Utilizing the Inspection Tool

Summary

This guide will describe the eGovFrame's Code Inspection tool called PMD and its basic usage.

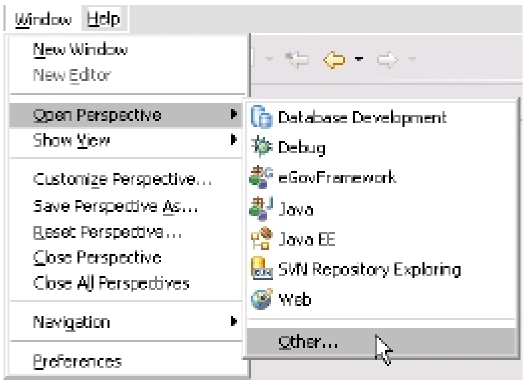
Basic manual

You can run Code Inspection from the IDE's PMD Perspective in order to batch-inspect code conveniently.

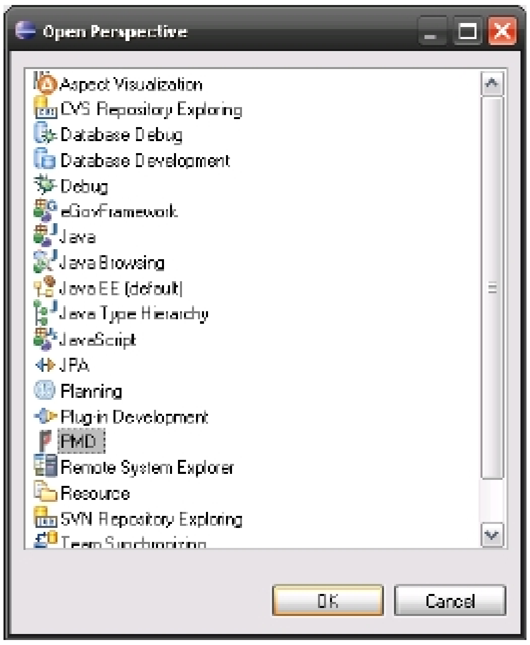
Switching to PMD Perpective

Following is how to switch to the PMD Perspective.

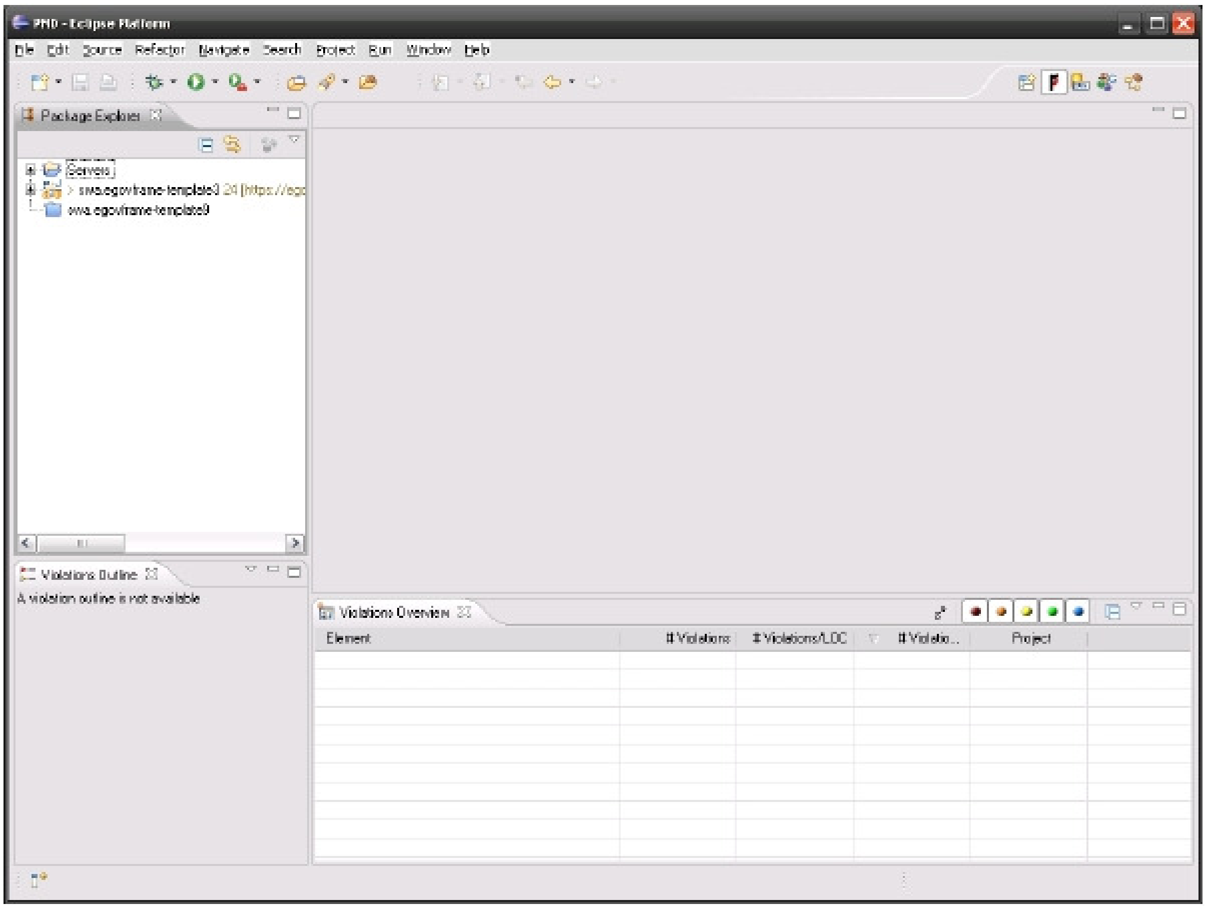
1. Choose 'Window' > 'Open Perspective' > 'Other...' from the menu



2. Choose PMD from the Open Perspective window



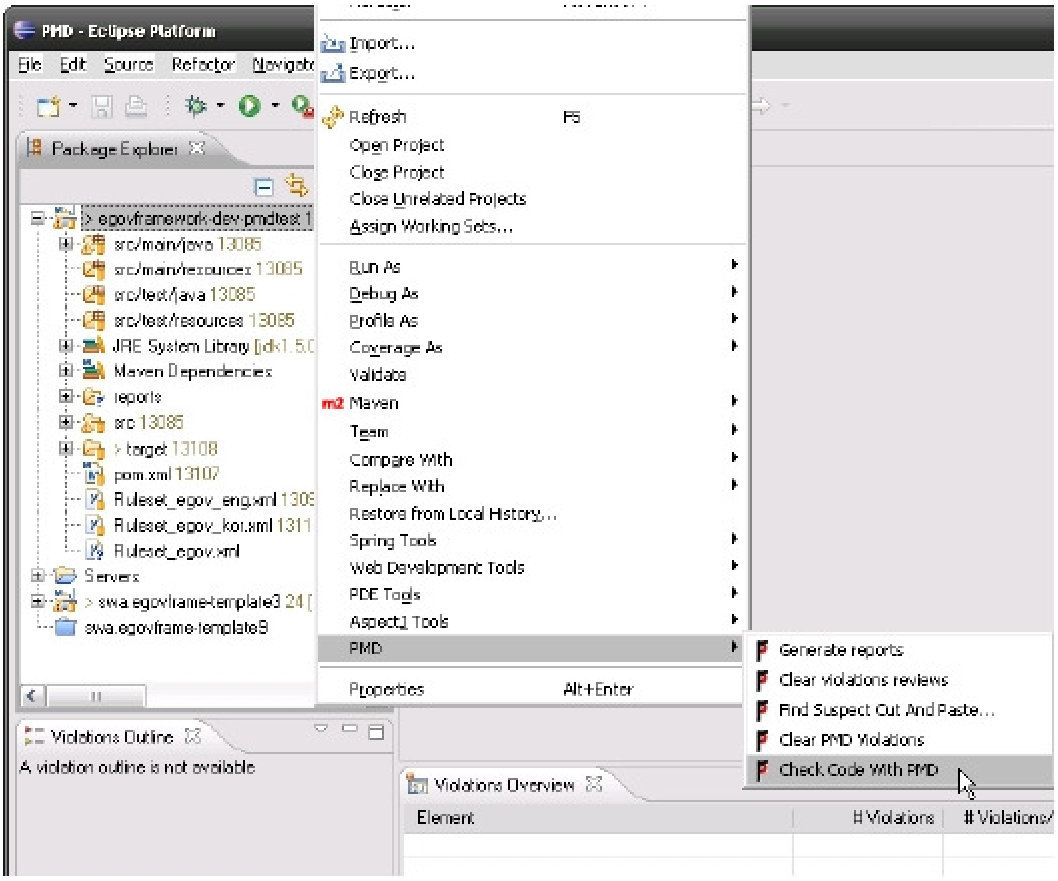
3. After switching to the PMD Perspective, you will see Package Explorer, Violation Outline, Violation Overview and other views within the IDE.



