Software Verification

Static Analysis Report #1

Team 3

200911429 한종철
200911391 박준모
201111364 신민용

1. PMD

   a. 룰 기반 정적 분석 도구

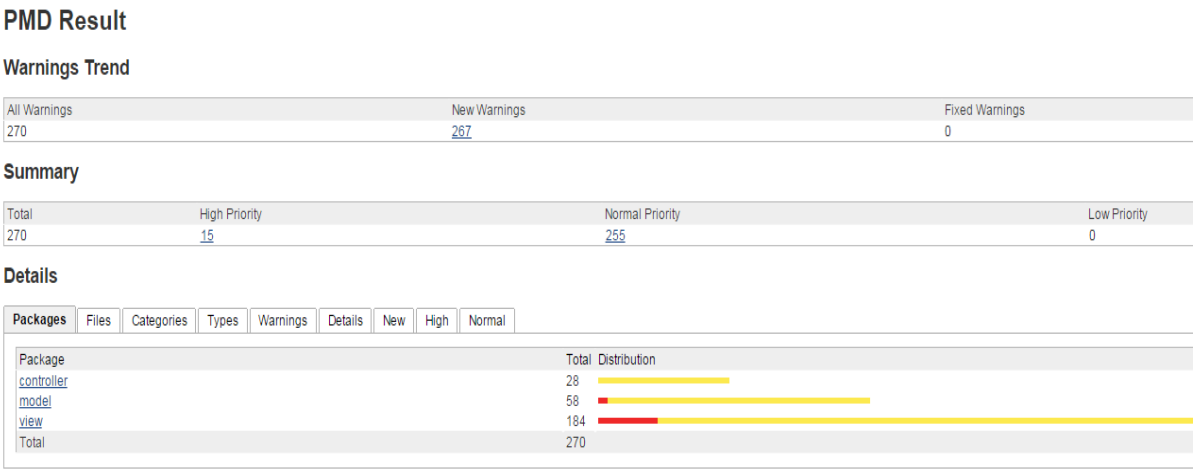
   b. 적용할 Rule을 지정하여 해당 Rule 위반 여부를 확인

   c. 실제 분석에 사용할 Rule을 따로 뽑아내서 Rule Set을 만들었다.

Rule set	설명
Basic	가장 기본적인 코딩에 대한 규칙
Braces	괄호에 대한 규칙
Code Size	CyclomaticComplexity 같은 코드 사이즈에 대한 규칙
Empty code	비어 있는 Catch문 과 같은 비어있는 코드에 대한 규칙
Naming	너무 긴 변수 명 과 같은 naming관련 규칙
Optimization	최적화에 관한 규칙
Unused Code	도달하지 못한 코드에 관한 규칙

   d. Report

(1) Rule Set 위반 총합 개수 : CI서버에 보고된 총 개수는 270개 이다.



(1)-1 Package - controller : 28개

PMD Warnings - Package controller

Summary

Total	High Priority	Normal Priority	Low Priority
28	0	28	0

Details

Files | Categories | Types | Warnings | Details

Category	Total	Distribution
Braces	1	
Naming	7	
Optimization	19	
Unused Code	1	
Total	28	

(1)-2 Package – model : 58개

PMD Warnings - Package model

Summary

Total	High Priority	Normal Priority	Low Priority
58	2	56	0

Details

Files | Categories | Types | Warnings | Details | High | Normal

Category	Total	Distribution
Code Size	1	
Naming	12	
Optimization	45	
Total	58	

(1)-3 Package – view : 184개

PMD Warnings - Package view

Summary

Total	High Priority	Normal Priority	Low Priority
184	13	171	0

Details

Files | Categories | Types | Warnings | Details | High | Normal

Category	Total	Distribution
Braces	12	
Code Size	14	
Empty Code	3	
Naming	45	
Optimization	100	
Unused Code	10	
Total	184	

(2) Rule Category

PMD Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
270	267	0

Summary

Total	High Priority	Normal Priority
270	15	255

Details

Packages | Files | **Categories** | Types | Warnings | Details | New | High | Normal

