애플리케이션 배포

Maven & PMD

❑ **웹기반어플리케이션 –> 애플리케이션 배포 -> 애플리케이션 배포 환경 구성하기**
-> 애플리케이션 소스코드 검증하기

– 참고 사이트

- https://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev
- http://maven.apache.org/

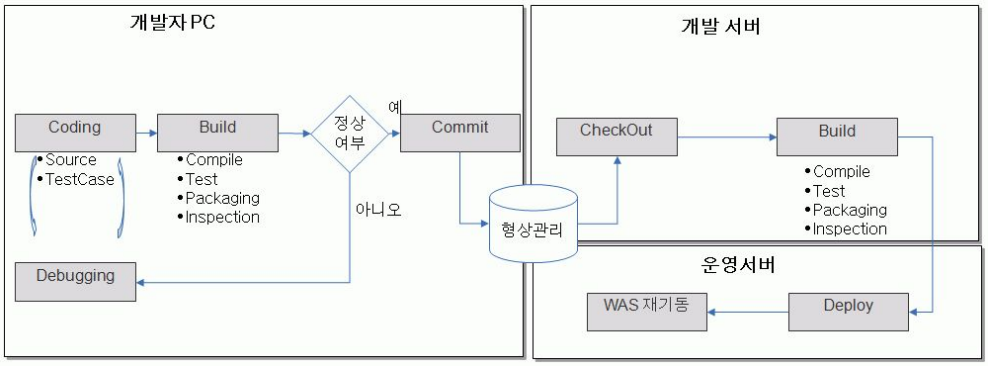
대덕인재개발원 : 최희연

❑ Build

빌드란 소스개발에서 최종 사용자에게 전달될때까지의 전 과정, 즉 프로젝트의 생명주기 전체를 아우르는 행위를 말하는데, 프로젝트의 규모가 클수록 정형화된 개발환경, 다양한 라이브러리 의존성 관리, 코드 품질 관리 및 결함 관리등의 필요성이 제기되며 빌드 관리 툴을 이용하여 이러한 요구를 해결할 수 있다. 전체 빌드 과정은 그 주체에 따라 개발자 개인의 PC 에서 빌드를 하여 작동 가능한 코드를 생성하는 개인 빌드와 개발 서버에서 프로젝트 개발자들의 작성 코드를 통합하여 빌드를 수행하는 통합 빌드로 구성되는데, 본 교재에서는 전자정부프레임워크 스펙에 기반하여 개인 빌드에 Maven을 통합 빌드에 오픈 소스 CI 서버인 Hudson 을 사용하도록 한다.

❑ 소프트웨어 개발 프로젝트의 빌드 프로세스

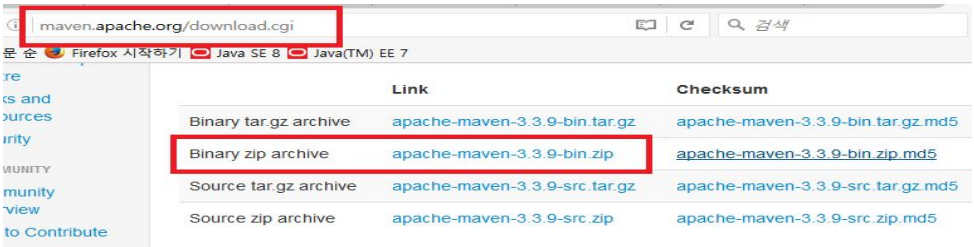
1. 개인빌드
   1) 개발자가 소스 코드를 작성한다.
   2) 개인 빌드를 통해 실행 가능한 바이너리 코드를 생성한다.
   3) 정상 코드 여부를 판별한 후 형상 관리 서버로 작성한 코드를 커밋한다.
2. 통합 빌드
   1) 개발 서버는 형상 관리 서버로부터 최신의 소스 코드를 check out 받는다.
   2) 주기적 자동 빌드 프로세세를 통해 코드 빌드 후 동작 가능한 소프트웨어 패키지를 생성한다.
   3) 운영 서버로 패키징한 소프트웨어를 배포한다.
3. 운영 서버의 WAS 재기동 작업을 통하여 작성한 소프트웨어를 실행시킨다.



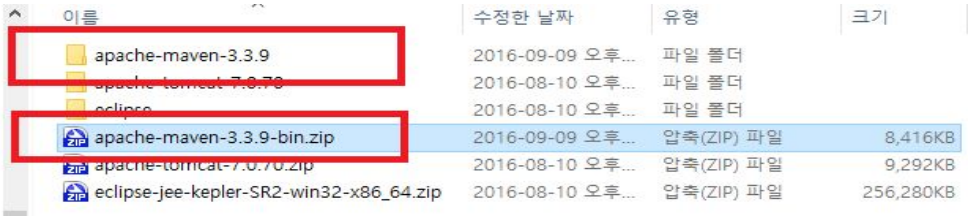
❑ 개인 빌드 툴 Maven

: 메이븐은 프로젝 트 객체 모델이라는 개념을 바탕으로 프로젝트 의존성 관리, 라이브러리 관리, 프로젝트 생명 주기 관리 기능 등을 제공하는 프로젝트 관리 도구이다. 또한 플러그인을 기반으로 소스 코드로부터 배포 가능한 산출물을 만들어 내는 빌드 기능 뿐만 아니라 레포팅 및 documentation 작성 등 프로젝트 라이프사이클 전반에 걸친 기능을 제공한다.

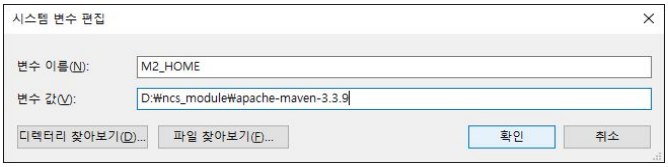
1. Maven 설치
   1) 메이븐 설치 파일 다운로드



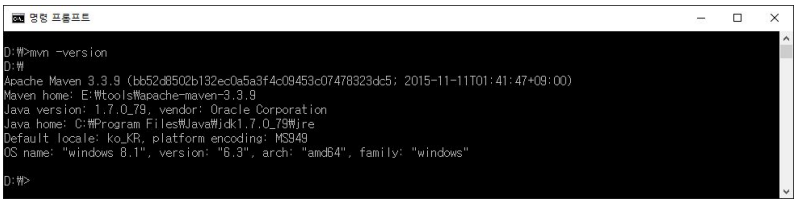
   2) 메이븐 설치



3) 환경변수 설정

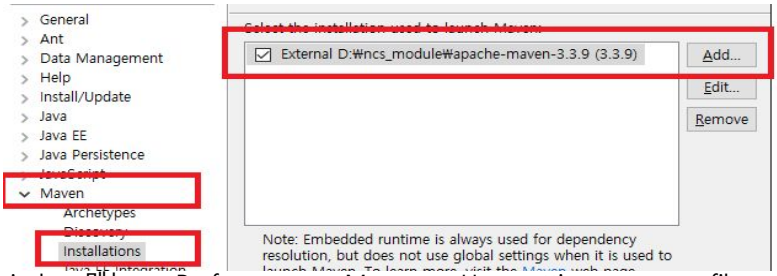


4) 설치 확인

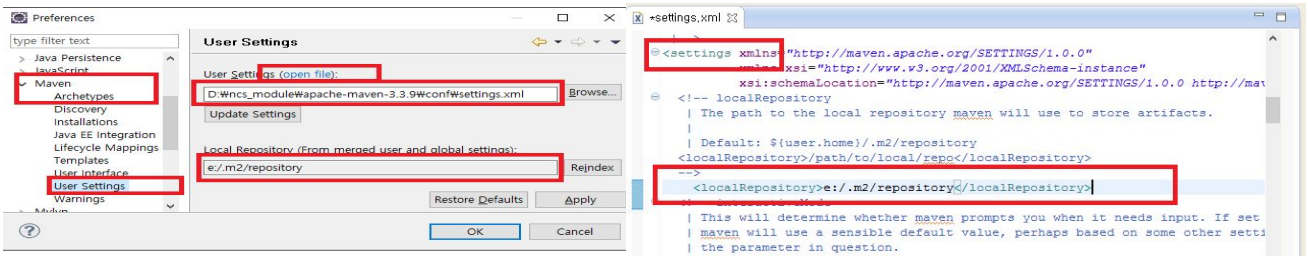


5) 이클립스 & 메이븐 연동

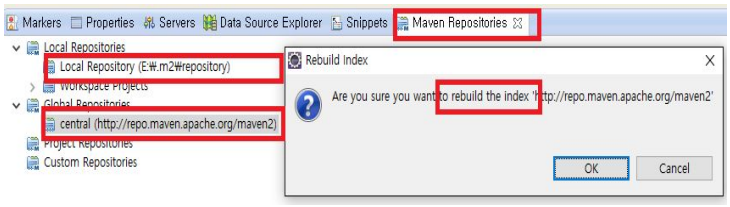
window 메뉴 -> Preferences -> Maven -> Installations -> Add -> %M2_HOME%



window 메뉴 -> Preferences -> Maven -> User settings -> open file -> Local Repository 위치 설정

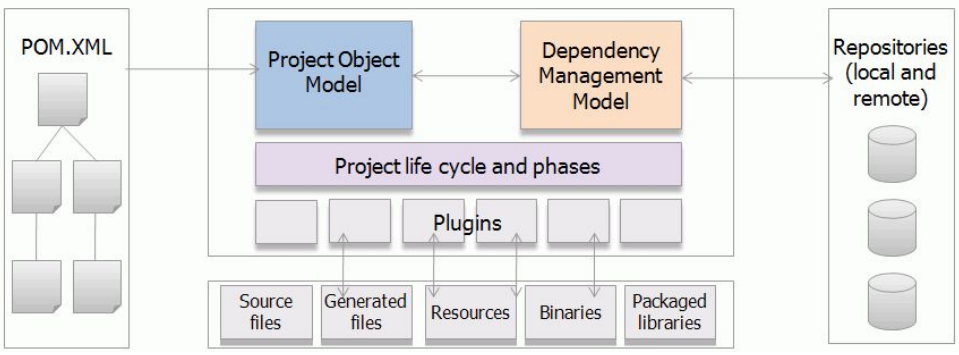


6) Global Repository 인덱싱



2. Maven 아키텍처

메이븐은 프로젝트 전체 내용에 대한 기술 및 설정을 담당하는 POM(Project Object model)과 라이브러리(artifact)관리에 대한 dependency/repository 관리 모델, 그리고 컴파일, 테스트, 패키징 등의 빌드 생명 주기를 다루는 라이프사이클 및 플러그인들과의 연동을 다루는 부분으로 구성된다.

