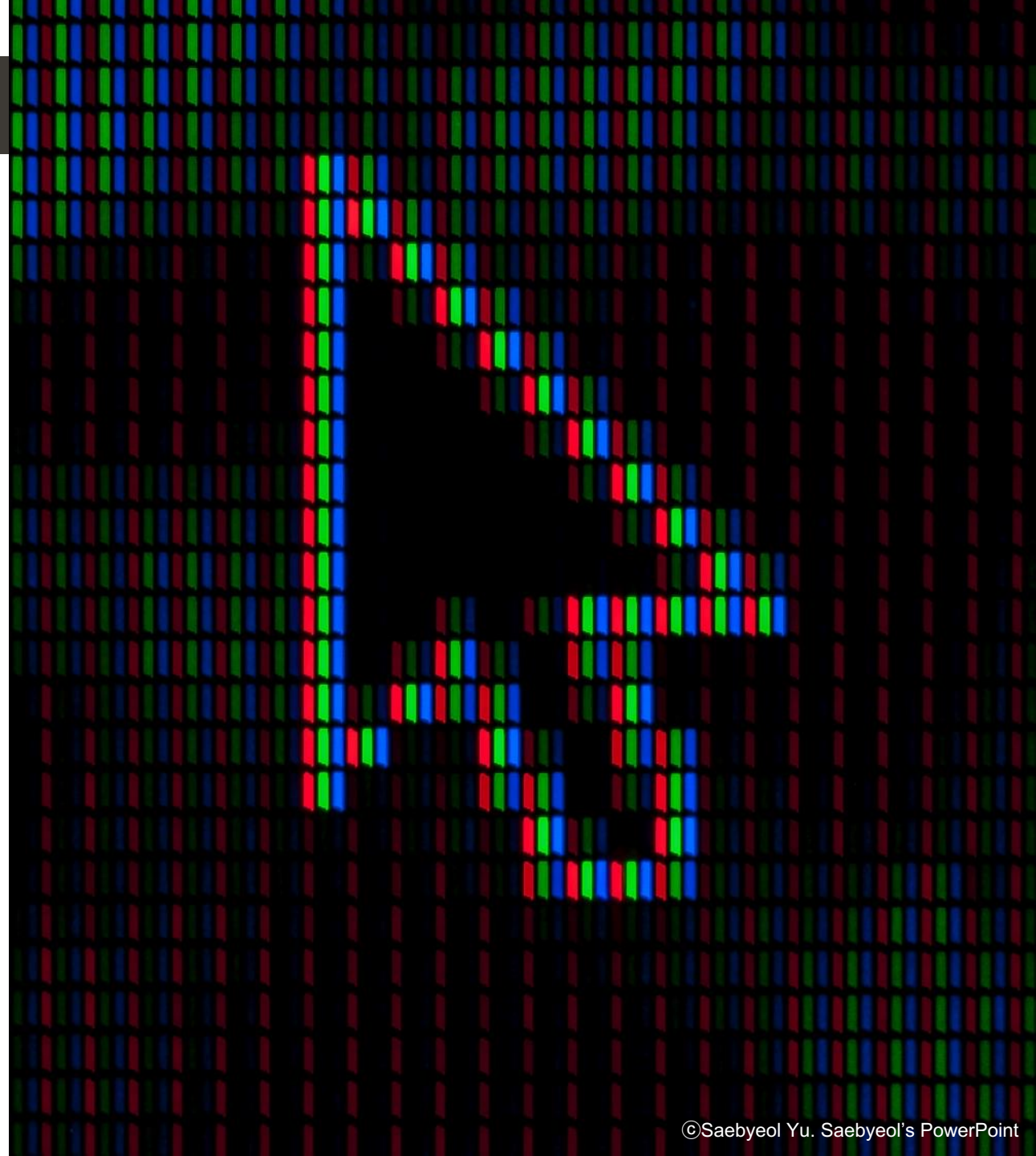
Maven

목차 a table of contents



Chapter 1 소개 및 설치

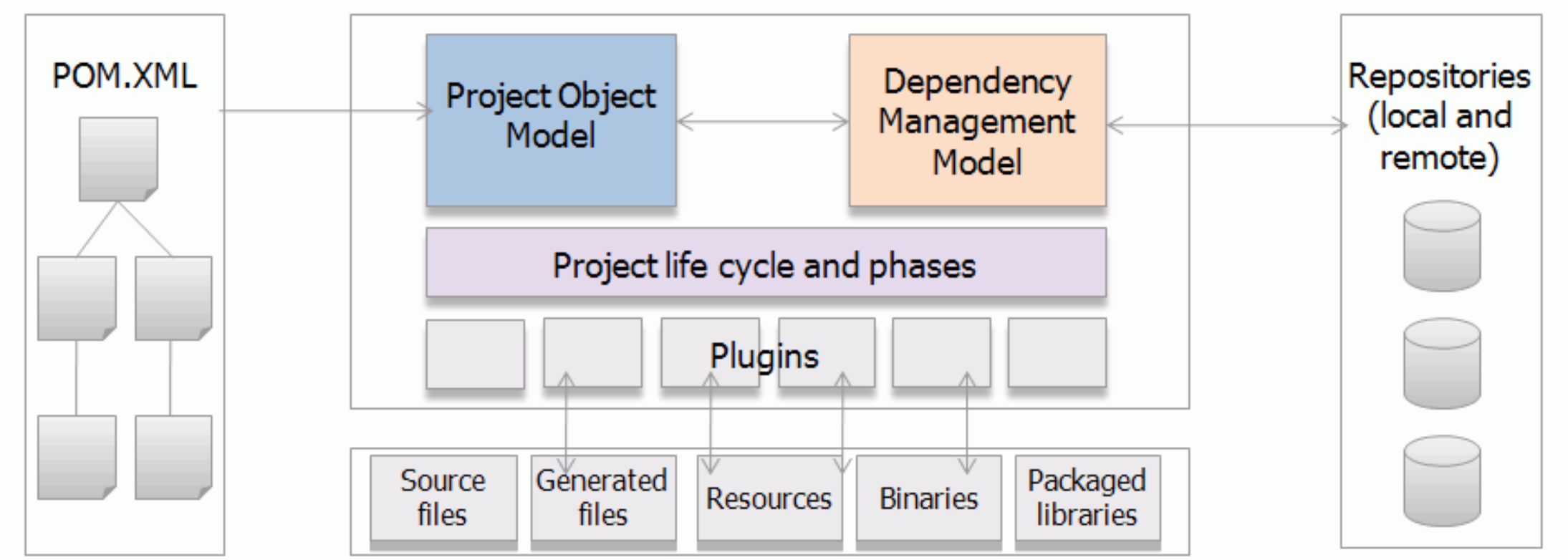

Maven이란?