Running code inspection

Choose a project to inspect.

- In the Package Explorer, right-click on the project, and choose PMD > Check Code with PMD in the context menu.

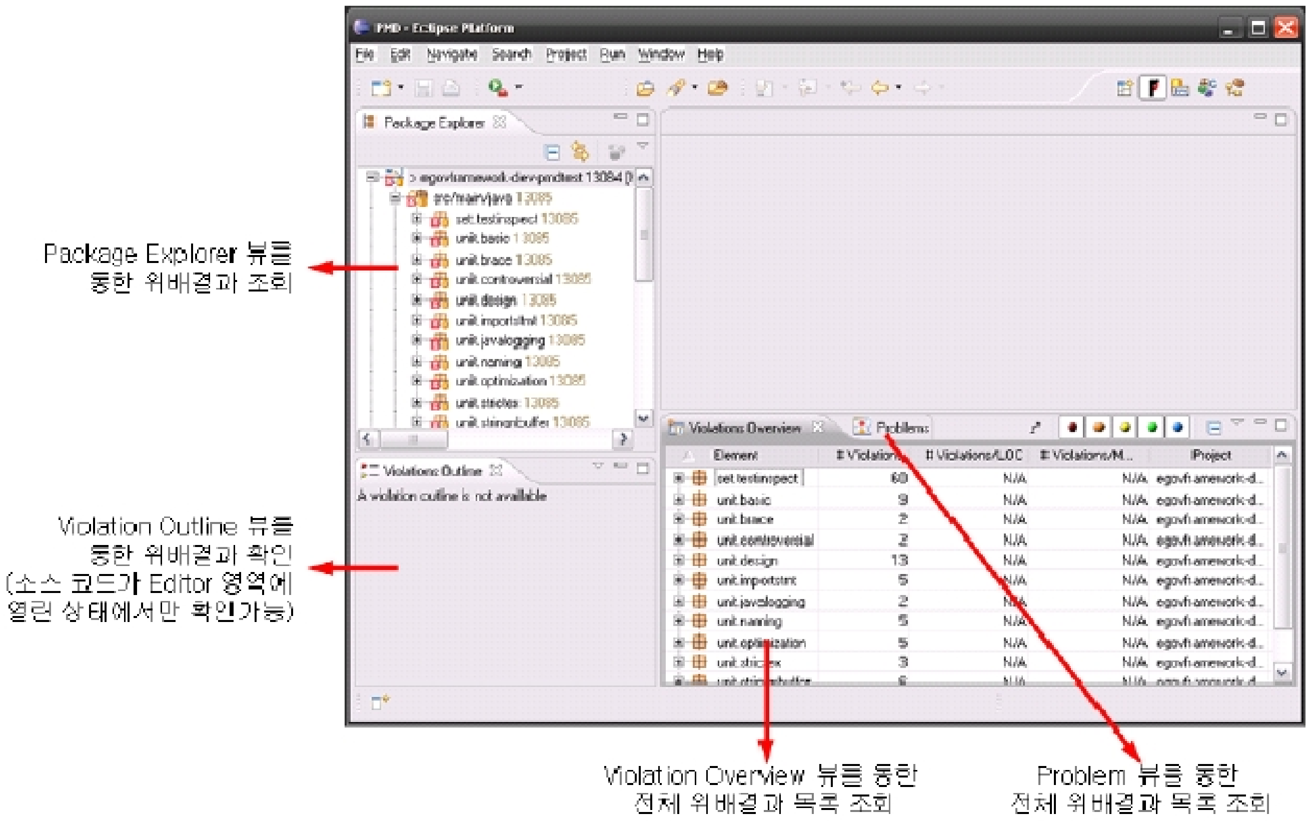


- The entire project’s source code will be inspected. If you want to run inspections only on individual source files, apply the same step as above on the files from the Package Explorer.
Only Java source codes will run through the inspection, and the following will be excluded.

Include files
JAR files and other binaries

Inspection results

After the inspection, you can check the results and see the violating codes.
PMD Perspective provides multiple views for you to see the inspection results.


PMD - Eclipse Platform
Package Explorer
Violations Outline
A violation outline is not available
Violations Overview
Problems
Package Explorer 뷰를 통한 위배결과 조회
Violation Outline 뷰를 통한 위배결과 확인 (소스 코드가 Editor 영역에 열린 상태에서만 확인가능)
Violation Overview 뷰를 통한 전체 위배결과 목록 조회
Problem 뷰를 통한 전체 위배결과 목록 조회

Element	# Violations	# Violations/LOC	# Violations/M...	Project
set.testinspect	60	N/A	N/A	egovframework-d...
unit.basic	9	N/A	N/A	egovframework-d...
unit.brace	2	N/A	N/A	egovframework-d...
unit.controversial	2	N/A	N/A	egovframework-d...
unit.design	13	N/A	N/A	egovframework-d...
unit.importstmt	5	N/A	N/A	egovframework-d...
unit.javalogging	2	N/A	N/A	egovframework-d...
unit.naming	5	N/A	N/A	egovframework-d...
unit.optimization	5	N/A	N/A	egovframework-d...
unit.strictex	3	N/A	N/A	egovframework-d...
unit.stringbuffer	6	N/A	N/A	egovframework-d...

Package Explorer

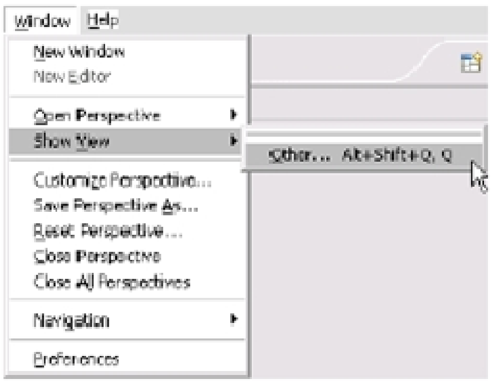
If there are violating code areas within the project, similar to compile errors, the project icon and the violating source icons in the Package Explorer will display red X boxes as below. ( )

These icons will persist until the violations have been corrected.

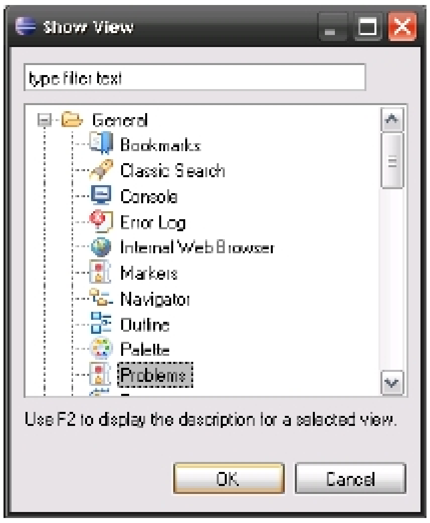
Problem view

Problem view will list the violating lines in the source code. Open the Problem view in the PMD Perspective as follows.

1. Window > Show View > Other... in the Eclipse menu

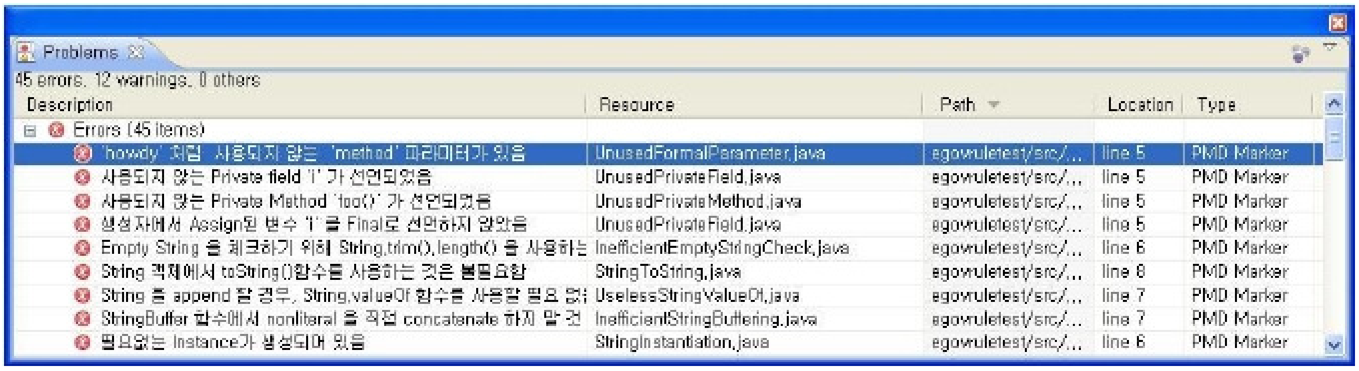


2. In the Show View dialog, choose Problem under the General category.



Use the Problem view to see the violating lines, and access them in the editor to fix the violated source codes.

