2. Maven.

Maven

- 프로젝트를 관리하는 도구
- 빌드 자동화 + 프로젝트 관리.

프로젝트(라이브러리) 관리

- pom.xml 파일 이용하여 프로젝트 관련 .jar 파일 다운로드 및 관리.
- 프로젝트 생명주기 일괄적으로 관리.

빌드 자동화

- 빌드 작업 간단. 쉽게, 일관성 있게 수행할 수 있는 통합 환경 제공
  - ↳ 소스 코드 파일을 실행 코드 파일로 변환하여 배포하는 과정.

프로젝트 관리 기능

- 일반 개발자 프로젝트 관리 과정 → Maven 설정으로 대체
- 정형화된 프로젝트 디렉토리 구조 관리 (Pom.xml)
  - → Convention over Configuration (CoC) 패러다임 따름 → 설정보다는 규약.
- 의존성 관리 기능 (편리한 라이브러리 기능 제공) → pom.xml, Repository.
  - → 프로젝트 빌드에 필요한 라이브러리, 플러그인을 개발자 PC에 자동 다운로드
  - → 중앙 저장소 (http://mvnrepository.com/, http://search.maven.org/)
  - → 로컬 저장소 (USER_HOME\.m2\repository)
- 빌드 프로세스를 관리 (pom.xml)
  - → 플러그인 설정을 통해 빌드 자동화.

프로젝트 디렉토리 기본 구조 (pom.xml)

- src/main/resources : 클래스 패스로 사용되는 디렉토리

Source code → ${basedir}/src/main/java
resources → 〃 /src/main/resources
Tests → 〃 /src/test
distributable JAR → 〃 /target
Compiled byte code → 〃 /target/classes

```
project home
├─ src
│   ├─ main
│   │    ├─ java
│   │    └─ resources
│   └─ test
│        ├─ java
│        └─ resources
├─ target
└─ pom.xml
```

웹 디렉토리 구조

- 자바 소스와 리소스 디렉토리는 메이븐 기본 디렉토리 유지, 웹자원 관리하는 별도의 src/main/webapp 디렉토리 사용
  이 값은 maven-war-plugin의 warSource Directory에 설정되어 있음

☆☆☆ 의존관계 설정 (라이브러리) pom.xml

- 프로젝트당 1개의 pom.xml 파일 관리.
- 최상위 엘리먼트 (root element) : project
- 3개의 필수 필드
  → groupId : 프로젝트 그룹 고유 도메인 ex) org.kpu.
  → artifactId : 프로젝트명
  → version : 프로젝트 버전
- 프로젝트 의존관계의 라이브러리 관리 : dependency

모든 POM은 Super POM (기본 POM)으로부터 상속받음.

기본 설정정보 포함.
수정하려면 POM에서
오버라이드.

프로젝트 빌드 설정

- 프로젝트 기본 정보, 저장소, 프로퍼티, 디렉터리 구조.
- 플러그인 (plugins)
- 골 (Goals)

Maven Repository

- 메이븐 저장소는 프로젝트에 사용되는 프로젝트 jar 파일, 라이브러리 jar 파일들 위치, 3가지 타입 존재
  1) 지역저장소 (local) : 메이븐을 빌드할 때 다운로드 하는 라이브러리. 플러그인을 관리하는 저장소.
     → USER_HOME/.m2/repository 디렉토리., groupId와 artifactId 아래에 각 버전에 따른 라이브러리 관리.
  2) 중앙저장소 (Central) : 오픈소스 라이브러리, 메이븐 플러그인, 메이븐 아키타입을 관리하는 저장소.
  3) 원격 저장소 (remote) : 모든 라이브러리가 중앙 저장소에 존재 X 중앙저장소에 존재하지 않는 라이브러리를 관리하기 위해 별도의 메이븐 저장소를 설치해 관리하는 것이 가능함

메이븐 의존성 검색 절차 (Maven Dependency Search Sequence)

- 1단계 : 지역 저장소 검색 └라이브러리 없으면 2단계
- 2단계 : 중앙 저장소 검색 찾은 라이브러리 지역 저장소에 저장. 없으면 에러 발생 및 종료.
- 3단계 : 원격 저장소 검색 〃 〃

의존 라이브러리 적용 스코프

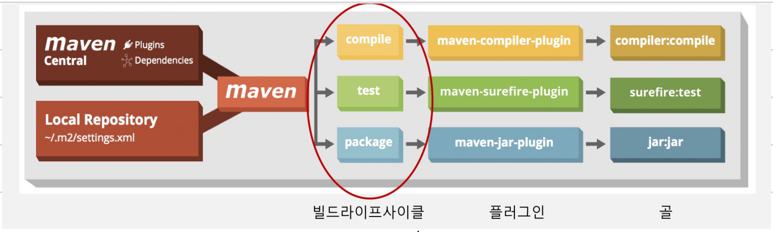
→ 의존라이브러리 적용시점 제한 가능 – Scope 설정.