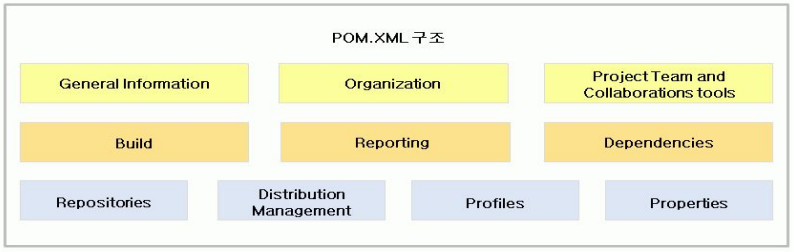


POM	메이븐 엔진 내장 + POM XML 파일에서 선언적으로 제공함.
의존성 관리 모델	로컬 및 리모트 저장소를 이용함.
프로젝트 생명주기 모델	메이븐 엔진은 프로젝트 생명주기 내에서 리소스에 대한 모든 접근 및 관리에 플러그인을 이용함.

3. POM

POM 은 프로젝트 관리 및 빌드에 필요한 환경 설정, 의존성 관리 등의 프로젝트 정보들을 담고 있으며, 이는 메이븐 프로젝트 생성 시 기본 설정 파일로 생성되는 pom.xml 파일에 기술된다.

pom.xml 파일은 프로젝트의 세부 메타 데이터 정보를 포함하여 크게 10개의 항목으로 구성되며, 프로젝트 일반 정보, 버전 및 설정 관리, 빌드 환경, 라이브러리 저장소 및 의존성 등의 내용을 포함한다.

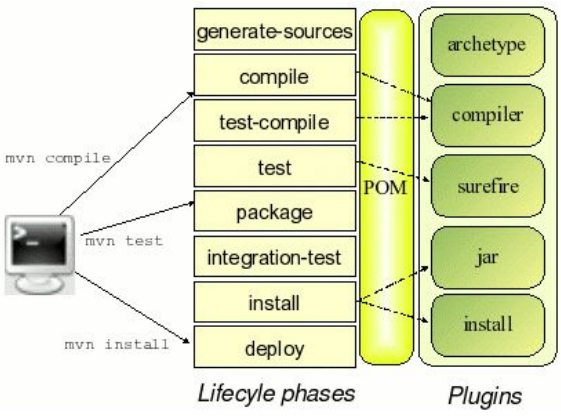


1) General Information : 프로젝트 이름, 설명, 버전 정보 등의 기술
2) Organization : 프로젝트 조직 정보(이름, 홈페이지, URL)
3) Project Team and Collaborations tools : 형상관리 서버, 이슈 트랙커, 통합 빌드 서버 정보 등
4) proferties : 프로젝트 프로퍼티 설정

```
<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/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>kr.or.ddit</groupId>
        <artifactId>sampleMVN</artifactId>
        <packaging>war</packaging>
        <version>0.0.1-SNAPSHOT</version>
        <name>sampleMVN Maven Webapp</name>
        <description>빌드툴 테스트 프로젝트 </decription>
        <url>http://maven.apache.org</url>
        <properties>
                <java-version>1.7</java-version>
                <encoding>UTF-8</encoding>
        </properties>
</project>
```

5) Build : 인코딩 정보 등 빌드 라이프사이클 환경 설정
   ① 메이븐 빌드는 소프트웨어 프로젝트의 핵심적인 빌드 라이프 사이클 개념을 따르고 있으며, 빌드 초기화 단계부터 artifact의 배포까지의 생명주기를 정의하고 있다.
   ② 생명주기내의 각 빌드 단계는 각각의 플러그인과 바인딩 되어 해당 플러그인의 명령을 실행함으로 생명주기 단계 순서에 따라 순차적인 빌드를 수행할 수 있다.

생명주기 단계	설명
validate	현재 설정과 POM의 내용이 유효한지 확인
generate-sources	코드 생성기가 이 다음의 단계들에서 컴파일되고 처리할 소스코드를 생성하기 시작하는 순간
Compile	소스 코드를 컴파일하고, 타겟 디렉토리에 트리 구조로 저장됨
test	컴파일된 단위 테스트를 실행하고, 그 결과를 표시함
package	실행 가능한 바이너리 파일들을 war 나 jar 같은 배포용 압축 파일로 압축함
install	압축 파일을 로컬 메이븐 저장소에 추가
deploy	압축 파일을 원격 메이븐 저장소에 추가



참고 : http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

```
<build>
	<finalName>sampleMVN</finalName>
	<extensions>
		<extension>
			<groupId>org.apache.maven.wagon</groupId>
			<artifactId>wagon-ssh</artifactId>
			<version>2.12</version>
		</extension>
	</extensions>
	<plugins>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-compiler-plugin</artifactId>
			<version>3.5</version>
			<configuration>
				<source>${java-version}</source>
				<target>${java-version}</target>
			</configuration>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-pmd-plugin</artifactId>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-war-plugin</artifactId>
			<version>2.6</version>
			<configuration>
				<warSourceDirectory>${basedir}/webapp</warSourceDirectory>
			</configuration>
		</plugin>
	</plugins>
</build>
```

③ 생명주기 내의 특정 phase 를 실행하기 위해 다음과 같은 명령을 사용할 수 있다.
ex) console$> mvn install
install phase 의 전단계들인 generate-sources 부터 compile, test 를 거쳐 install 과정이 수행되고, 로컬 저장소에 해당 프로젝트의 jar 파일이 저장된다.
두개 이상의 phase를 연속 실행하는 경우 다음과 같은 형태로 사용할 수 있다.
ex) console$> mvn clean install

6) Reporting : 리포트 생성에 대한 설정

메이븐은 프로젝트 정보에 대한 기본 리포트 뿐만 아니라 플러그인 설정은 통해, emma, JavaDoc 등의 문서 생성 기능을 제공한다.

```
<reporting>
	<outputDirectory>${basedir}/target/site</outputDirectory>
	<plugins>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-project-info-reports-plugin</artifactId>
			<version>2.9</version>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-javadoc-plugin</artifactId>
			<configuration>
				<minmemory>128m</minmemory>
				<maxmemory>512m</maxmemory>
				<encoding>${encoding}</encoding>
				<docencoding>${encoding}</docencoding>
			</configuration>
		</plugin>
	</plugins>
</reporting>
```

리포트 생성 명령 ex) console$>mvn site , mvn javadoc:javadoc

7) Dependencies : 프로젝트 의존성 설정

메이븐의 주요 기능 중의 하나인 의존성 관리를 위한 설정으로 프로젝트에서 필요한 라이브러리(artifact)를 선언하여 build path 에 포함시킨다.

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.8.1</version>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.0.1</version>
                <scope>provided</scope>
                </dependency>
        <dependency>
                <groupId>javax.servlet.jsp</groupId>
                <artifactId>javax.servlet.jsp-api</artifactId>
                <version>2.2.1</version>
                <scope>provided</scope>
        </dependency>
</dependencies>
```

① 의존성 분석 순서
   a. 로컬 저장소에서 의존성 확인
   b. 원격 저장소 리스트에서 의존성 확인
   c. 모두 실패시 의존성 에러 보고

② 의존성 SCOPE : 라이브러리 사용 범위를 설정
   a. compile : 기본설정이며, 모든 클래스 패스에서 사용 가능
   b. Provided : 컴파일과 유사하나 패키지에는 포함되지 않고 컨테이너나 실행환경의 라이브러리를 대신 사용
   c. runtime : 런타임시에 사용되는 라이브러리
   d. test : 테스트 단계에서만 사용됨
   e. system : 개발자가 직접 시스템 패스를 설정하여 직접 jar 파일을 제공하는 경우 사용되며, 저장소에서 해당 라이브러리를 찾지 않는다.

```
<dependency>
                <groupId>sun.jdk</groupId>
                <artifactId>tools</artifactId>
                <version>1.5.0</version>
                <scope>system</scope>
                <systemPath>${java.home}/../lib/tools.jar</systemPath>
</dependency>
```

의존성 Scope 가 생략된 경우, 기본값으로 compile 이 적용되지만, JDK 나 WAS등 실행 및 운영환경에서 제공하는 라이브러리와 dependency 에서 설정된 라이브러리 간의 충돌이 일어나는 경우가 발생할 수 있기때문에 적절한 scope 를 명시해야 한다.

8) Repositories : 라이브러리 저장소 위치 설정

repository 는 artifact 들의 저장소로 로컬 저장소와 리모트 저장소로 구성되며, dependency로 등록된 의존 라이브러리들을 지정된 저장소로부터 가져와 빌드패스에 추가하게 된다.

```
<repositories>
        <repository>
                <id>jahia</id>
                <name>ojdbc repository</name>
                <snapshots><enabled>false</enabled></snapshots>
                <url>http://maven.jahia.org/maven2/</url>
        </repository>
</repositories>
```

기본 설정된 중앙 저장소(central remote repository)는 effective POM 에서 확인할 수 있다.

```
<repository>
        <snapshots>
                <enabled>false</enabled>
        </snapshots>
        <id>central</id>
        <name>Central Repository</name>
        <url>https://repo.maven.apache.org/maven2</url>
</repository>
```

메이븐 저장소에 등록된 라이브러리들은 http://search.maven.org(Central Repository 검색사이트) 와 https://mvnrepository.com 에서 검색하여 빌드 스크립트를 확인할 수 있다.

9) Distribution Management : 리모트 저장소에 해당 프로젝트를 배포하거나 운영서버로 배포하기 위한 설정

```
<distributionManagement>
        <repository>
                <id>저장소아이디</id>
                <url>배포할 저장소 위치</url>
        </repository>
        <site>
        <id>배포서버아이디</id>
                <url>서버URL</url>
        </site>
<distributionManagement>
```

배포 명령 ex) console$>mvn site-deploy, mvn deploy

10) Profiles : 이기종 시스템간의 이식성을 확보하기 위해 빌드 환경 profile 을 등록함.

