Checkstyle

1. 도구 개요

checkstyle

1.1 도구 정보 요약

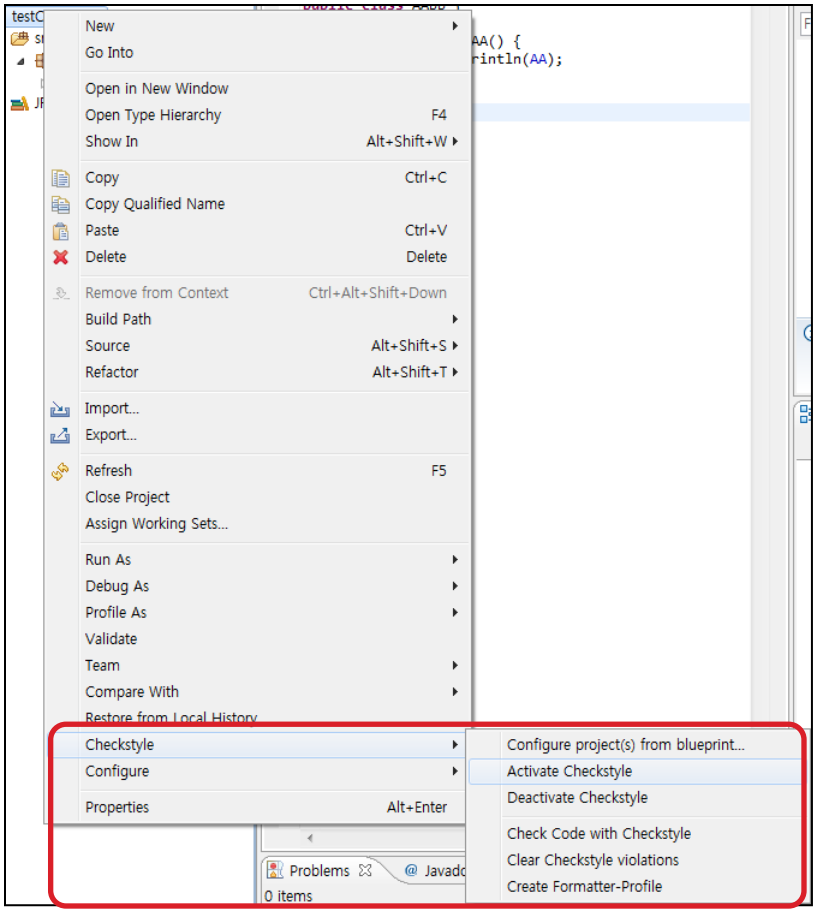
도구명	Checkstyle (http://checkstyle.sourceforge.net/)	라이선스	Eclipse Public License v1.0
소개	• 코딩 하면서 소스 코드 내에서 다양한 위반 사항에 대해 알 수 있고, 개발자들이 체크인 전에 위반 사항을 고칠 수 있음. 또한 정해놓은 코딩 규칙에 따라 팀원들이 보다 쉽게 규칙을 적용할 수 있게 도와주는 도구.		
특징	• 여러 사람과 작업 시 손쉽게 코딩 스타일을 맞출 수 있음. • 표준 코딩 스타일을 손쉽게 프로젝트에 적용. • 개발 초기에 소스 코드의 잠재적 결함을 발견		
주요기능	• 코딩 스타일 체크 및 통일 • 사용자 규칙 추가 가능		
실행환경	• Windows, Linux, Mac OS, UNIX	사전설치도구	• JDK, Eclipse
카테고리	• 품질관리	최신버전	• 6.1.1 (2014.11)
관련도구	• FindBugs, PMD, SonarQube		

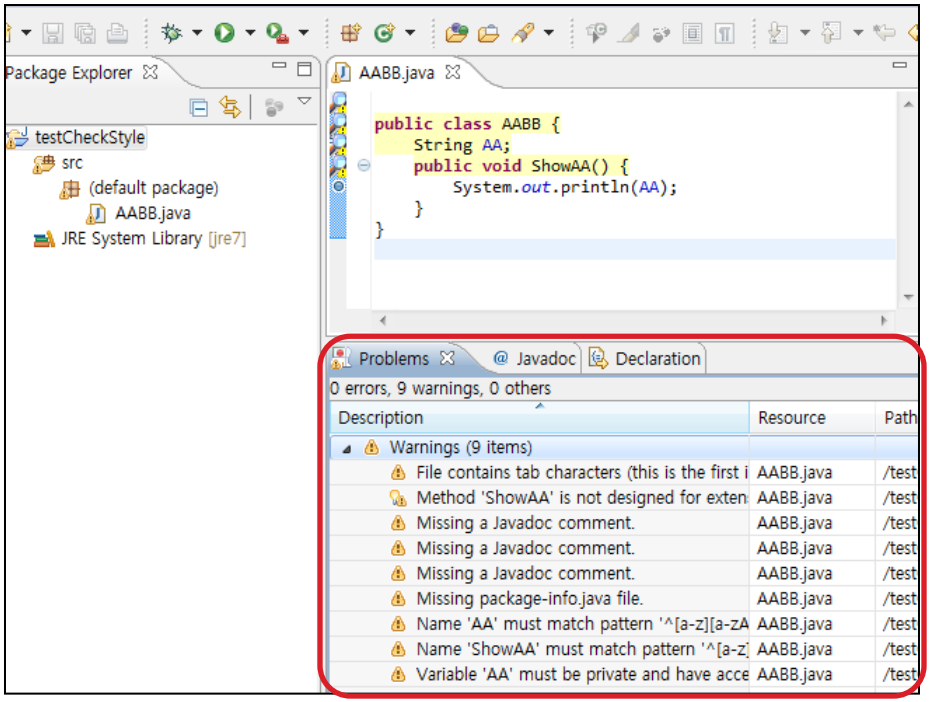
1. 도구 개요

checkstyle

1.2 스크린 캡쳐 및 주요 기능

- Eclipse의 Package Explorer에서 마우스 오른쪽 클릭시 팝업메뉴로 Checkstyle 기능 제공
- 코드 분석 결과는 Problems 탭에 표시





2. 설치 및 실행

checkstyle

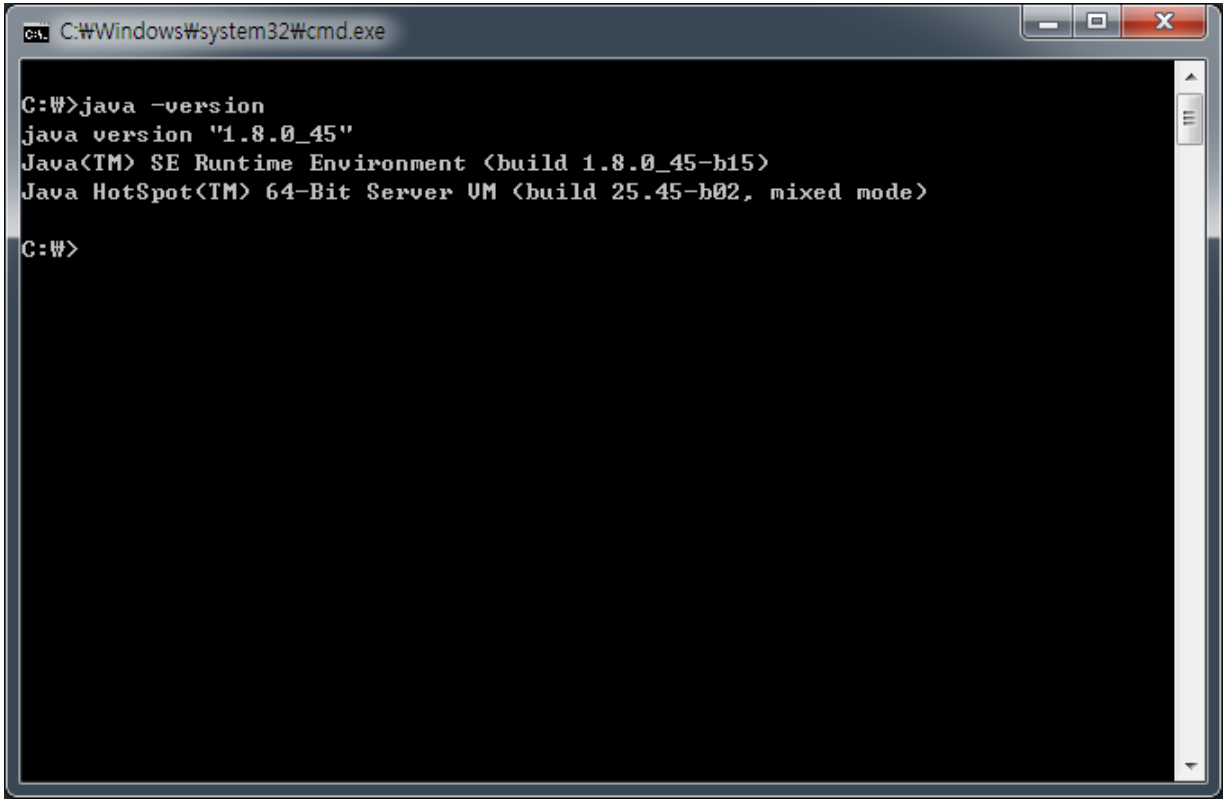
세부 목차

2. 설치 및 실행

checkstyle

2.1 사전 설치 사항 확인 (1/2)

- Windows의 경우 아래와 같이 Command창에서 'java –version'을 실행하여 설치된 JDK 설치 여부를 확인한다.
- JDK 1.6 이상을 권장한다.

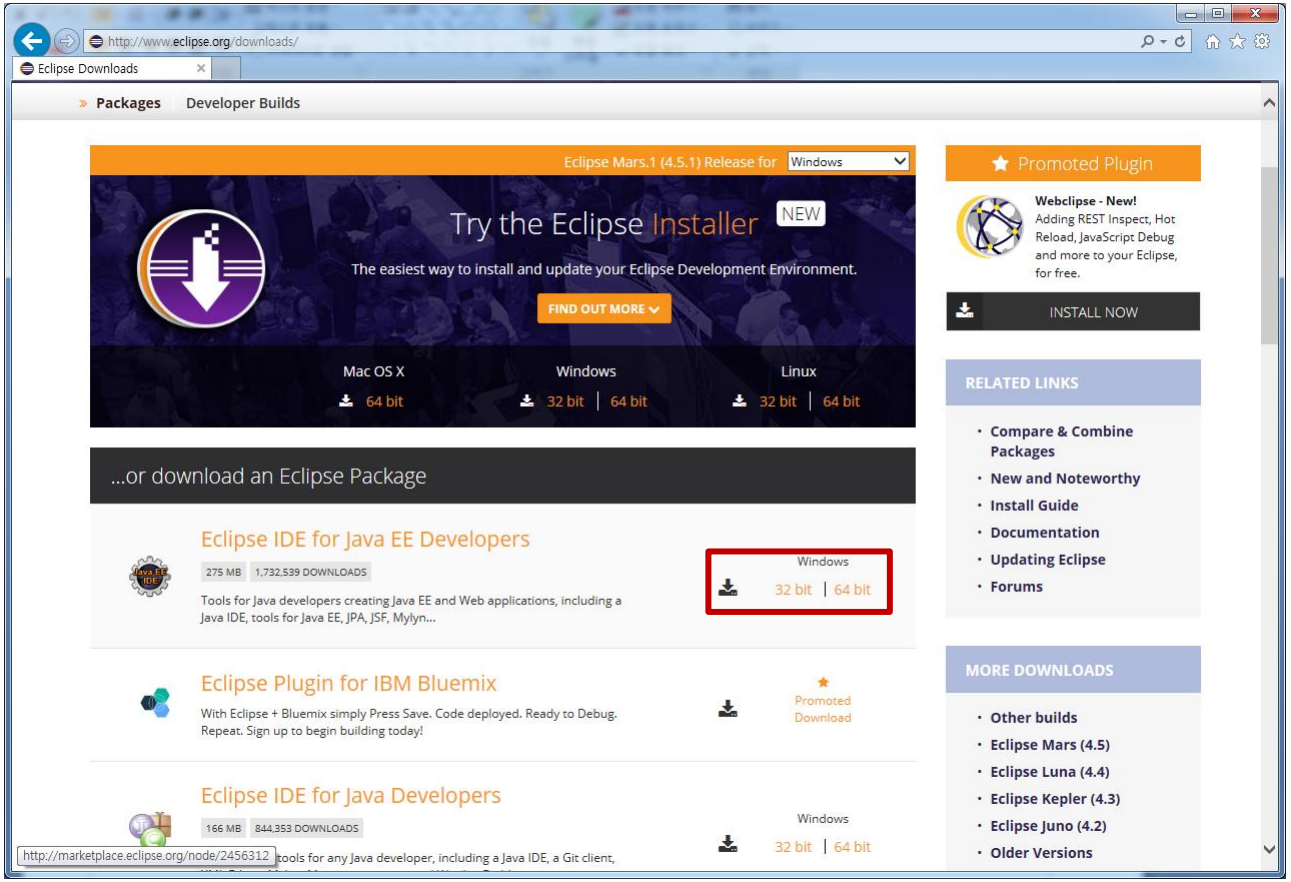


2. 설치 및 실행

checkstyle

2.1 사전 설치 사항 확인 (2/2)

- 이클립스가 설치되어 있어야 한다.
- 설치되어 있지 않다면 http://www.eclipse.org/downloads 에서 다운받을 수 있다.

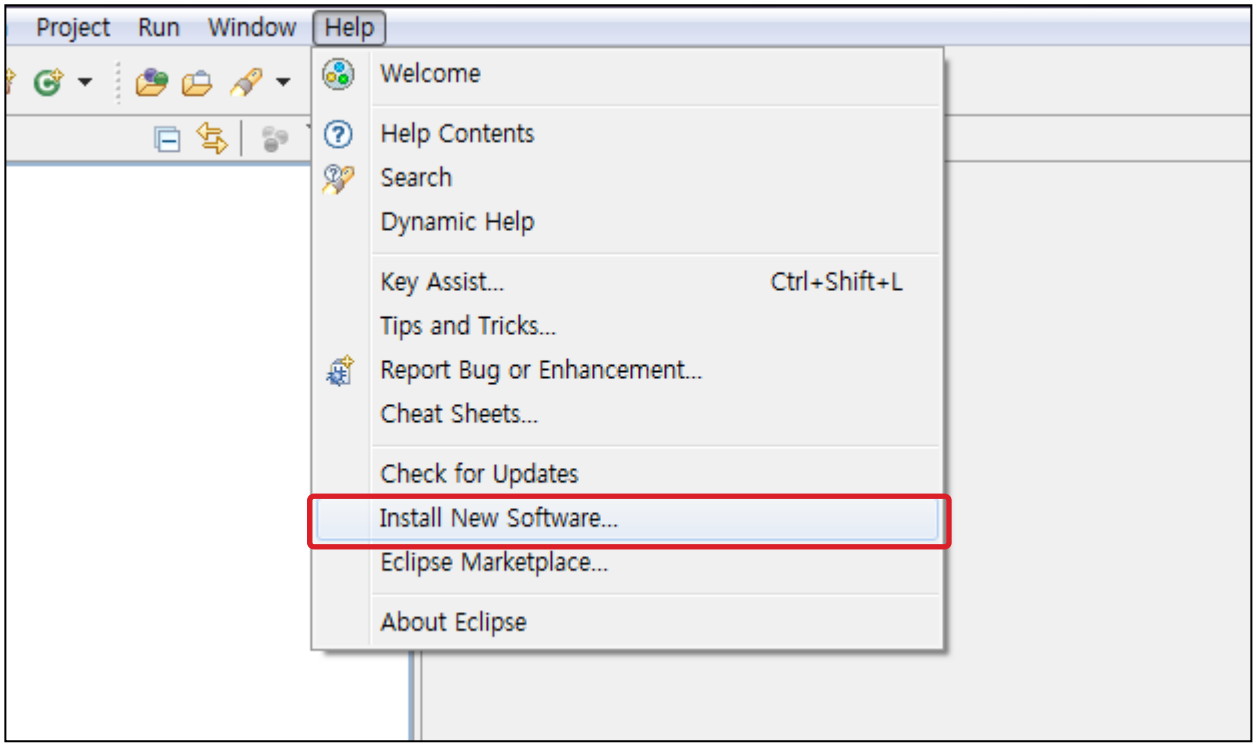


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (1/6)