Category	Total	Distribution
Braces	13	
Code Size	15	
Empty Code	3	
Naming	64	
Optimization	164	
Unused Code	11	
Total	270	

(3) Rule Type

Details

Packages | Files | Categories | **Types** | Warnings | Details | New | High | Normal

Type	Total	Distribution
AddEmptyString	1	
AvoidFieldNameMatchingTypeName	2	
AvoidInstantiatingObjectsInLoops	34	
CyclomaticComplexity	4	
EmptyIfStmt	2	
EmptyStatementNotInLoop	1	
ExcessiveMethodLength	1	
IfElseStmtsMustUseBraces	10	
IfStmtsMustUseBraces	3	
LocalVariableCouldBeFinal	38	
MethodArgumentCouldBeFinal	87	
ModifiedCyclomaticComplexity	2	
NPathComplexity	1	
NcssMethodCount	1	
RedundantFieldInitializer	4	
ShortClassName	1	
ShortVariable	46	
StdCyclomaticComplexity	2	
TooManyFields	3	
TooManyMethods	1	
UnusedLocalVariable	2	
UnusedPrivateField	9	
VariableNamingConventions	15	
Total	270	

(4) type 별 대표 분석 : 총 22가지 type에 Rule 체크를 했으며, 자세한 사항은 첨부한 xml 파일 또는 CI 서버를 참조(Http://125.187.155.70:8080)

1. AddEmptyString : 숫자를 문자열로 변환할 경우 비어있는 문자열("")을 이용하는 것은 매우 비효율적

Details

Details

Database.java:38, AddEmptyString, Priority: Normal

Do not add empty strings.

The conversion of literals to strings by concatenating them with empty strings is inefficient. It is much better to use one of the type-specific toString() methods instead.

```
String s = "" + 123;                    // inefficient
String t = Integer.toString(456);       // preferred approach
```

2. AvoidFieldNameMatchingTypeName : 클래스 명과 그 클래스에 포함된 필드명이 같다면 혼동됨

Details

Details

Alphabet.java:5, AvoidFieldNameMatchingTypeName, Priority: Normal

It is somewhat confusing to have a field name matching the declaring class name.

It is somewhat confusing to have a field name matching the declaring class name. This probably means that type and/or field names should be chosen more carefully.

```
public class Foo extends Bar {
        int foo;        // There is probably a better name that can be used
}
```

3. AvoidInstantiatingObjectsInLoops : 반복문 내에서 객체를 인스턴트화하지 말자.

```
224         Word temp = new Word();
225         temp.setName(rs.getString("name"));
226             temp.setImageURL(rs.getString("imageURL"));
227             temp.setSoundURL(rs.getString("soundURL"));
228             wordarr.add(temp);
```

4. EmptyIfStmt : 이 룰은 비어 있는 조건절을 찾아낸다

Details

Details

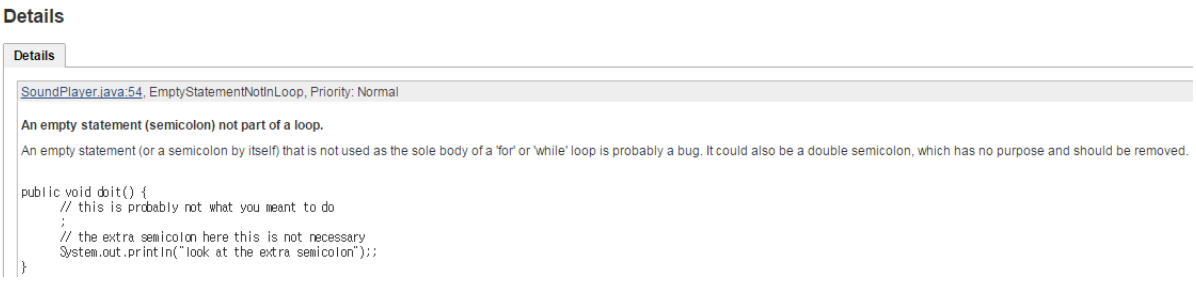
SoundPlayer.java:54, EmptyIfStmt, Priority: Normal

Avoid empty if statements.