```
<settings>
...
        <activeProfiles>
           <activeProfile>dev</activeProfile>
        </activeProfiles>
</settings>
```

```
<project>
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-war-plugin</artifactId>
                                <version>2.6</version>
                                <configuration>
                                <warSourceDirectory>${basedir}/webapp${envSuffix}</warSourceDirectory>
                                </configuration>
                        </plugin>
                </plugins>
        <build>
...
        <profiles>
                 <profile>
                        <id>dev</id>
                        <properties>
                                <envSuffix>dev</envSuffix>
                        </properties>
                </profile>
                <profile>
                        <id>stage</id>
                        <properties>
                                <envSuffix>stage</envSuffix>
                        </properties>
                </profile>
        </profiles>
</project>
```

❑ Deploy

개발 완료된 애플리케이션을 web archive file 인 war 형식으로 압축한 후 운영 환경의 WAS 로 배포하는 작업을 말하며, 이때 실제 stage 서버 의 기동/중지 및 배포 작업등을 Maven plugin 을 이용해 수행하거나 소스관리/빌드/배포의 작업이 지속적으로 반복되는 경우, 일정한 스케쥴에 따라 Continuos Integration 작업을 수행할 수 있는 툴로 Jenkins, Hudson 등이 이용될 수 있다.

❑ Maven Tomcat plugin을 이용한 배포

참고(http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/)

메이븐 플러그인을 이용한 배포 작업을 테스트하기 위해 다음과 같은 절차로 작업을 수행할 수 있다.

1. 운영 서버의 관리자 계정 설정

   : %CATALINA_HOME%₩conf₩tomcat-users.xml 수정

```
<tomcat-users>
        <role rolename="manager-gui"/>
        <role rolename="manager-script"/>
        <role rolename="manager-jmx"/>
        <role rolename="manager-status"/>
        <user username="admin" password="java" roles="manager-gui,manager-script,manager-jmx,manager-
       status"/>
</tomcat-users>
```

   톰캣의 manager 앱에서 사용할 수 있는 role의 종류는 manager 컨텍스트의 web.xml 을 통해 확인할 수 있다.

2. 운영 서버 구동
3. 메이븐 플러그인 설정 : tomcat7-maven-plugin

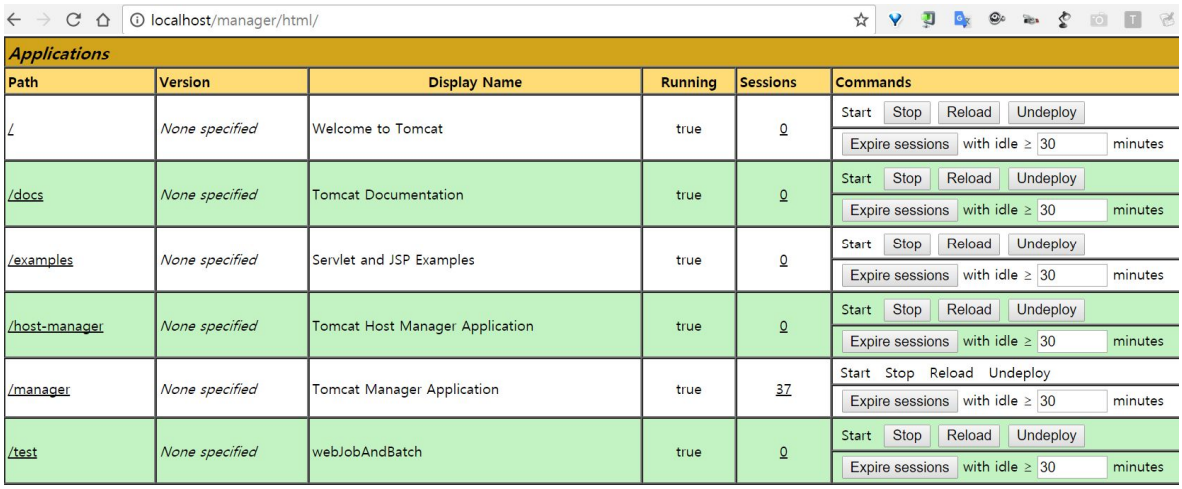
```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
                <url>http://localhost/manager/text</url>
                <username>admin</username>
                <password>java</password>
                <charset>UTF-8</charset>
                <path>/test</path>
        </configuration>
</plugin>
```

4. Plugin goal 의 실행

   : 플러그인에서 지원되는 goal 을 통해 manager 앱의 command 를 이용한 배포를 수행한다.