Chap 1

- Apache에서 만든 java용 프로젝트 관리도구.
- 이전에 있던 Apache Ant의 대안으로 만들어짐.
- Maven이 히트를 친 건, pom.xml에 필요한 라이브러리를 정의해 놓으면 내가 사용할 라이브러리 뿐만 아니라 해당 라이브러리가 작동하는데 필요한 다른 라이브러리들까지 자동으로 다운받아줘서 의존성을 관리해주기 때문.
- Maven은 중앙 저장소를 통한 자동 의존성 관리(회사내 사용 저장소도 구성 가능)

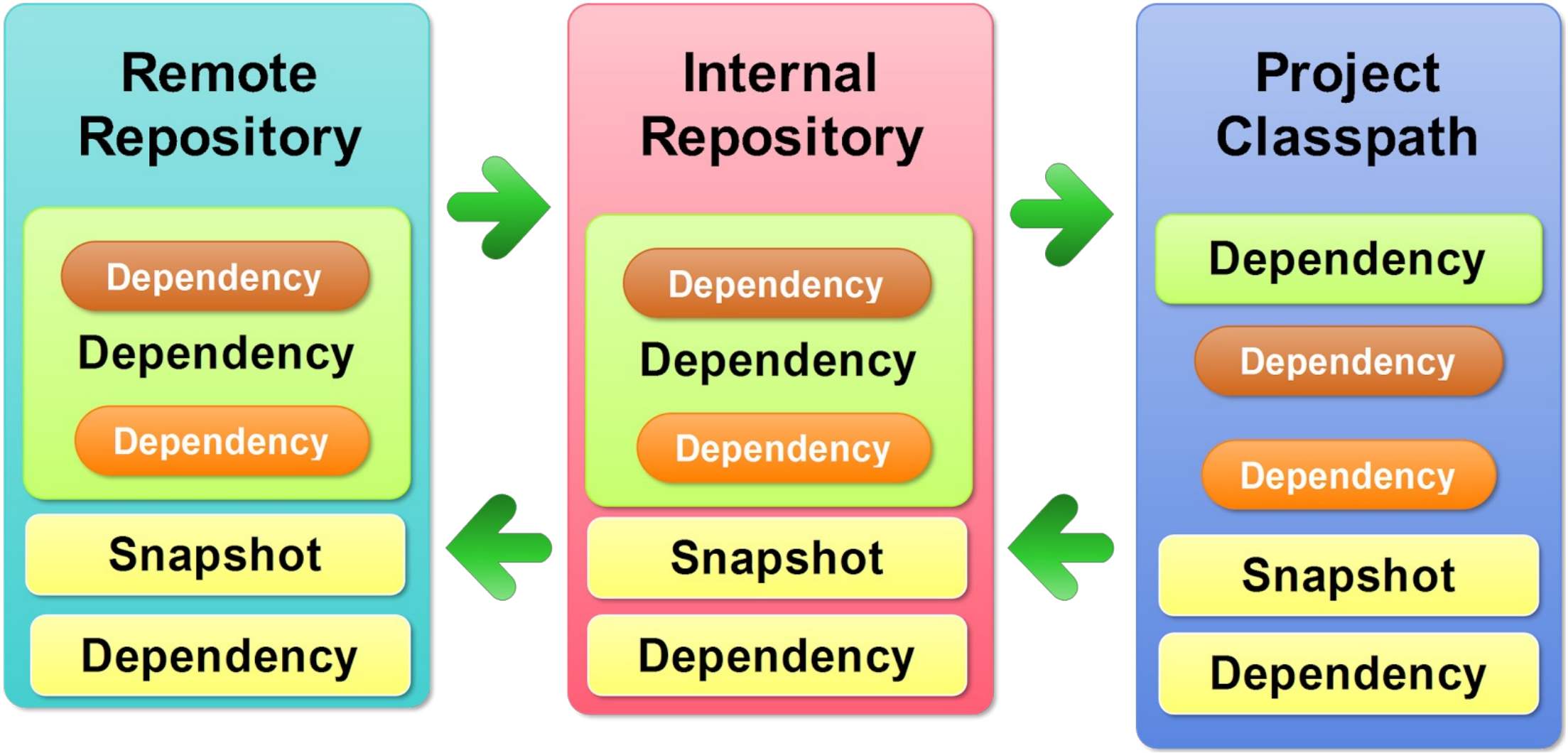
Chap 1

Maven Architecture



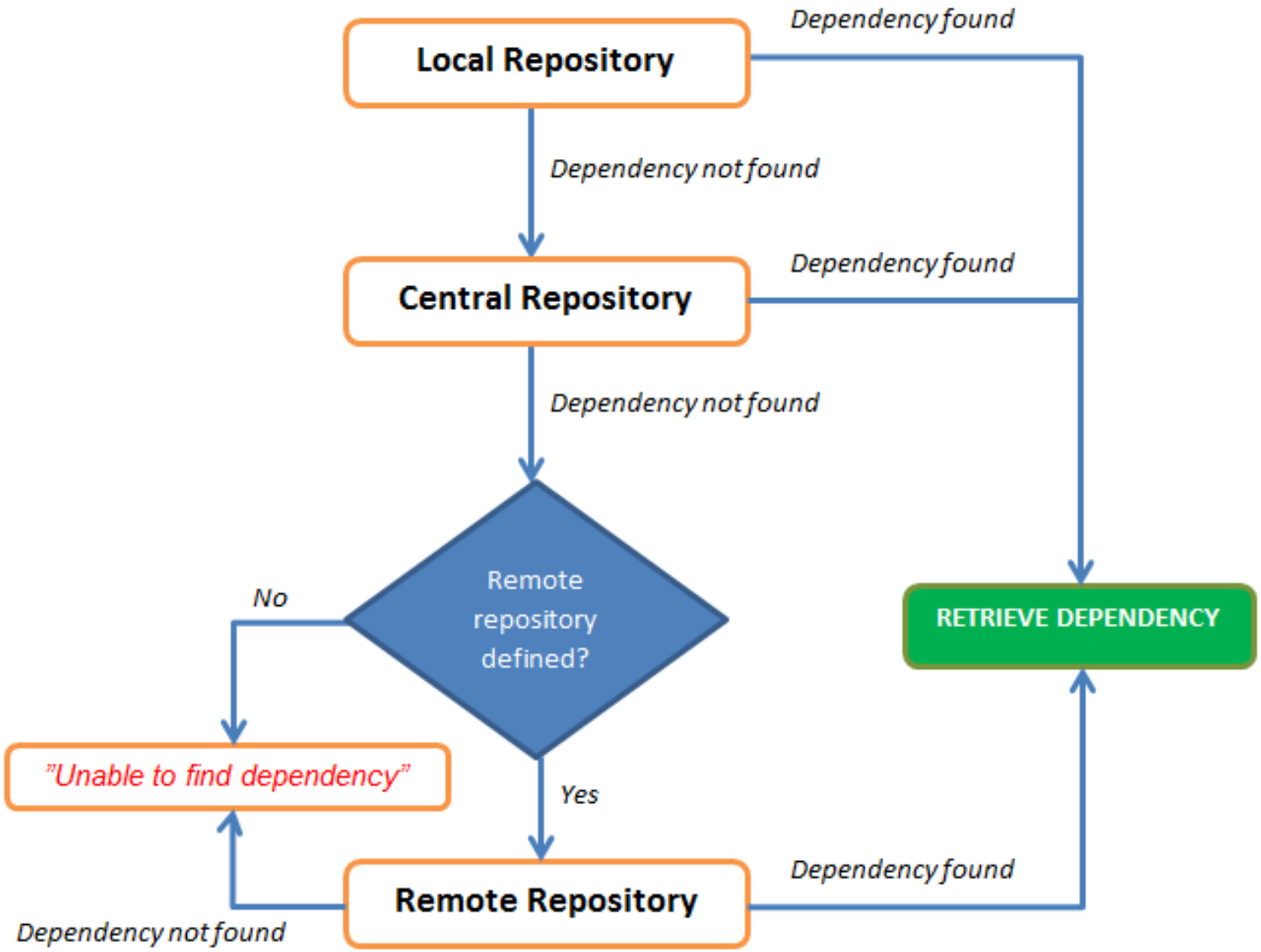
Maven Architecture

Maven interact with Repository



Chap 1

Maven flow



https://www.devopsschool.com/blog/how-to-configure-sonatype-nexus-repository-with-maven/