Empty If Statement finds instances where a condition is checked but nothing is done about it.

```
public class Foo {
 void bar(int x) {
  if (x == 0) {
   // empty!
  }
 }
}
```

5. EmptyStatementNotInLoop : 비어있는 구문(; <- 세미콜론(semicolon)으로 알려진)은 반복문 내에서는 두개의 ;(세미콜론)을 포함하고 있어야 하며, 하나만 나타날 경우 이는 버그일 가능성이 있다. 또한 코드상 ;(세미콜론) 혼자 나타거나 문장 끝에 중복되어 나타날 경우 불필요한 부분으로 삭제하여야 한다



6. ExcessiveMethodLength : 나의 메소드가 과도한 기능을 수행하는 것은 이 룰을 위반하는 것이며, 메소드의 사이즈를 줄이기 위하여 메소드 내의 기능을 분산 위임하여 실행할 수 있는 helper methods를 생성하거나 복사 붙여넣기로 만든 모든 코드를 제거한다.

```
public void doSomething() {
        System.out.println("Hello world!");
        System.out.println("Hello world!");
                // 98 copies omitted for brevity.
}
```

2. Metrics

a. cyclomatic complexity 등 코드의 복잡성에 대해 검사하는 정적 분석 도구

b. 메소드 개수, 클래스 개수, 총합 코드 라인수 등 다양한 정보 제공

c. 허용치를 넘는 class 및 method가 있는 부분은 빨간색, 정상적인 부분은 파란색으로 표기가 되므로 알아보기가 쉽다.

d. Jekins에서 metrics 관련 플러그인이 없어 Eclipse-Plugin을 통해 자체적으로 분석

e. Metrics 분석 결과 : 개개의 자세한 결과는 첨부한 xml형식의 report로 확인가능

Problems | Javadoc | Declaration | Console | Dependency Graph View | Metrics - Test_CI - Weighted methods per Class (avg/max per type)

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		0.64	0.945	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
Number of Static Attributes (avg/max per type)	0	0	0	0	/Test_CI/src/controller/DictionaryController.java	
Efferent Coupling (avg/max per packageFragment)		3	2.16	5	/Test_CI/src/view	
Specialization Index (avg/max per type)		0.176	0.706	3	/Test_CI/src/view/AlphabetBoard.java	
Number of Classes (avg/max per packageFragment)	17	5.667	1.247	7	/Test_CI/src/view	
Number of Attributes (avg/max per type)	130	7.647	7.412	23	/Test_CI/src/view/DictionaryPanel.java	
Abstractness (avg/max per packageFragment)		0	0	0	/Test_CI/src/controller	
Normalized Distance (avg/max per packageFragment)		0.574	0.34	1	/Test_CI/src/model	
Number of Static Methods (avg/max per type)	1	0.059	0.235	1	/Test_CI/src/controller/MainController.java	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/Test_CI/src/controller	
Total Lines of Code	1706					
Weighted methods per Class (avg/max per type)	235	13.824	9.476	34	/Test_CI/src/view/Board.java	
Number of Methods (avg/max per type)	135	7.941	3.873	17	/Test_CI/src/controller/GameController.java	
Depth of Inheritance Tree (avg/max per type)		2.235	1.864	6	/Test_CI/src/view/AlphabetBoard.java	
Number of Packages	3					
Instability (avg/max per packageFragment)		0.426	0.34	0.833	/Test_CI/src/view	
McCabe Cyclomatic Complexity (avg/max per method)		1.728	1.768	15	/Test_CI/src/view/Board.java	cycle
Nested Block Depth (avg/max per method)		1.412	0.836	6	/Test_CI/src/view/MainFrame.java	gameEvent
Lack of Cohesion of Methods (avg/max per type)		0.564	0.327	0.922	/Test_CI/src/view/DictionaryPanel.java	
Method Lines of Code (avg/max per method)	1132	8.324	18.816	176	/Test_CI/src/view/DictionaryPanel.java	viewDictionary
Number of Overridden Methods (avg/max per type)	3	0.176	0.706	3	/Test_CI/src/view/AlphabetBoard.java	
Afferent Coupling (avg/max per packageFragment)		3.667	1.886	5	/Test_CI/src/controller	
Number of Children (avg/max per type)	4	0.235	0.73	3	/Test_CI/src/model/Contents.java	