- Software Updates를 이용하여 Checkstyle을 설치
  - Eclipse → Help → Install New Software

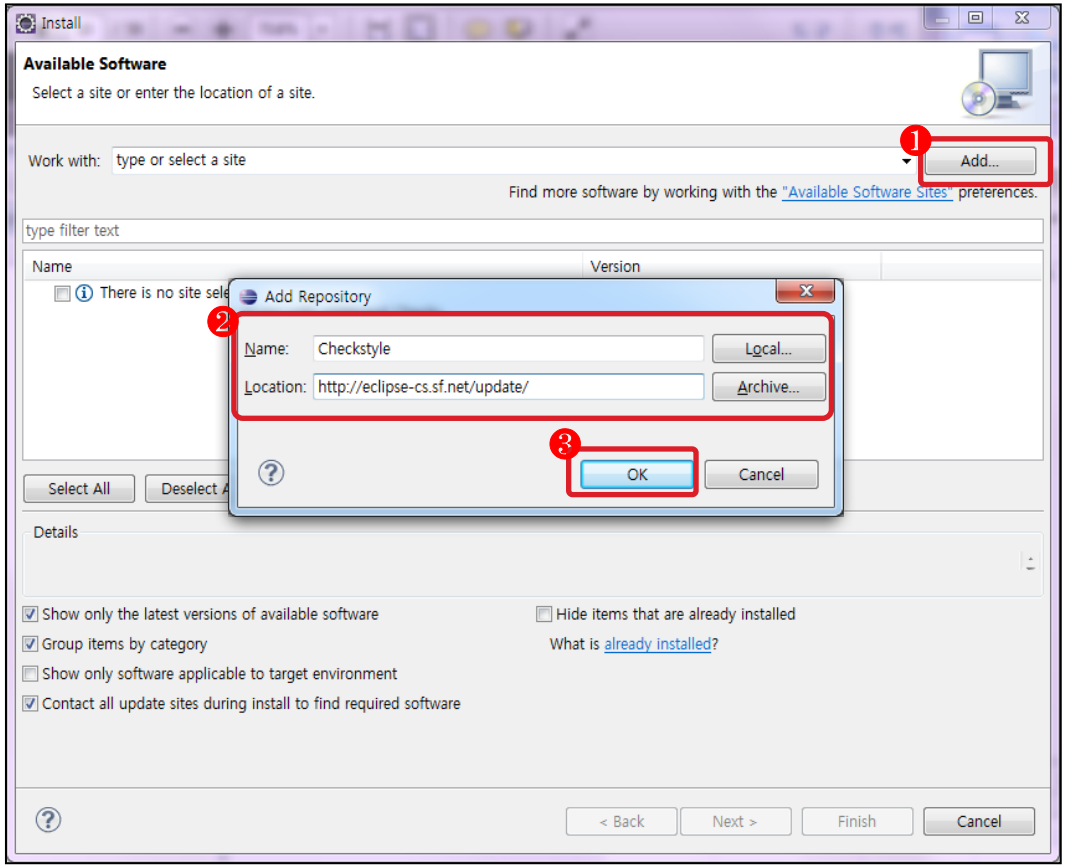


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (2/6)

- 새로운 업데이트 항목을 등록
  - Add → Name 입력 → Location 입력
  - Name : Checkstyle
  - Location : http://eclipse-cs.sf.net/update/

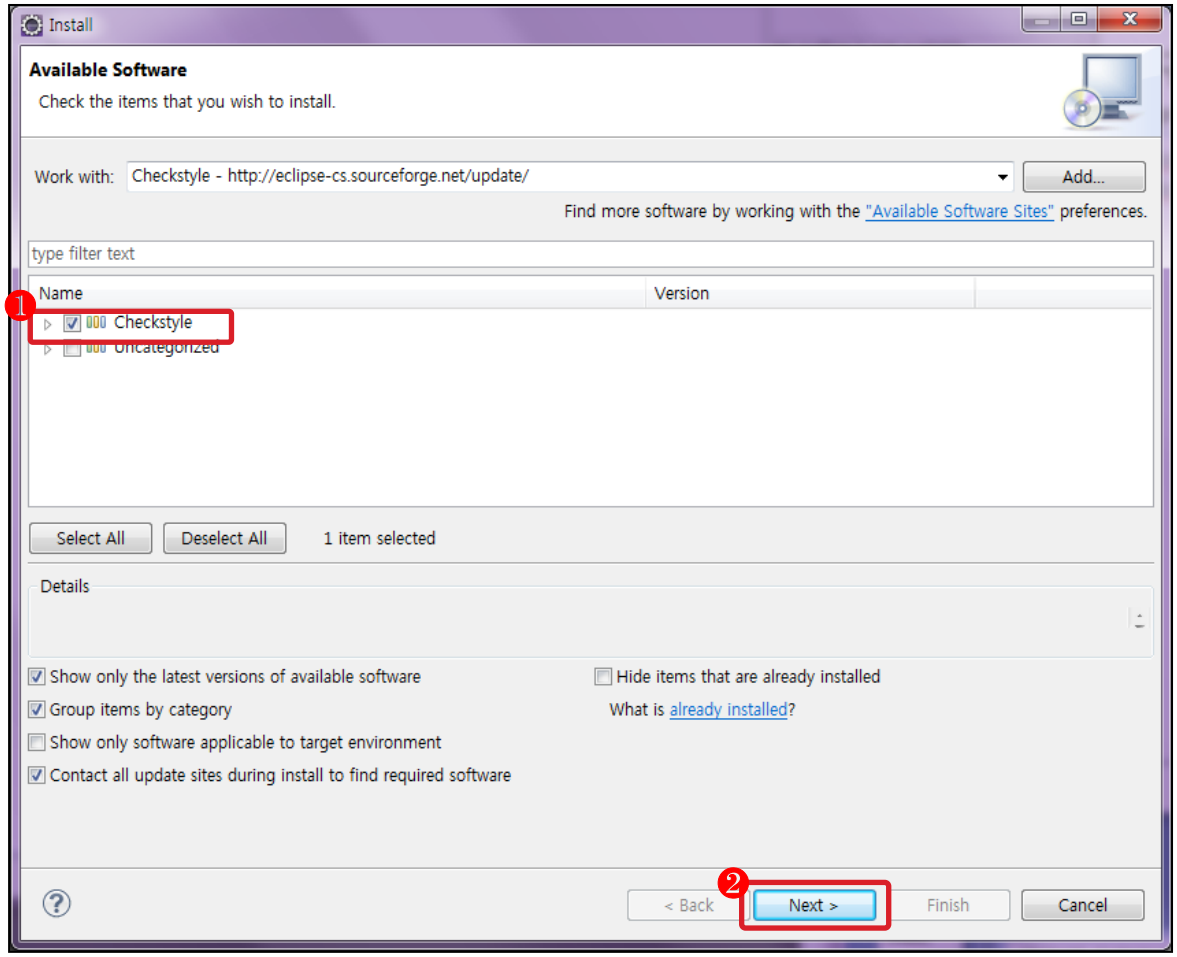


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (3/6)

- Checkstyle 항목이 추가
  - Checkstyle 체크 → Next

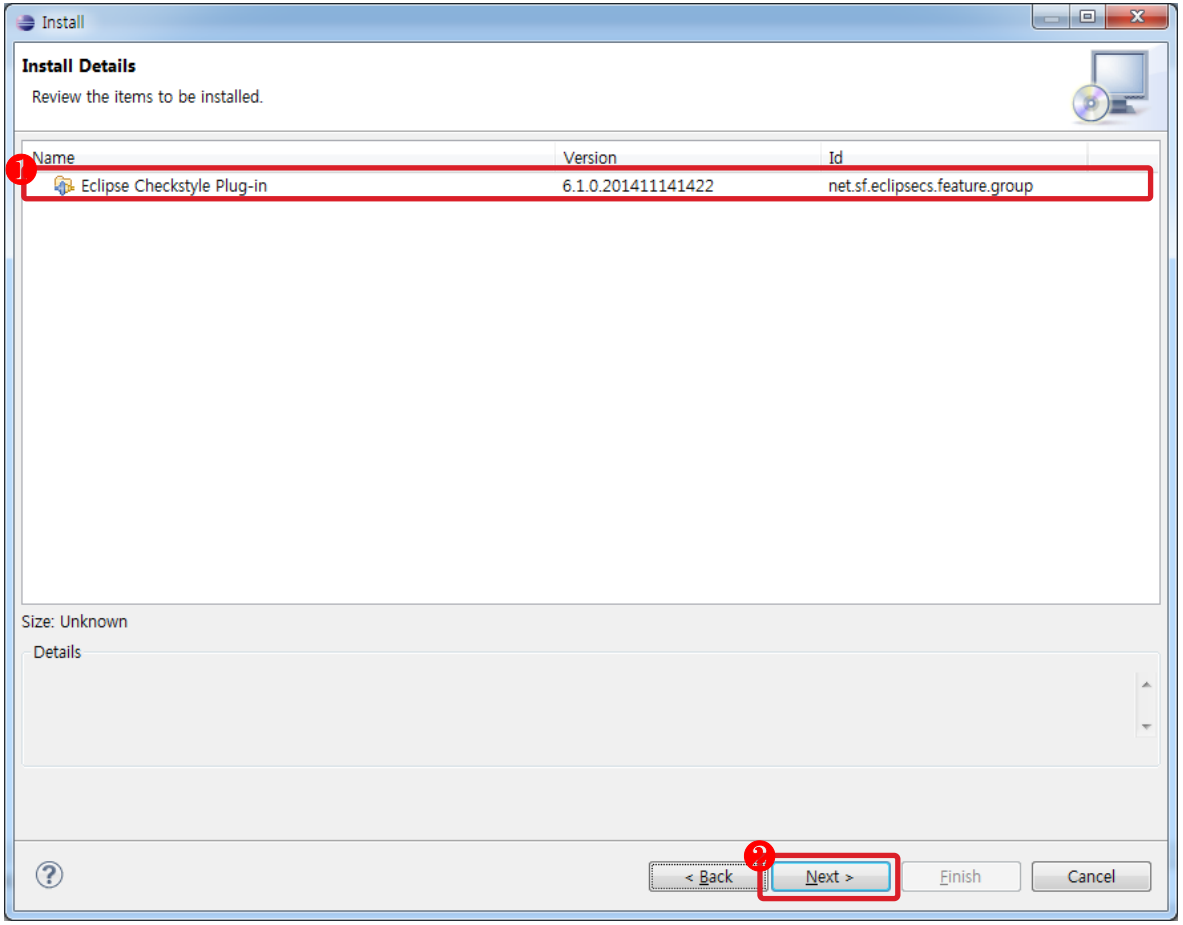


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (4/6)

- 설치할 대상을 선택
  - Eclipse Checkstyle Plug-in 선택 → Next 클릭

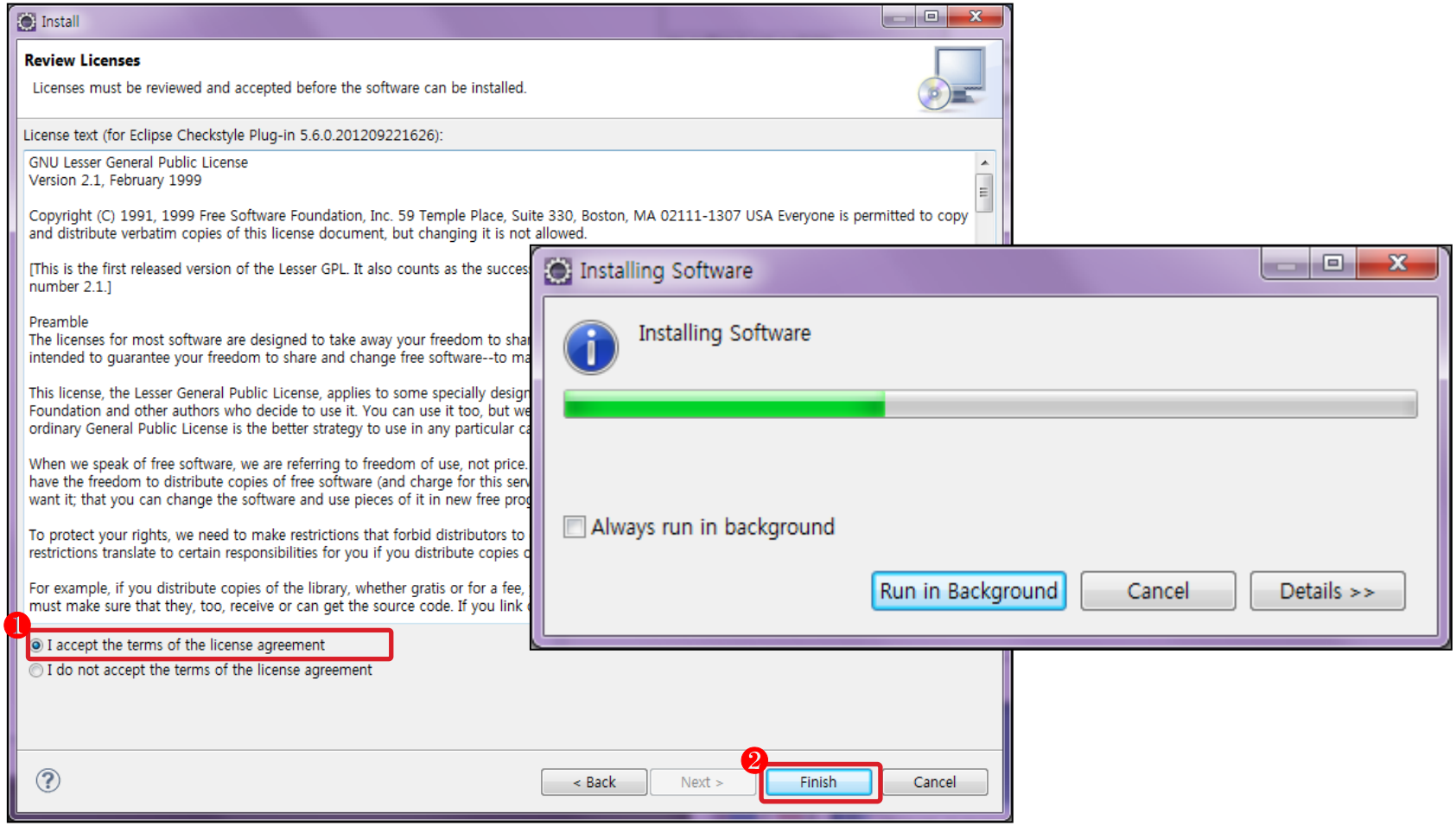


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (5/6)

- 설치할 프로그램의 버전과 라이선스를 확인
  - I accept the terms of the license agreement 체크 → Next 클릭