```
console$> mvn clean install
```

5. manager 앱을 통해 배포 여부 확인 (domain/manager/html)



❑ PMD : source code analyzer

소프트웨어 분석 기법은 분석 시점과 분석 대상을 기준으로 정적 분석과 동적 분석으로 나뉜다. 이때 정적분석이란 소프트웨어를 실행하지 않은 상태에서 소스 코드를 대상으로 한 분석을 말하는 반면, 동적 분석이란 실행 중인 프로그램을 대상으로 한 분석을 말한다. PMD 는 정적 소스 코드 분석기로 정의된 규칙을 기반으로 소스를 검사하여 오류 및 위험 요인을 식별한다. 그러나 PMD 분석 결과에 따른 이슈들은 실제 에러를 유발시킨다기 보다는 비효율적 코드를 의미하기 때문에 어플리케이션은 정상적으로 동작할 수도 있다. 참고로 단위 및 통합 테스트, 시스템 검사 인수 검사, 코드 커버리지 등에는 동적 분석 기법을 사용한다.

주요기능	설명
Syntax Error Inspection	작성한 소스 코드의 Syntax 오류를 검사하는 기능을 제공한다.
Logical Error Inspection	작성한 소스 코드에서 실행 시 발생 가능한 오류를 찾아내는 기능을 제공한다.
Reference Inspection	작성한 소스 코드에서 실행 시 사용되지 않는 부분을 찾아내는 기능을 제공한다.
Reporting	인스펙션을 수행한 결과를 Excel, HTML등이 문서 형식으로 제공하는 기능을 제공한다.
Rule Customizing	Rule을 정의하는 API를 사용하여 새로운 Rule을 정의하여 추가할 수 있는 기능을 제공한다.

1. 이클립스 플러그인 설치(https://github.com/pmd/pmd-eclipse-plugin)

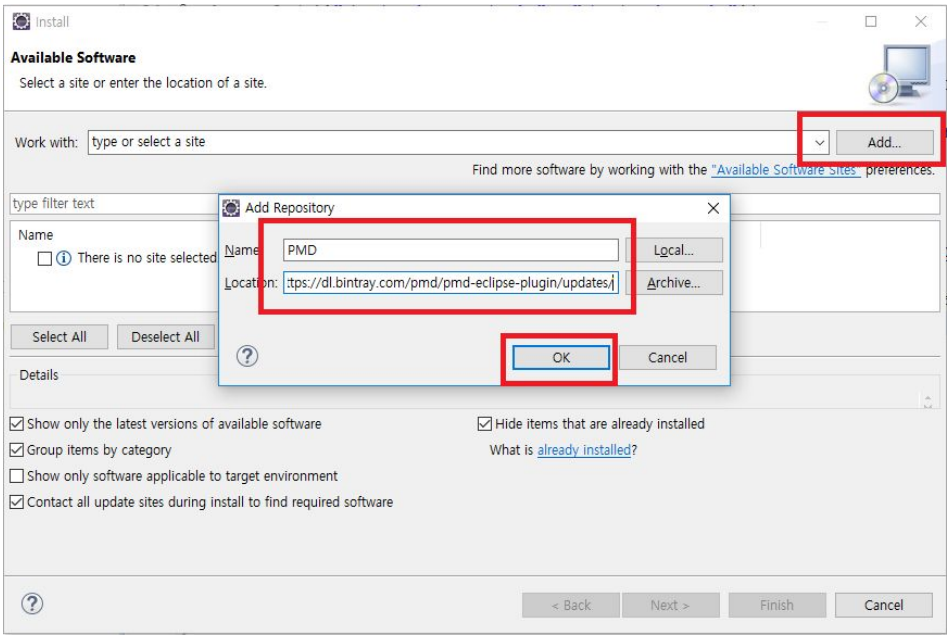
   : 단, Egovframework 개발환경에는 이미 설치되어 있음.

   1) Repository 추가

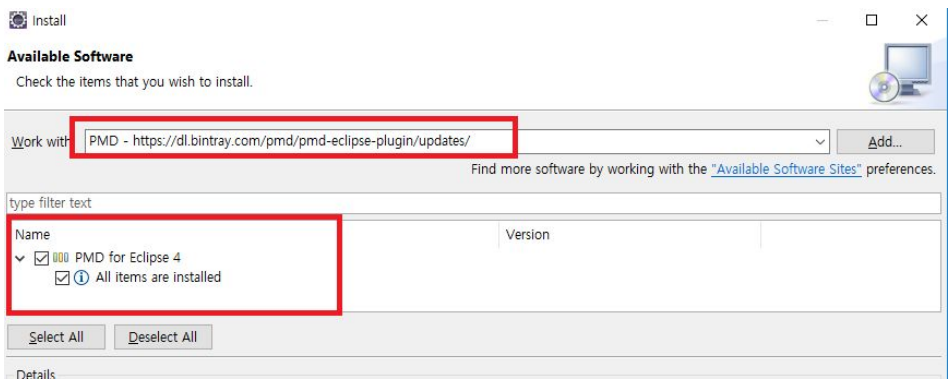
      Help -> Install New Software

      Name : PMD

      Location : https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/

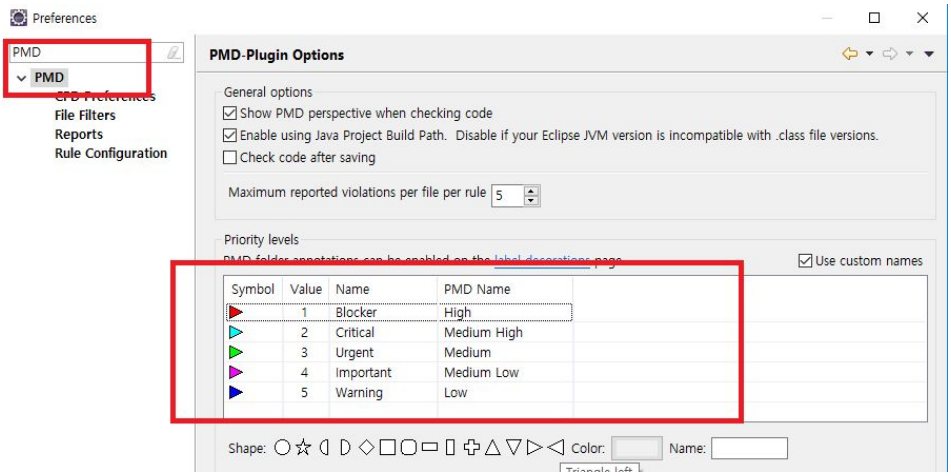


   2) 버전 확인 후 설치



3) 설치 여부 및 적용 룰의 우선순위 확인