Maven장점

Chap 1

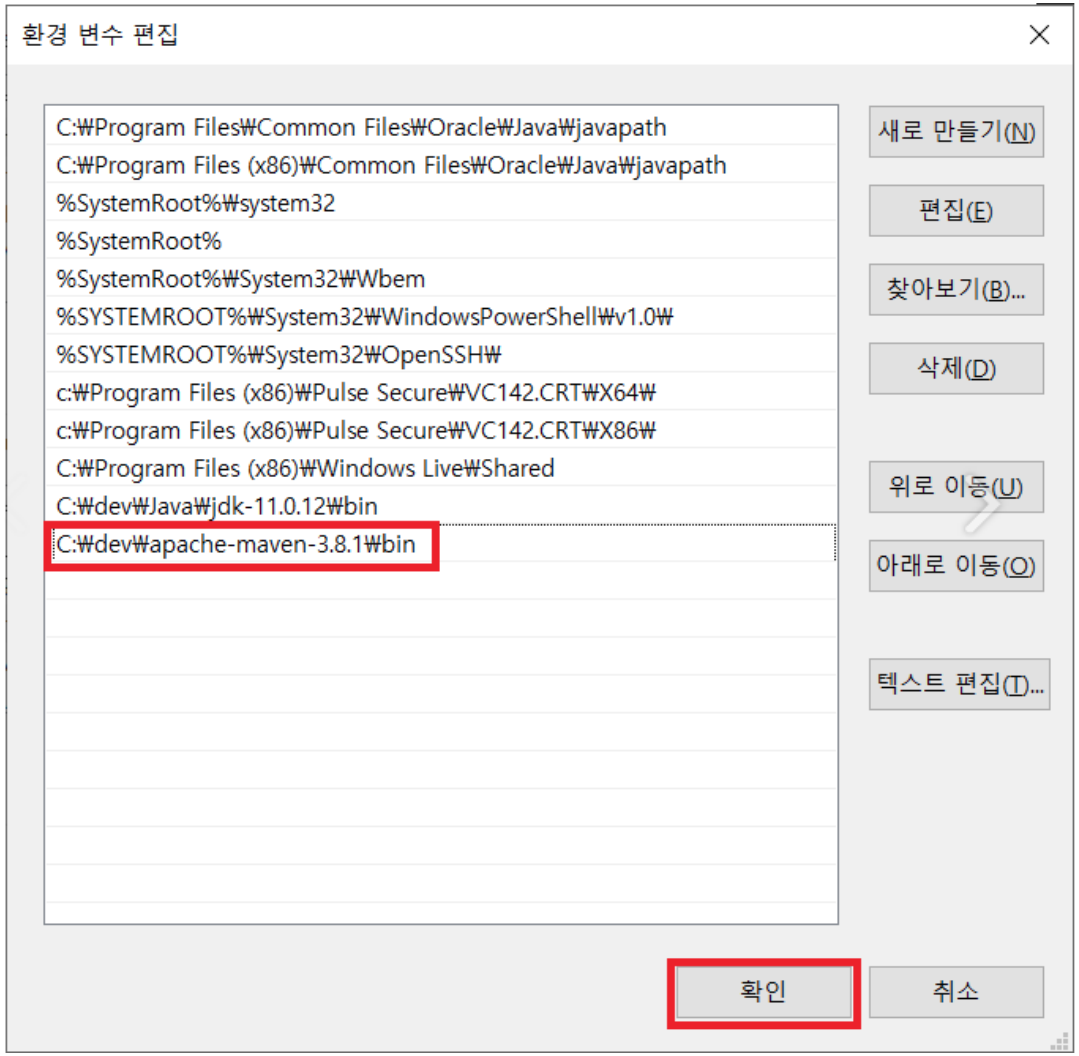
Maven의 장점

- 라이브러리 관리가 쉽다.
- 프로젝트의 작성부터 컴파일, 테스트 프로젝트 라이프사이클에 포함된 각 테스트를 지원.
- 빌드 과정을 쉽게 만들어 줌.
- 정형화된 빌드 시스템 제공.
- 다양한 플러그인을 통해서 많은 작업이 자동화 됨.

설치 및 환경설정

Chap 1 Maven 설치 및 환경설정

1. JDK 우선 설치
2. JAVA_HOME 환경 변수 설정.
3. maven 다운로드
    1. https://maven.apache.org/download.cgi
4. 압축 풀고, 환경변수에 등록
    1. JAVA_HOME
    2. M2_HOME
        1. C:₩/sw/maven
    3. PATH
        1. %JAVA_HOME%/bin
        2. %M2_HOME%/bin
5. mvn –v 로 확인.



Chapter 2 프로젝트 생성

프로젝트 구조

Chap 2 프로젝트의 구조

- first run
  - mvn archetype:generate
  - -DgroupId=com.mycompany.app
  - -DartifactId=my-app
  - -DarchetypeArtifactId=maven-archetype-quickstart
  - -DarchetypeVersion=1.4
  - -DinteractiveMode=false

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

```
|-- META-INF
|   |-- MANIFEST.MF
|   `-- application.properties
|   `-- maven
|       `-- com.mycompany.app
|           `-- my-app
|               |-- pom.properties
|               `-- pom.xml
`-- com
    `-- mycompany
        `-- app
            `-- App.class
```

Chap 2 프로젝트 구조 상세

- 프로젝트 루트 디렉토리: Maven 프로젝트의 최상위 디렉토리. 이 디렉토리에는 Maven 프로젝트를 설명하는 POM(Project Object Model) 파일인 pom.xml이 있음.
- 소스 디렉토리: Maven은 기본적으로 소스 코드를 "src/main/java" 디렉토리에 위치. Java 소스 코드 파일들은 이 디렉토리에 작성.
- 테스트 디렉토리: 테스트 코드를 작성하는 디렉토리로 "src/test/java"에 위치. 테스트를 위한 추가적인 리소스는 "src/test/resources" 디렉토리에 위치시킬 수 있음.
- 리소스 디렉토리: 프로덕션 코드에서 사용되는 리소스 파일들을 저장하는 디렉토리. 예를 들어, 속성 파일, 설정 파일, 이미지 파일 등을 "src/main/resources"에 위치시킬 수 있음.
- 빌드 디렉토리: Maven 빌드 중에 생성된 파일들이 저장되는 디렉토리. 기본적으로 "target" 디렉토리에 생성. 컴파일된 클래스 파일, 패키지 파일, JAR 파일 등이 여기에 위치.

아키타입의이해

Chap 2 Maven Archetype

- Apache Maven의 Archetype은 Maven 프로젝트의 기본 템플릿을 정의하는 도구.
- Archetype을 사용하면 Maven을 통해 새로운 프로젝트를 빠르게 설정 가능.
- Maven Archetype은 미리 구성된 프로젝트 템플릿을 제공하며, 이 템플릿은 특정 프로젝트 유형이나 프레임워크에 맞추어져 있음.
- 각 Archetype은 일반적으로 프로젝트의 기본 구조, 디렉토리 구성, 설정 파일, 의존성 등을 정의.
  - 예를 들어, 웹 애플리케이션을 개발하기 위한 Archetype은 기본적인 디렉토리 구조와 웹 관련 설정 파일, 서블릿 컨테이너 의존성 등을 포함할 수 있음.

Chap 2 Maven Archetype

- Maven Archetype은 Maven의 명령줄 도구를 통해 사용할 수 있음.
- 새로운 프로젝트를 생성하기 위해 Maven Archetype을 실행하면, 사용자는 프로젝트 유형과 Archetype을 선택하고 추가적인 설정을 입력할 수 있음.
- Maven은 선택한 Archetype에 따라 프로젝트를 생성하고 초기 구성을 완료.
- Archetype은 Maven 중앙 저장소(Central Repository)에서 다양한 유형의 프로젝트 템플릿을 제공.
- 개발자는 자체 Archetype을 정의하고 공유가능.
- 이를 통해 개발자들은 자신의 표준 프로젝트 템플릿을 만들고 재사용할 수 있으며, 팀 또는 조직 간에 일관된 프로젝트 구조를 유지할 수 있음.
- Maven Archetype은 Maven을 사용하여 프로젝트를 시작할 때 유용한 도구로, 초기 설정과 구성 작업을 간소화하고 효율성을 높일 수 있음.