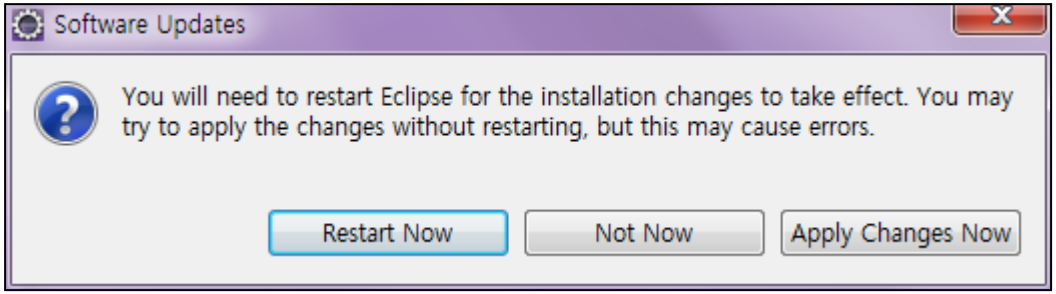


2. 설치 및 실행

checkstyle

2.2 Eclipse에서 Checkstyle 설치하기 (6/6)

- 설치를 완료
  - Restart Now를 클릭하여 Eclipse를 재 시작



3. 주요 기능

checkstyle

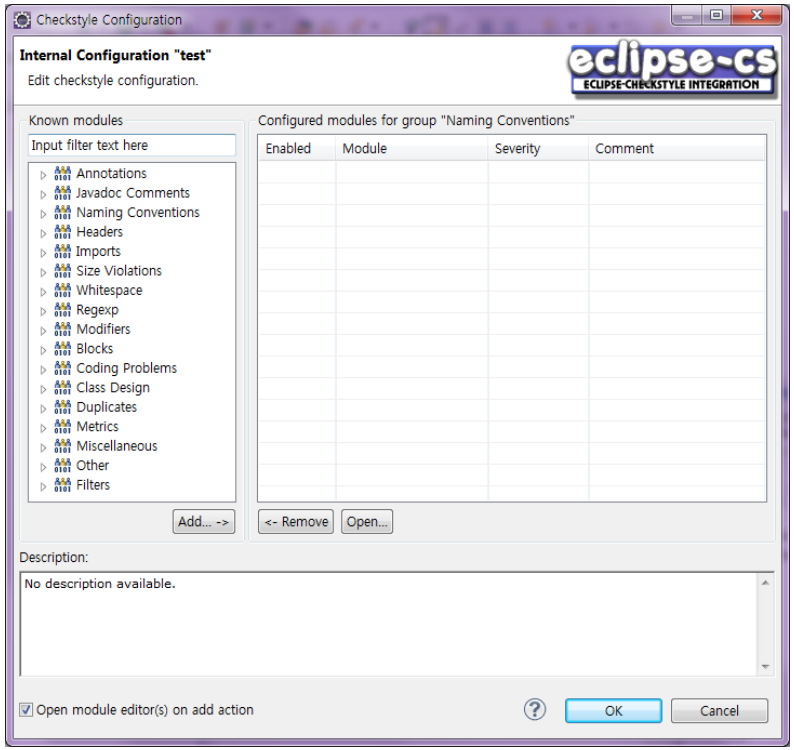
세부목차

3. 주요 기능

checkstyle

3.1 Checkstyle 주요기능

- 프로젝트 개발 시 개별적으로 코딩 된 소스를 표준에 적합하게 수정 가능
- 원하는 규칙을 선택하여 자신(팀)만의 규칙을 적용 가능

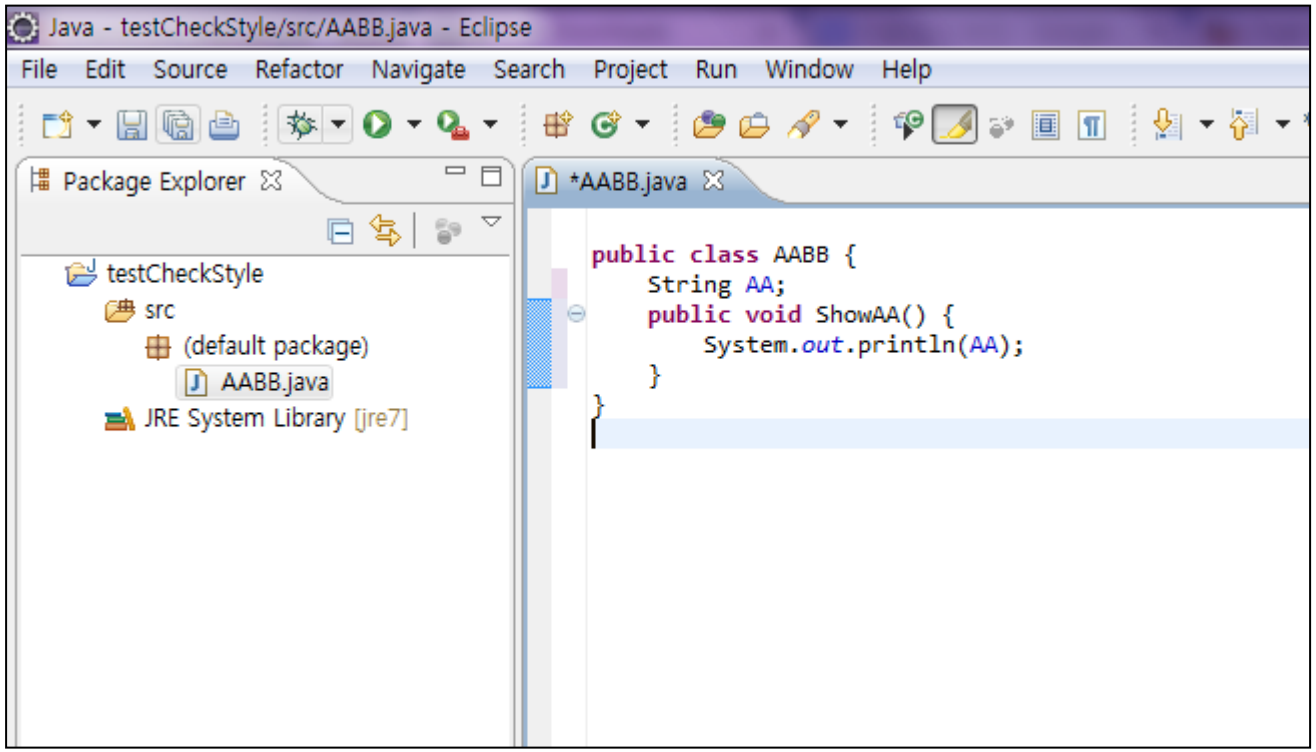


3. 주요 기능

checkstyle

3.2 예제 소개 (1/4)

- Checkstyle의 기능을 소개하기 위해 간단한 프린트문 예제를 작성
  - 예제는 일반적인 컴파일러로 컴파일 시 error나 warning 메시지가 없는 source
  - String 타입 AA를 화면에 출력하는 프로그램

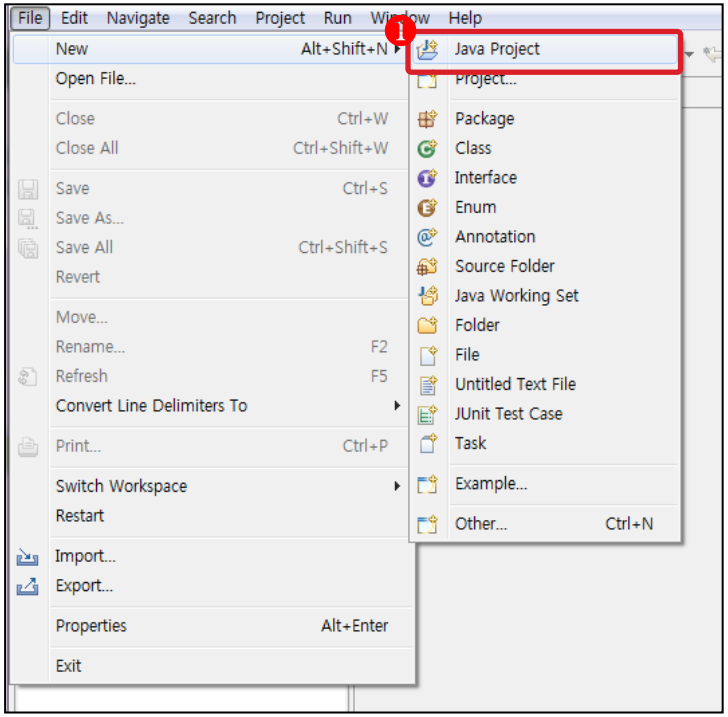


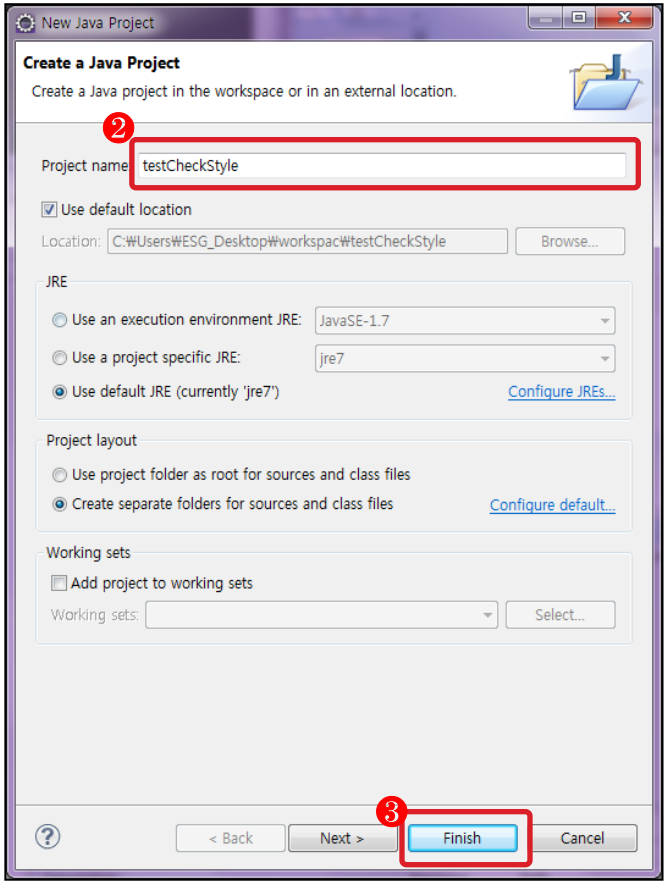
3. 주요 기능

checkstyle

3.2 예제 소개 (2/4)

- 프로젝트 생성
  - File → New → Java Project → Project name 입력 → Finish
  - 본 예제에서는 Project name을 testCheckStyle로 입력



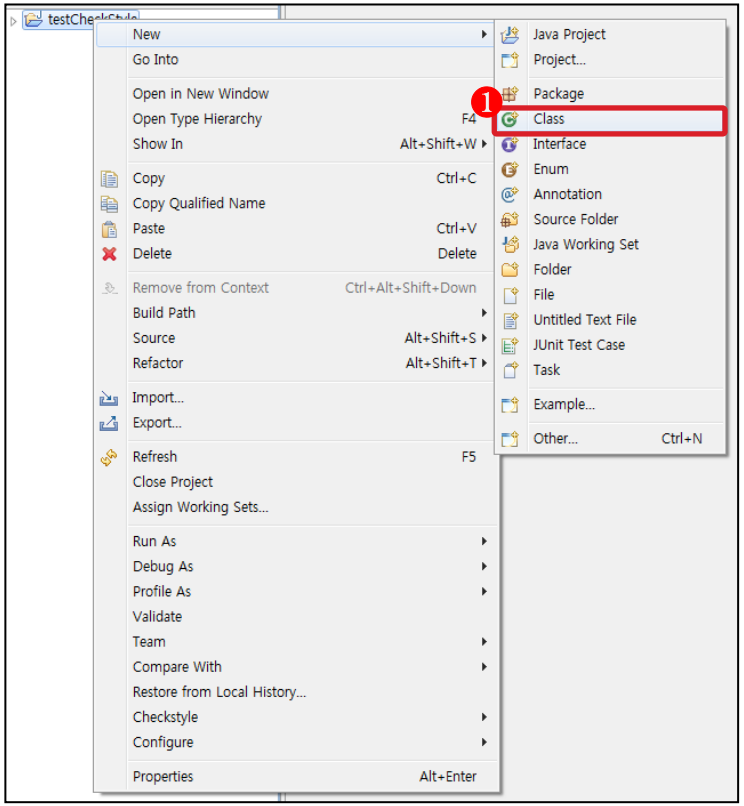


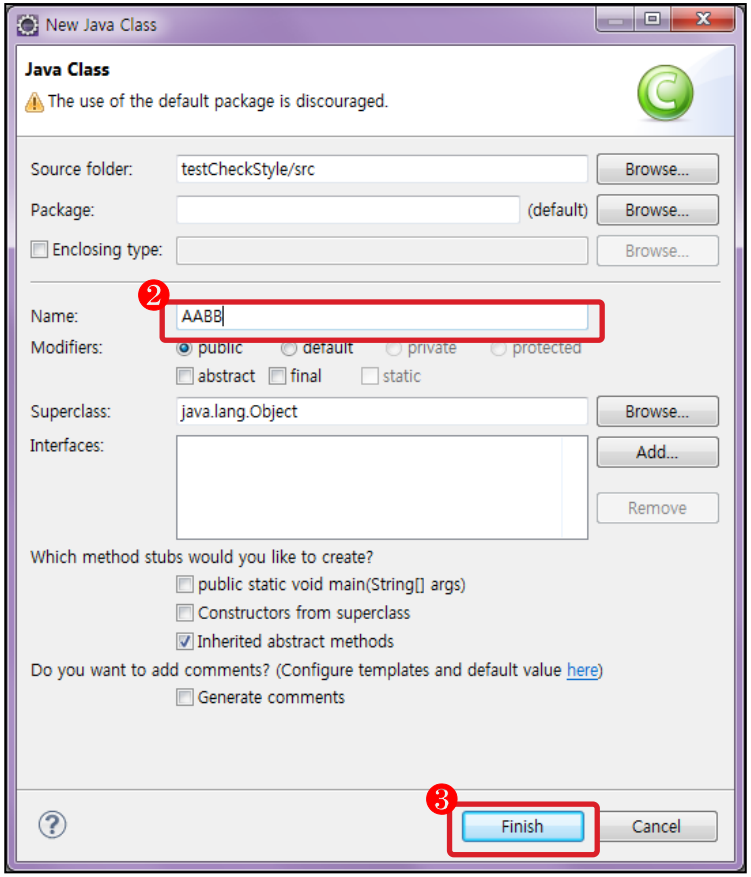
3. 주요 기능

checkstyle

3.2 예제 소개 (3/4)

- 클래스 생성
  - 프로젝트 선택 후 마우스 우 클릭 → New → Class → name 입력 → Finish
  - 본 예제에서는 name을 AABB로 입력





3. 주요 기능

checkstyle

3.2 예제 소개 (4/4)

- 소스코드 작성

```java
public class AABB {

}
```

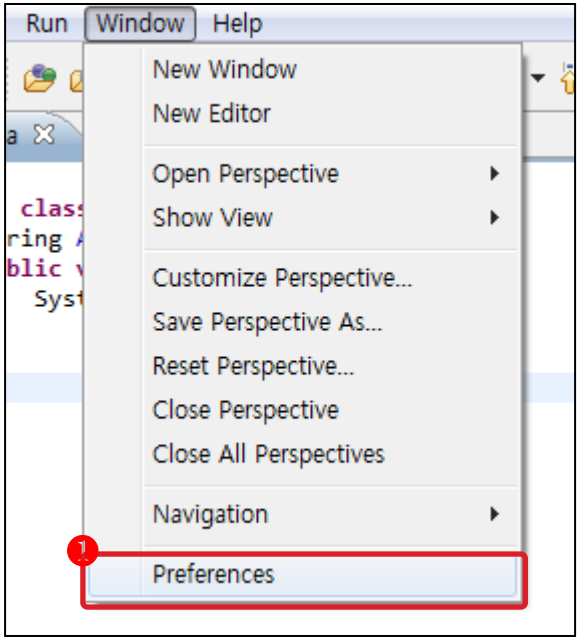
```java
public class AABB {
    String AA;
    public void ShowAA() {
        System.out.println(AA);
    }
}
```