- Number of Parameters, McCaBe Cyclomatic Complexity, Nested Block Depth 세 부분 적색으로 표시, 코드 복잡도가 높거나, 가진 인수가 지나치게 많은 부분임을 알 수 있다.

f. 위험 부분 분석

1. Number of Parameters : Parameter의 개수

Problems | Javadoc | Declaration | Console | Dependency Graph View | Metrics - Test_CI - Number of Static Methods (avg/max per type)

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		0.64	0.945	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
src		0.64	0.945	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
view		0.926	1.303	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
AlphabetBoard.java		1.833	2.794	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
AlphabetBoard		1.833	2.794	8	/Test_CI/src/view/AlphabetBoard.java	AlphabetBoard
AlphabetBoard	8					
loadImage	1					
paintComponent	1					
drawStar	1					
cycle	0					
run	0					

```
public AlphabetBoard(MainFrame frame, MainController mainControl, GamePanel panel, int width, int wordWidth , int dx, int dy, int index) {
    this.frame = frame;
    this.mainControl = mainControl;
    this.panel = panel;
    this.width = width;
    this.wordWidth = wordWidth;
    this.dx = dx;
    this.dy = dy;
    this.index = index;
    setDoubleBuffered(true);
    loadImage(index);
    this.setVisible(true);
}
```

-AlphabetBoard가 가진 Parameter가 8개로 너무 많다.

2. McCabe Cyclomatic Complexity : 코드 순환 복잡도. 프로그램의 복잡도를 측정하는 수치 이며, 프로그램 소스 코드의 독립된 경로의 수를 기초로 만들어짐. 컴포넌트를 통한 경로의 수를 측정하며 조건부 분기의 수와 복잡성에 의해 결정된다.

McCabe Cyclomatic Complexity (avg/max per method)		1.728	1.768	15	/Test_CI/src/view/Board.java	cycle
src		1.728	1.768	15	/Test_CI/src/view/Board.java	cycle
view		2.37	2.563	15	/Test_CI/src/view/Board.java	cycle
Board.java		2.833	3.955	15	/Test_CI/src/view/Board.java	cycle
Board		2.833	3.955	15	/Test_CI/src/view/Board.java	cycle
cycle	15					
setdemension	5					
run	5					
Board	1					
Board	1					

- Cyclomatic Complexity 수치가 15이다.

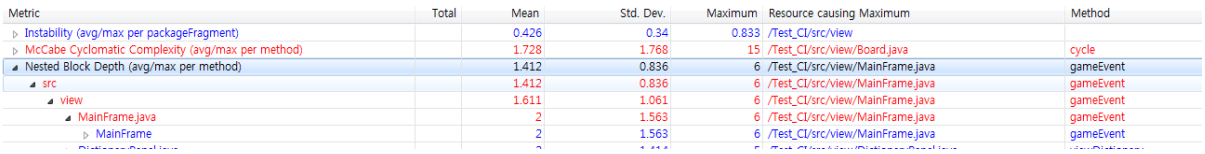
```
protected boolean cycle() {

    if (x > dx && y > dy) {
        x--;
        y--;
        if (x == dx || y == dy)
            return false;
        else
            return true;
    } else if (x > dx && y < dy) {
        x--;
        y++;
        if (x == dx || y == dy)
            return false;
        else
            return true;
    } else if (x < dx && y > dy) {
        x++;
        y--;
        if (x == dx || y == dy)
            return false;
        else
            return true;
    } else {
        x++;
        y++;
        if (x == dx || y == dy)
            return false;
        else
            return true;
    }

}
```

- 대채적으로 Cyclomatic Complexity가 10 이상인 경우에는 관리가 어렵다고 정의 한다. 그런데 해당 수치가 현재 cycle() 에서 15를 갖고있다. If 분기가 많고, if문안에 if문이 들어가 있는 형태로서 복잡도가 높아졌다. Switch case에서 실제 복잡도에 비해 복잡도가 늘어나지만, 현재 코드에는 해당 사항이 없다.