샘플

- mvn archetype:generate
- -DgroupId=com.mycompany.app
- -DartifactId=my-app
- -DarchetypeArtifactId=maven-archetype-quickstart
- -DarchetypeVersion=1.4
- -DinteractiveMode=false

프로젝트 생성/설정

Chap 2 프로젝트의 생성

- first run
  - mvn archetype:generate
  - -DgroupId=com.mycompany.app
  - -DartifactId=my-app
  - -DarchetypeArtifactId=maven-archetype-quickstart
  - -DarchetypeVersion=1.4
  - -DinteractiveMode=false

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

```
|-- META-INF
|   |-- MANIFEST.MF
|   `-- application.properties
|   `-- maven
|       `-- com.mycompany.app
|           `-- my-app
|               |-- pom.properties
|               `-- pom.xml
`-- com
    `-- mycompany
        `-- app
            `-- App.class
```

Chap 2 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Chapter 3 의존성 관리

의존성의이해/관리

Chap 3 의존성 관리

- 의존성 관리는 프로젝트가 필요로 하는 외부 라이브러리나 모듈을 정의하고 이를 자동으로 다운로드하고 관리하는 작업
- Maven의 의존성 관리 기능을 통해 개발자는 프로젝트에서 필요한 외부 라이브러리와 모듈을 쉽게 관리가능. 의존성을 선언하고 Maven이 이를 해결하도록 설정하면, Maven은 의존성을 자동으로 다운로드하고 클래스패스에 추가하여 개발 환경을 설정. 이는 개발자가 수동으로 외부 라이브러리를 관리하는 번거로움을 줄여주고, 프로젝트의 의존성 버전 관리와 충돌 문제를 해결
- Maven은 의존성을 중앙 저장소에서 가져오는데, 중앙 저장소에는 수많은 오픈 소스 라이브러리와 모듈이 포함됨. 따라서 Maven을 사용하면 많은 외부 의존성을 간편하게 추가하고 업데이트할 수 있음. Maven은 의존성을 관리하면서 버전 충돌을 방지하기 위해 의존성 트리를 통해 중복된 라이브러리를 제거하고 최적의 버전을 선택.
- Maven은 의존성을 스코프와 전이성을 통해 세밀하게 제어할 수 있음. 의존성의 스코프는 해당 의존성이 어느 단계에서 사용되는지를 지정하는데, 개발, 테스트, 런타임 등 다양한 단계에서 필요한 의존성을 구분할 수 있음. 또한 전이성은 의존성 간에 전파되는 정도를 조절하는데, Maven은 의존성 트리를 통해 필요한 의존성만 전파하여 충돌이나 불필요한 의존성을 방지.

Chap 3 의존성 관리 단계

- pom.xml 파일: Maven 프로젝트의 최상위 디렉토리에는 pom.xml 파일이 존재. 이 파일은 프로젝트의 메타데이터와 의존성 정보를 포함.
- 의존성 선언: pom.xml 파일에서 의존성을 선언하여 프로젝트가 사용해야 할 외부 라이브러리나 모듈을 명시. 의존성은 그룹ID, 아티팩트ID, 버전 등의 정보로 구성.
- 의존성 해결: Maven은 pom.xml 파일에 선언된 의존성을 기반으로 필요한 라이브러리나 모듈을 찾아 자동으로 다운로드하고 관리. 이를 위해 Maven은 중앙 저장소(Central Repository)라는 공용 레파지토리를 사용. 의존성이 해결되면 해당 라이브러리나 모듈은 프로젝트의 클래스패스에 추가.
- 의존성 트리: Maven은 의존성 간에 상속 관계를 유지. 즉, 프로젝트가 의존하는 라이브러리가 다시 다른 라이브러리에 의존하는 경우, Maven은 이러한 의존성 트리를 자동으로 해결. 이를 통해 프로젝트에서 필요한 모든 의존성을 효과적으로 관리할 수 있음.
- 스코프와 전이성: Maven은 의존성에 대해 스코프(scope)와 전이성(transitivity)을 설정가능. 스코프는 의존성이 어떤 상황에서 사용되는지를 정의하며, 예를 들어 컴파일 타임, 런타임, 테스트 등이 있음. 전이성은 의존성이 다른 의존성에 영향을 미치는 정도를 나타내며, Maven은 이를 통해 중복 의존성을 제거하고 충돌을 방지.

Chap 3 POM.xml

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:schemaLocation="http://maven.apache.org/P
OM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>net.daum.cafe</groupId>
  <artifactId>simple-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>simple-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