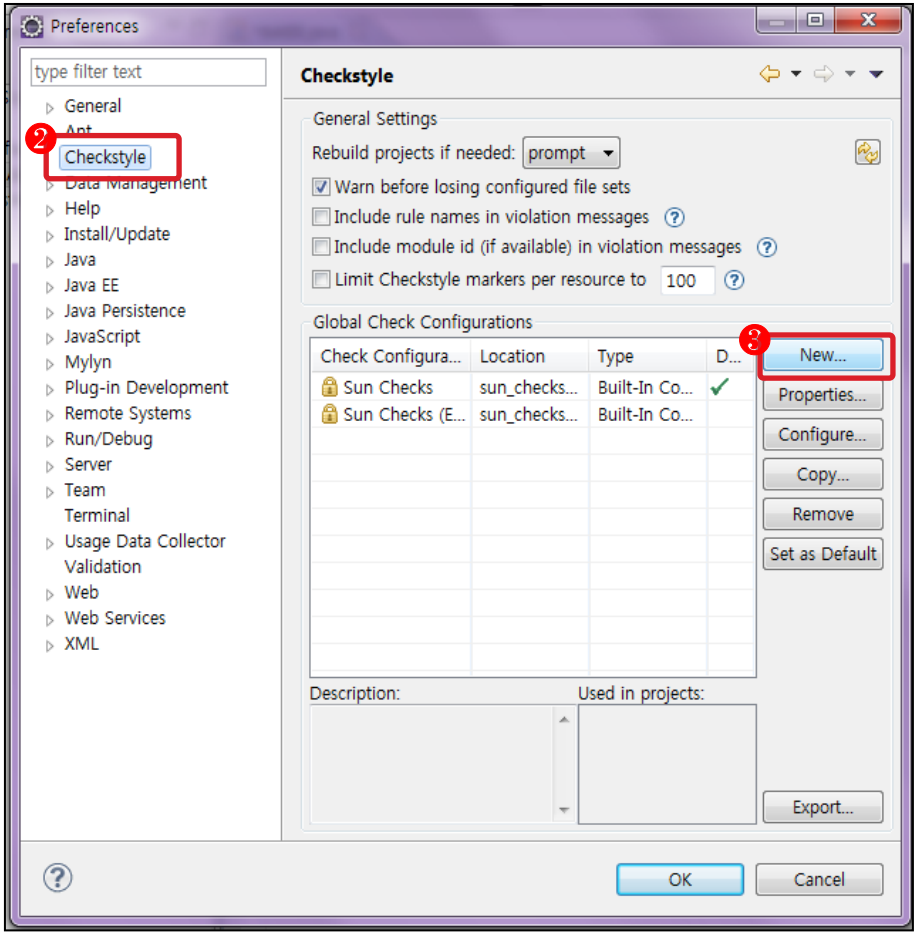
3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (1/8)

- 규칙 추가 과정
  - Window → Preferences → 왼쪽에 Checkstyle 선택 → New



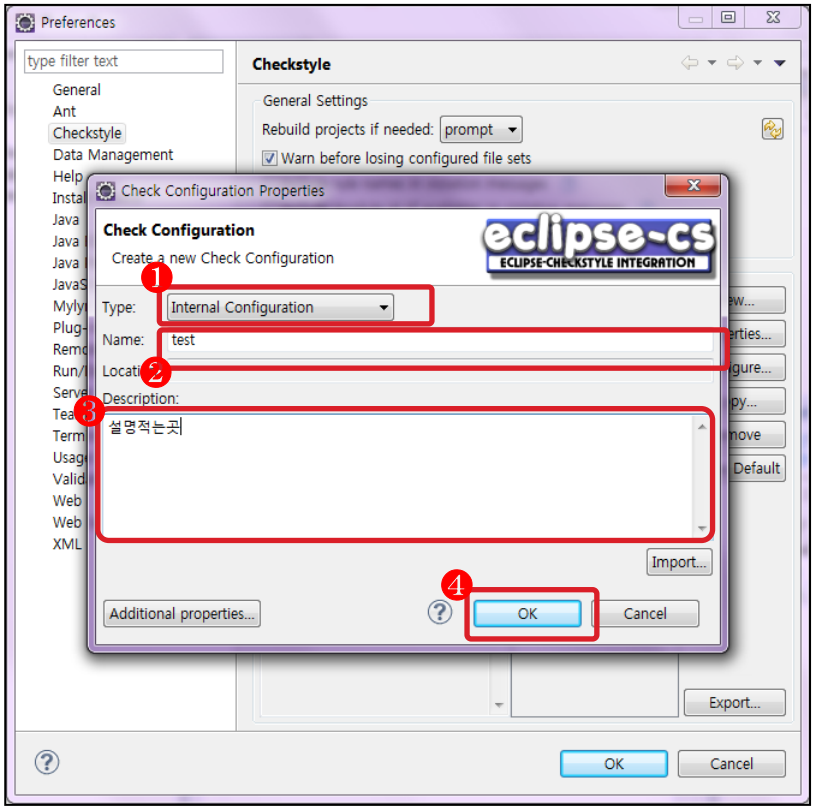


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (2/8)

- 규칙 추가 과정
  - Type선택 → Name입력 → Description 입력 → OK
  - Type을 선택. 본 예제에서는 Internal Configuration을 선택
  - Name을 test로 입력
  - Description은 규칙에 대한 설명을 입력

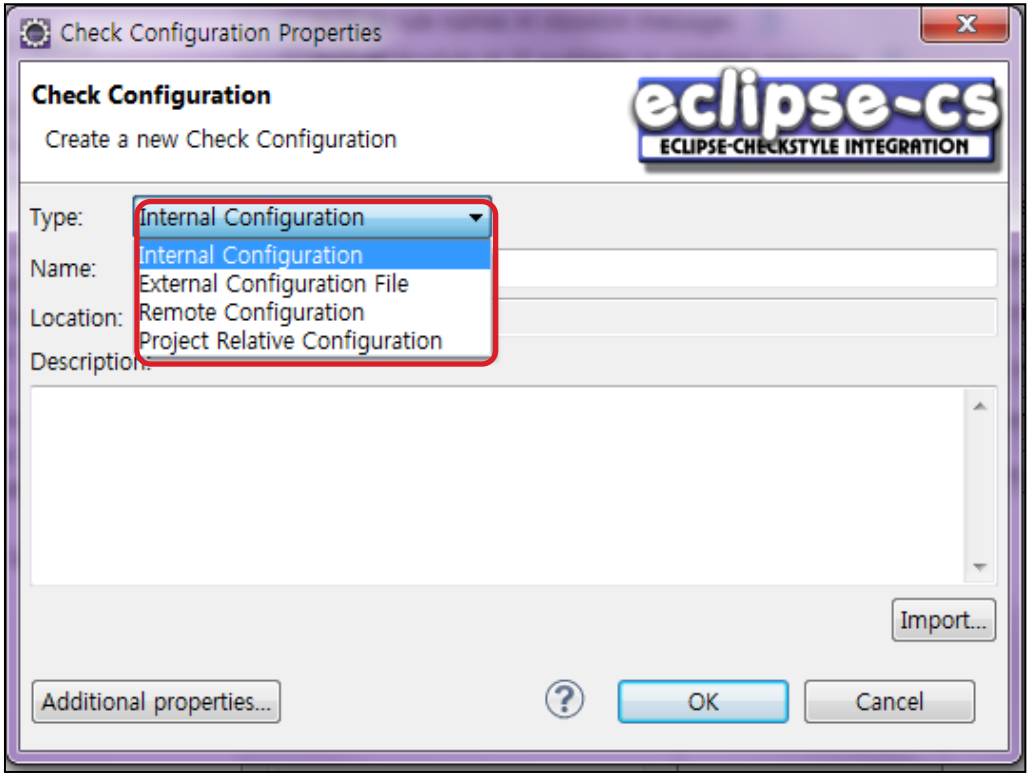


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (3/8)

- Type 설명
  - Internal Configuration , External Configuration File , Remote Configuration , Project Relative Configuration 4가지가 있음
  - 이번 예제에서는 internal Configuration을 선택



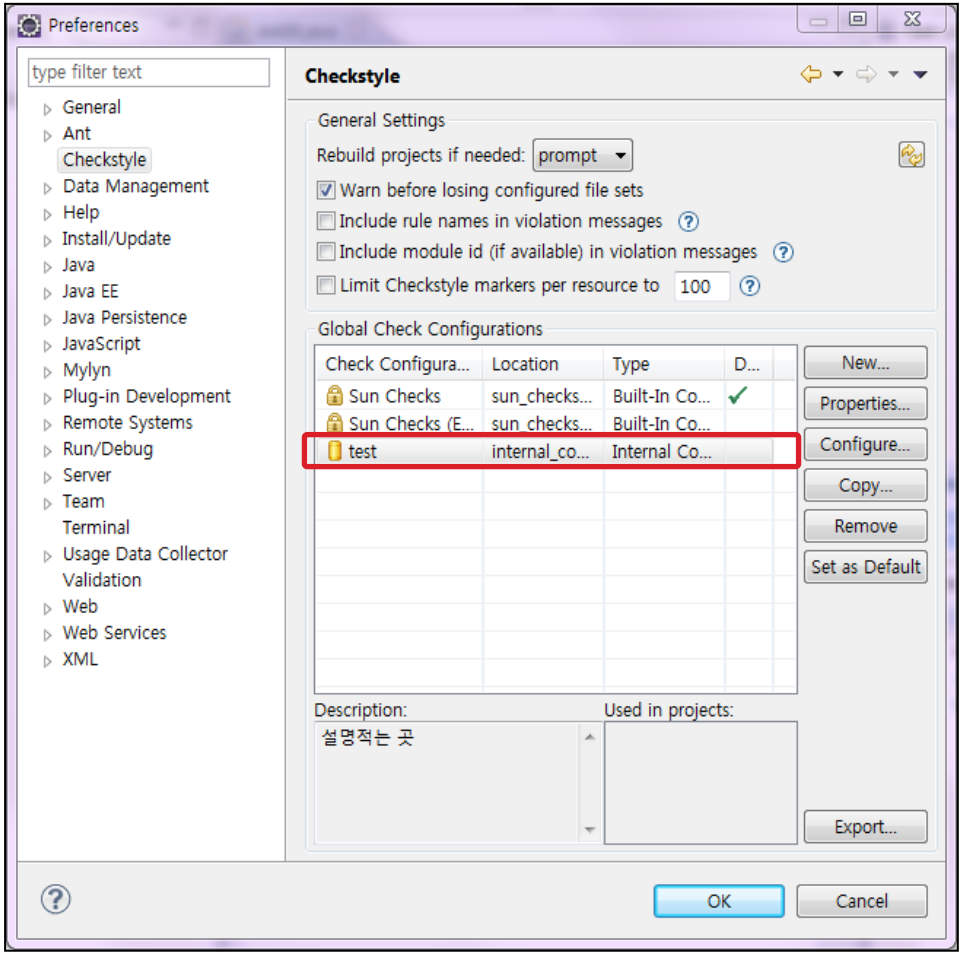
- **Internal Configuration : 규칙을 내부에 구성**
- **External Configuration File : 규칙을 외부 파일로 구성**
- **Remote Configuration : 규칙을 서버에 원격 구성**
- **Project Relative Configuration : 규칙을 프로젝트와 관련 구성**

3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (4/8)

- 규칙 추가 과정
  - Test를 더블 클릭

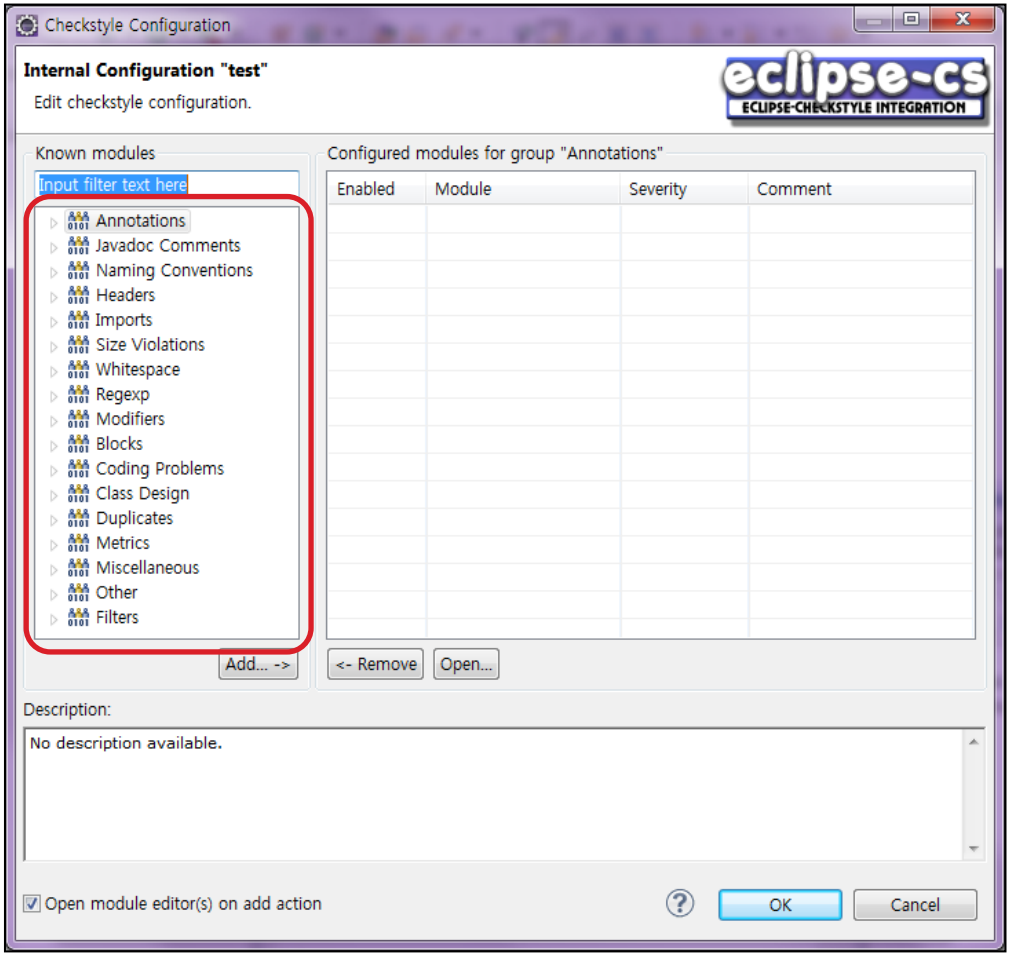


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (5/8)

- 규칙 추가 과정
  - 왼쪽 상자에 있는 규칙들을 사용해서 사용자가 원하는 규칙을 추가, 제거

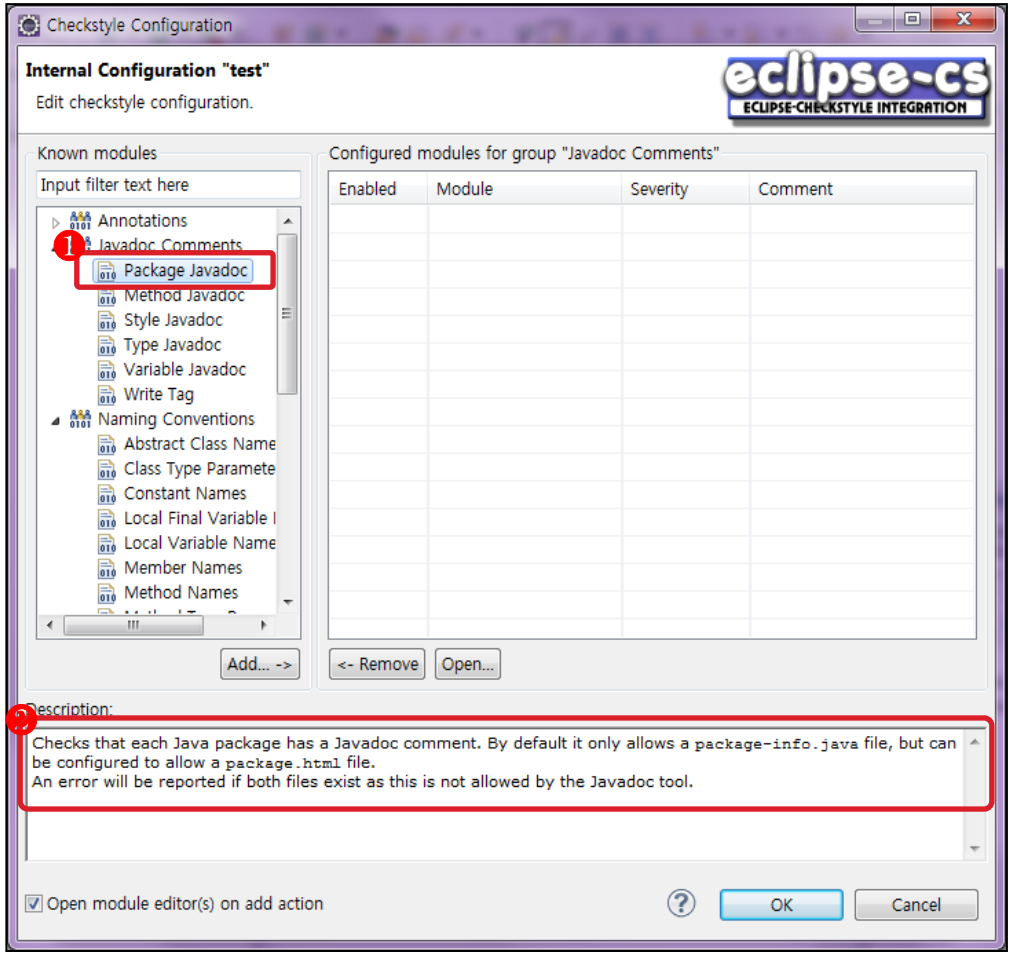


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (6/8)

