅲ. Gradle 플러그인**

build.gradle 파일에 플러그인을 등록해야만 그 플러그인의 task를 사용 가능

* 1. **플러그인 등록**

//build.gradle

apply plugin: '플러그인명'

* 1. **Gradle 플러그인에서 제공하는 작업 목록 확인**

c:\workspace\test> gradle tasks --all

**Ⅳ. wrapper**

시나리오: 어떤 개발자가 프로젝트를 만든 후 다른 개발자에게 전달하려 한다. 그런데 다른 개발자의 PC에 gradle이 깔려있지 않다. 그렇다고 그 개발자에게 gradle 설치라는 번거로움을 주기 싫다. 단지 실행하면 자동으로 gradle을 설치하게 하고 싶다. 또는 자신이 사용한 gradle 버전을 다운로드 받게 하고 싶다.

이런 경우를 위해 사용하는 것이 wrapper 이다. gradle에서는 가능한 wrapper 방식을 사용할 것을 권장하고 있다.

1. **wrapper 관련 파일 생성**

c:\workspace\test> gradle wrapper --gradle-version [버전]

예)

c:\workspace\test> gradle wrapper --gradle-version 4.2

1. **wrapper 파일 사용**

gradlew를 실행하면 명시된 버전의 gradle을 자동으로 다운로드 한다. 그런 후 gradle 명령을 수행한다.

c:\workspace\test> gradlew [작업명]