Category	Description
Description	Detailed description of the violation
Resource	Violating file name
Path	Violating file's path
Location	Violating line number



Problems

45 errors, 12 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (45 items)				
'howdy' 처럼 사용되지 않는 'method' 파라미터가 있음	UnusedFormalParameter.java	egovruletest/src/...	line 5	PMD Marker
사용되지 않는 Private field 'i' 가 선언되었음	UnusedPrivateField.java	egovruletest/src/...	line 5	PMD Marker
사용되지 않는 Private Method 'foo()' 가 선언되었음	UnusedPrivateMethod.java	egovruletest/src/...	line 5	PMD Marker
생성자에서 Assign된 변수 'i' 를 Final로 선언하지 않았음	UnusedPrivateField.java	egovruletest/src/...	line 5	PMD Marker
Empty String 을 체크하기 위해 String.trim().length() 을 사용하는	InefficientEmptyStringCheck.java	egovruletest/src/...	line 6	PMD Marker
String 객체에서 toString()함수를 사용하는 것은 불필요함	StringToString.java	egovruletest/src/...	line 8	PMD Marker
String 을 append 할 경우, String.valueOf 함수를 사용할 필요 없	UselessStringValueOf.java	egovruletest/src/...	line 7	PMD Marker
StringBuffer 함수에서 nonliteral 을 직접 concatenate 하지 말 것	InefficientStringBuffering.java	egovruletest/src/...	line 7	PMD Marker
필요없는 Instance가 생성되어 있음	StringInstantiation.java	egovruletest/src/...	line 6	PMD Marker

Double-clicking an entry will jump to the violating line in the Editor.

Violations Overview view

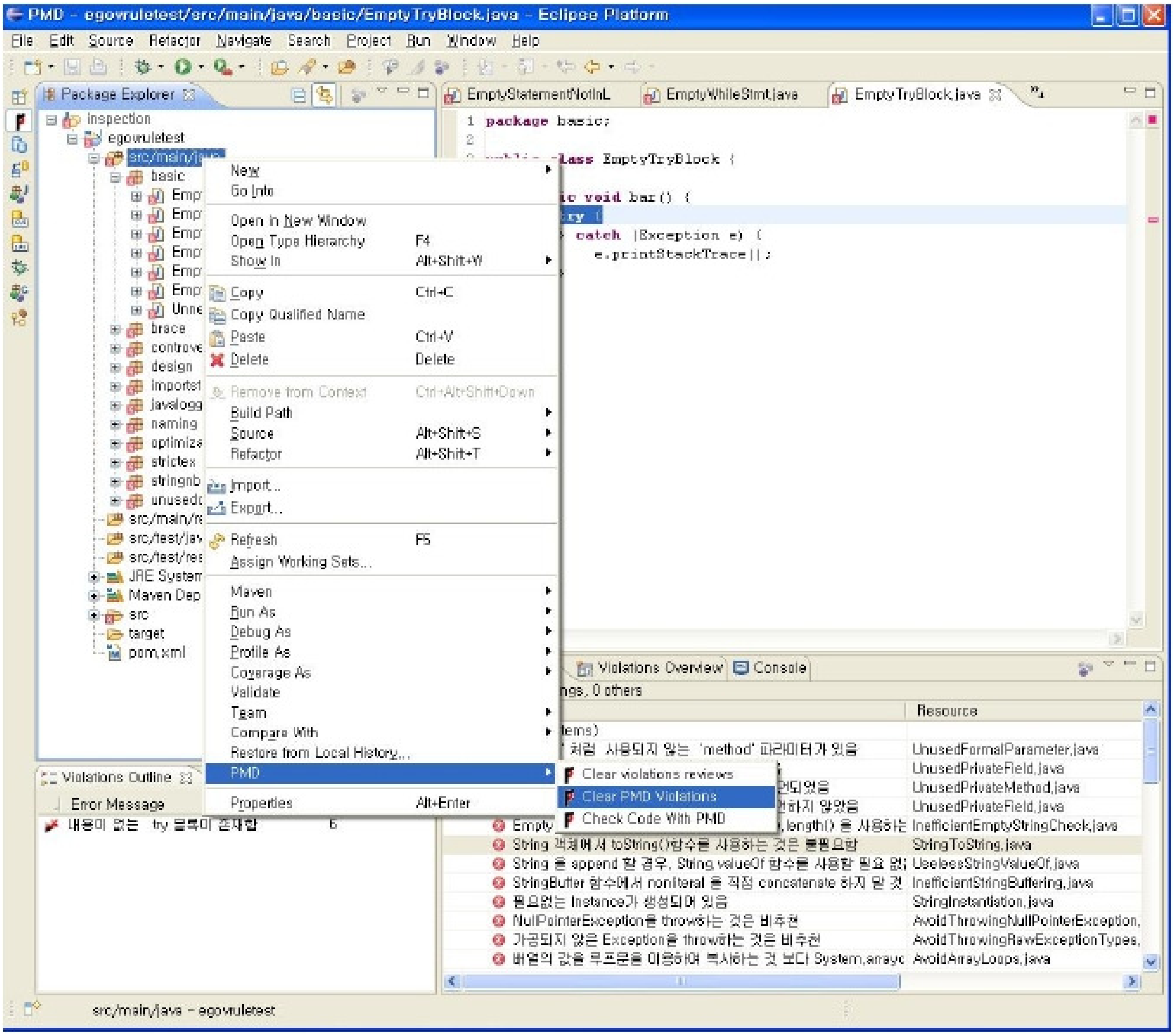
Using Violations Overview will be described in the Reporting inspection results section later on.

Resetting inspection results

Inspection results will persist on the source until you edit them.
You may need to reset the inspection results because it may be confusing to distinguish between inspection results and code compile error results. Or, if you want to re-run inspection, you need to reset inspection results.

In order to reset inspection results, choose a project and reset it as below.

- In the Package Explorer, right-click on the project, and choose PMD > Clear PMD Violations from the context menu.



```
package basic;

public class EmptyTryBlock {

    ic void bar() {
        try {
        } catch (Exception e) {
            e.printStackTrace();
```

내용이 없는 try 블록이 존재함 6

ngs, 0 others

	Resource
tems)	
' 처럼 사용되지 않는 'method' 파라미터가 있음	UnusedFormalParameter.java
	UnusedPrivateField.java
언되었음	UnusedPrivateMethod.java
언하지 않았음	UnusedPrivateField.java
Empty .length() 을 사용하는	InefficientEmptyStringCheck.java
String 객체에서 toString()함수를 사용하는 것은 불필요함	StringToString.java
String 을 append 할 경우, String.valueOf 함수를 사용할 필요 없;	UselessStringValueOf.java
StringBuffer 함수에서 nonliteral 을 직접 concatenate 하지 말 것	InefficientStringBuffering.java
필요없는 Instance가 생성되어 있음	StringInstantiation.java
NullPointerException을 throw하는 것은 비추천	AvoidThrowingNullPointerException.
가공되지 않은 Exception을 throw하는 것은 비추천	AvoidThrowingRawExceptionTypes.
배열의 값을 루프문을 이용하여 복사하는 것 보다 System.arrayc	AvoidArrayLoops.java

This will reset the inspection results, and will hide the Problem view, Violations Overview view, and Package Explorer view in the IDE.

eGovFrame standard inspection rules

eGovFrame defines total of 39 rules for Code Inspection in terms of logical/phrase/reference errors. eGovFrame rule set needs to be installed according to the standard installation guideline; each rule will be explained in the following sections, as well as examples of violating codes, remedies, and compliant codes.

Rule#01. EmptyCatchBlock

- Description: using empty catch statements
- Problematic code:

```
public void doSomething() {
    try
    {
        FileInputStream fis = new FileInputStream("/tmp/bugger");
    }
    catch (IOException ioe)
    {
        //
    }
}
```

- Recommendation: Always use handling code in catch blocks

Rule#02. EmptyIfStmt

- Description: empty if conditions

- Problematic code:

```
public class Foo {
      void bar(int x) {
             if (x == 1) {
                   //
             }
       }
}
```

- Recommendation: avoid empty if conditions

Rule#03. EmptyWhileStmt

- Description: empty while conditions
- Problematic code:

```
public class Foo {
      void bar(int a, int b) {
            while (a == b) {
                  //
            }
      }
}
```

- Recommendation: avoid empty while conditions

Rule#04. EmptyTryBlock

- Description: empty try conditions
- Problematic code:

```
public class Foo {
      public void bar() {
            try
            {
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }
}
```

- Recommendation: avoid empty try conditions

Rule#05. EmptyFinallyBlock

- Description: empty finally statements
- Problematic code:

```
public class Foo {
      public void bar() {
            try
            {
                  int x=3;
            }
            finally
            {
                  //
            }
      }
}
```

- Recommendation: there must not be any empty finally statments

ule#06. UnnecessaryConversionTemporary

- Unnecessary String conversions into temporary variables
- Problematic code:

```
public String convert(int x) {
```

```
public String convert(int x) {
        String foo = new Integer(x).toString();
        return foo;
}
```

- Recommendation: when converting data type into String, avoid creating temporary String variables

Rule#07. EmptyStatementNotInLoop

- Description: unnecessary semi-colons
- Problematic code:

```
public class MyClass {
        public void doit() {
                ;
                System.out.println("look at the extra semicolon");;
        }
}
```

- Recommendation: avoid using empty statements with semi-colons only.