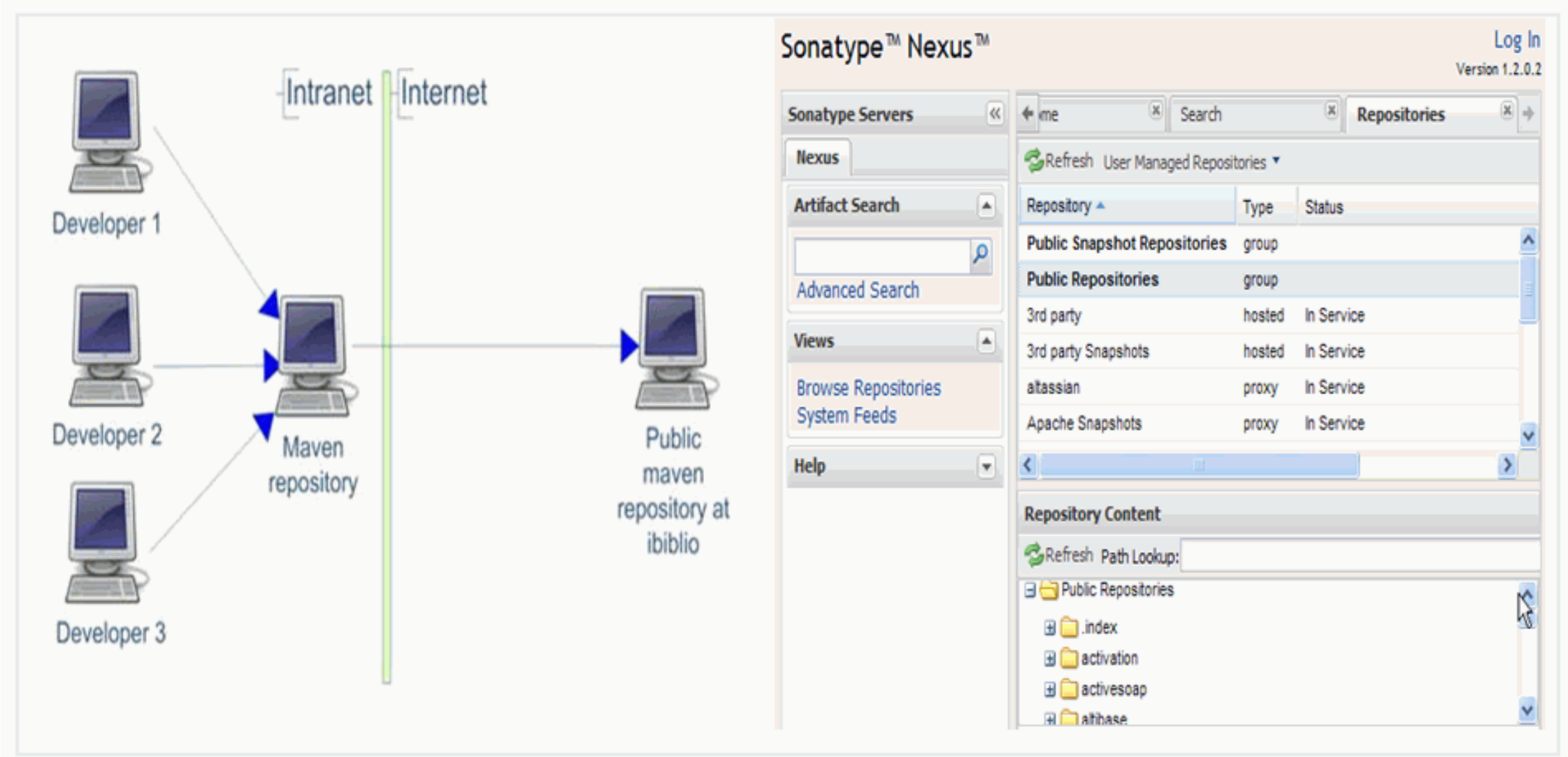
- 위 POM 파일에서 프로젝트 정보를 기술하는 태그는 다음과 같다.
  - <name> - 프로젝트 이름
  - <url> - 프로젝트 사이트 URL
- POM 연관 정보는 프로젝트간 연관 정보를 기술하는데, 관련 태그는 다음과 같다.
  - <groupId> - 프로젝트의 그룹 ID 설정
  - <artifactId> - 프로젝트의 Artifact ID 설정
  - <version> - 버전 설정
  - <packaging> - 패키징 타입 설정. 