Spring Convention

태그	네이버 캠퍼스 핵데이 Java 코딩 컨벤션
출처	https://naver.github.io/hackday-conventions-java/

1. 이름 (Naming)

1.1. 식별자에는 영문/숫자/언더스코어만 허용

변수명, 클래스명, 메서드명 등에는 영어와 숫자만을 사용한다. 상수에는 단어 사이의 구분을 위하여 언더스코어(`_`)를 사용한다. 정규표현식 `[^A-Za-z0-9_]`에 부합해야 한다.

1.2. 한국어 발음대로의 표기 금지

식별자의 이름을 한글 발음을 영어로 옮겨서 표기하지 않는다. 한국어 고유명사는 예외이다.

- 나쁜 예 : `moohyungJasan` (무형자산)
- 좋은 예 : `intangibleAssets` (무형자산)

1.3. 대문자로 표기할 약어 명시

클래스명, 변수명에 쓰일 단어 중 모든 글자를 대문자로 표기할 약어의 목록을 프로젝트별로 명시적으로 정의한다.

약어의 대소문자 표기는 JDK의 클래스나 라이브러리들 사이에서도 일관된 규칙이 없다. `javax.xml.bind.annotation.XmlElement`처럼 약어를 소문자로 표기하기도 하고, `java.net.HttpURLConnection`처럼 한 클래스 안에서 단어별로 다르게 쓰기도 했다. 그러나 단일 프로젝트에서는 규칙이 명확하지 않으면 같은 단어의 조합을 다른 이름으로 표기할 수 있어서 혼동을 유발한다.

약어가 클래스명에서 대문자로 들어가면 단어 간의 구분을 인지하기에 불리하다. 약어가 연속된 경우 더욱 가독성을 해친다. 예를 들면 XMLRPCHTTPAPIURL과 같은 경우이다. 그래서 기본 정책으로는 약어의 중간단어를 소문자로 표기하고 프로젝트별로 모두 대문자로 표기할 약어의 목록을 명시하는 방식이 가독성을 높이고 규칙을 단순화하는데 유리하다. 즉 프로젝트 내에서 정의한 단어 목록이 없다면 'XmlRpcHttpApiUrl'과 같이 쓴다.

좋은 예

HTTP + API + URL 의 클래스 이름의 경우

- 대문자로 표기할 약어의 목록을 정의하지 않는 경우 : HttpApiUrl
- API만 대문자로 표기할 약어의 목록에 있을 경우 : HttpAPIUrl
- HTTP, API, URL이 대문자로 표기할 약어의 목록에 있을 경우 : HTTPAPIURL

1.4. 패키지 이름은 소문자로 구성

패키지 이름은 소문자를 사용하여 작성한다. 단어별 구문을 위해 언더스코어(`_`)나 대문자를 섞지 않는다.

나쁜 예

```
package com.navercorp.apiGateway
package com.navercorp.api_gateway
```

좋은 예

```
package com.navercorp.apigateway
```

1.5. 클래스/인터페이스 이름에 대문자 카멜표기법 적용

클래스 이름은 단어의 첫 글자를 대문자로 시작하는 대문자 카멜표기법(Upper camel case)을 사용한다. 파스칼표기법(Pascal case)으로도 불린다.

나쁜 예

```
public class reservation
public class Accesstoken
```

좋은 예

```
public class Reservation
public class AccessToken
```

1.6. 클래스 이름에 명사 사용

클래스 이름은 명사나 명사절로 짓는다.

1.7. 인터페이스 이름에 명사/형용사 사용

인터페이스(interface)의 이름은 클래스 이름은 명사/명사절로 혹은 형용사/형용사절로 짓는다.

좋은 예

```
public interface RowMapper {

public interface AutoClosable {
```

1.8. 메서드 이름은 동사/전치사로 시작

메서드명은 기본적으로는 동사로 시작한다. 다른 타입으로 전환하는 메서드나 빌더 패턴을 구현한 클래스의 메서드에는 전치사를 쓸 수 있다.

좋은 예

- 동사사용 : `renderHtml()`
- 전환메서드의 전치사 : `toString()`
- Builder 패턴 적용한 클래스의 메서드의 전치사 : `withUserId(String id)`

1.9. 상수는 대문자와 언더스코어로 구성

상태를 가지지 않는 자료형이면서 `static final`로 선언되어 있는 필드일 때를 상수로 간주한다. 상수 이름은 대문자로 작성하며, 복합어는 언더스코어(`_`)를 사용하여 단어를 구분한다.

좋은 예

```
public final int UNLIMITED = -1;
public final String POSTAL_CODE_EXPRESSION = “POST”;
```

1.10. 변수에 소문자 카멜표기법 적용

상수가 아닌 클래스의 멤버변수/지역변수/메서드 파라미터에는 소문자 카멜표기법(Lower camel case)을 사용한다.

나쁜 예

```
private boolean Authorized;
private int AccessToken;
```

좋은 예

```
private boolean authorized;
private int accessToken;
```

2. 선언 (Declarations)

클래스, 필드, 메서드, 변수값, import문 등의 소스 구성요소를 선언할 때 고려해야할 규칙이다.