Window -> Preferences -> PMD



2. 취약성 점검을 위한 테스트 애플리케이션 설치(WebGoat)

- OWASP(Open Web Application Security Project) : 웹 어플리케이션 보안과 관련된 다양한 이슈들이 다뤄지는 보안 프로젝트로 주로 웹상의 정보 노출, 악성 파일 및 스크립트, 보안 취약점 등을 연구하며 주기적으로 10대 웹 애플리케이션 취약점 등을 발표하는 등의 활동이 이루어진다.

- OWASP TOP 10 : 웹 애플리케이션을 대상으로 보안 취약점 중 빈도가 높고, 크리티컬한 이슈 10 가지를 선정하여, 2004, 2007, 2010, 2013, 2017년을 기준으로 발표하고, 문서를 공개했다.

(https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

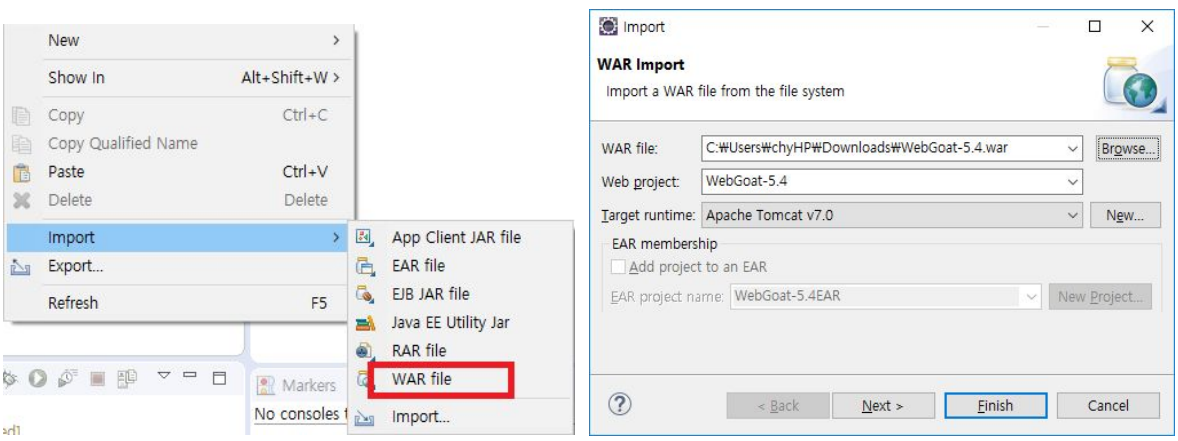
- WebGoat : OWASP 에서 웹애플리케이션의 보안 취약점을 테스트하기 위해 만든 애플리케이션

(참고, https://www.owasp.org/index.php/WebGoat_Getting_Started)

- WebGoat 설치

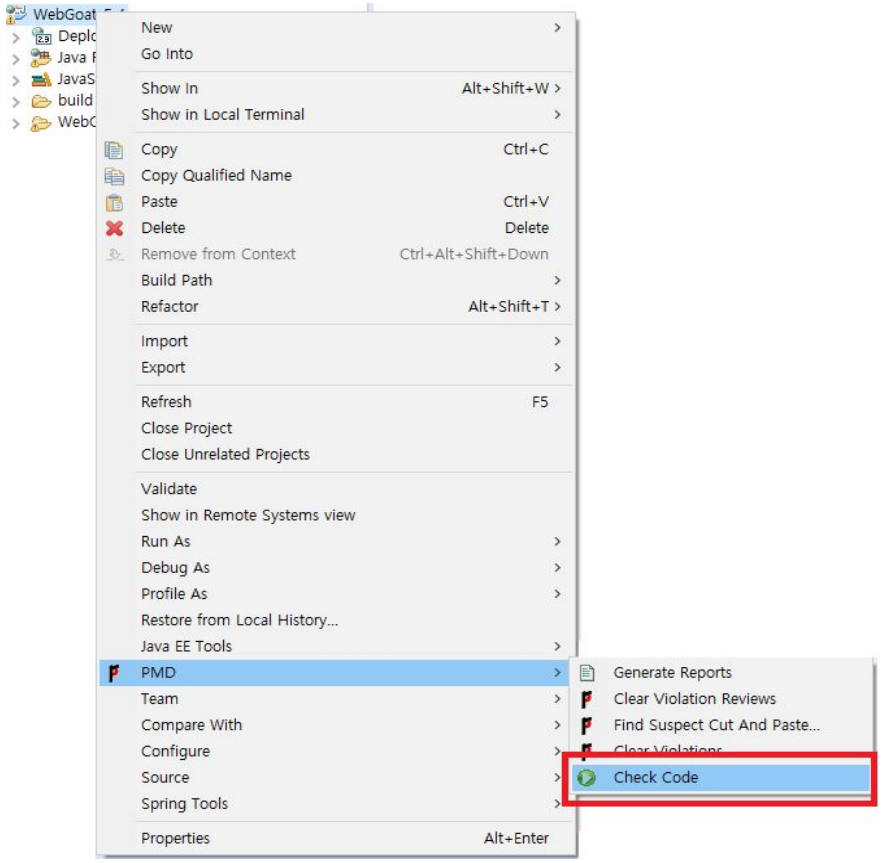
1) https://code.google.com/archive/p/webgoat/downloads -> WebGoat-5.4.war

2) WebGoat 프로젝트 임포트

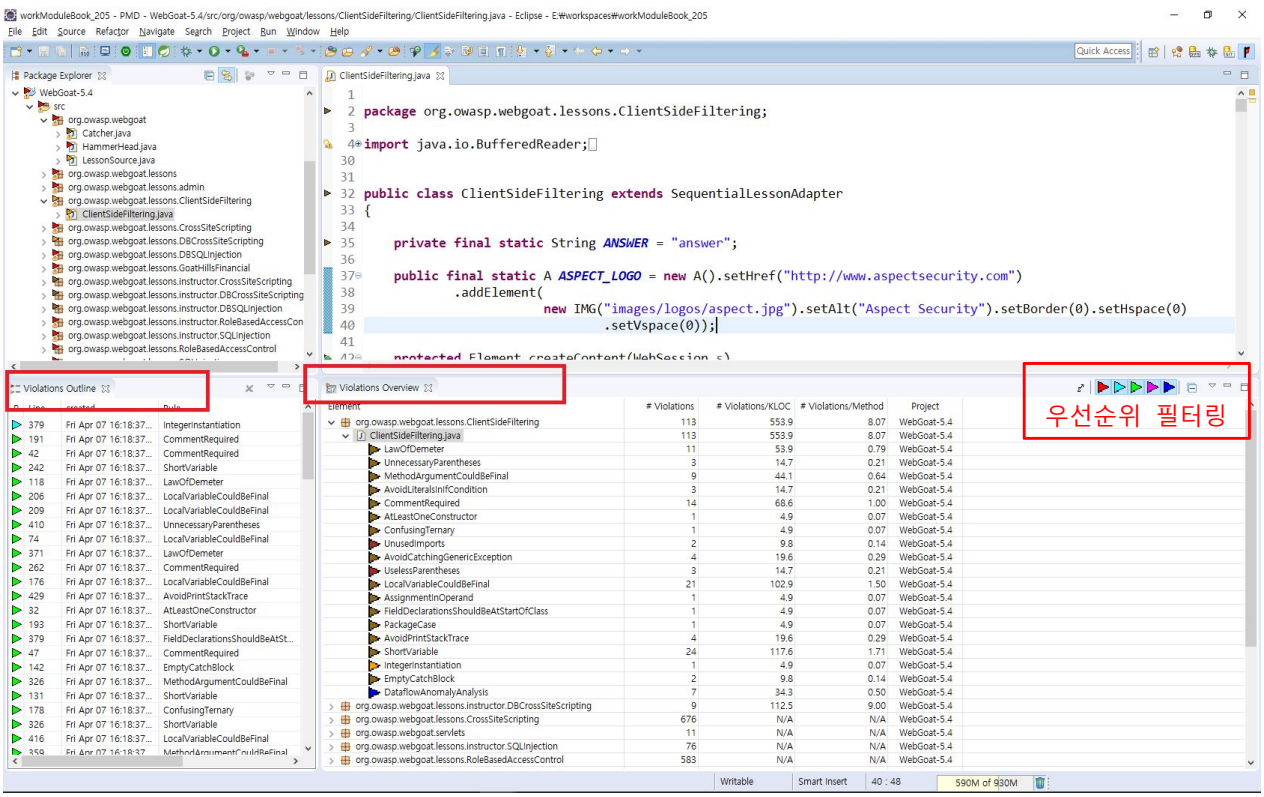


3. PMD의 이용

1) 소스 코드 인스펙션

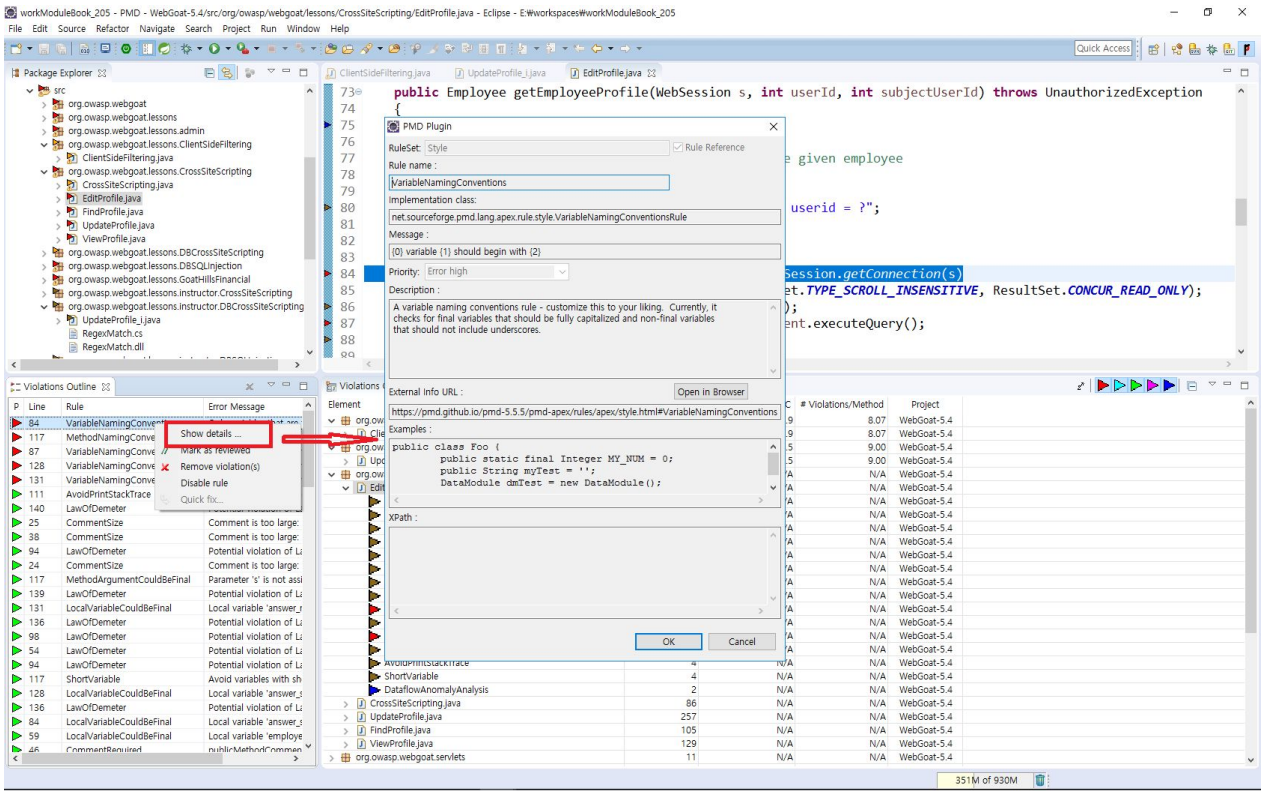


2) 검사 결과 분석

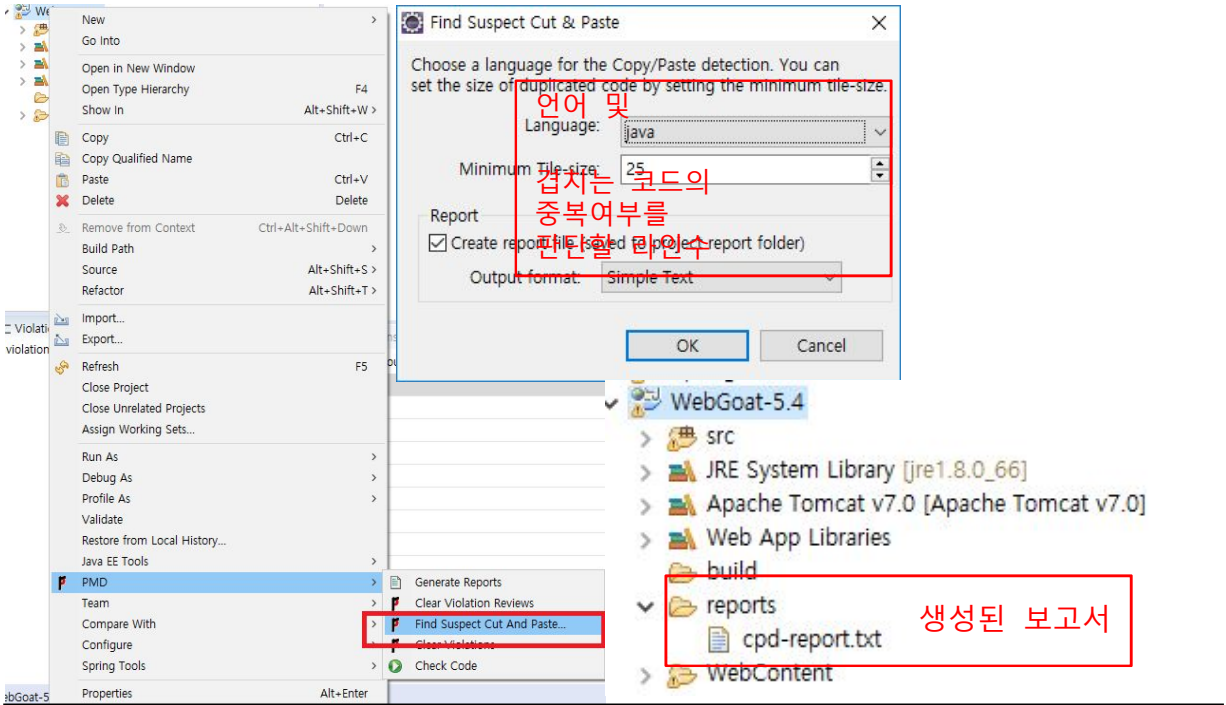


- Violation Outline : 선택한 클래스의 위반 규칙들이 PMD 규칙 우선순위에 따라 지정된 컬럼으로 나타난다.
("라인 작성일 규칙명 메시지")
- Violaiont Overview : 프로젝트 전체 소스를 대상으로 한 패키지 구조의 규칙 위반 클래스

- 위반 사항 상세 보기 : Violation Outline -> each rule -> Show details



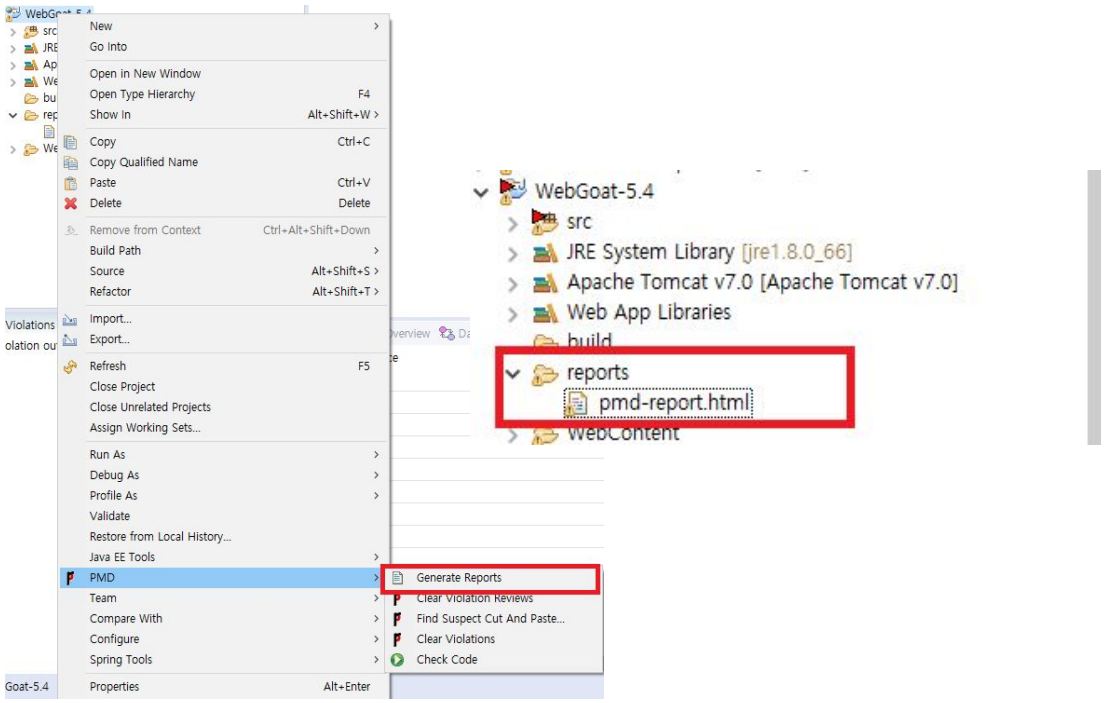
3) CPD(Copy and Paste Detector) : 중복 코드 검사



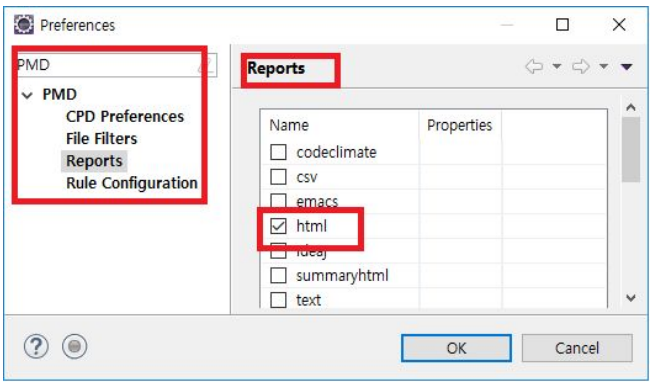