Rule#08. WhileLoopsMustUseBraces

- Description: bracket-less while statements
- Problematic code:

```
public void doSomething() {
        while (true)
                x++;
}
```

- Recommendation: avoid using while statements without brackets

Rule#09. AssignmentInOperand

- Description: assignments within conditions/operands
- Problematic code:

```
public class Foo {
        public void bar() {
                int x = 2;
                if ( (x = getX() ) == 3) {
                        System.out.println("3!");
                }
        }
        private int getX() {
                return 3;
        }
}
```

- Recommendation: avoid using assignments in conditions/operands, because it decreases code readability and increases complixity

Rule#10. UnnecessaryParentheses

- Description: unnecessary parentheses
- Problematic code:

```
public class Foo {
        boolean bar() {
                return (true);
        }
}
```

- Recommendation: avoid unnecessary parentheses, because it decreases code readability

Rule#11. SimplifyBooleanExpressions

- Description: unnecessary boolean comparisons
- Problematic code:

```
public class Bar {
         private boolean bar = (isFoo() == true);
```

```
    private boolean bar = (isFoo() == true);

    public isFoo() {
        return false;
    }
}
```

- Recommendation: avoid using unnecessary boolean comparisons

---

Rule#12. SwitchStmtsShouldHaveDefault

- Description: default-less switch statements
- Problematic code:

```
public class Foo {
    public void bar() {
        int x = 2;
        switch (x) {
            case 2;
            int j = 8;
        }
    }
}
```

- Recommendation: always define switch label in switch statements

---

Rule#13. AvoidReassigningParameters

- Description: reassigning parameters
- Problematic code:

```
public class Foo {
    private void foo(String bar) {
        bar = "something else";
    }
}
```

- Recommendation: parameters shall not be modified in value.

Rule#14. FinalFieldCouldBeStatic

- Description: using static instead of final
- Problematic code:

```
public class Foo {
    public final int BAR = 42;
}
```

- Recommendation: switching to static instead of final can reduce overhead

---

Rule#15. EqualsNull

- Description: null comparison using equals()
- Problematic code:

```
class Bar {
     void foo() {
         String x = "foo";
         if (x.equals(null)) {
             doSomething();
         }
     }
}
```

- Recommendation: equals method shall not be used for comparing null values

---

Rule#16. SimpleDateFormatNeedsLocale

- Description: locale-less SimpleDateFormat
- Problematic code:

```
public class Foo {
      private SimpleDateFormat sdf = new SimpleDateFormat("pattern");
}
```

- Recommendation: always assign locale when using SimpleDateFormat

---

Rule#17. ImmutableField

- Description: use final for constructor variables
- Problematic code:

```
public class Foo {
       private int x;
       public Foo() {
             x = 7;
       }
       public void foo() {
            int a = x + 2;
       }
}
```

- Recommendation: use final variable type for constructor variables

---

Rule#18. AssignmentToNonFinalStatic

- Description: erroneous usage of static type
- Problematic code:

```
public class StaticField {
      static int x;
      public FinalFields(int y) {
            x = y;
      }
}
```

- Recommendation: avoid using static fields in unsafe manner

Rule#19. AvoidSynchronizedAtMethodLevel

- Description: overusing synchronization at method level
- Problematic code:

```
public class Foo {
      synchronized void foo() {
      }
}
```

- Recommendation: use synchronization only for blocks rather than methods

---

Rule#20. AbstractClassWithoutAbstractMethod

- Description: defining abstract class without abstract methods
- Problematic code:

```
public abstract class Foo {
      void int method1() {
            // ...
      }
      void int method2() {
            // ...
      }
}
```

- Recommendation: always define abstract methods in abstract classes

---

Rule#21. UncommentedEmptyMethod

- Description: comment-less empty methods
- Problematic code:

```

public void doSomething() {
}
```

```
}
```

- Recommendation: always indicate empty methods with comments

---

Rule#22. AvoidConstantsInterface

- Description: using constants in interfaces
- Problematic code:

```
public interface ConstantsInterface {
        public static final int CONSTANT1 = 0;
        public static final String CONSTANT2 = "1";
}
```

- Recommendation: use interfaces only for defining class behaviors

---

Rule#23. DuplicateImports

- Description: duplicate import statements
- Problematic code:

```
import java.lang.String;
import java.lang.*;

public class Foo {
}
```

- Recommendation: avoid using duplicate import statements

Rule#24. ImportFromSamePackage

- Description: importing from same package
- Problematic code:

```
package foo;
import foo.Buz;
import foo.*;

public class Bar {
}
```

- Recommendation: avoid importing from same package

Rule#25. SystemPrintln

- Description: using System.out.print
- Problematic code:

```
class Foo{
        public void testA () {
                System.out.println("Entering test");
        }
}
```

- Recommendation: avoid System.out.print, instead use customized message printing

---

Rule#26. VariableNamingConventions

- Description: under-bars in variable names
- Problematic code:

```
public class Foo {
  public static final int MY_NUM = 0;
  public String myTest = "";
  DataModule dmTest = new DataModule();
}
```

- Recommendation: do not include underlines for variables that are not final

---

Rule#27. MisleadingVariableName

- Description: using erroneous prefixes for variables
- Problematic code:

```
public class Foo {
    public void bar(String m_baz) {
        int m_boz = 42;
    }
}
```

- Recommendation: avoid using m_ prefixes for non-fields

Rule#28. AvoidArrayLoops

- Description: loops for copying arrays
- Problematic code:

```
public class Test {
    public void bar() {
        int[] a = new int[10];
        int[] b = new int[10];
        for (int i=0;i<10;i++) {
            b[i]=a[i];
        }
    }
}
```

- Recommendation: avoid loops and instead use System.arraycopy() for array copies

---

Rule#29. UnnecessaryWrapperObjectCreation

- Description: unnecessary WrapperObjects
- Problematic code:

```
public int convert(String s) {
    int i, i2;
    i = Integer.valueOf(s).intValue();
    i2 = Integer.valueOf(i).intValue();
```

- Recommendation: use custom parse-related methods instead

Rule#30. AvoidThrowingRawExceptionTypes

- raw exception types
- Problematic code:

```
public class Foo {
    public void bar() throws Exception {
        throw new Exception();
    }
}
```

- Recommendation: use more specific exceptions

Rule#31. AvoidThrowingNullPointerException

- Description: using null pointer exceptions
- Problematic code:

```
public class Foo {
    void bar() {
        throw new NullPointerException();
    }
}
```

- Recommendation: avoid using NullPointerException

Rule#32. StringInstantiation

- Description: using unnecessary String instances
- Problematic code:

```
public class Foo {
```

```
public class Foo {
        private String bar = new String("bar");
}
```

- Recommendation: use more simple variables

Rule#33. StringToString

- Description: using toString() on a String instance
- Problematic code:

```
public class Foo {
        private String baz() {
                String bar = "howdy";
                return bar.toString();
        }
}
```

- Recommendation: avoid calling toString() on String instances

---

Rule#34. InefficientStringBuffering

- Description: combining strings inside StringBuffer() constructor
- Problematic code:

```
StringBuffer sb = new StringBuffer( "tmp =" +
                                            System.getProperty("java.io.tmpdir") );
```

- Recommendation: use append method instead

Rule#35. InefficientEmptyStringCheck

- Description: using null check or zero size check on Strings
- Problematic code:

```
public class Foo {
        void bar(String string) {
                if (string != null && string.trim().size() > 0) {
        doSomething();
                }
        }
}
```

- Recommendation: use custom logic instead to differentiate whitespace and non-whitespace

---

Rule#36. UselessStringValueOf

- Description: using String.valueOf() when appending
- Problematic code:

```
public String convert(int i) {
        String s;
        s = "a" + String.valueOf(i);
        return s;
}
```