2.1. 소스파일당 1개의 탑레벨 클래스를 담기

탑레벨 클래스(Top level class)는 소스 파일에 1개만 존재해야 한다. (탑레벨 클래스 선언의 컴파일타임 에러 체크에 대해서는 Java Language Specification 7.6 참조)

나쁜 예

```
public class LogParser {
}

class LogType {
}
```

좋은 예

```
public class LogParser {
    // 굳이 한 파일안에 선언해야 한다면 내부 클래스로 선언
    class LogType {
    }
}
```

2.2. 제한자 선언의 순서

클래스/메서드/멤버변수의 제한자는 Java Language Specification에서 명시한 아래의 순서로 쓴다.

```
public protected private abstract static final transient volatile synchronized native strictfp
```

(Java Language Specification - Chapter 18. Syntax 참조)

2.3. 한 줄에 한 문장

문장이 끝나는 `;` 뒤에는 새줄을 삽입한다. 한 줄에 여러 문장을 쓰지 않는다.

나쁜 예

```
int base = 0; int weight = 2;
```

좋은 예

```
int base = 0;
int weight = 2;
```

2.4. 하나의 선언문에는 하나의 변수만

변수 선언문은 한 문장에서 하나의 변수만을 다룬다.

나쁜 예

```
int base, weight;
```

좋은 예

```
int base;
int weight;
```

2.5. 배열에서 대괄호는 타입 뒤에 선언

배열 선언에 오는 대괄호(`[]`)는 타입의 바로 뒤에 붙인다. 변수명 뒤에 붙이지 않는다.

나쁜 예

```
String names[];
```

좋은 예

```
String[] names;
```

2.6. `long`형 값의 마지막에 `L`붙이기

long형의 숫자에는 마지막에 대문자 'L'을 붙인다. 소문자 'l'보다 숫자 '1'과의 차이가 커서 가독성이 높아진다.

나쁜 예

```
long base = 54423234211l;
```

좋은 예

```
long base = 54423234211L;
```

3. 중괄호 (Braces)

중괄호(`{`, `}`) 는 클래스, 메서드, 제어문의 블럭을 구분한다.

3.1. K&R 스타일로 중괄호 선언

클래스 선언, 메서드 선언, 조건/반복문 등의 코드 블럭을 감싸는 중괄호에 적용되는 규칙이다. 중괄호 선언은 K&R 스타일(Kernighan and Ritchie style)을 따른다. 줄의 마지막에서 시작 중괄호 `{`를 쓰고 열고 새줄을 삽입한다. 블럭을 마친후에는 새줄 삽입 후 중괄호를 닫는다.

나쁜 예

```
public class SearchConditionParser
{
    public boolean isValidExpression(String exp)
    {

        if (exp == null)
        {
            return false;
        }

        for (char ch : exp.toCharArray())
        {
             ....
        }

        return true;
    }
}
```

좋은 예

```
public class SearchConditionParser {
    public boolean isValidExpression(String exp) {

        if (exp == null) {
            return false;
        }

        for (char ch : exp.toCharArray()) {
            ....
        }

        return true;
    }
}
```

3.2. 닫는 중괄호와 같은 줄에 `else`, `catch`, `finally`, `while` 선언

아래의 키워드는 닫는 중괄호(`}`) 와 같은 줄에 쓴다.

- else
- catch, finaly
- do-while 문에서의 while

나쁜 예

```
if (line.startWith(WARNING_PREFIX)) {
    return LogPattern.WARN;
}
else if (line.startWith(DANGER_PREFIX)) {
    return LogPattern.DANGER;
}
else {
    return LogPattern.NORMAL;
}
```

좋은 예

```
if (line.startWith(WARNING_PREFIX)) {
    return LogPattern.WARN;
} else if (line.startWith(DANGER_PREFIX)) {
    return LogPattern.NORMAL;
} else {
    return LogPattern.NORMAL;
}
```

나쁜 예

```
try {
    writeLog();
}
catch (IOException ioe) {
    reportFailure(ioe);
}
finally {
    writeFooter();
}
```

좋은 예

```
try {
    writeLog();
} catch (IOException ioe) {
    reportFailure(ioe);
} finally {
    writeFooter();
}
```

나쁜 예

```
`do {
    write(line);
    line = readLine();
}
while (line != null);`

*좋은 예*

`do {
    write(line);
    line = readLine();
} while (line != null);`
```

3.3. 빈 블럭에 새줄 없이 중괄호 닫기 허용

내용이 없는 블럭을 선언할 때는 같은 줄에서 중괄호를 닫는 것을 허용한다.

좋은 예

```
public void close() {}
```

3.4. 조건/반복문에 중괄호 필수 사용

조건, 반복문이 한 줄로 끝더라도 중괄호를 활용한다. 이 문서에 언급된 중괄호의 전후의 공백, 제어문 앞 뒤의 새줄 규칙도 함께 고려한다.

나쁜 예

```
if (exp == null) return false;

for (char ch : exp.toCharArray()) if (ch == 0) return false;
```

좋은 예

```
if (exp == null) {
    return false;
}

for (char ch : exp.toCharArray()) {

    if (ch == 0) {
        return false;
    }

}
```

인텔리제이 자동 컨밴션 에디터, 검사기 설정

- 아래 파일을 다운받는다.