- 규칙 추가 과정
  - 각 항목을 클릭하면 아래의 description에 각 규칙에 대한 자세한 설명이 나옴

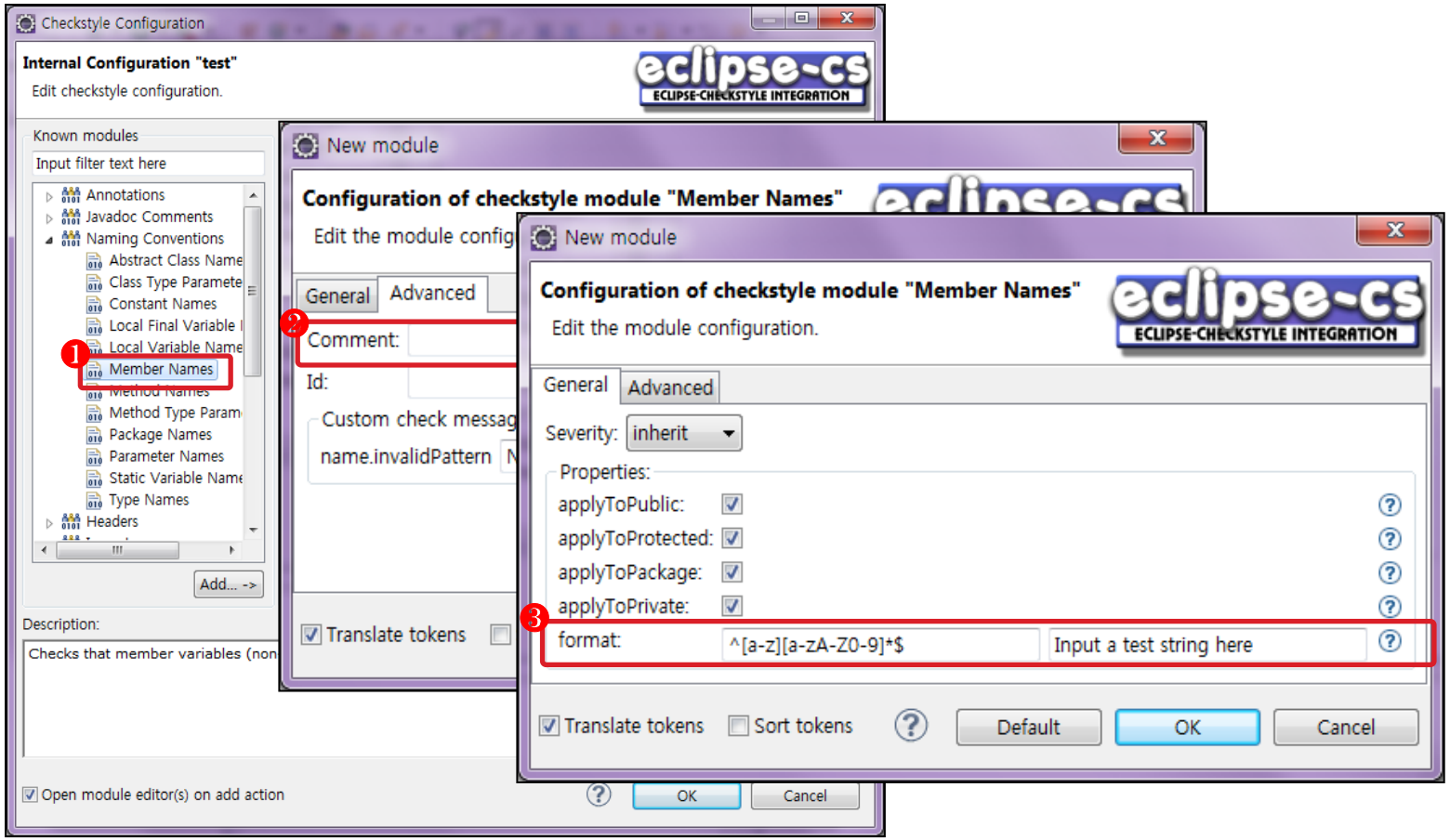


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (7/8)

- Member Names 규칙을 추가
  - Member Names → Advance → Commnet 입력 → General → Format → OK
  - Comment : 규칙에 대한 간단한 설명을 적는 곳
  - Format : 규약의 형식을 정하는 곳

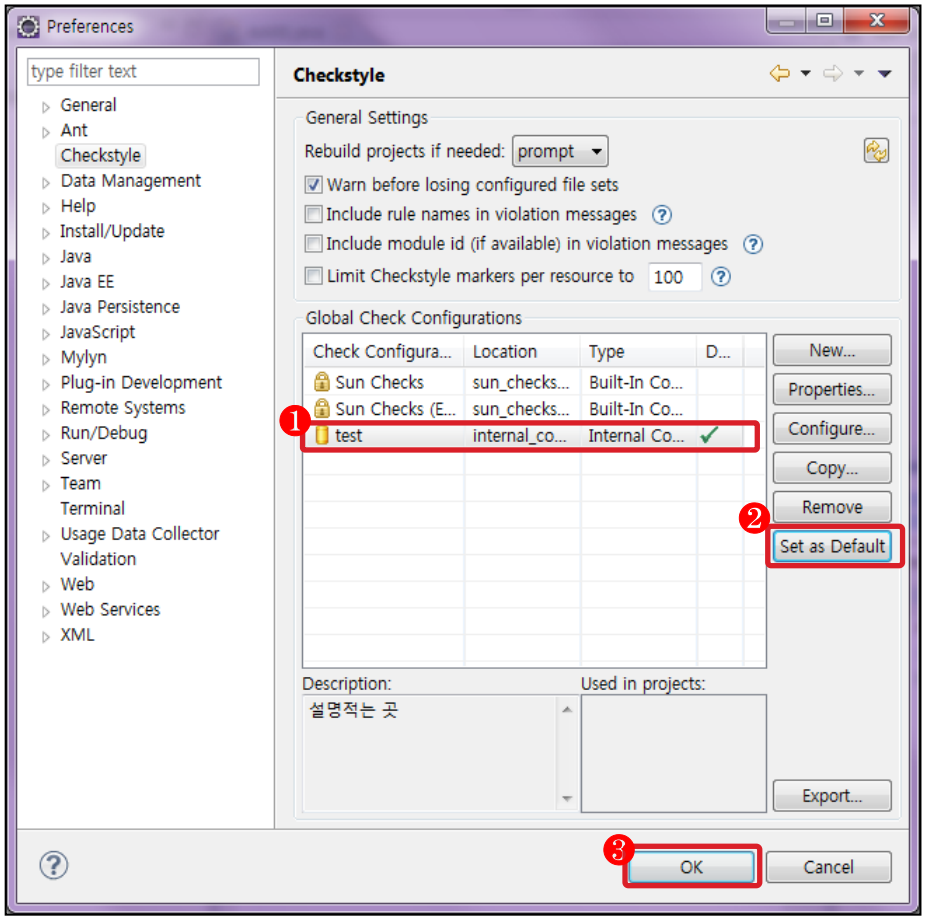


3. 주요 기능

checkstyle

3.3 Checkstyle 규칙 추가하기 (8/8)

- 규칙 추가 과정
  - Test라는 규칙이 생성
  - test규칙을 default로 하고 싶다면 test 클릭 → set default → OK

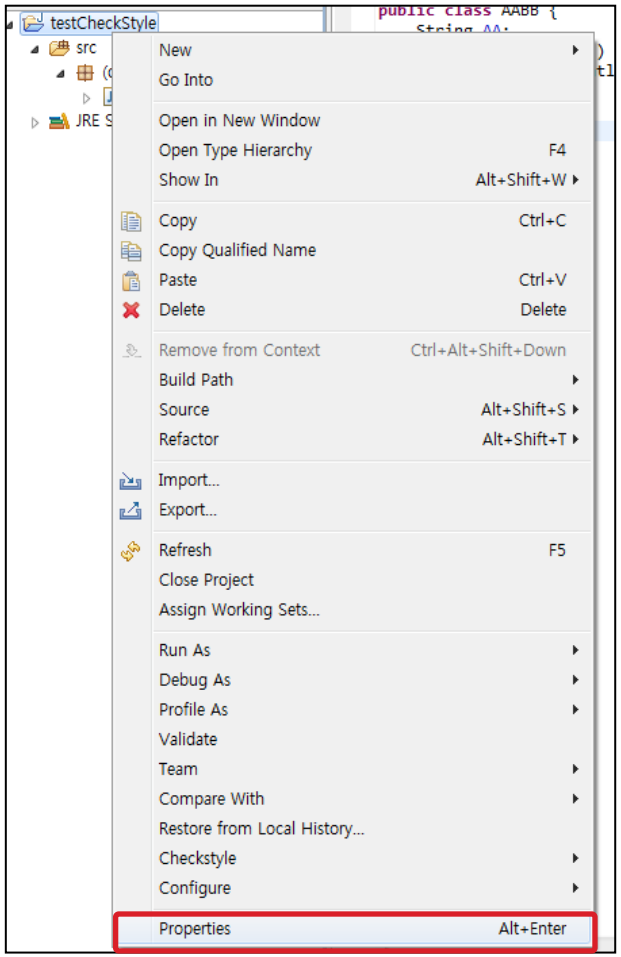


3. 주요 기능

checkstyle

3.4 Checkstyle 규칙 적용하기 (1/5)

- Checkstyle을 적용
  - testCheckStyle 프로젝트 선택 후 마우스 우 클릭 → Properties

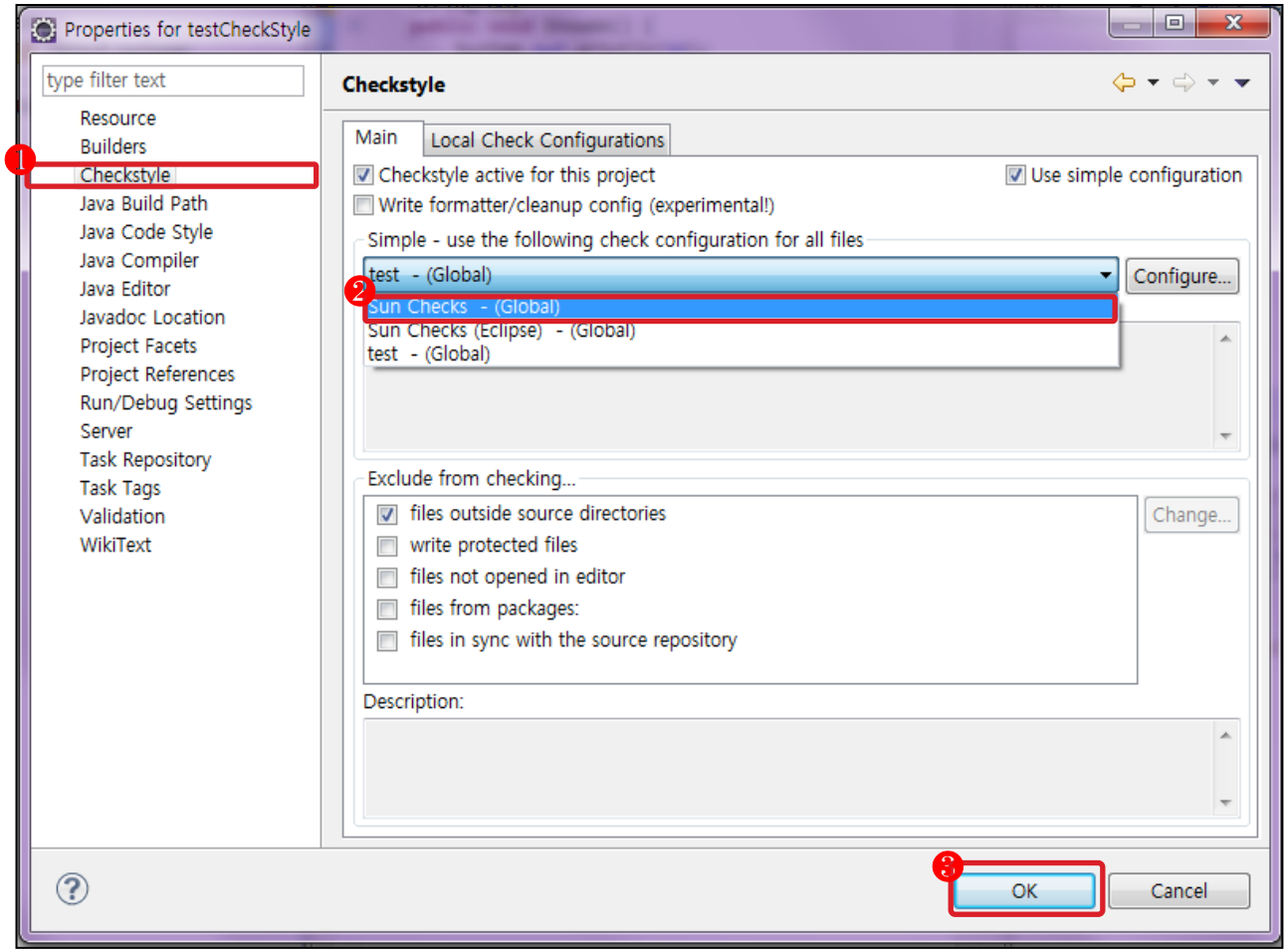


3. 주요 기능

checkstyle

3.4 Checkstyle 규칙 적용하기 (2/5)

- Checkstyle을 클릭
  - Checkstyle active for this project 체크 → 원하는 규칙 선택 : Sun Checks – (Global) 선택 → OK

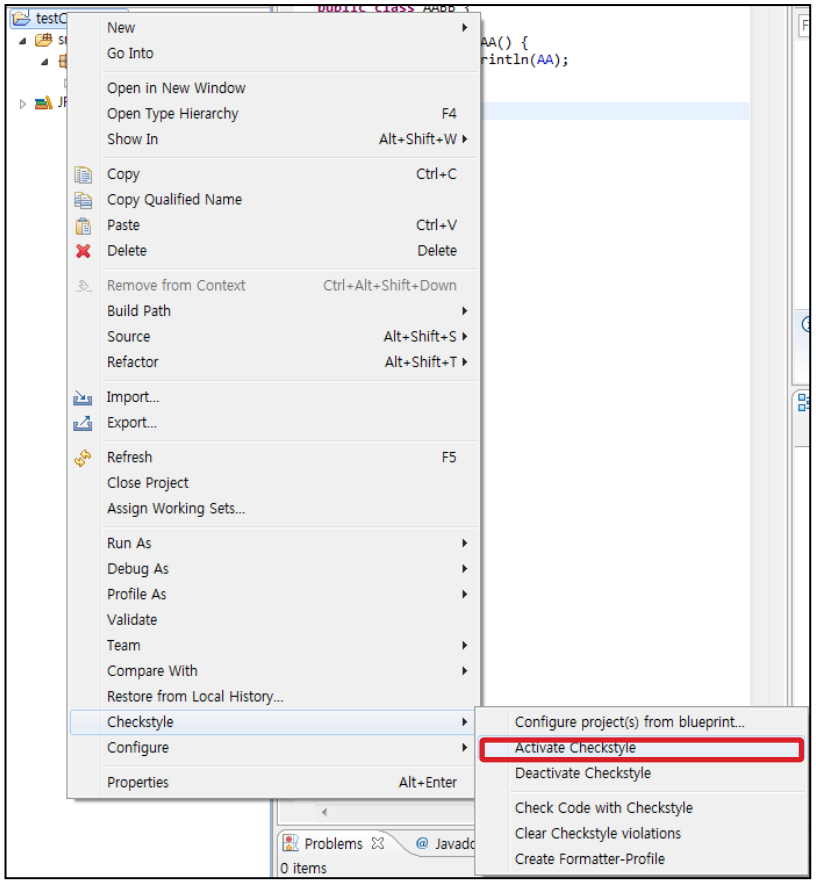


3. 주요 기능

checkstyle

3.4 Checkstyle 규칙 적용하기 (3/5)

- 팝업 메뉴 사용하기
  - 단 default값 Sun Checks - (Global)로만 적용됨
  - testCheckStyle project선택 후 마우스 우 클릭 → Checkstyle → Active Checkstyle 선택