위 코드의 경우 프로젝트의 결과 Artifact가 jar 파일로 생성됨을 의미한다. jar 뿐만 아니라 웹 어플리케이션을 위한 war나 JEE를 위한 ear 등의 패키징 타입이 존재한다.
  - <dependencies> - 이 프로젝트에서 의존하는 다른 프로젝트 정보를 기술한다.

Chap 3 의존 설정

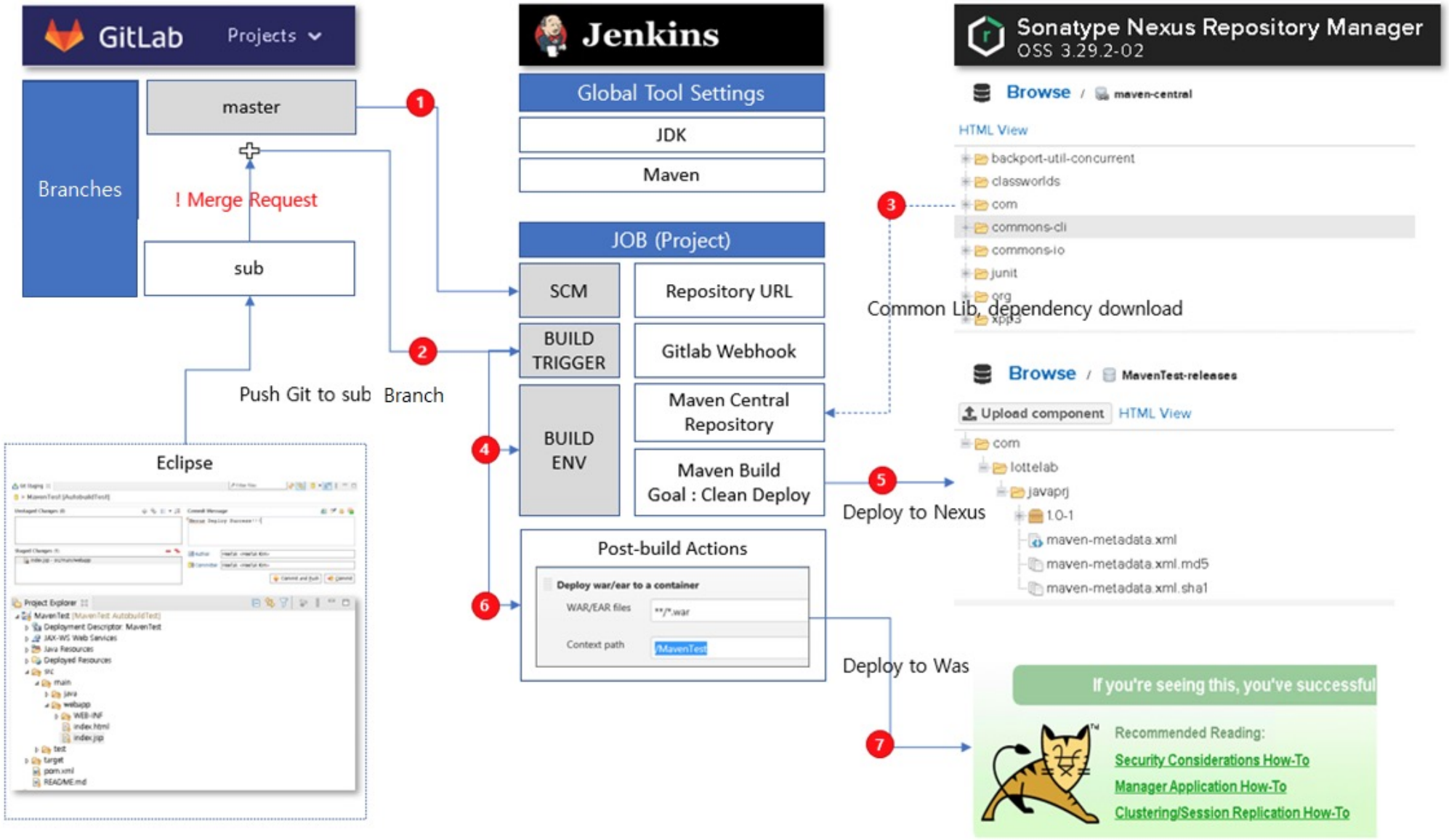
- <dependency> - 의존하는 프로젝트 POM 정보를 기술
  - <groupId> - 의존하는 프로젝트의 그룹 ID
  - <artifactId> - 의존하는 프로젝트의 artifact ID
  - <version> - 의존하는 프로젝트의 버전
  - <scope> - 의존하는 범위를 설정