- Recommendation: avoid calling String.valueOf() when appending

---

Rule#37. UnusedPrivateField

- Description: unused private fields
- Problematic code:

```
public class Something {
        private static int FOO = 2; // Unused
        private int i = 5; // Unused
        private int j = 6;

      public int addOne() {
                return j++;
        }
        }
}
```

- Recommendation: avoid unused private fields

Rule#38. UnusedPrivateMethod

- Description: unused private methods
- Problematic code:

```
public class Something {
        private void foo() {} // unused
}
```

- Recommendation: avoid unused private methods

Rule#39. UnusedFormalParameter

- Description: unused method parameters
- Problematic code:

```
public class Foo {
        private void bar(String howdy) {
        // howdy is not used
        }
}
```

- Recommendation: avoid unused method parameters

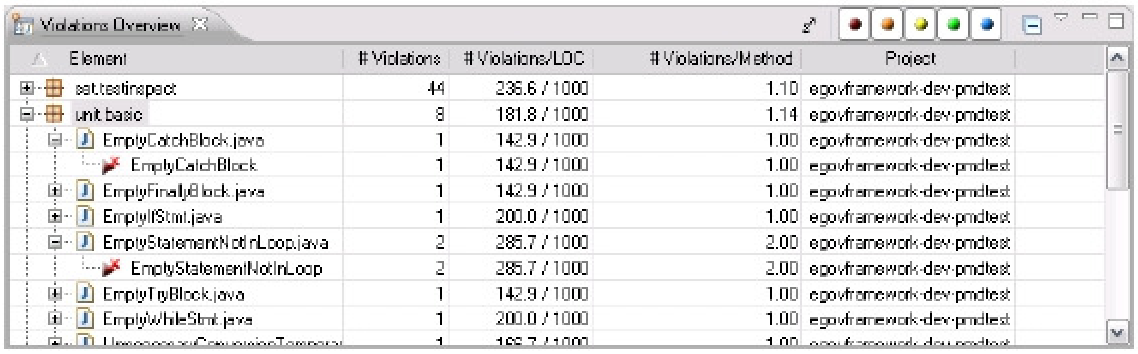
Reporting inspection results

You can aggregate and report inspection results as below.

- Personal IDE: Check results in Violations Overview of Eclipse IDE or use file-based reporting in CSV[1], HTML, TXT, or XML.
- Server IDE: Using CI server, Hudson[2], Hudson PMD Hudson's PMD Plugin can be used for inspection reports. See eGovFrame's Hudson Code Inspection Tools guide.

Reports in Violations Overview

Inspection results can be checked immediately in the IDE as below.



Element	# Violations	# Violations/LOC	# Violations/Method	Project
sat.testinspect	44	236.6 / 1000	1.10	egovframework-dev-pmdtest
unit.basic	8	181.8 / 1000	1.14	egovframework-dev-pmdtest
EmptyCatchBlock.java	1	142.9 / 1000	1.00	egovframework-dev-pmdtest
EmptyCatchBlock	1	142.9 / 1000	1.00	egovframework-dev-pmdtest
EmptyFinallyBlock.java	1	142.9 / 1000	1.00	egovframework-dev-pmdtest
EmptyIfStmt.java	1	200.0 / 1000	1.00	egovframework-dev-pmdtest
EmptyStatementNotInLoop.java	2	285.7 / 1000	2.00	egovframework-dev-pmdtest
EmptyStatementNotInLoop	2	285.7 / 1000	2.00	egovframework-dev-pmdtest
EmptyTryBlock.java	1	142.9 / 1000	1.00	egovframework-dev-pmdtest
EmptyWhileStmt.java	1	200.0 / 1000	1.00	egovframework-dev-pmdtest

Violations Overview displays a grid for listing statistic violating codes, and control buttons to control the grid on the top right.

Button functionalities

- Statistics button: refreshes the statistics grid. Should be used after using the priority button, or any violation is fixed.
- ● ● ● ● ● Priority buttons: filters violating items in the grid. Five, different-colored buttons that denote from 1 to 5 (low to high, from left to right). 1s and 2s (red and orange) are must-fixes. Each button are toggle-style buttons.
- Collapse statistics button: collapses all the statistics items in the grid to initial state.
- View menu: selects sorting method for the grid. The options are as follows:
  - Show violations to packages: sorts violating items by packages.
  - Show violations to files: sorts violating items by files.
  - Show packages with files: sort by packages, along with displaying file information.

Grid components

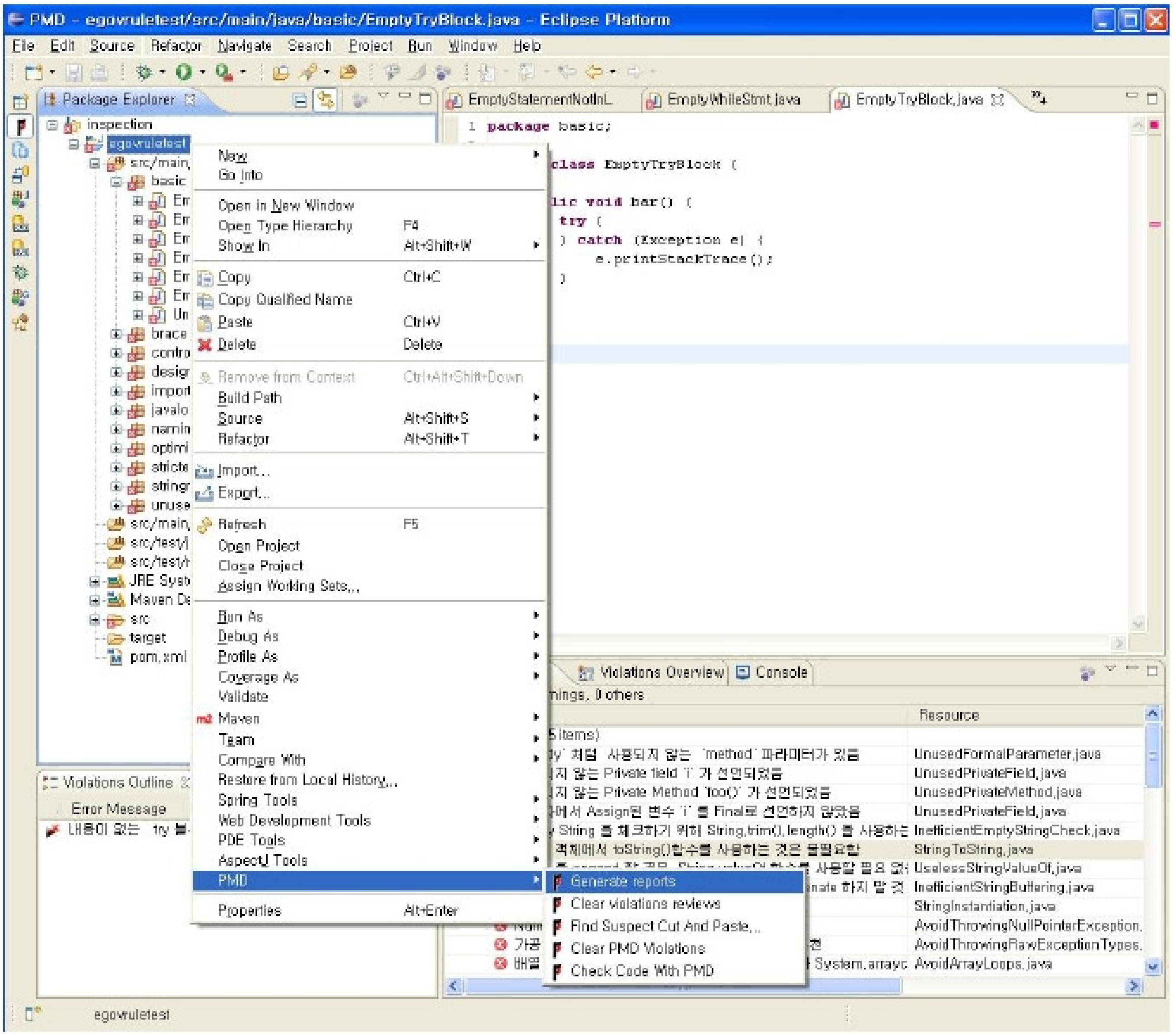
- Element: violating packages, files, and detailed information.
- Violations: violation count for the selected element.
- Violations/LOC: parts-per-thousand (per-mil), as in violations per 1,000 lines of code.[3)]
- Violations/Method: violations per method, or average violations per method for package elements.
- Project: name of project that the selected violating element belongs to.

These statistical information can be utilized by the developer, directly from the IDE, for improving code quality.

Per-file report generation

To generate per-file reports, choose a project, then proceed as follows.

- In Package Explorer, choose Project, then right-click on it.
- In the context menu, choose PMD > Generate reports.