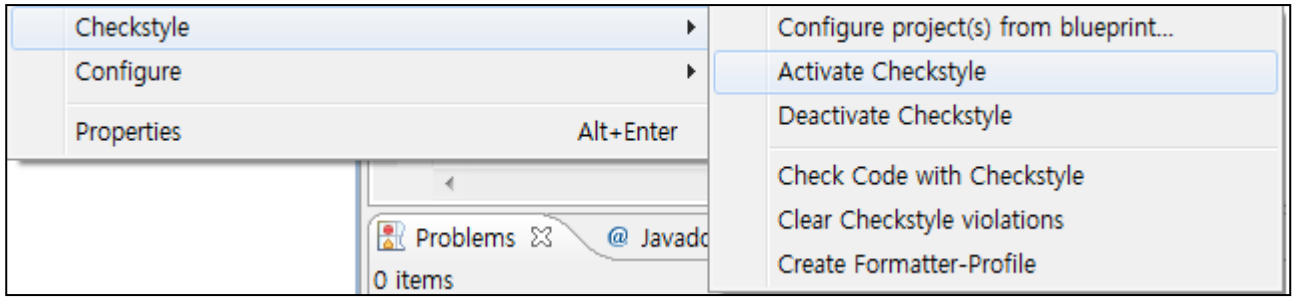


3. 주요 기능

checkstyle

3.4 Checkstyle 규칙 적용하기 (4/5)

- 팝업 메뉴 설명



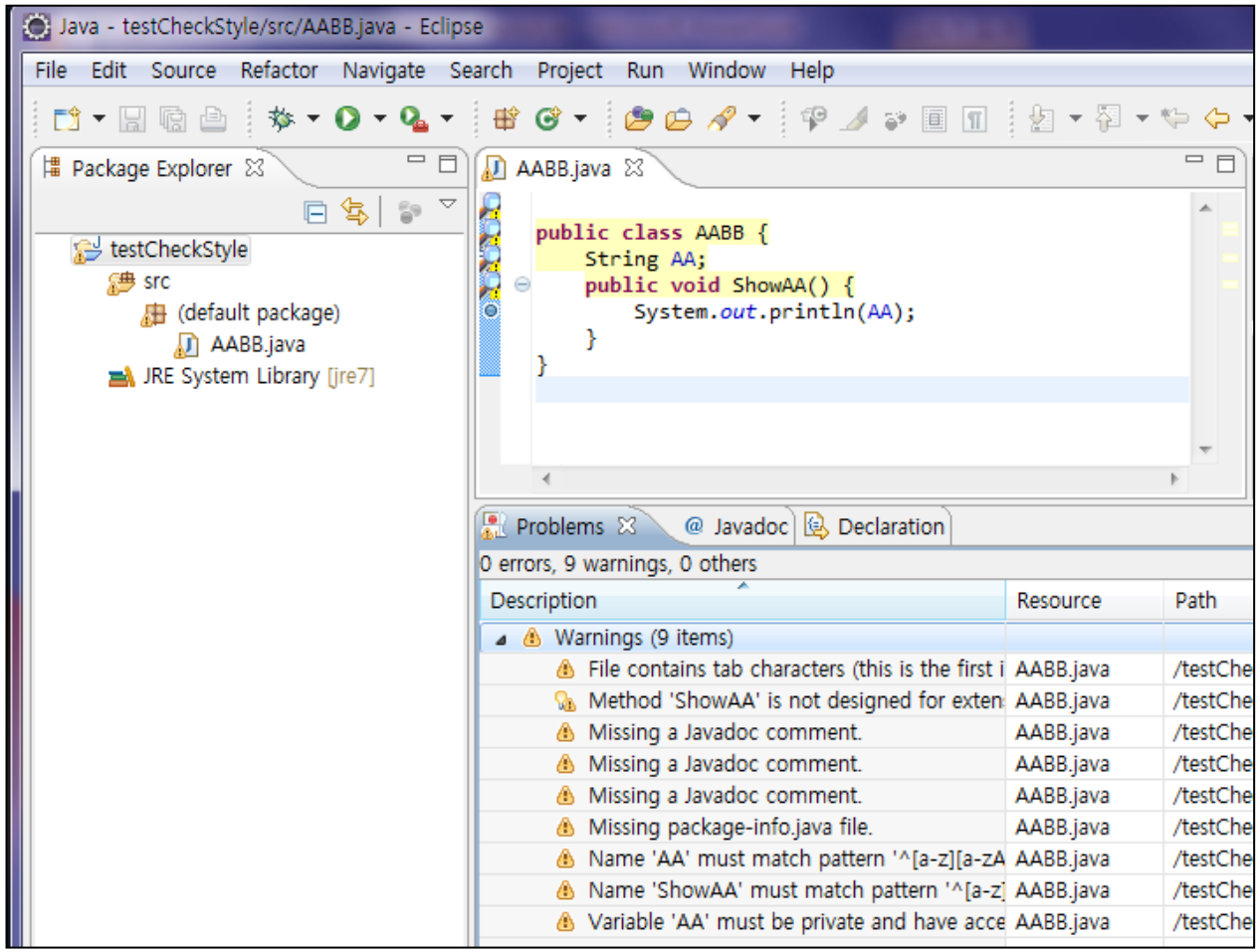
이름	설명
Configure projects(s) from blueprint...	Blueprint로 부터 선택된 프로젝트에 Checkstyle을 적용
Active Checkstyle	Checkstyle 적용
Deactive Checkstyle	Checkstyle 해지
Check Code with Checkstyle	Checkstyle로 code 검사
Clear Checkstyle violations	Checkstyle 에 의해 발견된 에러 제거
Create Formatter-Profile	Formatter-Profile 생성

3. 주요 기능

checkstyle

3.4 Checkstyle 규칙 적용하기 (5/5)

- Checkstyle 적용
  - Checkstyle의 Sun Checks – (Global) 규칙에 맞지 않아 경고 표시와 함께 노란색으로 표시



3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (1/8)

- Checkstyle은 소스 코드에 적용할 수 있는 많은 검사 규칙을 제공함.
  그 분류는 다음과 같다.

- Annotations
- Javadoc Comments
- Naming Conventions
- Headers
- Imports
- Size Violations
- Whitespace
- Regexp
- Block Checks
- Coding
- Class Design
- Metrics
- Miscellaneous

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Annotations

규칙명	규칙개요
AnnotationUseStyle	This check controls the style with the usage of annotations.
MissingDeprecated	This class is used to verify that both the java.lang.Deprecated annotation is present and the @deprecated Javadoc tag is present when either is present.
MissingOverride	This class is used to verify that the java.lang.Override annotation is present when the {@inheritDoc} javadoc tag is present.
PackageAnnotation	This check makes sure that all package annotations are in the package-info.java file.
SuppressWarnings	This check allows you to specify what warnings that SuppressWarnings is not allowed to suppress.
SuppressWarningsHolder	This check allows for finding code that should not be reported by Checkstyle.
AnnotationLocation	Check location of annotation on language elements.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Javadoc Comments

규칙명	규칙개요
JavadocPackage	Checks that all packages have a package documentation.
JavadocType	Checks the Javadoc of a type.
JavadocMethod	Checks the Javadoc of a method or constructor.
JavadocVariable	Checks that a variable has Javadoc comment.
JavadocStyle	Custom Checkstyle Check to validate Javadoc.
WriteTag	Outputs a JavaDoc tag as information.
NonEmptyAtclauseDescription	Checks that the at-clause tag is followed by description .
JavadocTagContinuationIndentation	Checks the indentation of the continuation lines in at-clauses.
SummaryJavadoc	Checks that Javadoc summary sentence does not contain phrases that are not recommended to use.
AtclauseOrder	Checks the order of at-clauses.
JavadocParagraph	Checks Javadoc paragraphs.
SingleLineJavadoc	Checks that a JavaDoc block which can fit on a single line and doesn't contain at-clauses.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Naming Conventions

규칙명	규칙개요
AbbreviationAsWordInName	The Check validate abbreviations(consecutive capital letters) length in identifier name, it also allow in enforce camel case naming.
AbstractClassName	Ensures that the names of abstract classes conforming to some regular expression.
ClassTypeParameterName	Checks that class type parameter names conform to a format specified by the format property.
ConstantName	Checks that constant names conform to a format specified by the format property.
InterfaceTypeParameterName	Checks that interface type parameter names conform to a format specified by the format property.
LocalFinalVariableName	Checks that local final variable names conform to a format specified by the format property.
LocalVariableName	Checks that local, non-final variable names conform to a format specified by the format property.
MemberName	Checks that instance variable names conform to a format specified by the format property.
MethodName	Checks that method names conform to a format specified by the format property.
MethodTypeParameterName	Checks that class type parameter names conform to a format specified by the format property.
PackageName	Checks that package names conform to a format specified by the format property.
ParameterName	Checks that parameter names conform to a format specified by the format property.
StaticVariableName	Checks that static, non-final variable names conform to a format specified by the format property
TypeName	Checks that type names conform to a format specified by the format property.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Headers

규칙명	규칙개요
Header	Checks the header of the source against a fixed header file.
RegexpHeader	Checks the header of a source file against a header that contains a regular expression for each line of the source header.

- Imports

규칙명	규칙개요
AvoidStarImport	Check that finds import statements that use the * notation.
AvoidStaticImport	Check that finds static imports.
IllegalImport	Checks for imports from a set of illegal packages.
RedundantImport	Checks for imports that are redundant.
UnusedImports	Checks for unused import statements.
ImportOrder	Ensures that groups of imports come in a specific order.
ImportControl	Check that controls what packages can be imported in each package.
CustomImportOrder	Checks that the groups of import declarations appear in the order specified by the user.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Size Violations

규칙명	규칙개요
ExecutableStatementCount	Restricts the number of executable statements to a specified limit (default = 30).
FileLength	Checks for long source files.
LineLength	Checks for long lines.
MethodLength	Checks for long methods.
AnonInnerLength	Checks for long anonymous inner classes.
ParameterNumber	Checks the number of parameters that a method or constructor has.
OuterTypeNumber	Checks for the number of defined types at the "outer" level.
MethodCount	Checks the number of methods declared in each type.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Whitespace

규칙명	규칙개요
GenericWhitespace	Checks that the whitespace around the Generic tokens < and > are correct to the *typical* convention.
EmptyForInitializerPad	Checks the padding of an empty for initializer; that is whether a space is required at an empty for initializer, or such spaces are forbidden.
EmptyForIteratorPad	Checks the padding of an empty for iterator; that is whether a space is required at an empty for iterator, or such spaces are forbidden.
MethodParamPad	Checks the padding between the identifier of a method definition, constructor definition, method call, or constructor invocation; and the left parenthesis of the parameter list.
NoWhitespaceAfter	Checks that there is no whitespace after a token.
NoWhitespaceBefore	Checks that there is no whitespace before a token.
OperatorWrap	Checks line wrapping for operators.
ParenPad	Checks the padding of parentheses; that is whether a space is required after a left parenthesis and before a right parenthesis, or such spaces are forbidden, with the exception that it does not check for padding of the right parenthesis at an empty for iterator.
TypecastParenPad	Checks the padding of parentheses for typecasts.
FileTabCharacter	Checks to see if a file contains a tab character.
WhitespaceAfter	Checks that a token is followed by whitespace, with the exception that it does not check for whitespace after the semicolon of an empty for iterator.
WhitespaceAround	Checks that a token is surrounded by whitespace.
NoLineWrap	Checks that chosen statements are not line-wrapped.
EmptyLineSeparator	Checks for blank line separators.
SeparatorWrap	Checks line wrapping with separators.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Regexp

규칙명	규칙개요
Regexp	A check that makes sure that a specified pattern exists (or not) in the file.
RegexpHeader	Checks the header of the source against a header file that contains a
RegexpMultiline	Implementation of a check that looks that matches across multiple lines in any file type.
RegexpSingleline	Implementation of a check that looks for a single line in any file type.
RegexpSinglelineJava	Implementation of a check that looks for a single line in Java files.

- Block Checks

규칙명	규칙개요
AnnotationUseStyle	This check controls the style with the usage of annotations.
MissingDeprecated	This class is used to verify that both the java.lang.Deprecated annotation is present and the @deprecated Javadoc tag is present when either is present.
MissingOverride	This class is used to verify that the java.lang.Override annotation is present when the {@inherit Doc} javadoc tag is present.
PackageAnnotation	This check makes sure that all package annotations are in the package-info.java file.
SuppressWarnings	This check allows you to specify what warnings that SuppressWarnings is not allowed to suppress.
SuppressWarningsHolder	This check allows for finding code that should not be reported by Checkstyle.
AnnotationLocation	Check location of annotation on language elements.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Coding

규칙명	규칙개요
ArrayTrailingComma	Checks if array initialization contains optional trailing comma.
AvoidInlineConditionals	Detects inline conditionals.
CovariantEquals	Checks that if a class defines a covariant method equals, then it defines method equals(java.lang.Object).
EmptyStatement	Detects empty statements (standalone ';').
EqualsAvoidNull	Checks that any combination of String literals with optional assignment is on the left side of an equals() comparison.
EqualsHashCode	Checks that classes that override equals() also override hashCode().
FinalLocalVariable	Ensures that local variables that never get their values changed, must be declared final.
HiddenField	Checks that a local variable or a parameter does not shadow a field that is defined in the same class.
IllegalInstantiation	Checks for illegal instantiations where a factory method is preferred.
IllegalToken	Checks for illegal tokens.
IllegalTokenText	Checks for illegal token text.
InnerAssignment	Checks for assignments in subexpressions, such as in String s = Integer.toString(i = 2);.
MagicNumber	Checks for magic numbers.
MissingSwitchDefault	Checks that switch statement has "default" clause.
ModifiedControlVariable	Check for ensuring that for loop control variables are not modified inside the for block.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Coding

규칙명	규칙개요
SimplifyBooleanExpression	Checks for overly complicated boolean expressions.
SimplifyBooleanReturn	Checks for overly complicated boolean return statements.
StringLiteralEquality	Checks that string literals are not used with == or !=.
NestedForDepth	Restricts nested for blocks to a specified depth (default = 1).
NestedIfDepth	Restricts nested if-else blocks to a specified depth (default = 1).
NestedTryDepth	Restricts nested try-catch-finally blocks to a specified depth (default = 1).
NoClone	Checks that the clone method is not overridden from the Object class.
NoFinalizer	Checks that no method having zero parameters is defined using the name *finalize*.
SuperClone	Checks that an overriding clone() method invokes super.clone().
SuperFinalize	Checks that an overriding finalize() method invokes super.finalize().
IllegalCatch	Catching java.lang.Exception, java.lang.Error or java.lang.RuntimeException is almost never acceptable.
IllegalThrows	Throwing java.lang.Error or java.lang.RuntimeException is almost never acceptable.
PackageDeclaration	Ensures there is a package declaration and (optionally) in the correct directory.
ReturnCount	Restricts return statements to a specified count (default = 2).