중앙저장소/로컬저장소

Chap 3 중앙 저장소 와 원격저장소(with Nexus)



Chap 3 중앙 저장소 와 원격저장소(with Jenkins)



https://emptyreset.tistory.com/96

의존성추가/버전관리

의존 성 추가와 관리

- 의존성 추가를 통해서 해당 의존성의 의존성은 개발자가 신경쓸 필요가 없다.

```
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.2.1</version>
</dependency>
```

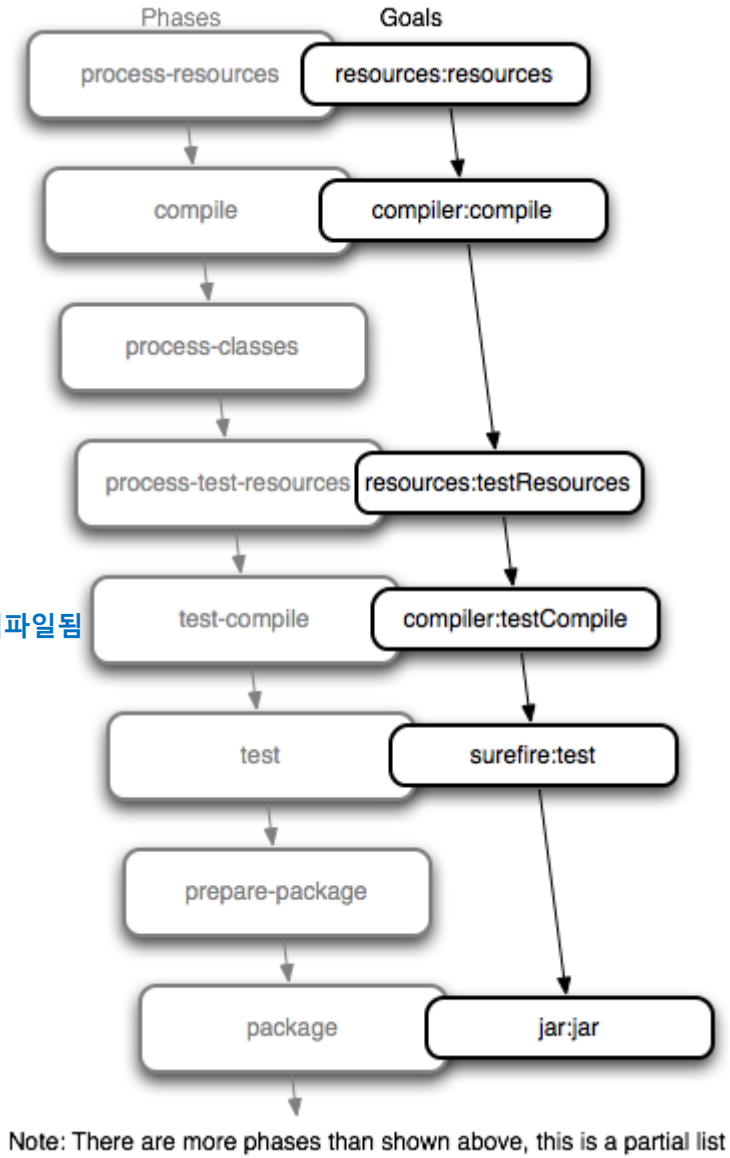
```
<dependencies>
  <dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>2.1</version>
  </dependency>
  <dependency>
    <groupId>commons-pool</groupId>
    <artifactId>commons-pool</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>javax.sql</groupId>
    <artifactId>jdbc-stdext</artifactId>
    <version>2.0</version>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>xml-apis</groupId>
    <artifactId>xml-apis</artifactId>
    <version>2.0.2</version>
  </dependency>
  <dependency>
    <groupId>xerces</groupId>
    <artifactId>xerces</artifactId>
    <version>2.0.2</version>
  </dependency>
</dependencies>
```

Chapter 4 빌드 및 테스트

빌드 lifecycle

Chap 4 Maven Life Cycle

- clean : 빌드 시 생성되었던 산출물을 삭제
  - pre-clean : clean 작업 전에 사전작업
  - clean : 이전 빌드에서 생성된 모든 파일 삭제
  - post-clean : 사후작업



단위테스트용 파일은 여기서 컴파일됨

https://www.devopsschool.com/blog/how-to-configure-sonatype-nexus-repository-with-maven/

- default : 프로젝트 배포절차, 패키지 타입별로 다르게 정의됨
  - validate : 프로젝트 상태 점검, 빌드에 필요한 정보 존재유무 체크
  - initialize : 빌드 상태를 초기화, 속성 설정, 작업 디렉터리 생성
  - generate-sources : 컴파일에 필요한 소스 생성
  - process-sources : 소스코드를 처리
  - generate-resources : 패키지에 포함될 자원 생성
  - compile : 프로젝트의 소스코드를 컴파일
  - process-classes : 컴파일 후 후처리
  - generate-test-source : 테스트를 위한 소스 코드를 생성
  - process-test-source : 테스트 소스코드를 처리