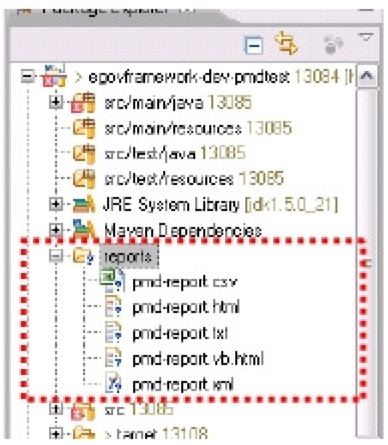


```
package basic;

class EmptyTryBlock {

lic void bar() {
 try {
 } catch (Exception e) {
    e.printStackTrace();
 }
```

```
nings, 0 others
5 items)
y' 처럼 사용되지 않는 'method' 파라미터가 있음          UnusedFormalParameter.java
지 않는 Private field 'i' 가 선언되었음                 UnusedPrivateField.java
지 않는 Private Method 'foo()' 가 선언되었음            UnusedPrivateMethod.java
에서 Assign된 변수 'i' 를 Final로 선언하지 않았음       UnusedPrivateField.java
String 을 체크하기 위해 String.trim().length() 를 사용하는 InefficientEmptyStringCheck.java
객체에서 toString()함수를 사용하는 것은 불필요함        StringToString.java
사용할 필요 없                                          UselessStringValueOf.java
nate 하지 말 것                                         InefficientStringBuffering.java
                                                        StringInstantiation.java
                                                        AvoidThrowingNullPointerException.
                                                        AvoidThrowingRawExceptionTypes.
System.arrayc                                           AvoidArrayLoops.java
```

Confirm generated report

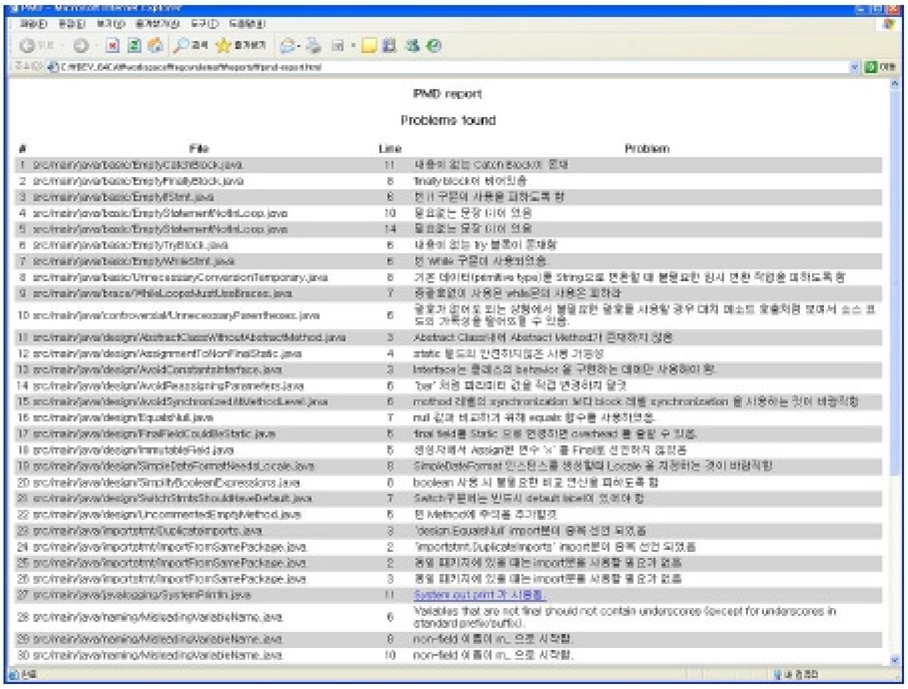
Multiple file formats are supported for per-file inspection report generation.
CSV, HTML, TXT, and XML files can be created under the reports folder inside the project.



```
egovframework-dev-pmdtest 13084
  src/main/java 13085
  src/main/resources 13085
  src/test/java 13085
  src/test/resources 13085
  JRE System Library [jdk1.5.0_21]
  Maven Dependencies
  reports
    pmd-report.csv
    pmd-report.html
    pmd-report.txt
    pmd-report.vb.html
    pmd-report.xml
  src 13085
  > target 13108
```

Reviewing reports

Reports display all the inspection results in a single file view. Following is an example of an HTML report file opened in a browser.



PMD report

Problems found

#	File	Line	Problem
1	src/main/java/basic/EmptyCatchBlock.java	11	내용이 없는 catch block이 존재
2	src/main/java/basic/EmptyFinallyBlock.java	8	finally block이 비어있음
3	src/main/java/basic/EmptyIfStmt.java	6	빈 if 구문의 사용을 피하도록 함
4	src/main/java/basic/EmptyStatementNotInLoop.java	10	필요없는 문장 (;)이 있음
5	src/main/java/basic/EmptyStatementNotInLoop.java	14	필요없는 문장 (;)이 있음
6	src/main/java/basic/EmptyTryBlock.java	6	내용이 없는 try 블록이 존재함
7	src/main/java/basic/EmptyWhileStmt.java	6	빈 While 구문이 사용되었음.
8	src/main/java/basic/UnnecessaryConversionTemporary.java	8	기존 데이터(primitive type)를 String으로 변환할 때 불필요한 임시 변환 작업을 피하도록 함
9	src/main/java/braces/WhileLoopsMustUseBraces.java	7	중괄호없이 사용된 while문의 사용은 피하라
10	src/main/java/controversial/UnnecessaryParentheses.java	6	괄호가 없어도 되는 상황에서 불필요한 괄호를 사용할 경우 마치 메소드 호출처럼 보여서 소스 코드의 가독성을 떨어뜨릴 수 있음.
11	src/main/java/design/AbstractClassWithoutAbstractMethod.java	3	Abstract Class내에 Abstract Method가 존재하지 않음
12	src/main/java/design/AssignmentToNonFinalStatic.java	4	static 필드의 안전하지않은 사용 가능성
13	src/main/java/design/AvoidConstantsInterface.java	3	Interface는 클래스의 behavior 을 구현하는 데에만 사용해야 함.
14	src/main/java/design/AvoidReassigningParameters.java	6	'bar' 처럼 파라미터 값을 직접 변경하지 말것
15	src/main/java/design/AvoidSynchronizedAtMethodLevel.java	6	method 레벨의 synchronization 보다 block 레벨 synchronization 을 사용하는 것이 바람직함
16	src/main/java/design/EqualsNull.java	7	null 값과 비교하기 위해 equals 함수를 사용하지말 것.
17	src/main/java/design/FinalFieldCouldBeStatic.java	5	final field를 Static 으로 변경하면 overhead 를 줄일 수 있음.
18	src/main/java/design/ImmutableField.java	5	생성자에서 Assign된 변수 'x' 를 Final로 선언하지 않았음
19	src/main/java/design/SimpleDateFormatNeedsLocale.java	8	SimpleDateFormat 인스턴스를 생성할때 Locale 을 지정하는 것이 바람직함
20	src/main/java/design/SimplifyBooleanExpressions.java	6	boolean 사용 시 불필요한 비교 연산을 피하도록 함
21	src/main/java/design/SwitchStmtsShouldHaveDefault.java	7	Switch구문에는 반드시 default label이 있어야 함
22	src/main/java/design/UncommentedEmptyMethod.java	5	빈 Method에 주석을 추가할것
23	src/main/java/importstmt/DuplicateImports.java	3	'design.EqualsNull' import문이 중복 선언 되었음
24	src/main/java/importstmt/ImportFromSamePackage.java	2	'importstmt.DuplicateImports' import문이 중복 선언 되었음
25	src/main/java/importstmt/ImportFromSamePackage.java	2	동일 패키지에 있을 때는 import문을 사용할 필요가 없음
26	src/main/java/importstmt/ImportFromSamePackage.java	3	동일 패키지에 있을 때는 import문을 사용할 필요가 없음
27	src/main/java/javalogging/SystemPrintln.java	11	System.out.print 가 사용됨.
28	src/main/java/naming/MisleadingVariableName.java	6	Variables that are not final should not contain underscores (except for underscores in standard prefix/suffix).
29	src/main/java/naming/MisleadingVariableName.java	8	non-field 이름이 m_ 으로 시작함.
30	src/main/java/naming/MisleadingVariableName.java	10	non-field 이름이 m_ 으로 시작함.

Fixing broken Korean fonts in reports

PMD's report files use UTF-8 encoding to generate Korean text, so if you see broken Korean fonts follow the guideline below.