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Coding

규칙명	규칙개요
IllegalType	Checks that particular class are never used as types in variable declarations, return values or parameters.
DeclarationOrder	Checks that the parts of a class or interface declaration appear in the order suggested by the Code Conventions for the Java Programming Language.
ParameterAssignment	Disallow assignment of parameters.
ExplicitInitialization	Checks if any class or object member explicitly initialized to default for its type value (null for object references, zero for numeric types and char and false for boolean.
DefaultComesLast	Check that the default is after all the cases in a switch statement.
MissingCtor	Checks that classes (except abstract one) define a ctor and don't rely on the default one.
FallThrough	Checks for fall through in switch statements Finds locations where a case contains Java code - but lacks a break, return, throw or continue statement.
MultipleStringLiterals	Checks for multiple occurrences of the same string literal within a single file.
MultipleVariableDeclarations	Checks that each variable declaration is in its own statement and on its own line.
RequireThis	Checks that code doesn't rely on the "this" default.
UnnecessaryParentheses	Checks if unnecessary parentheses are used in a statement or expression.
OneStatementPerLine	Checks there is only one statement per line.
VariableDeclarationUsageDistance	Checks the distance between declaration of variable and its first usage.
OverloadMethodsDeclarationOrder	Checks that overload methods are grouped together.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Class Design

규칙명	규칙개요
VisibilityModifier	Checks visibility of class members.
FinalClass	Checks that class which has only private ctors is declared as final.
InterfaceIsType	Implements Bloch, Effective Java, Item 17 - Use Interfaces only to define types.
HideUtilityClassConstructor	Make sure that utility classes (classes that contain only static methods) do not have a public constructor.
DesignForExtension	Checks that classes are designed for inheritance.
MutableException	Ensures that exceptions (defined as any class name conforming to some regular expression) are immutable.
ThrowsCount	Restricts throws statements to a specified count (default = 1).
InnerTypeLast	Check nested (internal) classes/interfaces are declared at the bottom of the class after all met hod and field declarations.
OneTopLevelClass	Checks that each top-level class, interfaces or enum resides in a source file of its own.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Metrics

규칙명	규칙개요
BooleanExpressionComplexity	Restricts nested boolean operators (&&, ||, &, | and ^) to a specified depth (default = 3).
ClassDataAbstractionCoupling	This metric measures the number of instantiations of other classes within the given class.
ClassFanOutComplexity	The number of other classes a given class relies on.
CyclomaticComplexity	Checks cyclomatic complexity against a specified limit.
NPathComplexity	Checks the npath complexity against a specified limit (default = 200).
JavaNCSS	This check calculates the Non Commenting Source Statements (NCSS) metric for Java source files and methods.

3. 주요 기능

checkstyle

3.5 Checkstyle 제공 규칙 (2/8)

- Miscellaneous

규칙명	규칙개요
NewlineAtEndOfFile	Checks that there is a newline at the end of each file.
TodoComment	A check for TODO comments.
CommentsIndentation	CommentsIndentation
Translation	The TranslationCheck class helps to ensure the correct translation of code by checking property files for consistency regarding their keys.
UncommentedMain	Checks for uncommented main() methods.
UpperEll	Checks that long constants are defined with an upper ell.
ArrayTypeStyle	Checks the style of array type definitions.
FinalParameters	Check that method/constructor/catch/foreach parameters are final.
DescendantToken	Checks for restricted tokens beneath other tokens.
Indentation	Checks correct indentation of Java Code.
TrailingComment	The check to ensure that requires that comments be the only thing on a line.
OuterTypeFilename	Checks that the outer type name and the file name match.
UniqueProperties	Detects duplicated keys in properties files.
AvoidEscapedUnicodeCharacters	Restrict using Unicode escapes.
FileContentsHolder	Holds the current file contents for global access when configured as a TreeWalker sub-module.

4. 활용 예제

checkstyle

세부 목차

4. 활용 예제

checkstyle

4.1 예제 소개 (1/3)

- 퀵 정렬 구현
  - 분할 알고리즘을 이용한 Quicksort의 code를 작성하고 code style을 체크하는 예
  - 먼저 분할 알고리즘과 퀵 정렬에 대한 설명

퀵정렬 알고리즘 설명

- 퀵 정렬은 1962년에 C.A.R Hoare가 만든 정렬 알고리즘으로 정렬 알고리즘 중 가장 우수한 평균 수행속도를 자랑

- 분할 알고리즘(Partition Algorithm)은 퀵 정렬의 기본 개념

- 분할 알고리즘을 기본으로 사용하여 정렬하고자 하는 배열을 2개의 하위 배열로 분할하고 각각의 하위 배열에서는 다시 재귀적으로 자신의 배열을 분할하여 결국 배열을 정렬하는 알고리즘

- 멀리 떨어진 자료를 직접적으로 치환함으로써 정렬의 수행속도를 개선한 알고리즘

분할 알고리즘 설명

- 자료들을 분할한다는 것은 그것을 2개의 그룹들로 나눈다는 의미이며 배열의 양끝 방향 즉, 왼쪽 끝에서 오른쪽으로 그리고 오른쪽 끝에서 왼쪽으로 탐색을 시작해서 각각 피봇 값보다 큰 값과 작은 값을 발견하면 그 값들을 서로 치환하는 방식

예제)
```
//피봇 값보다 크거나 같은 값 찾기
while(theArray[++left] < pivot);

//피봇 값보다 작은 값찾기
while(theArray[--right] >= pivot) ;

//피봇값보다 크거나 같은 값과 작은 값을 서로 치환
swap(left, right);
```

4. 활용 예제

checkstyle

4.1 예제 소개 (2/3)

- 퀵 정렬 구현
  - 퀵 정렬을 구현하는 데 필요한 요구사항

요구사항

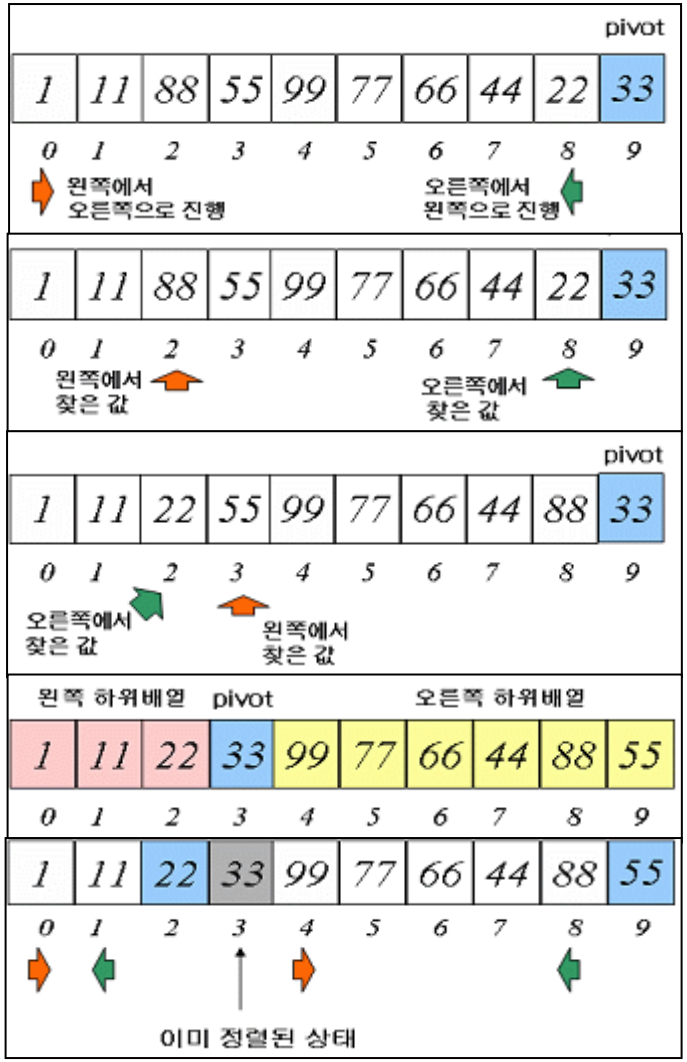
- 다음의 순서로 퀵 정렬 알고리즘을 구현

- 첫 번째, 배열을 2개의 하위배열 즉, 왼쪽 하위배열은 피봇 값보다 작거나 같은 값으로 이루어진 배열 그리고 오른쪽 하위배열은 피봇 값보다 큰 값으로 이루어진 배열로 분할

- 두 번째, 왼쪽 하위배열에 퀵 정렬을 다시 실행

- 세 번째, 오른쪽 하위배열에 퀵 정렬을 다시 실행

- 이 과정을 계속해서 진행하되 서로의 자리 값이 엇갈리면 피봇 값과 왼쪽에서 찾은 값을 서로 바꿈



4. 활용 예제

checkstyle

4.1 예제 소개 (3/3)

- 퀵 정렬을 구현하는 데에 있어 몇 가지 이슈 상황

각자 작성한 코드를 한 눈에 파악하는 데 어려움

코드가 정리되지 않고 특정한 규칙이 없는 상황

다른 개발자가 구현한 코드를 한 가지 규칙을 적용하기가 어려움

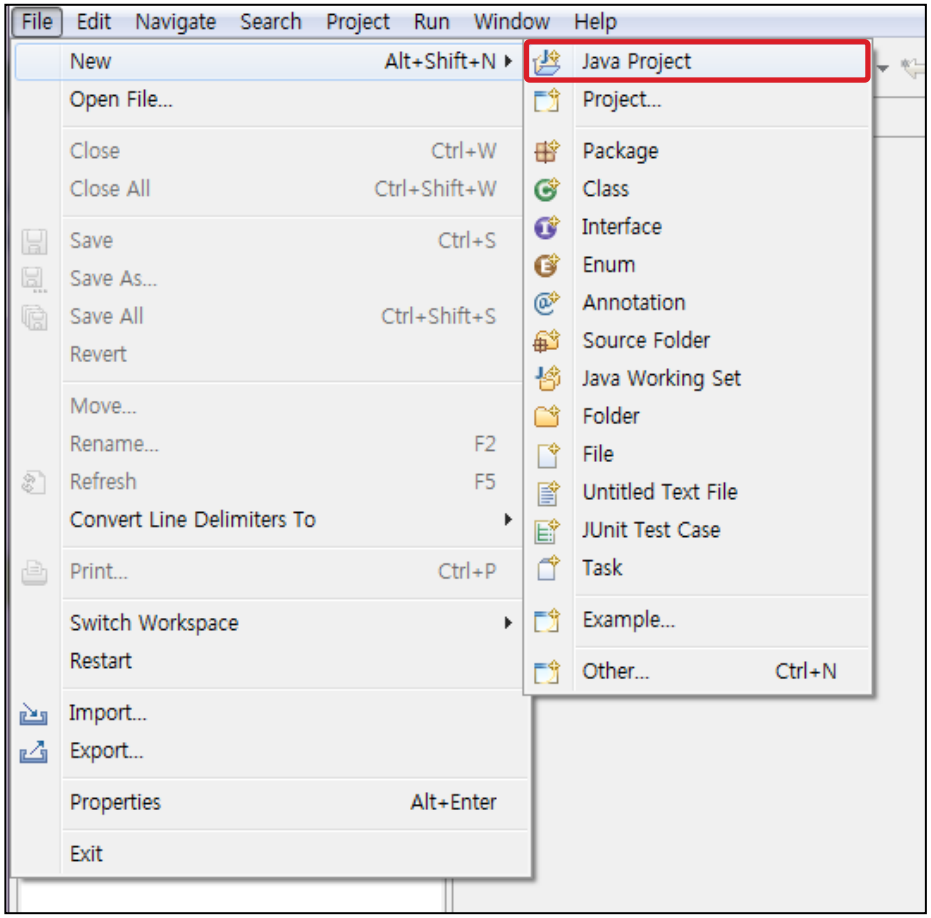
Checkstyle을 도입하여 소스 스타일 관리를 진행하기로 결정

4. 활용 예제

checkstyle

4.2 예제 생성하기 (1/5)

- 퀵 정렬을 위한 코드를 작성
- 임의의 Java 프로젝트를 생성하기 위해 다음의 절차를 따름
  - File → New → Java Project

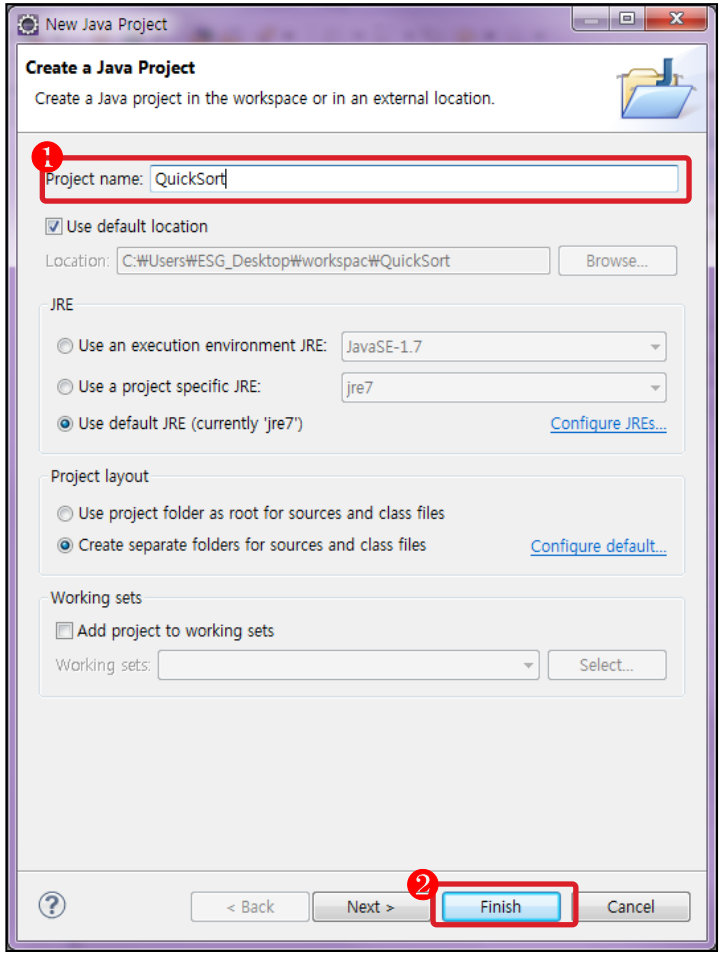


4. 활용 예제

checkstyle

4.2 예제 생성하기 (2/5)

– Project 이름 입력 → Finish
  • 본 예제에서는 Project name을 QuickSort로 입력

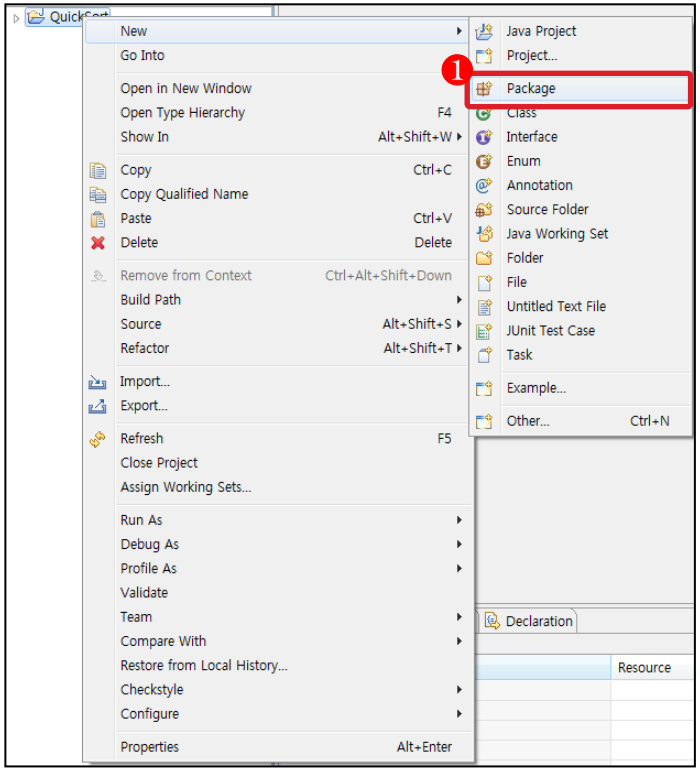


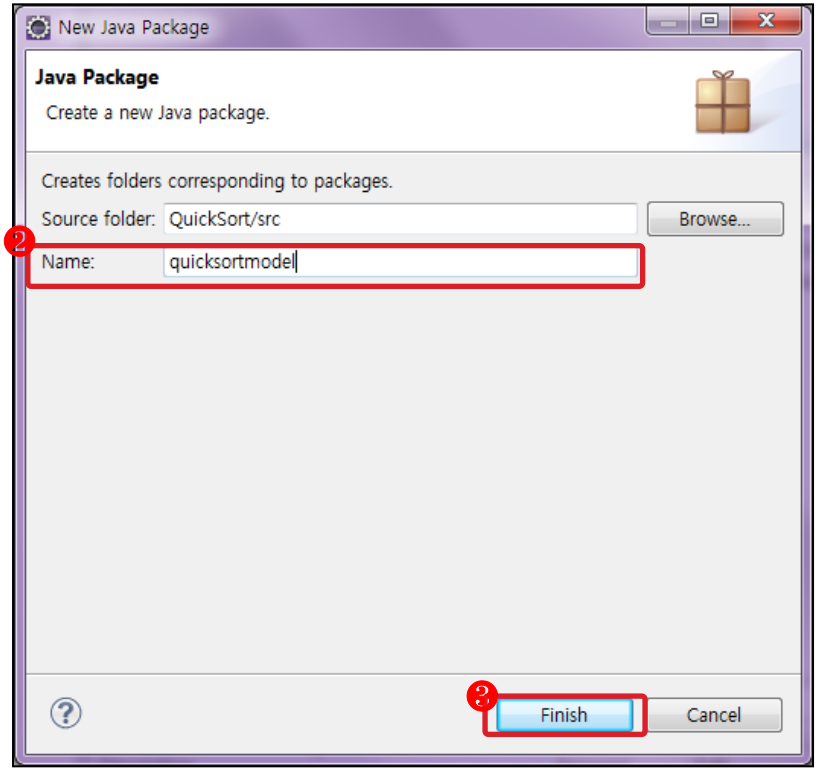
4. 활용 예제

checkstyle

4.2 예제 생성하기 (3/5)