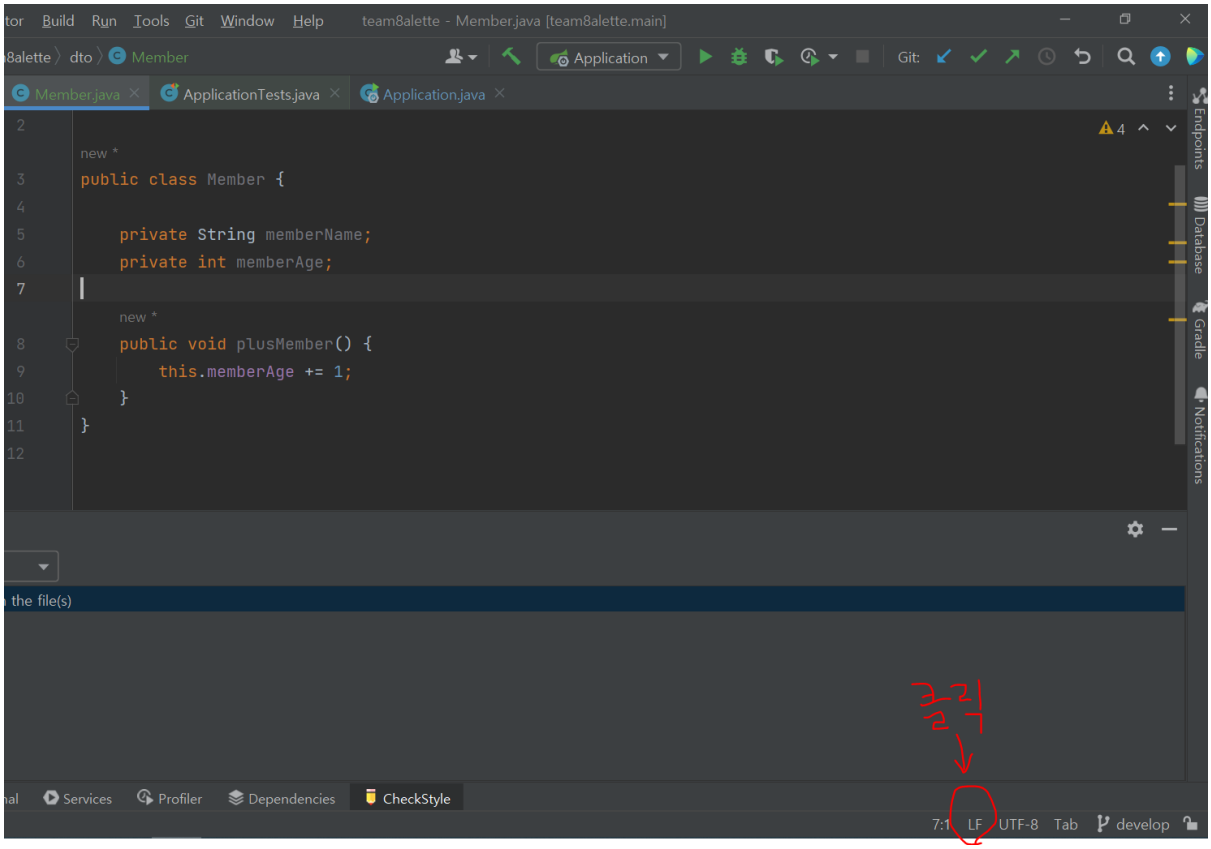
naver-checkstyle-rules.xml

naver-checkstyle-suppressions.xml

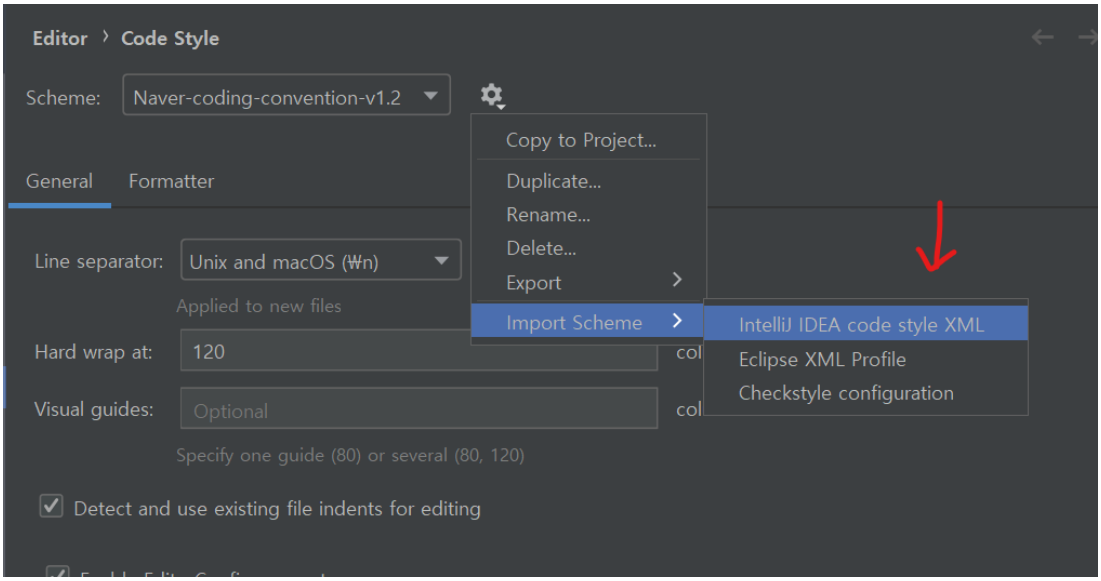
naver-intellij-formatter.xml

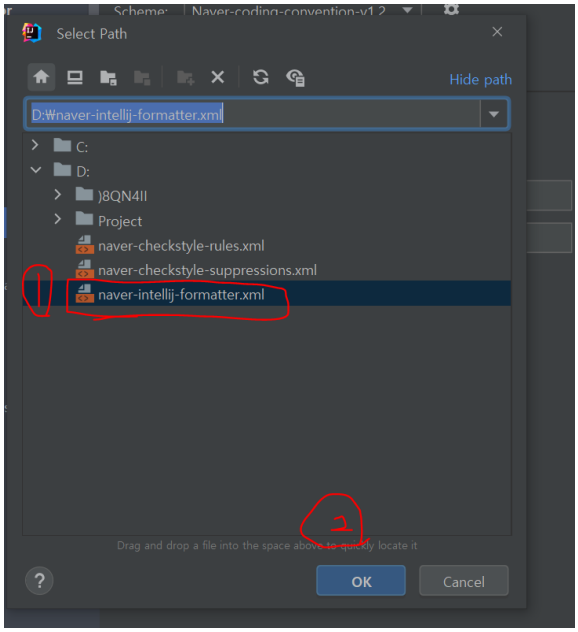
❗checkstyle-rules의 경우 파일경로가 바뀌면 읽을수 없으니 프로젝트 파일내에 저장하는걸 추천드립니다.