- Web browser : opening HTML reports, and your encoding is set to Korean:
  - Switch to UTF-8 (Unicode).
  - Microsoft Internet Explorer:
    - From the menu bar, choose View > Encoding > Unicode (UTF-8)
  - Mozilla Firefox:
    - From the menu bar, choose View > Character encoding > Unicode (UTF-8)
  - Google Chrome:
    - Click the Customize and Control button to the right of the URL field, then choose Encoding > Unicode (UTF-8)
- Microsoft Excel: does not support CSV files in UTF-8
  - Open such CSV files with text editors such as Notepad, store in a different encoding other than UTF-8, then open with Excel.
  - Use Notepad as below.
    1. Open the CSV file with Notepad, then choose File > Save as ... from the menu.
    2. In the dialog, choose All files in the file type drop-down menu.
    3. Choose ANSI or Unicode encoding, then save the file.
    4. Open the file with Excel.
- Hudson PMD Plugin tool has a bug of Hudson PMD Plugin used in reporting PMD using the remote CI server, Hudson and cannot recognize UTF-8-based Korean (or other Asian) rule set files.
  - Until the bug fix is patched, you need to use english rule set file in order to use the PMD plug-in.

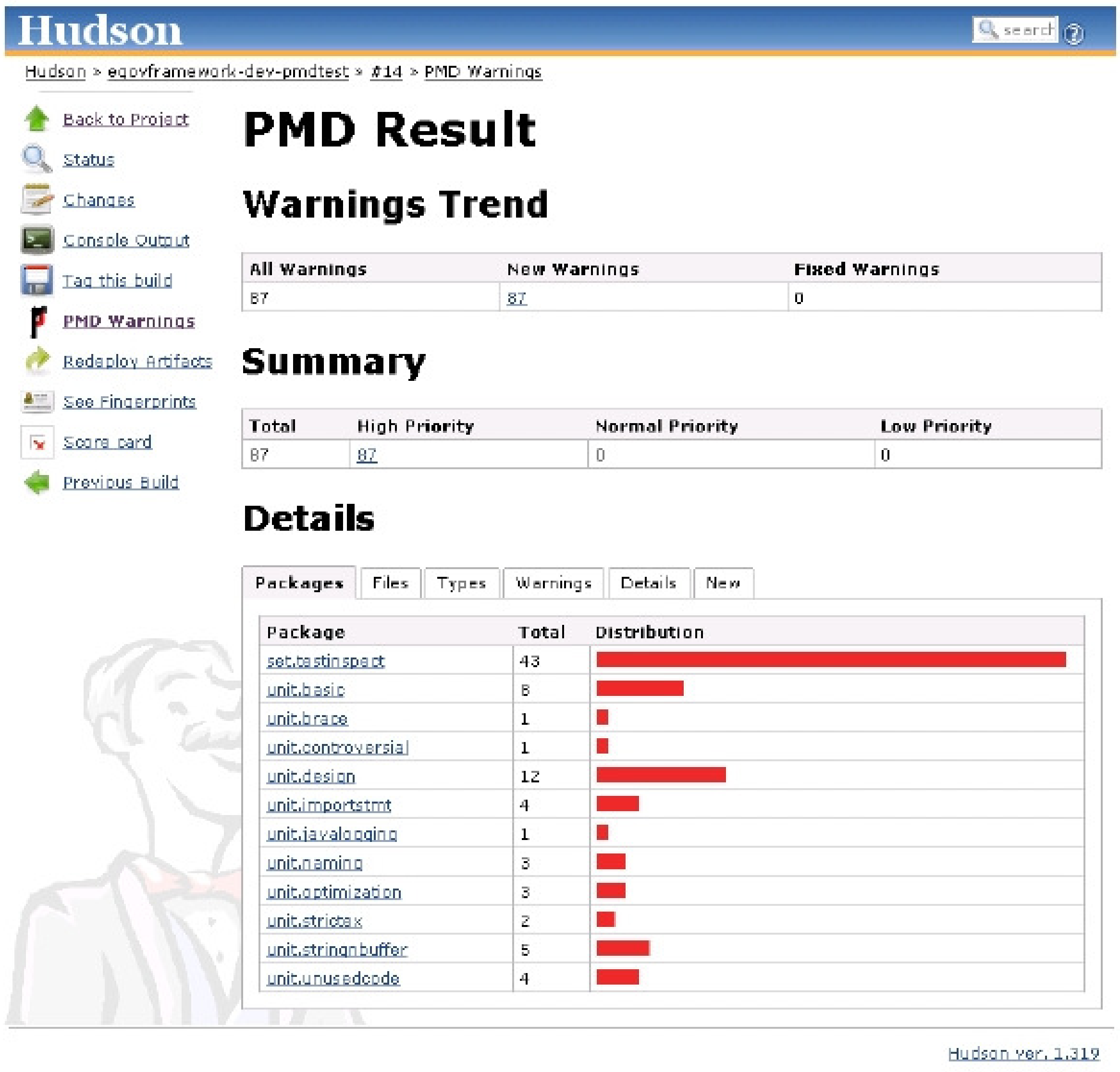
Utilizing inspection report statistics

You can use the hudson PMD Plugin to utilize the statistical information in the inspection reports.

To search statistics on violations from Hudson, it is confirmed as following.

- In the Hudson Dashboard, choose a project to review
- Choose PMD Warning from the left menu

Next is the initial statistics screen.



PMD Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
87	87	0

Summary

Total	High Priority	Normal Priority	Low Priority
87	87	0	0

Details

Package	Total	Distribution
set.testinspect	43	
unit.basic	8	
unit.brace	1	
unit.controversial	1	
unit.design	12	
unit.importstmt	4	
unit.javalogging	1	
unit.naming	3	
unit.optimization	3	
unit.strictex	2	
unit.stringbuffer	5	
unit.unusedcode	4	

PMD result view

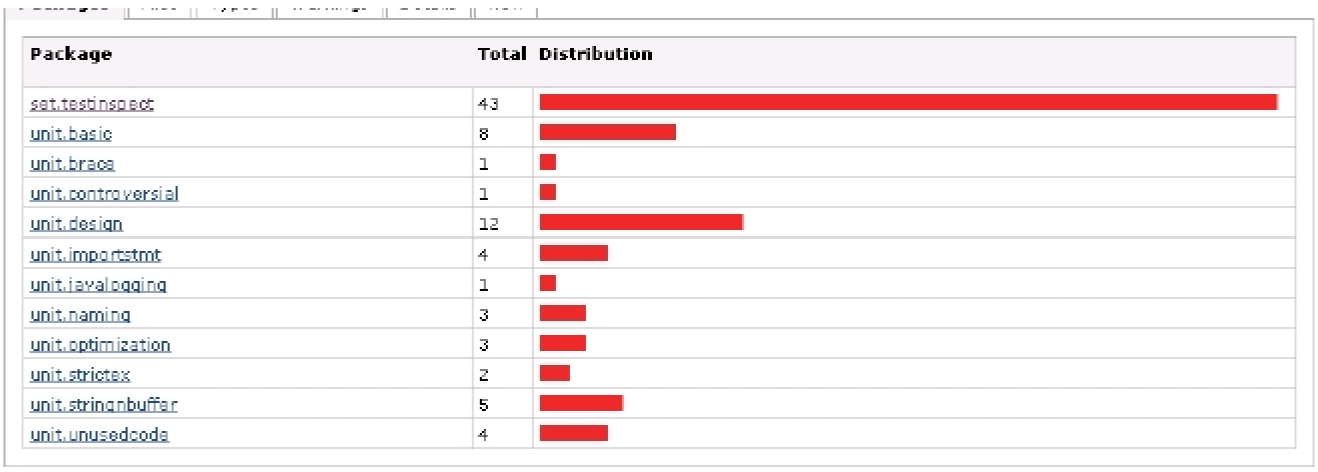
- Warnings Trend: statistical trend of code inspection warnings in the project
  - All Warnings: total number of warnings occurred in the project
  - New Warnings: number or warnings from the most recent build
  - Fixed Warnings: number of fixed warnings in the most recent build
- Summary: displays statistics sorted by priorities (High/Normal/Low)
- Detail: displays detailed statistics with tabbed items as below
  - Packages: statistics per packages
  - Files: statistics per files
  - Types: statistics per violation types
  - Warnings: all warnings
  - Details: all warnings with detailed information
  - New: new or unfixed warnings with detailed information

Detailed statistics view

Hudson PMD Plugin’s statistics can be used to measure individual or team performances as well as metrics for code quality improvement.