- generate-test-resources : 테스팅을 위한 자원 생성
- process-test-resources : 테스트 대상 디렉터리에 자원을 복사하고 가공
- test-compile : 테스트 코드를 컴파일
- process-test-classes : 컴파일 후 후처리
- test : 단위 테스트 프레임워크를 이용해 테스트 수행
- prepare-package : 패키지 생성 전 사전작업
- package : 개발자가 선택한 war, jar 등의 패키징 수행
- pre-integration-test : 통합테스팅 전 사전작업
- integration-test : 통합테스트
- post-integration : 통합테스팅 후 사후작업
- verify : 패키지가 품질 기준에 적합한지 검사
- install : 패키지를 로컬 저장소에 설치
- deploy : 패키지를 원격 저장소에 배포

Chap 4

- site : 프로젝트 문서화 절차
  - pre-site : 사전작업
  - site : 사이트문서 생성
  - post-site : 사후작업 및 배포 전 사전작업
  - site-deploy : 생성된 문서를 웹 서버에 배포

빌드명령어/옵션

Maven build

- mvn clean package
  - clean : 이전 빌드에서 생성된 파일을 제거하는 작업 target을 지움
  - package : 프로젝트를 패키징하여 결과물을 생성하는 작업
  - compile : 소스 코드를 컴파일
  - test : 단위 테스트를 실행
  - install : 빌드된 패키지를 로컬 저장소에 설치
  - deploy : 패키지를 원격 저장소에 배포하는 작업
- 빌드 작업은 profile을 지정하여 특정 환경에 맞는 빌드 구성 가능.


테스트실행/리포트

Chap 4 mvn test

- 프로젝트의 단위 테스트를 실행하고 테스트 리포트를 생성하는 기능을 제공
- 개발자는 자동화된 테스트 실행과 테스트 결과의 분석을 통해 소프트웨어 품질을 개선
- test/documantation plugin
  - Surefire
  - Failsafe
- test 실행
  - “target/surefire-reports” 에 HTML 형식의 레포트 생성.
  - surefire는 XML, CSV 형식 지원
  - Cobertura는 테스트 커버리지 리포트도 생성

Chapter 5 배포

배포과정의이해

Chap 5 mvn 배포 과정

1. 프로젝트 설정: Maven 프로젝트의 최상위 디렉토리에 있는 pom.xml 파일을 열어 프로젝트 정보와 배포 설정을 확인. pom.xml 파일에는 프로젝트의 메타데이터, 의존성, 빌드 설정 등이 포함.
2. 버전 관리: 배포할 버전을 결정하고 pom.xml 파일에 해당 버전을 명시. 일반적으로 배포할 때마다 버전을 업데이트하여 고유한 버전 관리를 수행.
3. 빌드: Maven build 명령어(mvn clean package 등)를 사용하여 프로젝트를 빌드. 이 단계에서 소스 코드 컴파일, 테스트 실행, 패키징 등의 작업이 수행.
4. 로컬 설치: Maven의 install 명령어(mvn install)를 사용하여 빌드된 패키지를 로컬 저장소에 설치. 이를 통해 다른 프로젝트에서 해당 패키지를 의존성으로 사용가능.
5. 원격 저장소 구성: 배포할 원격 저장소를 구성. 일반적으로는 Maven의 중앙 저장소 이외에도 사내 또는 외부 저장소를 사용가능.
6. 배포: Maven의 deploy 명령어(mvn deploy)를 사용하여 빌드된 패키지를 원격 저장소에 배포. 이 단계에서는 배포 설정에 따라 패키지와 관련 메타데이터가 원격 저장소로 전송.
7. 배포 확인: 원격 저장소에 배포된 패키지를 확인. 이를 통해 다른 개발자나 시스템에서 해당 패키지를 사용가능.

배포 설정

Chap 5 배포 설정의 핵심

- 두개 파일
  - settings.xml
  - pom.xml(<distributionManagement>)
- 설정 상세
  - settings.xml 파일:

private한 정보
외부 공개X

    - Maven의 전역 설정 파일로, Maven 설치 디렉토리의 conf 폴더에 위치. 이 파일은 모든 프로젝트에 적용되는 전역 설정을 포함. 배포에 필요한 원격 저장소의 URL, 인증 정보 등을 설정 가능.
  - pom.xml 파일:
    - 프로젝트의 루트 디렉토리에 위치한 Maven 프로젝트 설정 파일. pom.xml 파일에서는 프로젝트의 의존성, 빌드 설정과 함께 배포 설정을 관리 가능.
      <distributionManagement> 요소를 사용하여 원격 저장소의 위치와 배포 설정을 지정.

Chap 5 배포설정시 중요 요소

- <distributionManagement> : 원격 저장소의 위치와 배포 설정을 지정.
- <repository> : 원격 저장소의 URL과 식별자를 설정.
- <snapshotRepository> : 스냅샷(Snapshot) 버전을 위한 원격 저장소를 설정.
- <server> : 인증 정보를 지정합니다.
- <id> : 인증 정보를 식별하는 식별자입니다. <username>과 <password> 요소를 사용하여 인증 정보를 설정.
- <releases>와 <snapshots> : 릴리스 버전과 스냅샷 버전에 대한 배포 설정을 지정. 각각 <enabled> 요소를 사용하여 릴리스와 스냅샷 버전의 배포를 활성화 또는 비활성화.

설정 예

```
<distributionManagement>
  <repository>
    <id>my-repository</id>
    <name>My Repository</name>
    <url>https://example.com/repository</url>
  </repository>
  <snapshotRepository>
    <id>my-snapshot-repository</id>
    <name>My Snapshot Repository</name>
    <url>https://example.com/snapshot-repository</url>
  </snapshotRepository>
</distributionManagement>
```

Chap 5

- 예제에서는 <repository> 요소로 원격 저장소의 URL과 식별자를 설정하고, <snapshotRepository> 요소로 스냅샷 버전을 위한 원격 저장소의 URL과 식별자를 설정.
- 배포 설정은 settings.xml 파일에서도 구성
- settings.xml 파일에서는 <servers> 요소를 사용하여 인증 정보를 설정.

settings.xml 에서의 인증정보 설정

```
<settings>
  <servers>
    <server>
      <id>my-repository</id>
      <username>my-username</username>
      <password>my-password</password>
    </server>
  </servers>
</settings>
```

감사합니다