```
2182 =====================================================================
2183 Found a 27 line (196 tokens) duplication in the following files:
2184 Starting at line 150 of E:\workspaces\workModuleBook_205\WebGoat-5.4\src\org\owasp\webgoat\lessons\CrossSiteScripting\FindProfile.java
2185 Starting at line 119 of E:\workspaces\workModuleBook_205\WebGoat-5.4\src\org\owasp\webgoat\lessons\GoatHillsFinancial\FindProfile.java
2186
2187             try
2188             {
2189                 PreparedStatement answer_statement = WebSession.getConnection(s)
2190                         .prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
2191                 answer_statement.setString(1, "%" + pattern + "%");
2192                 answer_statement.setString(2, "%" + pattern + "%");
2193                 ResultSet answer_results = answer_statement.executeQuery();
2194
2195                 // Just use the first hit.
2196                 if (answer_results.next())
2197                 {
2198                     int id = answer_results.getInt("userid");
```

```
2199                    // Note: Do NOT get the password field.
2200                    profile = new Employee(id, answer_results.getString("first_name"), answer_results
2201                            .getString("last_name"), answer_results.getString("ssn"),
2202                            answer_results.getString("title"), answer_results.getString("phone"), answer_results
2203                                    .getString("address1"), answer_results.getString("address2"), answer_results
2204                                    .getInt("manager"), answer_results.getString("start_date"), answer_results
2205                                    .getInt("salary"), answer_results.getString("ccn"), answer_results
2206                                    .getInt("ccn_limit"), answer_results.getString("disciplined_date"), answer_results
2207                                    .getString("disciplined_notes"), answer_results.getString("personal_description"));
2208
2209                    /*
2210                     * System.out.println("Retrieved employee from db: " + profile.getFirstName() +
2211                     * " " + profile.getLastName() + " (" + profile.getId() + ")");
2212                     */
2213                    setRequestAttribute(s, getLessonName() + "." + CrossSiteScripting.EMPLOYEE_ID, Integer.toString(id));
2214 ================================================================
```

4) 보고서 생성



5) 보고서 문서 타입 설정 : window -> Preferneces -> PMD -> Reports



6) 보고서 확인

PMD report

Problems found

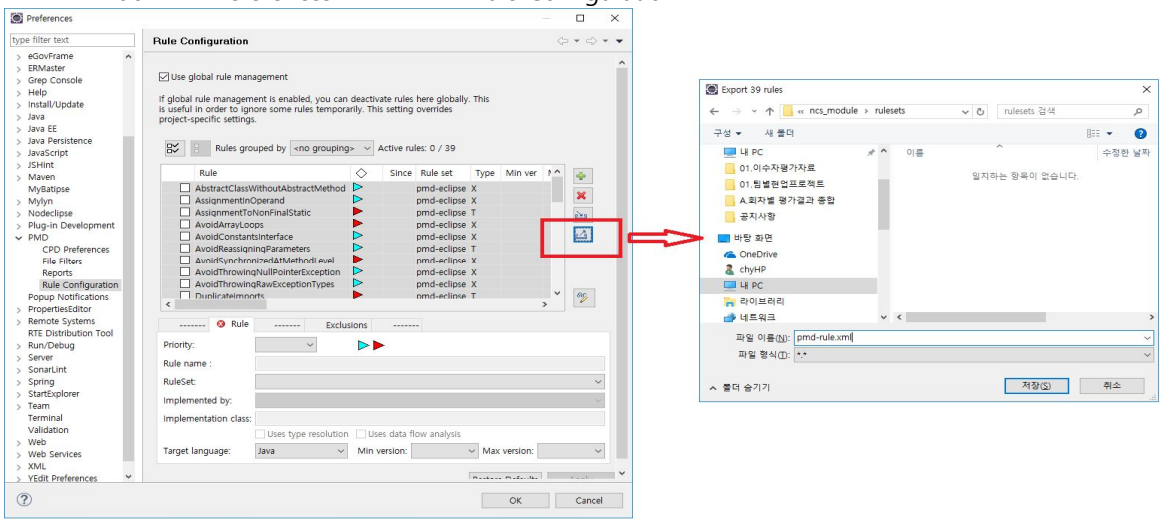
#	File	Line	Problem
1322	src/org/owasp/webgoat/HammerHead.java	77	Variables that are final and static should be all capitals, 'sessionTimeoutSeconds' is not all capitals.
1323	src/org/owasp/webgoat/HammerHead.java	77	Variables that are final and static should be all capitals, 'sessionTimeoutSeconds' is not all capitals.
1324	src/org/owasp/webgoat/HammerHead.java	272	RULE#39:Consider using a logger rather than System.out.print() statement
1325	src/org/owasp/webgoat/HammerHead.java	378	RULE#31:Sometimes expressions are wrapped in unnecessary parentheses
1326	src/org/owasp/webgoat/HammerHead.java	378	RULE#31:Sometimes expressions are wrapped in unnecessary parentheses
1327	src/org/owasp/webgoat/HammerHead.java	450	RULE#31:Sometimes expressions are wrapped in unnecessary parentheses

3. Ruleset 변경

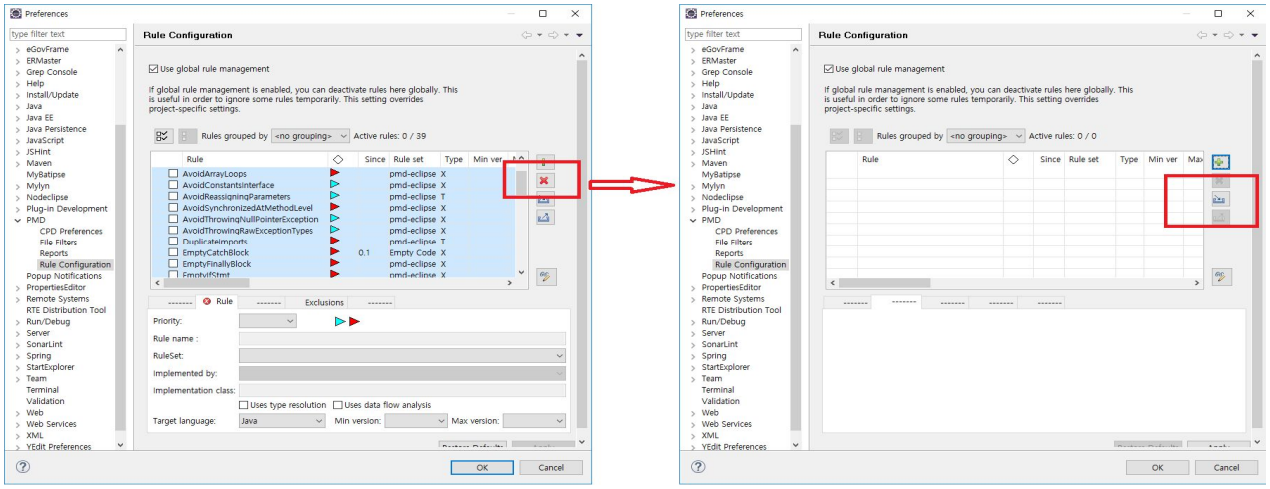
1) 전자정부프레임워크 표준 인스펙션 룰셋 적용

- 기존 룰셋 백업

window -> Preferences -> PMD -> Rule Configuration



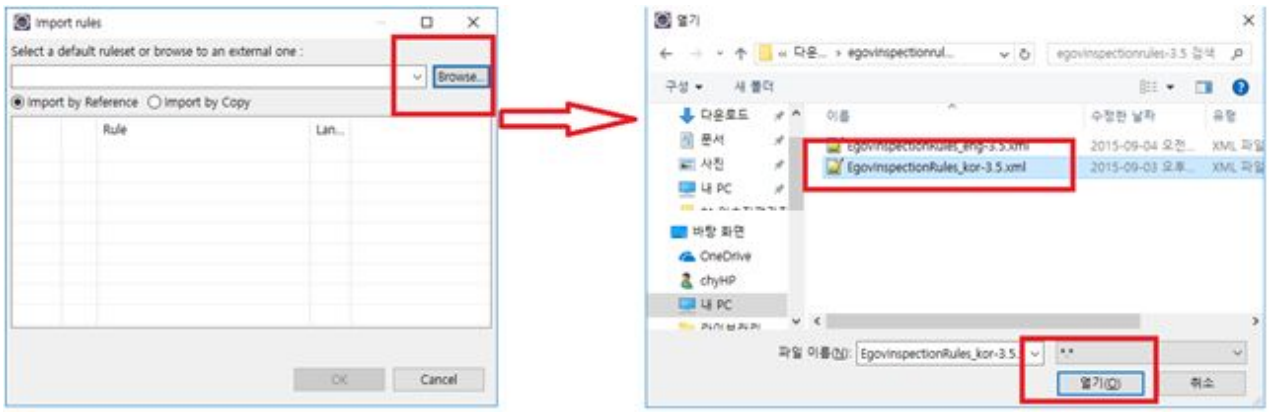
- 기존 룰셋 제거



- 전자정부프레임워크 표준 인스펙션 룰셋 다운로드(개발환경 3.5 이상 버전 사용)
(https://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev2:imp:inspection#전자정부_표준프레임워크_표준_inspection_룰셋_적용하기)

■ 표준 Inspection 룰셋 한글/영문판의 압축파일 : 개발환경 3.5 이상 버전 사용

- 새로운 룰셋 추가



2) Rules Configuration 항목

구성 항목	유형	설명
Rules	그리드	전자정부 표준 Inspection 룰셋을 구성하고 있는 전체 룰의 목록을 표시하는 그리드
Remove rule	버튼	Rules 그리드에서 선택한 항목의 룰을 삭제
Edit rule	버튼	Rules 그리드에서 선택한 항목의 룰을 수정
Add rule	버튼	Rules 그리드의 룰셋에 새로운 룰을 추가
Import rule set...	버튼	하나 이상의 룰들을 파일단위로 룰셋에 추가
Export rule set...	버튼	하나 이상의 룰들을 추출하여 외부 파일로 저장
Clear all	버튼	Rules 그리드에 나타난 모든 룰들을 삭제
Rule Designer	버튼	XPath 기반의 새로운 룰을 만들기위한 외부 프로그램 실행(PMD 플러그인 제공)
Rule properties	그리드	개별 룰에 대한 Key-Value 형태의 속성 목록을 표시, 주로 룰에 대한 XPath 쿼리(XQuery)를 표시
Add property...	버튼	개별 룰에 새로운 property 추가
Exclude patterns	그리드	전체 룰셋을 프로젝트에 적용할 때, 예외 대상 패턴을 정의한 목록을 표시
Include patterns	그리드	전체 룰셋을 프로젝트에 적용할 때, 포함 대상 패턴을 정의한 목록을 표시
Add Exclude Patterns	버튼	룰셋 적용 예외 대상 패턴을 새로이 추가
Add Include Patterns	버튼	룰셋 적용 포함 대상 패턴을 새로이 추가

3) Basic Rules

Rule Name	Rule Description
EmptyCatchBlock	비어있는 catch 블록을 찾는다.
EmptyIfStmt	조건절 블록이 비어있는 조건문을 찾는다.
EmptyWhileStmt	반복 구문이 비어있는 while문을 찾는데, 타이밍 조절용으로 쓰이는 while구문에서는 Thread.sleep() 을 사용하는 것이 낫지만, 종료 구문이 지나치게 많은 경우 많은 실행 비용을 발생시킨다.
EmptyTryBlock	비어있는 try 블록을 찾는다.
EmptyFinallyBlock	비어있는 finally 블록을 찾는다.
EmptySwitchStatements	비어있는 switch 블록을 찾는다.
JumbledIncrementer	
ForLoopShouldBeWhileLoop	while 문으로 변환하는 것이 더 나은 for 문을 찾는다.
UnnecessaryConversionTemporary	기본형을 문자로 변환시 불필요한 String 임시 변수가 사용되는 코드를 찾는다.
OverrideBothEqualsAndHashcode	equals 나 hashCode 메소드를 오버라이딩하는 경우 반드시 두 메소드 모두를 재정의해야 한다.
DoubleCheckedLocking	멀티 쓰레딩 환경에서 객체 생성 지연 및 싱글턴 패턴을 사용하기 위한 코드에서 객체 중복 확인을 위한 코드가 잘못된 경우를 찾는다.
ReturnFromFinallyBlock	finally 구문에서 리턴값이 반환되는 메소드를 찾는다.
EmptySynchronizedBlock	쓰레드 동기화를 위한 synchronized 구문이 비어있는 블록을 찾는다.
UnnecessaryReturn	void 리턴타입의 메소드와 같이 불필요한 return 구문이 사용되는 코드를 찾는다.
EmptyStaticInitializer	비어있는 static 초기화 블록을 찾는다.
UnconditionalIfStatement	조건절에 단순 boolean 값이 들어가는 코드를 찾는다.
EmptyStatementNotInLoop	반복문 밖에서 비어있는 구문을 찾는다.
BooleanInstantiation	Boolean.TRUE 등의 상수 대신 Boolean 객체를 생성하는 코드를 찾는다.
UnnecessaryFinalModifier	class 에 final 을 사용하는 경우, 해당 클래스의 모든 멤버들이 자동으로 상수가 되기때문에 불필요한 final 이 지정되는 경우를 찾는다.
CollapsibleIfStatements	하나의 if 조건문으로 변경 가능한 중첩 if 문을 찾는다.
UselessOverridingMethod	메소드 상속시 어노테이션을 재정의 하는 등의 제한적인 경우가 아님에도 상위 메소드를 재호출하는 구문만을 갖는 코드를 찾는다.
ClassCastExceptionWithToArray	List.toArray() 메소드의 리턴값을 배열로 캐스팅하는 코드를 찾는다.

AvoidDecimalLiteralsInBigDecimal Constructor	BigDecimal 객체 생성시 숫자 인자로 생성하는 경우, double로 처리하기 때문에 의도하지 않은 실수가 생기기때문에 문자열 인자로 생성해야 하는데, 그렇지 않은 코드를 찾는다.
UselessOperationOnImmutable	String, BigDecimal 등의 이뮤터블 객체는 스스로 상태를 변경할 수 없기 때문에 연산의 결과로 새로운 객체를 생성한다. 이때 연산의 결과를 사용하지 않는 코드를 찾는다.
MisplacedNullCheck	null 체크 연산의 위치가 잘못된 코드를 찾는다. and/or 연산자의 경우, 일항의 피연산자 값에 따라 이항의 피연산자 체크 여부가 달라지기 때문에 null 체크시의 위치가 코드에 영향을 미치기 때문이다.
UnusedNullCheckInEquals	equals 를 이용한 객체의 상태 비교전에 null체크를 통해 안정성을 확보하지 않는 코드를 찾는다.
AvoidThreadGroup	쓰레드에 안전하지 않은 코드를 포함하고 있는 ThreadGroup을 사용하는 코드를 찾는다.
BrokenNullCheck	and/or 연산과 null 체크 연산을 동시에 사용하는 경우 위치에 따라 논리 연산자의 종류가 달라져야 하는데, 그 위치와 연관된 연산자의 종류를 잘 못 사용한 코드를 찾는다.
BigIntegerInstantiation	BigInteger 에 상수로 이미 선언되어 있는 인스턴스들을 굳이 생성하는 코드를 찾는다. BigDecimal 과 유사함.
AvoidUsingOctalValues	0으로 시작하는 8진수 값을 사용하는 코드를 찾는다.
AvoidUsingHardCodedIP	IP 주소가 하드코딩된 코드를 찾는다.
CheckResultSet	ResultSet의 네비게이션 메소드(next, previous, last, first) 를 사용할 때 리턴값을 확인하지 않는 코드를 찾는다.
AvoidMultipleUnaryOperators	단항 연산자를 복합적으로 사용시 적절한 괄호처리가 없는 코드를 찾는다.
EmptyInitializer	초기화 코드 블록이 비어있는 경우를 찾는다.
DontCallThreadRun	쓰레드를 시작할때 start대신 run 메소드를 호출하는 구문을 찾는다.

- **참고 : http://pmd.sourceforge.net/pmd-4.3.0/rules/index.html#Basic_Rules**

- EmptyCatchBlock