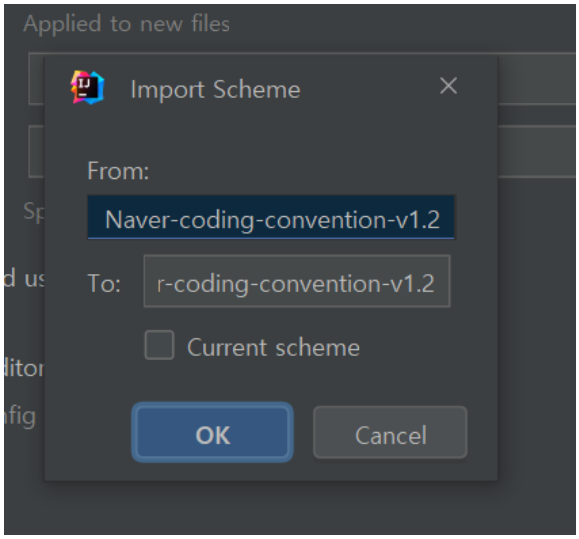
1. IntelliJ CRLF —> LF로 변경



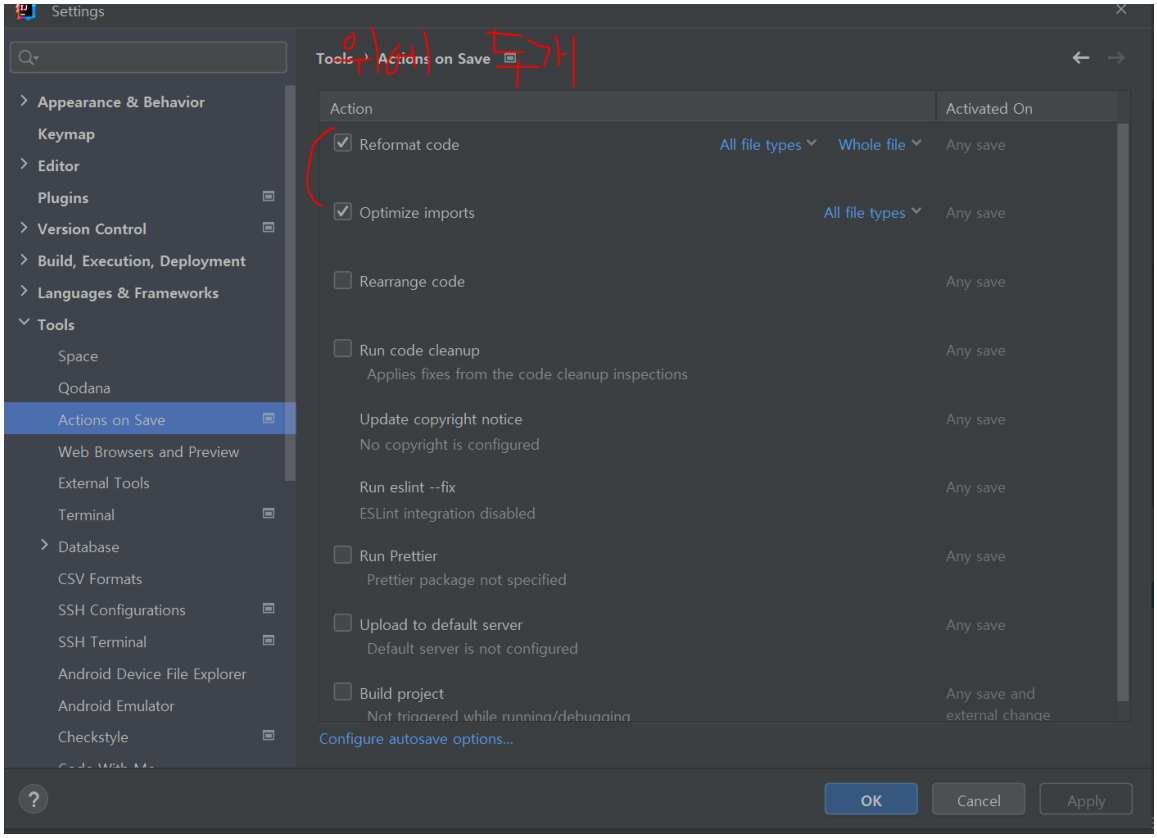
2. Code Formatter 설정
    a. Setting → Editor → Code Style
    b. Schema → 톱니바퀴 → Import Schema → IntelliJ IDEA code style XML
    c. naver-intellij-formatter.xml 선택 후 OK
    d. To : 에다가 원하는 이름 쓰고 OK
    e. 이후에 alt + ctrl + L 누르면 자동으로 코드정렬이 된다.