Packages

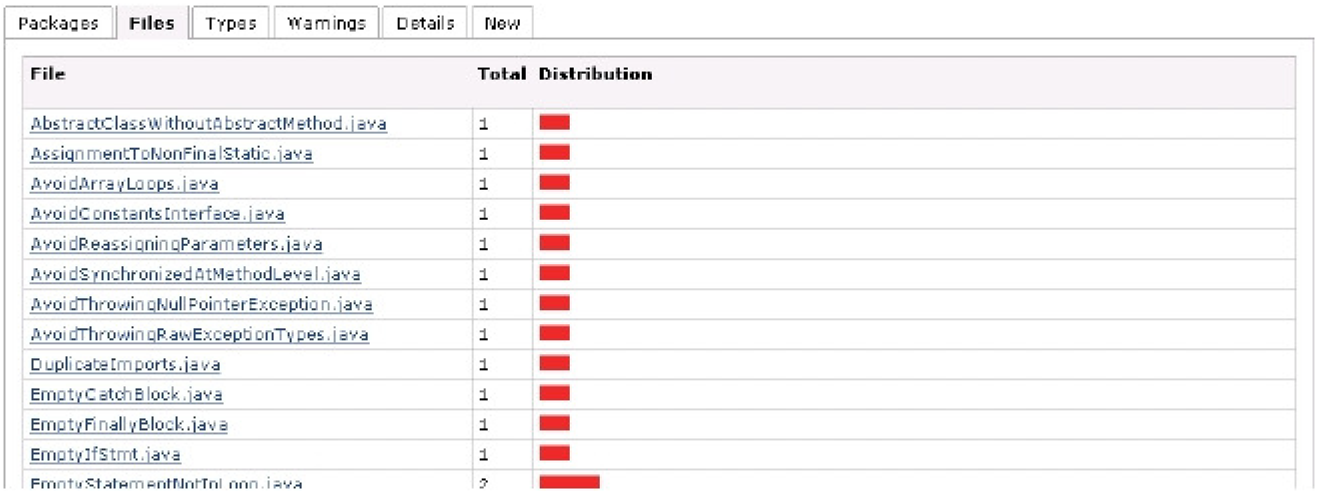
Displays per-package violations as below.



Package	Total	Distribution
sat.testinspect	43	🟥
unit.basic	8	🟥
unit.brace	1	🟥
unit.controversial	1	🟥
unit.design	12	🟥
unit.importstmt	4	🟥
unit.javalogging	1	🟥
unit.naming	3	🟥
unit.optimization	3	🟥
unit.strictex	2	🟥
unit.stringbuffer	5	🟥
unit.unusedcode	4	🟥

Files

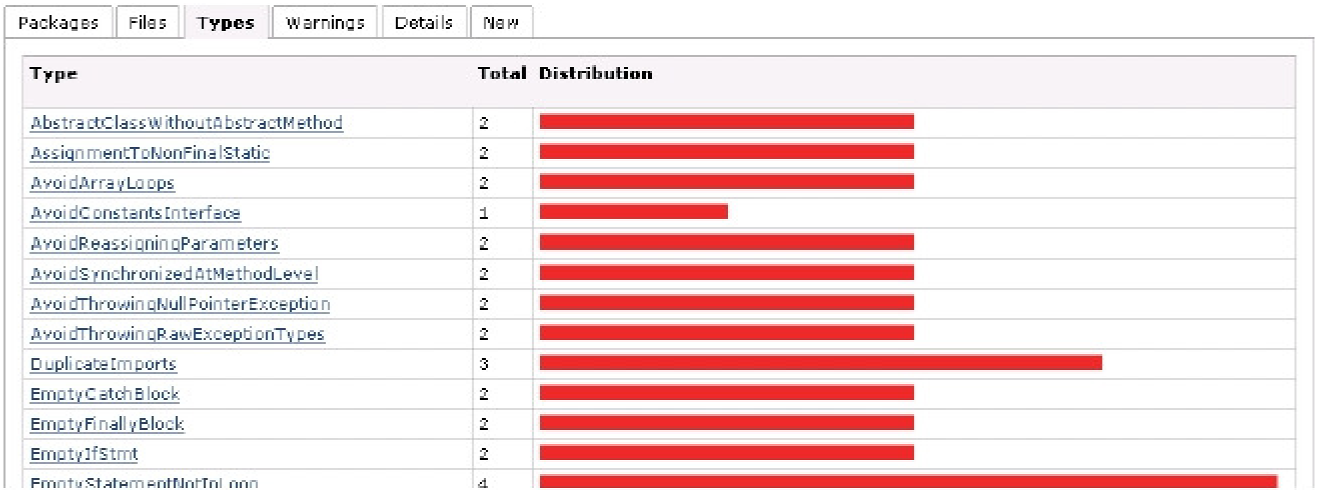
Displays per-file violations as below.



Packages | **Files** | Types | Warnings | Details | New

File	Total	Distribution
AbstractClassWithoutAbstractMethod.java	1	🟥
AssignmentToNonFinalStatic.java	1	🟥
AvoidArrayLoops.java	1	🟥
AvoidConstantsInterface.java	1	🟥
AvoidReassigningParameters.java	1	🟥
AvoidSynchronizedAtMethodLevel.java	1	🟥
AvoidThrowingNullPointerException.java	1	🟥
AvoidThrowingRawExceptionTypes.java	1	🟥
DuplicateImports.java	1	🟥
EmptyCatchBlock.java	1	🟥
EmptyFinallyBlock.java	1	🟥
EmptyIfStmt.java	1	🟥
EmptyStatementNotInLoop.java	2	🟥

Types
Displays per-type violations as below..



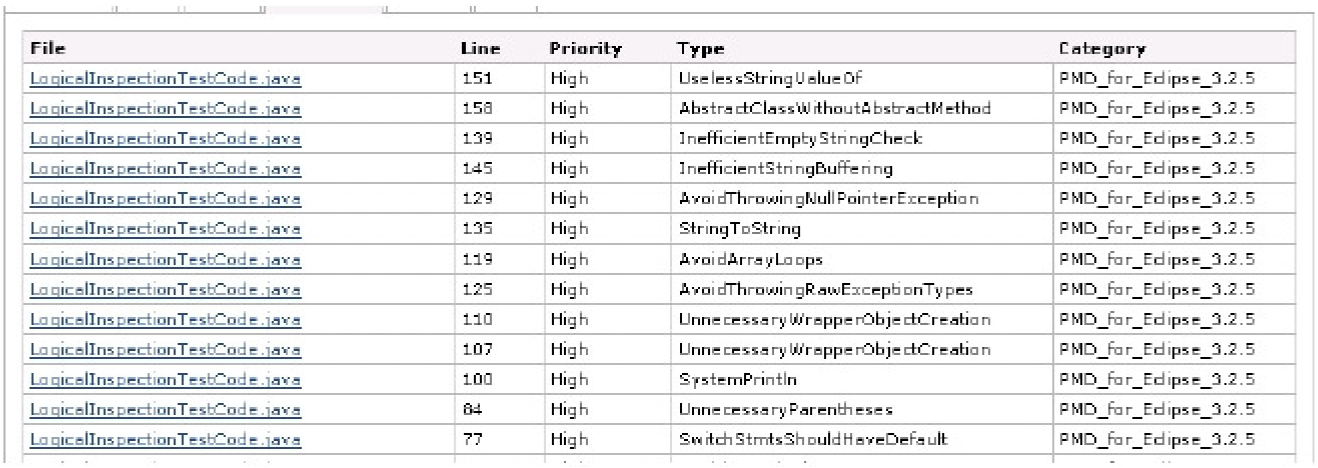
Packages | Files | **Types** | Warnings | Details | New

Type	Total	Distribution
AbstractClassWithoutAbstractMethod	2	🟥
AssignmentToNonFinalStatic	2	🟥
AvoidArrayLoops	2	🟥
AvoidConstantsInterface	1	🟥
AvoidReassigningParameters	2	🟥
AvoidSynchronizedAtMethodLevel	2	🟥
AvoidThrowingNullPointerException	2	🟥
AvoidThrowingRawExceptionTypes	2	🟥
DuplicateImports	3	🟥
EmptyCatchBlock	2	🟥
EmptyFinallyBlock	2	🟥
EmptyIfStmt	2	🟥
EmptyStatementNotInLoop	4	🟥

Warnings

Displays all warnings as below.

Packages | Files | Types | **Warnings** | Details | New



File	Line	Priority	Type	Category
LogicalInspectionTestCode.java	151	High	UselessStringValueOf	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	158	High	AbstractClassWithoutAbstractMethod	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	139	High	InefficientEmptyStringCheck	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	145	High	InefficientStringBuffering	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	129	High	AvoidThrowingNullPointerException	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	135	High	StringToString	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	119	High	AvoidArrayLoops	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	125	High	AvoidThrowingRawExceptionTypes	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	110	High	UnnecessaryWrapperObjectCreation	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	107	High	UnnecessaryWrapperObjectCreation	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	100	High	SystemPrintln	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	84	High	UnnecessaryParentheses	PMD_for_Eclipse_3.2.5
LogicalInspectionTestCode.java	77	High	SwitchStmtsShouldHaveDefault	PMD_for_Eclipse_3.2.5

1) Comma Separated Value

2) Continuous integration

3) Line of Code