```
try{
        InputStream in = new FileInputStream("path");
}catch(IOException e){
      // 이곳이 비어있는 경우 발생
}
```

- EmptyIfStmt

```
if(x==null){
      // 이곳이 비어있는 경우 발생
}
```

- EmptyWhileStmt

```
while(x==null){
      // 이곳이 비어있는 경우 발생
}
```

- EmptyTryBlock

```
try{
       // 이곳이 비어있는 경우 발생
}catch(Exception e){
       e.printStackTrace();
}
```

- EmptyFinallyB lock

```
try{
        Statement 1… N..;
}finally{
       // 이곳이 비어있는 경우 발생
}
```

- EmptySwitchStatements

```
switch(x){
    // 주석처리 등으로 기존의 실행 코드가 실행되지 않는 경우. 비어있는 블록으로 판단.
}
```

- JumbledIncrementer

```
for(int i=0; i<10; i++){
        for(int k=0; k<10; i++){
            // 실행 구문
        }
}
```

```
for(int i=0; i<10; i++){
        for(int k=0; k<10; k++){
            // 중첩 반복시 변수를 명확히 구분
        }
}
```

- ForLoopShouldBeWhileLoop

```
for(; true; ){
    //실행 구문
}
```

```
while(true){
    //실행 구문
}
```

- UnnecessaryConversionTemporary

```
String foo = new Integer(x).toString();
return foo;
```

```
return Integer.toString(x);
```

- OverrideBothEqualsAndHashcode

```
public class Baz{
        public boolean equals(Object o){
            // 객체 비교
        }
}
```

```
public class Baz{
        public boolean equals(Object o){
            // 객체 비교
        }
        public int hashcode(){
            // 해시코드 반환
        }
}
```

- DoubleCheckedLocking
  a. 객체 생성 지연 코드

```
public class MySingleton{
        private static MySingleton instance = null;
        public static MySingleton getInstance(){
            If(instance ==null)
                instance = new MySingleton();    -- 1
            return instance;
        }
        private MySingleton(){ }
}
```

멀티 쓰레드 환경에서 null 체크를 하는 코드가 동기화 되지 않는 경우, getter를 호출하는 각 쓰레드가 각기 다른 객체를 반환받게 될 위험이 존재한다.

b. 동기화를 고려한 객체 생성 지연

```
public class MySingleton{
        private static MySingleton instance = null;
        public static synchronized MySingleton getInstance(){
            If(instance ==null)
                instance = new MySingleton();    -- 1
            return instance;
        }
        private MySingleton(){ }
}
```

객체를 생성하는 1번 코드만을 동기화하면 되므로 메소드 전체를 동기화할때 발생하는 낭비를 막기위해 동기화가 필요한 코드만을 묶는다.

c. 동기화 블록 최소화

```
public class MySingleton{
        private static MySingleton instance = null;
        public static MySingleton getInstance(){
                if(instance ==null)
                        synchronized(MySingleton.class){
                            instance = new MySingleton();    -- 1
                         }
                return instance;
        }
        private MySingleton(){ }
}
```

한 쓰레드가 동기화된 코드 실행을 종료하면, 대기상태의 다른 쓰레드가 해당 코드에 접근하게 되고, null 체킹을 하지않은채 다시 객체를 생성하게 된다.

d. DCL(Double Checked Locking)

```
public class MySingleton{
        private static MySingleton instance = null;
        public static MySingleton getInstance(){
                if(instance ==null)
                        synchronized(MySingleton.class){
                                if(instance ==null)
                                        instance = new MySingleton();    -- 1
                         }
                return instance;
        }
        private MySingleton(){
                 statement 1….  N;
         }
}
```

동기화 블록 내에 진입한 쓰레드가 실제 객체의 null 여부를 한번 더 판단하기 때문에 c 단계에서 발생하는 문제를 해결할 수는 있다. 그러나 만일 MySingleton의 생성자내에서 많은 작업을 해야 하는 경우, 생성자가 모든 작업을 완료하기 전에 not-null 상태가 될수 있다. 예를 들어,
1. Thread1 이 getInstance 메소드에 진입한다.
2. thread1은 동기화 블록으로 들어간다. (is null)
3. Thread1이 non-null 인스턴스를 만든다. (아직 생성자는 호출되기 전이다)
4. Thread1 이 Thread 2 에 의해 선점당한다.
5. Thread2가 null 여부를 판단할때 아직 생성자가 호출되기 전이지만, 레퍼런스가 존재하기 때문에 not-null 로 판단되고, 생성자가 완료되지 않은 반쪽짜리 객체를 반환받는다.
6. 다시 Thread1이 동작하면서 생성자의 나머지 부분이 실행되고, 객체의 초기화를 완료하고 레퍼런스를 반환받는다.
이러한 경우, Thread2가 반환받은 객체는 온전하지 못한 객체가 되는 out-of-order write 문제가 발생한다.

e. out-of-order write 를 해결한 singletonholder 패턴

```
public class MySingleton{
        private MySingleton(){
                 statement 1….  N;
         }
        private static class SingletonHolder{
                public static final MySingleton INSTANCE = new MySingleton();
        }
        public static MySingleton getInstance(){
                return SingletonHolder.INSTANCE;
        }
}
```

- ReturnFromFinallyBlock

```
public class Bar{
        public String foo(){
                try{
                        throw new Exception("Exception");
                }catch(Exception e){
                        throw e;
                }finally{
                        // 발생한 예외에 대한 정보가 소실됨.
                        return "return Value";
                }
        }
}
```

```
public class Bar{
        public String foo() throws Exception{
                try{
                        throw new Exception("Exception");
                        return "return Value";
                }catch(Exception e){
                        throw e;
                }finally{
                        // 예외 처리 여부와 관계없는 필수 처리 구문
                }
        }
}
```

- EmptySynchronizedBlock

```
public class Foo{
        public void bar(){
                synchronized(this){
                        // 이곳이 비어있는 경우 발생
                }
        }
}
```

- UnnecessaryReturn

```
public class Foo{
        public void bar(){
                int num = 23;
                return;
        }
}
```

- EmptyStaticInitializer

```
public class Foo{
        static {
        // 이곳이 비어있는 경우 발생
        }
}
```

- UnconditionalStatement

```
public void sample(){
        if(true){       // 실행 구문          }
}
```

- EmptyStatementNotInLoop

```
public class Bar{
        public String foo(){
                ;
                // 실행구문
        }
}
```

- BooleanInstantiation

```
Boolean bar = new Boolean("true");
Boolean baz = Boolean.valueOf(false);
```

```
Boolean bar = Boolean.TRUE;
Boolean baz = Boolean.FALSE;
```

- UnnecessaryFinalModifier

```
public final class Foo{
        public fnial String foo(){ // final 클래스의 모든 멤버는 final 이기때문에 굳이 사용할 필요 없음.
                // 실행구문
        }
}
```

- CollapsibleIfStatements

```
public void foo(){
        if(x){
                if(y){
                        // 실행 구문
                }
        }
}
```

```
public void foo(){
        if(x & y){
                // 실행 구문
        }
}
```

- UselessOverridingMethod

```
public void foo(String bar){
        super.foo(bar); // 굳이 재정의 할 필요 없는 구문
}

@TestAnnotation
public void foo(String bar){
        super.foo(bar); // 어노테이션을 설정해야 하는 경우에는 재정의가 필요함.
}
```

- ClassCastExceptionWithToArray

```
Collection  c = new ArrayList();
Integer num = new Integer(1);
c.add(num);

Integer[] a = (Integer[]) c.toArray(); // ClassCastException 발생

Integer[] b = (Integer[]) c.toArray(new Integer[c.size]);
```

- AvoidDecimalLiteralsInBigDecimalConstructor

```
BigDecimal bd 1= new BigDecimal(1.1);
System.out.println(bd1);    // 1.100000000000000088817841970012523233890533447265625
System.out.println(bd1.doubleValue());  // 1.1

BigDecimal bd 2= new BigDecimal("1.1");
System.out.println(bd2);    // 1.1
System.out.println(bd2.doubleValue());  // 1.1
```

- UselessOperationOnImmutable
  수정불가능한 객체(BigDecimal, String, Integer 등)에 대한 연산의 결과로 도출된 새로운 객체를 사용하지 않는 경우 발생.

```
BigDecimal bd = new BigDecimal("1.1");
bd.add(new BigDecimal("2.2");  // BigDecimal 은 immutable 객체로, add 메소드의 리턴값으로 연산
                               // 결과를 가진 새로운 객체가 반환된다. 이 새로운 객체를 사용하지
                               //  않는 경우 발생.
bd = bd.add(new BigDecimal("2.2");
```

- MisplacedNullCheck

```
String bar = method();
// null 인지 모르는 상태에서 잘못된 널 체크
if(bar.equals(baz) && bar!=null){
        // 실행구문
}
bar = "test";
// 널이 될수 없는 상태에서 불필요한 널체크
if(bar!=null && bar.equals(baz)){
        //  실행구문
}
```

```
String bar = method();
If( bar!=null && bar.equals(baz)){
        // 실행구문
}
bar = "test"
if(bar.equals(baz)){
        // 실행구문
}
```

- UnusedNullCheckInEquals
  특정 객체에 대해 null 체크를 했다면, 다른 객체의 equals 메소드를 호출하면서 해당 객체를 아규먼트로 넘길 것이 아니라, 해당 객체의 equals 메소드를 호출하여 상태 비교를 해야함.