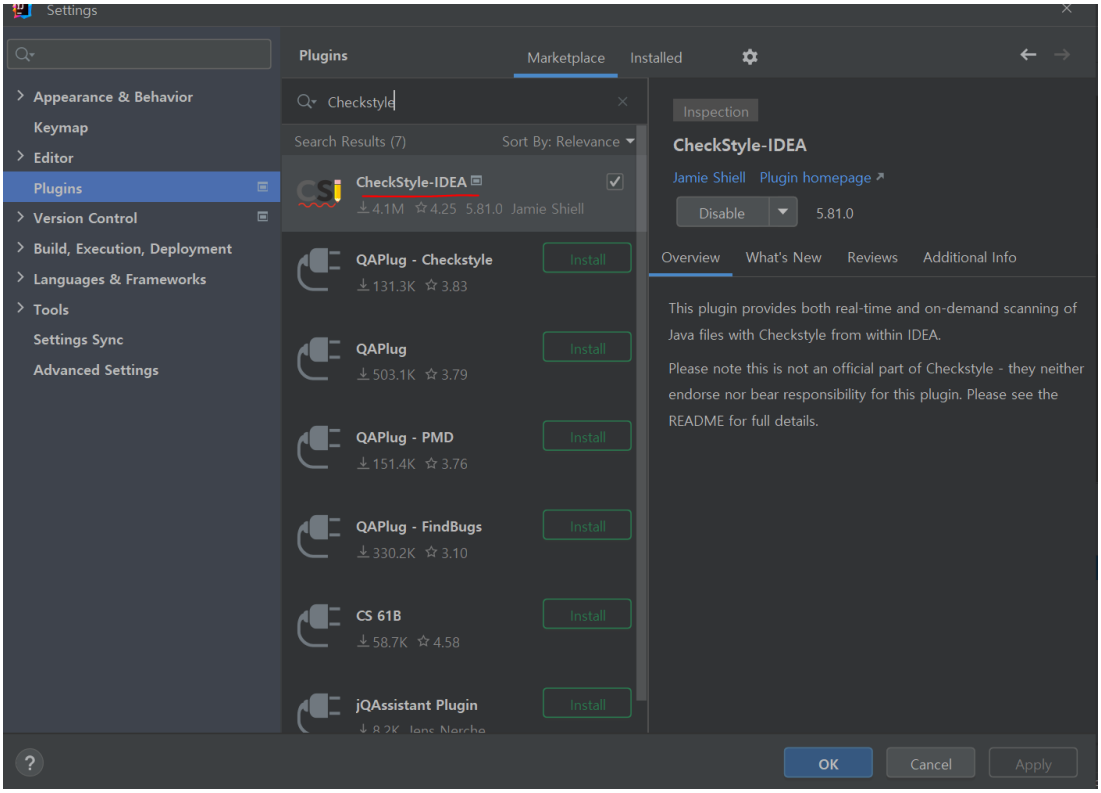




3. 저장시 자동으로 코드 정렬 설정
    a. Setting → Tool → Action on Save
    b. reformat code(fomatter 대로 코드 정렬), Optimize imports(안쓰는 import 삭제) 체크
    c. OK



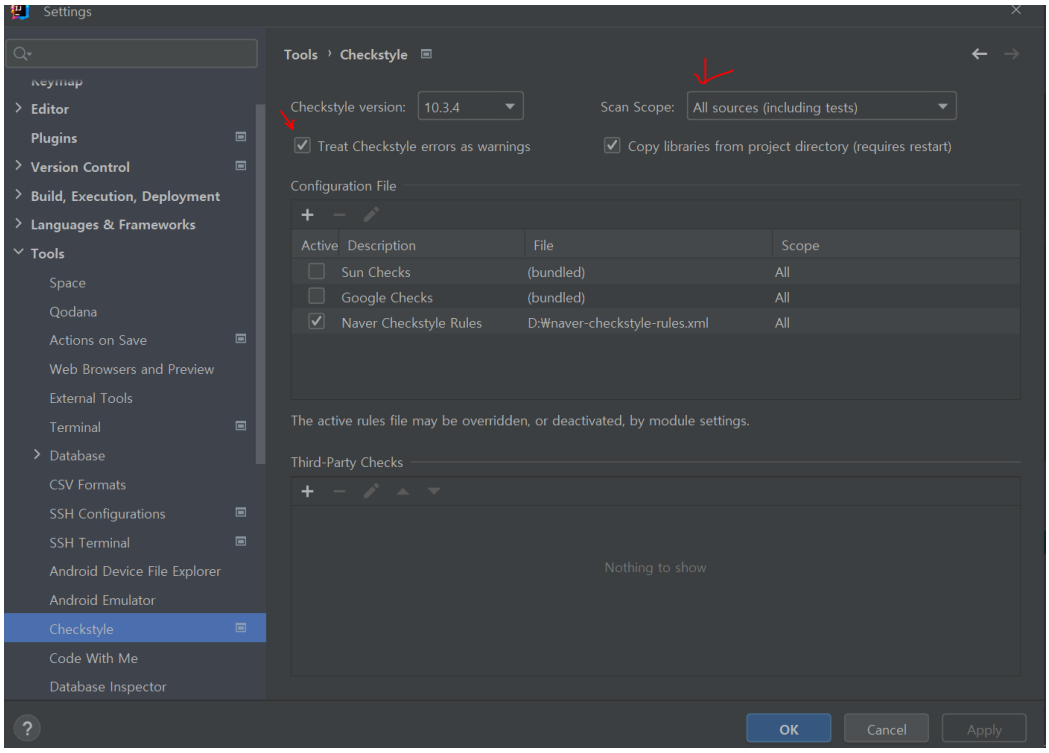
4. CheckStyle-IDEA 설정
   a. Setting → Plugins → CheckStyle-IDEA 설치 후 IntelliJ 재시작



b. Setting → Tools → Checkstyle

   i. Scan scope : All sources (including tests)