- quicksortmodel 패키지 생성하기
  - Quicksort 프로젝트 선택 후 마우스 우 클릭 → Package 선택 → Package 이름 입력 → Finish
    - 본 매뉴얼에서는 Name을 quicksortmodel로 입력



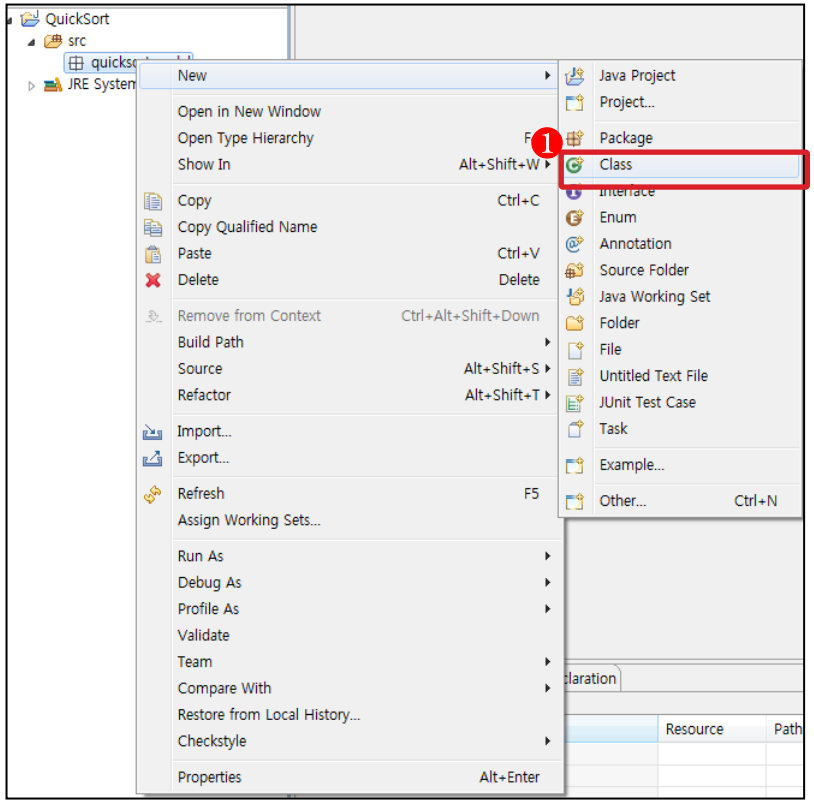


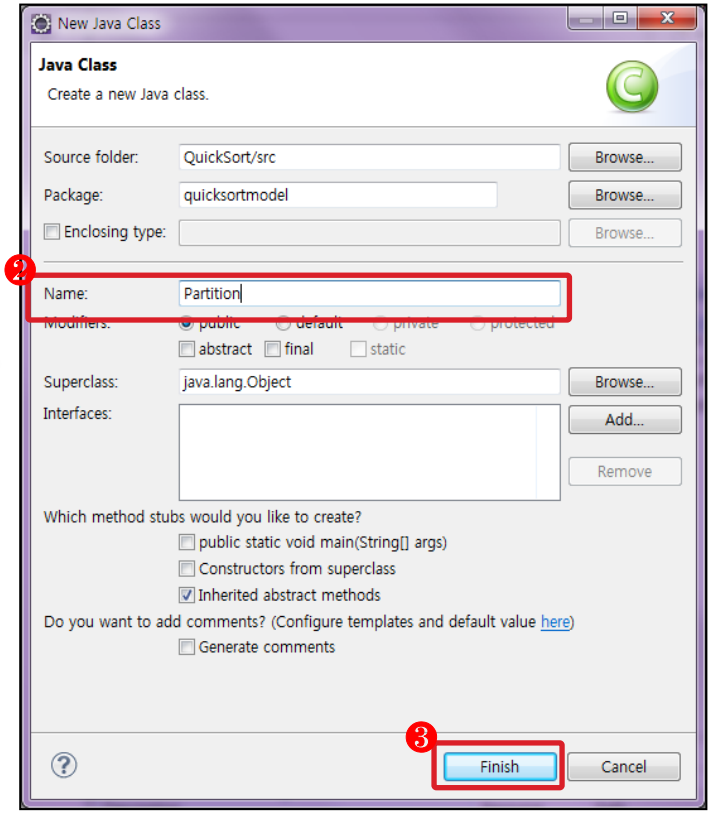
4. 활용 예제

checkstyle

4.2 예제 생성하기 (4/5)

- Partition 클래스 생성하기
  - Quicksortmodel패키지에서 클래스를 생성
    - > Quicksortmedel 선택 후 마우스 우 클릭 → New → Class
    - > 본 매뉴얼에서는 Partition이란 클래스를 생성





4. 활용 예제

checkstyle

4.2 예제 생성하기 (5/5)

- Partition클래스에 테스트 할 quicksort 코드를 작성
  - 아래 그림은 quicksort의 전체 소스코드예제
    - Public Int getRandomNum() – 난수를 발생하여 리턴
    - Public int randomDivide() – 난수들을 가진 배열을 이용하여 퀵 정렬
    - Public int swap() – 퀵정렬 중 swap이 필요할 때 호출

```java
package quicksortmodel;

import java.util.Random;

import com.sun.org.apache.xalan.internal.xsltc.compiler.sym;

public class Partition {
    static int cnt=1, ind=1;
    public int getRandomNum(int num) {
        int result = 0;
        Random rangen = new Random();
        result = rangen.nextInt() % num;

        if(result < 0) result =result * -1;
        return result;
    }

    public int randomDivide(int[] arr, int leftmost, int rightmost) {
        Partition middle = new Partition();
        int key;
        if((rigthmost-leftmost) <= 0)
            return 0;
        else
        {
            key = middle.getRandomNum(rightmost-leftmost+1)+leftmost;
            int pivot = swap(arr, key, rightmost);
            if(ind%10==0) {
                ind++;
            }else{
                ind++;
            }
            int i = leftmost-1;
            int j = rightmost;
            while(true) {
                while(arr[++i] < pivot);
                while(j>leftmost && arr[--j] >= pivot);
                if(i >= j)
                    break;
                else
                    swap(arr, i, j);
            }
            swap(arr, i, rightmost);
            randomDivide(arr, leftmost, i-1);
            randomDivide(arr, i+1, rightmost);
        }
        return key;
    }
```

```java
    public int swap(int[] arr, int left, int right) {
        int temp;
        temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
        return temp;
    }

    public static void main(String args[]) {
        int i, size=1000, left, right, pivot;
        int[] theArray = new int[size];
        Partition middle = new Partition();

        for(i=0;i<size;i++) {
            theArray[i] = middle.getRandomNum(size+1);
        }
        left=0;
        right=size-1;
        pivot = middle.randomDivide(theArray, left, right);
        System.out.println("\n\n-정렬된 후의 값>\n");
        System.out.println("----------------------------------------\n\t");
        int a=0;
        for(i=0;i<size;i++) {
            System.out.println("["+(a++)+"]"+"\t");
            int ct = i;
            for(int j=ct;j<ct+20;j++) {
                System.out.println(theArray[j]+"\t");
                i++;
            }
            i -= 1;
            System.out.println();
        }
    }
}
```

4. 활용 예제

checkstyle

4.3 checkstyle 적용하기 (1/4)

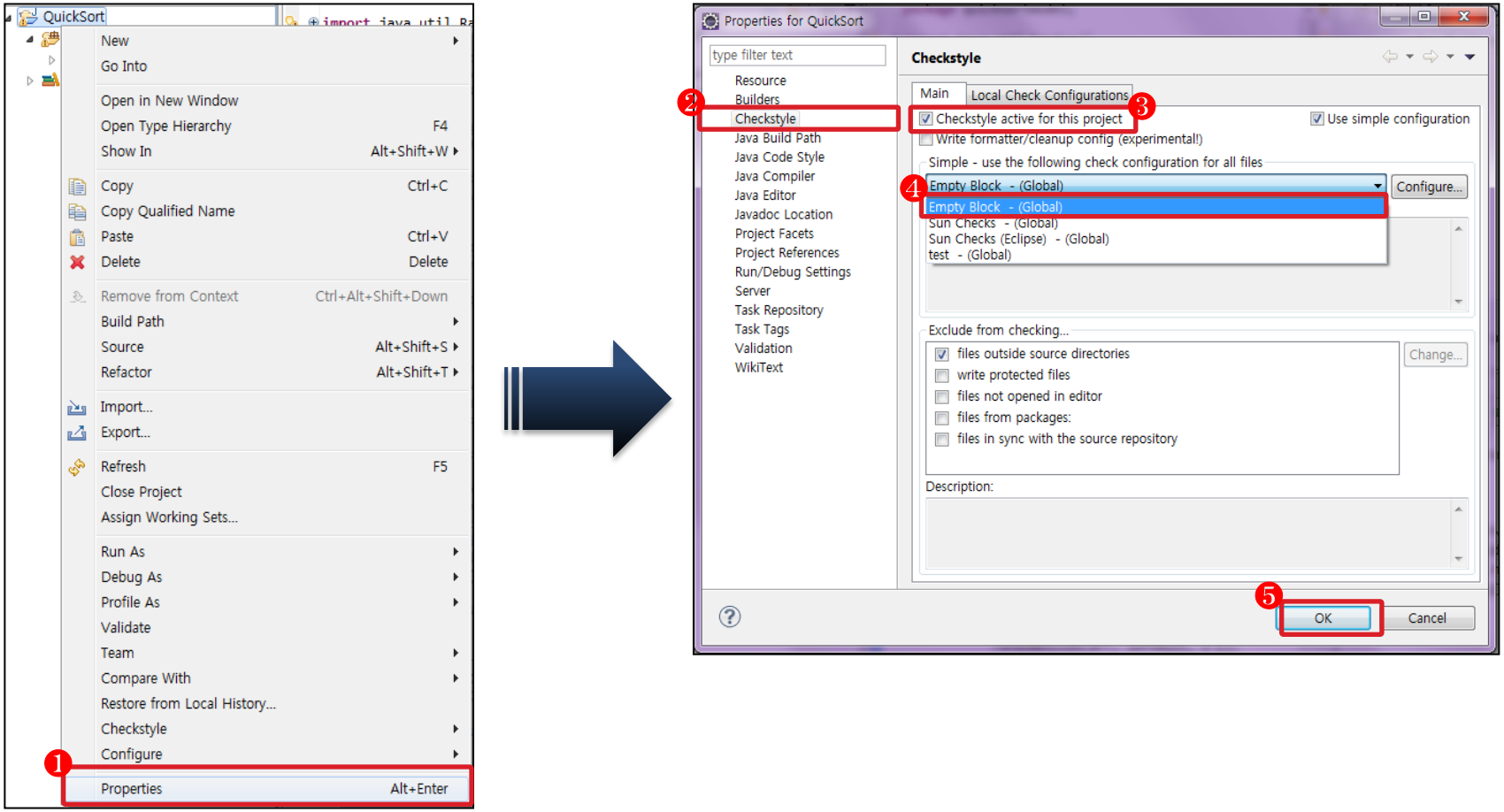
- 규칙을 생성
  - 규칙의 이름은 Empty Block으로 빈 block을 체크하는 규칙

4. 활용 예제

checkstyle

4.3 checkstyle 적용하기 (2/4)

- 새롭게 생성한 Checkstyle을 적용
  - QuickSort 선택 후 마우스 우 클릭 → Properties → Checkstyle 선택 → Checkstyle active for this project에 체크 → Empty Block – (Global)을 선택 → OK



4. 활용 예제

checkstyle

4.3 checkstyle 적용하기 (3/4)

- 작성된 코드를 Checkstyle을 이용하여 스타일 체크
  - 코드 상에 마우스 우 클릭 → Checkstyle → Check Code with Checkstyle

```
package quicksortmodel;

import java.util.Random;

public class Partition {
    static int cnt=1, ind=1;
    public int getRandomNum(int num) {
        int result = 0;
        Random rangen = new Random();
        result = rangen.nextInt() % num;

        if(result < 0) result =result * -1;
        return result;
    }

    public int randomDivide(int[] arr, int leftmost, int ri
        Partition middle = new Partition();
        int key;
        if((rightmost-leftmost) <= 0)
            return 0;
        else
        {
            key = middle.getRandomNum(rightmost-leftmost+1)
            int pivot = swap(arr, key, rightmost);
            if(ind%10==0) {
                ind++;
            }else{
                ind++;
            }
            int i = leftmost-1;
            int j = rightmost;
            while(true) {
                while(arr[++i] < pivot);
                while(j>leftmost && arr[--j] >= pivot);
                if(i >= j)
                    break;
                else
                    swap(arr, i, j);
            }
            swap(arr, i, rightmost);
            randomDivide(arr, leftmost, i-1);
            randomDivide(arr, i+1, rightmost);
        }
        return key;
    }

    public int swap(int[] arr, int left, int right) {
        int temp;
```

Problems | @ Javadoc | Declaration

errors, 2 warnings, 0 others

Undo Typing	Ctrl+Z
Revert File	
Save	Ctrl+S
Open Declaration	F3
Open Type Hierarchy	F4
Open Call Hierarchy	Ctrl+Alt+H
Show in Breadcrumb	Alt+Shift+B
Quick Outline	Ctrl+O
Quick Type Hierarchy	Ctrl+T
Open With	▸
Show In	Alt+Shift+W ▸
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Quick Fix	Ctrl+1
Source	Alt+Shift+S ▸
Refactor	Alt+Shift+T ▸
Local History	▸
References	▸
Declarations	▸
Add to Snippets...	
Run As	▸
Debug As	▸
Profile As	▸
Validate	
Apply Checkstyle fixes	Ctrl+Alt+C
Team	▸
Compare With	▸
Replace With	▸
Checkstyle	▸
Preferences...	
Remove from Context	Ctrl+Alt+Shift+Down

- Check Code with Checkstyle
- Clear Checkstyle violations

4. 활용 예제

checkstyle

4.3 checkstyle 적용하기 (4/4)

- 아래 그림과 같이 code 스타일을 체크해주는 것을 확인 가능
- Else문의 block이 비어 있음을 알려줌

```java
package quicksortmodel;

import java.util.Random;

public class Partition {
    static int cnt=1, ind=1;
    public int getRandomNum(int num) {
        int result = 0;
        Random rangen = new Random();
        result = rangen.nextInt() % num;

        if(result < 0) result =result * -1;
        else {
        }
        return result;
    }
```

```java
package quicksortmodel;

import java.util.Random;

public class Partition {
    static int cnt=1, ind=1;
    public int getRandomNum(int num) {
        int result = 0;
        Random rangen = new Random();
        result = rangen.nextInt() % num;

        if(result < 0) result =result * -1;
        else {
        }
        return result;
    }
```

Must have at least one statement.