```
// 널체크를 하지 않아도 예외가 발생하지 않음.
if( bar!=null && baz.equals(bar)){
        // 실행구문
}
```

```
If( bar!=null && bar.equals(baz)){
        // 실행구문
}
```

- AvoidThreadGroup
  쓰레드에 안전하지 않은 환경에서 ThreadGroup 을 사용하는 것은 피해야 함.
- BrokenNullCheck

```
// 널체크시  || / && 연산자를 잘 못 사용한 경우
if( bar!=null || bar.equals(baz)){ //
}
if( bar==null && bar.equals(""){ //
}
```

```
If( bar!=null && bar.equals(baz)){
}
If( bar==null || bar.equals(baz)){
}
```

- BigIntegerInstantiation

```
// BigInteger.ONE, BigInteger.TEN, BigInteger.ZERO 등의 상수가 존재하는 경우, 직접 BigInteger
// 객체를 생성하지 말 것.
BigInteger bi = new BigInteger(1);
BigInteger bi = BigInteger.ONE;
```

- AvoidUsingOctalValues

```
// 8진수 사용은 제한적으로.
int num = 012;
int num = 10;
```

- AvoidUsingHardCodedIP

```
String ip = "192.168.205.5";
        HttpGet hGet = new HttpGet(ip);
```

```
public static void main(String[] args){
        HttpGet hGet = new HttpGet(args[0]);
}
```

- CheckResultSet
  ResultSet 의 네비게이션 메소드(next, previous, first, last) 등의 메소드를 사용할때는 반드시 반환값을 확인해야 함.

- CheckResultSet
  ResultSet 의 네비게이션 메소드(next, previous, first, last) 등의 메소드를 사용할때는 반드시 반환값을 확인해야 함.

```
String sql = "SELECT MEM_NAME FROM MEMBER WHERE MEM_ID = 'ab0012'";
ResultSet rs = stmt.executeQuery(sql);
rs.next();
String mem_name = rs.getString("MEM_NAME");
if(rs.next()){
        String mem_name = rs.getString("MEM_NAME");
}
```

```
String ip = "192.168.205.5";
HttpGet hGet = new HttpGet(ip);
```

```
public static void main(String[] args){
        HttpGet hGet = new HttpGet(args[0]);
}
```

- AvoidMultipleUnaryOperators
  단항연산자는 중첩해서 사용하지 않도록 함.
- EmptyInitializer

```
public class Foo{
        // 초기화 블록 내에서 아무 작업도 하지 않는 경우에 발생
        static{}
        {}
}
```

- DontCallThreadRun
  쓰레드를 제어하는 과정에서 run() 메소드를 직접 호출하지 않도록 함.

4) Braces Rules

Rule Name	Rule Description
IfStmtsMustUseBraces	조건에 따라 수행할 구문이 단문일지라도 조건문에는 괄호를 사용함
WhileLoopsMustBraces	while 문의 반복구문이 단문일지라도 반복구문에는 괄호를 사용함
IfElseStmtsMustUseBraces	if~else 구문에는 괄호를 사용함
ForLoopsMustUseBraces	for 반복문의 구문이 단문일지라도 괄호를 사용함.

5) Clone Implementation Rules

Rule Name	Rule Description
ProperCloneImplementation	객체의 clone 메소드를 재정의시에는 super.clone 코드를 통해 객체를 생성함.
CloneThrowsCloneNotSupportedException	Cloneable 의 구현체가 아닌 객체에 대해 clone 메소드를 호출시 CloneNotSupportedException 이 발생하게 되므로, clone 메소드를 호출하는 경우 해당 예외에 대한 처리가 필요함.
CloneMethodMustImplementCloneable	clone 메소드는 Cloneable 의 구현체에 대해서만 호출 가능함.

```
// CloneMethodMustImplementCloneable 통과
public class Foo implements Cloneable{
        private String type;
        private String value;
        // CloneThrowsCloneNotSupportedException 통과
        public Object clone() throws CloneableNotSupportedException{
        // ProperCloneImplementation 통과
                Foo foo = (Foo)super.clone();
                foo.type = type;
                foo.value = value;
                return foo;
        }
}
```

6) Code Size Rules

Rule Name	Rule Description
NPathComplexity	한 메소드당 비순환 실행 구문의 적정 한계는 200개로 이를 넘지않도록 함
ExcessiveMethodLength	한 메소드에 너무 많은 기능을 부여하여 소스가 길어지지 않도록 함
ExcessiveParameterList	메소드 전달 파라미터가 너무 많지 않도록 하고, 필요한 경우, 객체(VO)를 활용함
ExcessiveClassLength	클래스 소스가 너무 길어지지 않도록 적절한 모델링이 필요함
CyclomaticComplexity	순환복잡도는 일반적으로 if, while, for, swith~case등의 결정포인트를 통해서 결정되며, 1~4 : 낮은복잡도, 5~7 : 중간복잡도, 8~9 : 높은 복잡도, 11 : 매우 높은 복잡도를 의미함.
ExcessivePublicCount	한 클래스에 너무 많은 public 메소드나 속성들이 존재한다면 테스트 비용이 증가되는 상황을 고려하여, 좀 더 명확한 기능을 갖도록 클래스를 분리함.
TooManyuFields	한 클래스가 너무 많은 필드를 갖고 있다면, 해당 필드들의 특성에 따라 재모델링하여 그룹화된 객체로 묶어야 함.
NcssMethodCount	메소드 내의 NCSS(Non Commenting Source Statements) 구문을 카운트함.
NcssTypeCount	클래스 내의 NCSS 코드를 카운트함.
NcssConstructorCount	생성자 내의 NCSS 구문을 카운트함.
TooManyMethods	클래스의 복잡도를 낮추고 결합도를 높이기 위해 한 클래스 내에서 메소드 수는 기본적으로 10개를 넘지 않도록 함.

7) Controversial Rules

Rule Name	Rule Description
UnnecessaryConstructor	불필요한 생성자를 찾아냄
NullAssignment	변수의 초기화 코드를 제외하고 변수에 null 값을 할당하는 경우를 찾아냄
OnlyReturn	한 메소드 내에서 return 구문은 한번만 사용하도록 함.
UnusedModifier	의미 없는 modifier 를 찾아냄 ex) 인터페이스 내의 abstract, 생성자의 static 등.
AssignmentInOperand	피연산자에 사용된 할당연산자를 찾아냄. 이런 경우, 연산자의 우선순위를 고려해야함.
AtLeastOneConstructor	적어도 하나 이상의 생성자를 정의하도록 함.
DontImportSun	sum.** 패키지는 임포트 하지 않도록 함.
SuspiciousOctalEscape	escape sequence : 8진수형 확장 특수문자로 오인받을 만한 구문은 피하도록 함. ex) "₩038" 는 특수문자와 숫자 8이 출력됨.
CallSuperInConstructor	생성자에서는 상위의 생성자를 호출하도록 함. ex) super();
UnnecessaryParentheses	불필요한 괄호는 생략하도록 함.
DefaultPackage	기본 패키지의 private 레벨 대신 명확한 패키지 범위를 사용하도록 함.
BooleanInversion	불린 데이터를 도치할때 논리연산 대신 비트 연산을 사용하도록 함.
DataflowAnomalyAnalysis	지역변수의 흐름을 분석하고 추적하기 위한 룰로 다음과 같은 타입들로 분류됨. UR(anomaly) : 정의되지 않은 변수. DU(anomaly) : 최근에 정의된 변수를 null로 해제. DD : 최근에 정의된 변수를 재정의함. ex) UR : foo(buz); DU : Integer buz = 4; foo(buz); buz=null; DD : Integer buz = 4; foo(buz); buz = 2;
AvoidFinalLocalVariable	지역변수의 상수형은 피하도록 함.
AvoidUsingShortType	산술연산시 short 형의 사용은 int 로 대신하도록 함.
AvoidUsingVolatile	JVM의 메모리 관리 구조에 대해 명확한 지식이 없다면 volatile 은 사용하지 않도록 함.
AvoidUsingNativeCode	JNI 코드는 제한적으로 사용하도록 함.
AvoidAccessiblilityAlteration	캡슐화되어있는 객체에 대해 setAccessible 과 같은 메소드를 사용하여 강제로 접근 제한을 풀지않도록 함.
DoNotCallGarbage	GC 는 JVM의 영역, 기준없이 gc(), runFilalization() 메소드들을 사용하지 않도록 함.
AvoidLiteralsInIfCondition	조건문에 하드코딩된 리터럴이나 문자열을 사용하지 않도록 함. ex) if(num == 24) { } -> 24를 static 이나 private 변수로 선언할 것.
UseConcurrentHashMap	java5 부터 병행처리에 지원되는 ConcurrentHashMap 을 활용하도록 함.

8) Import Statement Rules

Rule Name	Rule Description
DuplicateImports	중복 import 구문은 피하도록 함.
DontImportJavaLang	java.lang 패키지 import 피하도록 함.
UnusedImports	사용되지 않는 import구문은 삭제하도록 함.
ImportFromSamePackage	동일 패키지의 클래스는 import 하지 않도록 함.
TooManyStaticImports	static import 구문을 지나치게 남발하지 않도록 함.

4. Maven 과 PMD 연동 : maven-pmd-plugin

1) 플러그인 등록

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.7</version>
</plugin>
```

① 타겟 JDK 버전 설정

```
<configuration>
        <targetJDK>1.6</targetJDK>
</configuration>
```

② Ruleset 커스터마이징 : pmd 플러그인이 기본으로 사용하는 룰셋은 java/javascript/jsp 에 대한 검증룰들로, basic.xml, empty.xml, imports.xml, unnecessary.xml, unusedcode.xml 다섯개의 기본 룰셋이 사용된다. 이외의 룰셋을 사용하도록 하기 위해서 rulesets 와 ruleset 엘리먼트를 사용함

```
<configuration>
        <rulesets>
                <!-- 추가 룰셋 지정 -->
                <ruleset>/rulesets/java/naming.xml</ruleset>
                <!– 커스텀 룰셋 지정 -->
                <ruleset>d:\rulesets\strings.xml</ruleset>
        </rulesets>
</configuration>
```

2) 플러그인을 이용한 소스 검증 ex) mvn pmd:pmd

① pmd:pmd (pmd:check) – 룰셋에 따라 소스 검증 후 target/pmd.xml 로 보고서 생성

② pmd:cpd (pmd:cpd-check) – CPD 실행 후 target/cpd.xml 에 보고서 생성