   ii. Treat Checkstyle errors as warnings

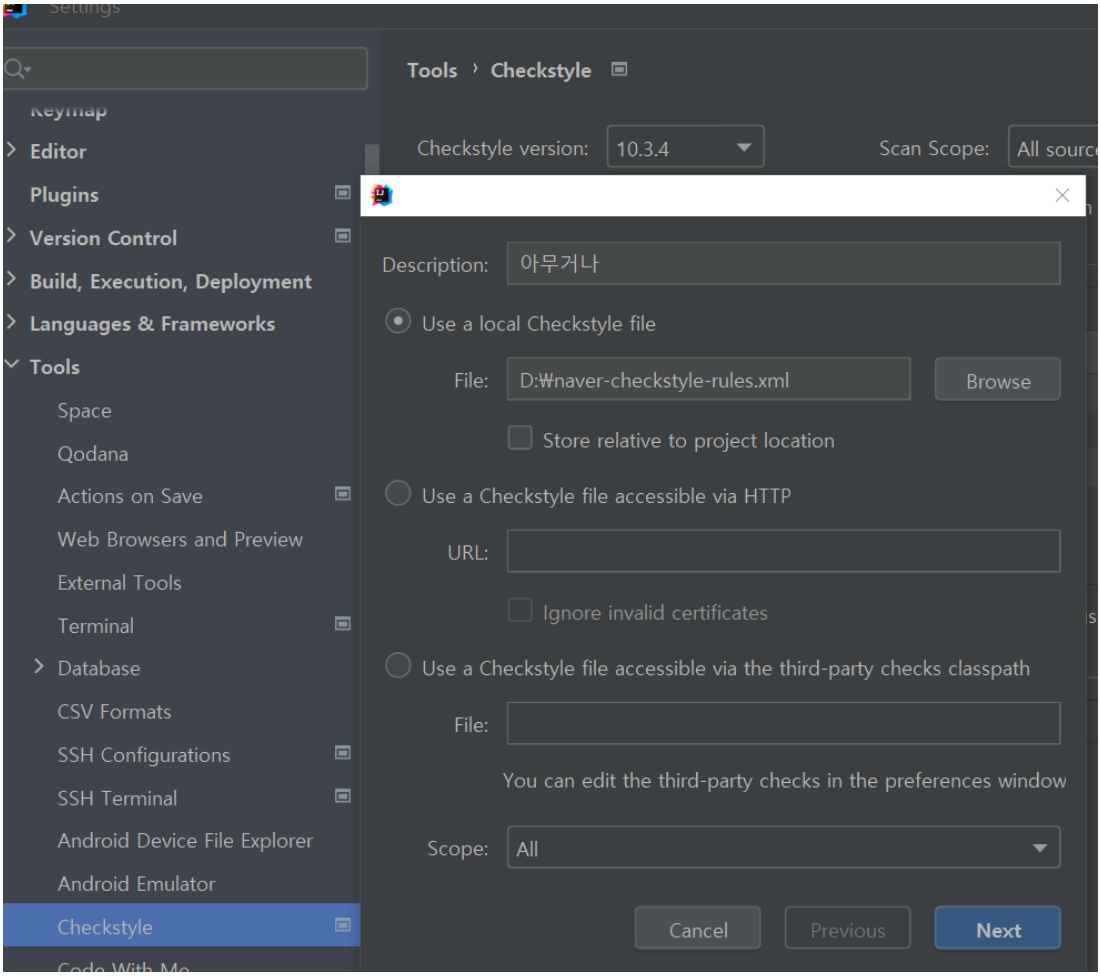


c. Configuration file → +버튼

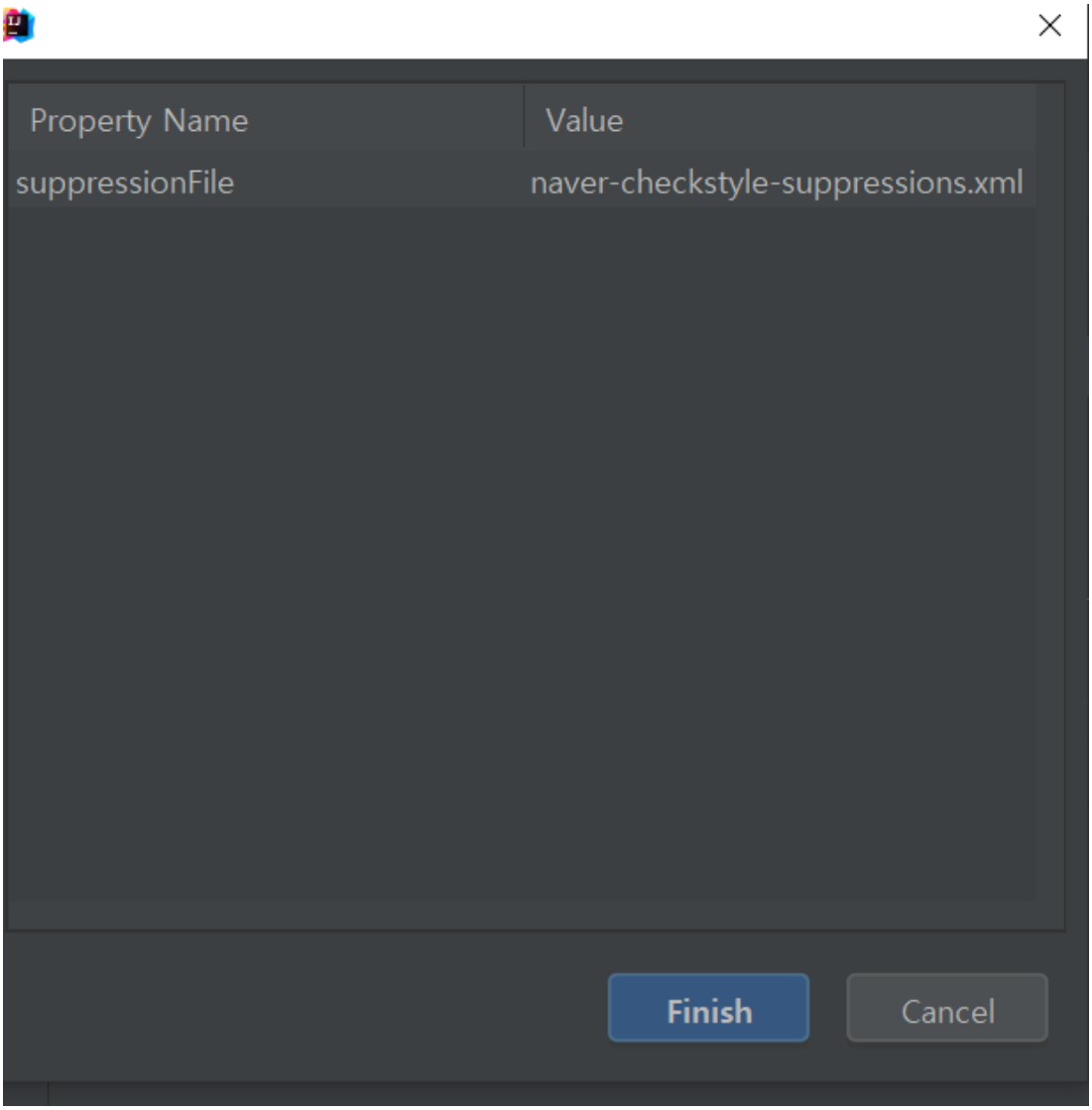
   i. Description 입력

   ii. Use a local Checkstyle file 체크

   iii. Browse해서 naver-checkstye-rules.xml 선택 후 NEXT

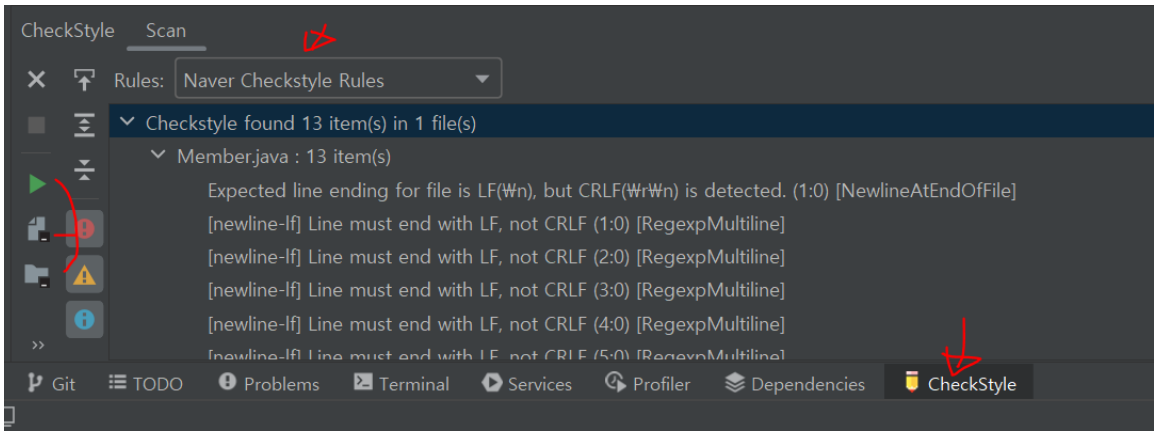


d. Value에 naver-checkstyle-suppressions.xml 라고 적고 Finish

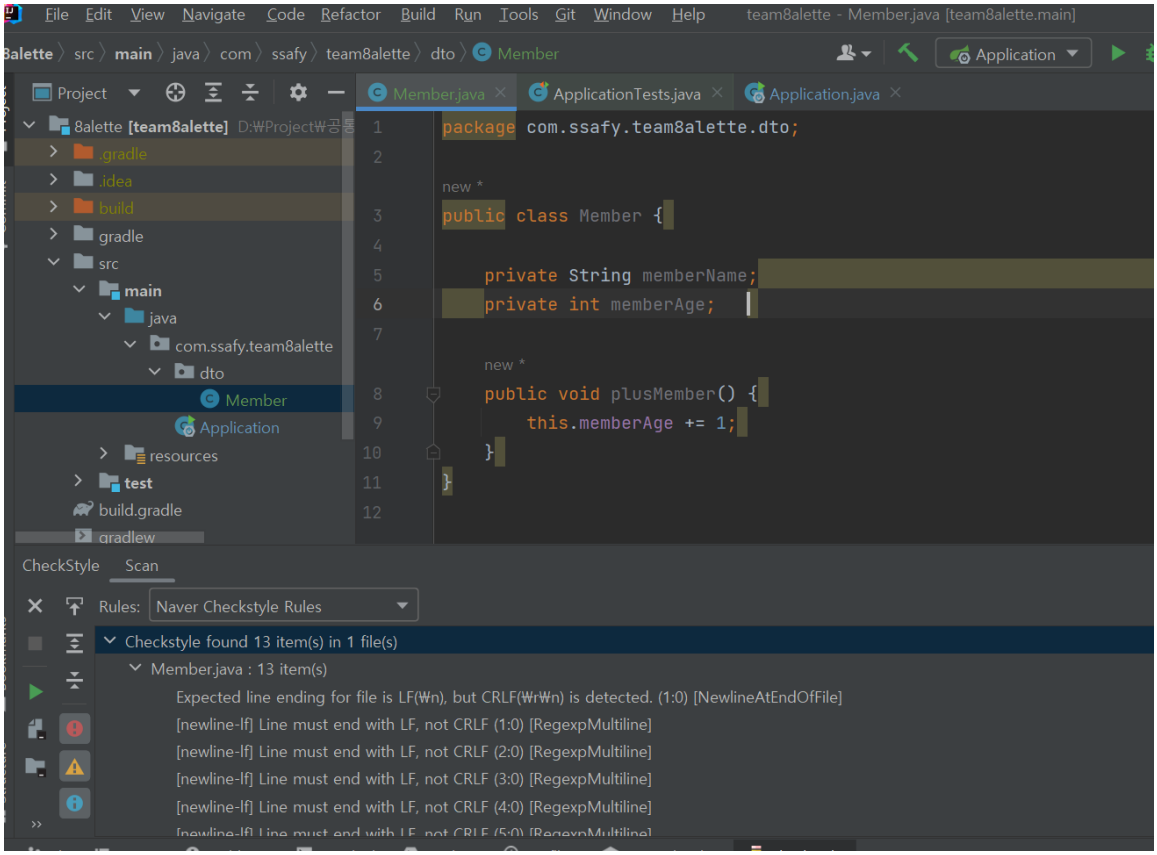