Scope	Description
compile	설정X, 기본스코프.
provided	컴파일 시 직접의존성 참조, 런타임시 다른 환경에서의 의존성 제공 받음 (해당 컨테이너의 서블릿 API)
runtime	컴파일시 사용X, 앱 실행할 때 사용되는 라이브러리 경우 설정.
test	테스트 시점에만 적용되는 라이브러리에 대한 스코프 설정 시 사용 (JUnit)
System	provided와 비슷, 우리가 직접 jar파일 제공해야 함
import	다른 POM 설정 파일에 정의되어 있는 의존 관계 설정을 현재 프로젝트로 가져옴. <dependencyManagement/>에서만 사용 가능함

빌드 자동화 기능

메이븐은 소프트웨어 빌드를 위한 공통 인터페이스를 제공하는 프레임워크

→ 플러그인 설정을 통해 기능 확장.



↓ 빌드단계 (컴파일, 테스트, 배포)

골은 플러그인에 의해 실행됨.

빌드단계에서 수행되는 작업

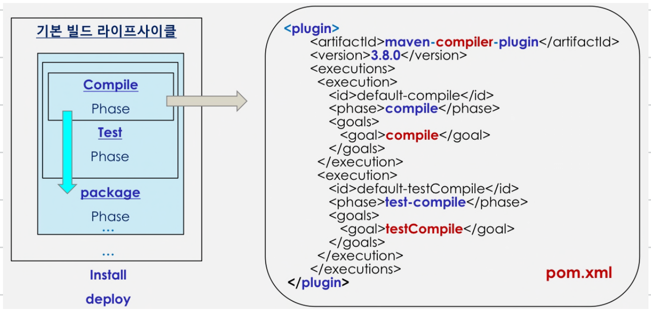
Maven 빌드 라이프사이클(lifecycle)

- 기본, clean, site 3개의 라이프 사이클 있음.
- 기본라이프사이클은 여러 단계의 페이즈로 나뉘어져 있으며, 각 페이즈 의존관계 가짐.
  compile, test, package deploy 순서로 진행됨.
- clean 라이프사이클은 clean 페이즈를 이용하여 이전 빌드에서 생성된 모든 파일(target 디렉토리)들을 삭제함.
- Site 라이프사이클은 site, site-deploy 페이즈를 이용하여 생성된 문서들을 대상 사이트에 배포함.
- 기본 빌드 라이프 사이클 일부 단계.

기본 phase	Description.
리소스관리(resource)	리소스 경로에 복사함
컴파일(compile)	소스코드 컴파일
테스트(Test)	JUnit과 같은 테스트 프레임워크로 단위테스트함
인스톨(install)	패키지 로컬 저장소에 배포
배포(deploy)	원격 메이븐 저장소에 압축한 파일 배포

Maven Phase & Goal

- 빌드 라이프사이클은 하나 이상의 골을 수행하는 페이즈(phase)들로 구성
- 각 페이즈별로 플러그인이 작업을 수행 → Goal.



```
<plugin>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.8.0</version>
     <executions>
      <execution>
        <id>default-compile</id>
        <phase>compile</phase>
        <goals>
          <goal>compile</goal>
        </goals>
      </execution>
      <execution>
        <id>default-testCompile</id>
        <phase>test-compile</phase>
        <goals>
          <goal>testCompile</goal>
        </goals>
      </execution>
     </executions>
</plugin>
```

Maven Plugin

- 대부분 기능들은 플러그인을 통해서 제공

plugin	Description
clean	빌드이후 target 디렉 삭제
compiler	자바소스파일 컴파일
surefire	JUnit 단위 테스트 실행, 리포트 생성
jar	현재 프로젝트로부터 jar 파일생성
war	〃 war 〃
javadoc	프로젝트의 javadoc 문서 생성
antrun	빌드페이즈단계에서 특정 ant 작업 수행

Maven 빌드 라이프 사이클에는 기본, clean, site 3개의 라이프 사이클 존재,
기본 라이프 사이클은 여러 단계 페이즈로 나뉘어져 있고 compile, test, package, deploy 순서로 진행됨.