사용법

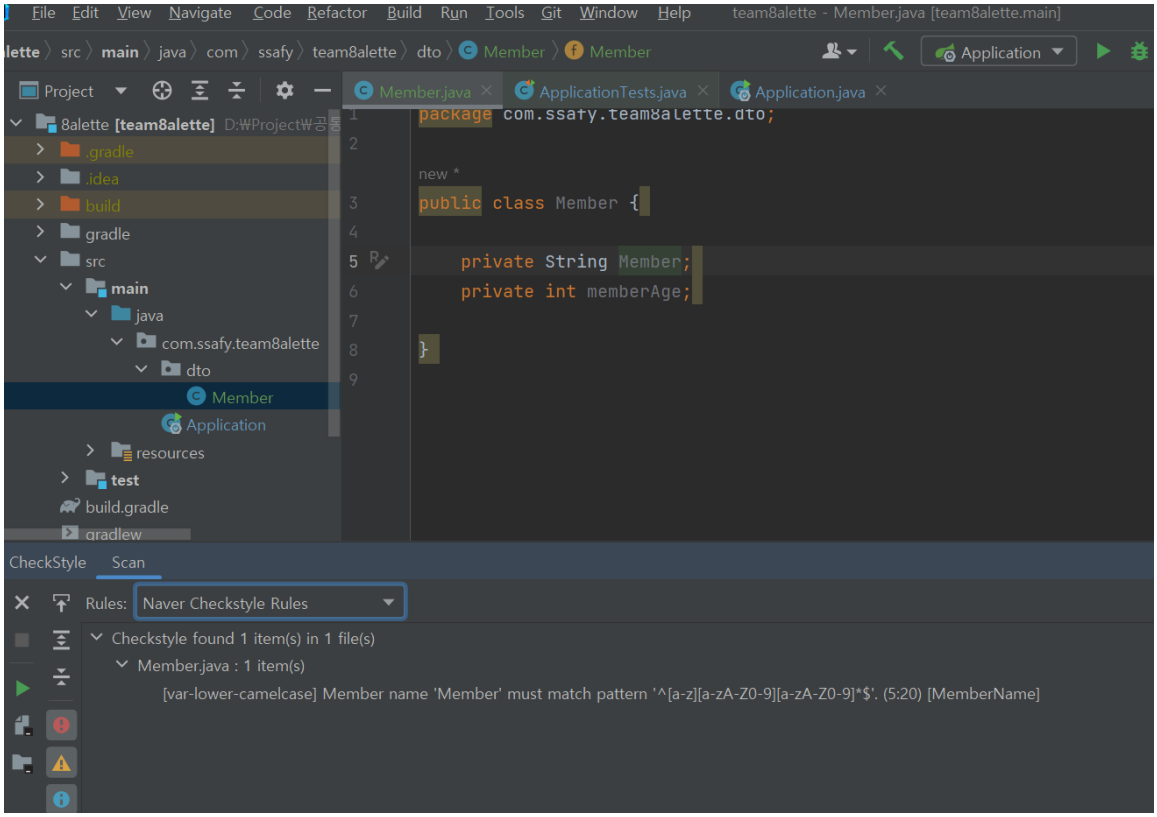
하단에생긴 CheckStyle로 검사를 할 수 있다.



쓸데 없는 공백 있을때 Warning



파스칼 케이스를 지키지 않았을때


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
team8alette - Member.java [team8alette.main]
Application
Project
Member.java
ApplicationTests.java
Application.java
8alette [team8alette] D:₩Project₩공통
.gradle
.idea
build
gradle
src
main
java
com.ssafy.team8alette
dto
Member
Application
resources
test
build.gradle
gradlew
package com.ssafy.team8alette.dto;
new *
public class Member {
private String Member;
private int memberAge;
}
CheckStyle
Scan
Rules:
Naver Checkstyle Rules
Checkstyle found 1 item(s) in 1 file(s)
Member.java : 1 item(s)
[var-lower-camelcase] Member name 'Member' must match pattern '^[a-z][a-zA-Z0-9][a-zA-Z0-9]*$'. (5:20) [MemberName]