3. Nested Block Depth : 코드의 인접 block의 depth



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Instability (avg/max per packageFragment)		0.426	0.34	0.833	/Test_CI/src/view	
McCabe Cyclomatic Complexity (avg/max per method)		1.728	1.768	15	/Test_CI/src/view/Board.java	cycle
Nested Block Depth (avg/max per method)		1.412	0.836	6	/Test_CI/src/view/MainFrame.java	gameEvent
src		1.412	0.836	6	/Test_CI/src/view/MainFrame.java	gameEvent
view		1.611	1.061	6	/Test_CI/src/view/MainFrame.java	gameEvent
MainFrame.java		2	1.563	6	/Test_CI/src/view/MainFrame.java	gameEvent
MainFrame		2	1.563	6	/Test_CI/src/view/MainFrame.java	gameEvent

- 인접한 block의 depth 를 측정한 수치이며, 기본 maximum은 5인데, view package에서 Total 6의 수치를 갖고 있다.

3.FindBugs

FindBugs Result

Warnings Trend

All Warnings	New this build	Fixed Warnings
20	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
20	4	16	0

Details

Packages | Files | Categories | **Types** | Warnings | Details | High | Normal

Type	Total ↑	Distribution
SS_SHOULD_BE_STATIC	4	
SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	3	
SF_SWITCH_NO_DEFAULT	3	
DLS_DEAD_LOCAL_STORE	2	
IS2_INCONSISTENT_SYNC	1	
DM_STRING_VOID_CTOR	1	
DM_NEXTINT_VIA_NEXTDOUBLE	1	
DM_STRING_CTOR	1	
UCF_USELESS_CONTROL_FLOW	1	
URF_UNREAD_FIELD	1	
URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD	1	
UWF_UNWRITTEN_FIELD	1	
Total	20	

- 총 20개의 Warning 발생

1) SS_SHOULD_BE_STATIC (4 warnings)

- AlphabetBoard.java:12, Board.java:32, 33, 34

- 해당 field 를 static으로 지정해야 한다.

예)

AlphabetBoard.java:12, SS_SHOULD_BE_STATIC, Priority: Normal

SS: Unread field: view.AlphabetBoard.DELAY; should this field be static?

This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.

```
private final int DELAY = 2;
```

2) SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE (3 Warnings)

- Database.java:85, 148, 192
- SQL문에 사용되는 string을 constant로 지정해야 한다. SQL Injection에 덜 취약하고 좀더 효율적이다.

예)

Database.java:85, SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE, Priority: High

SQL: model.Database.updateWord(Word) passes a nonconstant String to an execute method on an SQL statement

The method invokes the execute or addBatch method on an SQL statement with a String that seems to be dynamically generated. Consider using a prepared statement instead. It is more efficient and less vulnerable to SQL injection attacks.

```
082         String sql = "update Word set correct = 1 where name = '"+word.getName()+"'";
083         try{
084           stmt = conn.createStatement();
085           stmt.executeUpdate(sql);
```

3) SF_SWITCH_NO_DEFAULT (3 Warnings)

- AlphabetBoard.java:43, Board.java:87, DictionaryPanel.java:242,
- Switch문에 default case를 추가해야한다.

예)

AlphabetBoard.java:43, SF_SWITCH_NO_DEFAULT, Priority: Normal

SF: Switch statement found in view.AlphabetBoard.loadImage(int) where default case is missing

This method contains a switch statement where default case is missing. Usually you need to provide a default case.

Because the analysis only looks at the generated bytecode, this warning can be incorrect triggered if the default case is at the end of the switch statement and the switch statement doesn't contain break statements for other cases.

```
043         switch(result)
044         {
045         case 0:
046           this.x = (int)((double)Math.random()*((double)this.width-(70*ratio)));
047           break;
048         case 1:
049           this.x = (int)((double)Math.random()*((double)this.width-(70*ratio)));
050           this.x = this.x+ this.width+this.wordWidth;
```

```
051             break;
```

4) DLS_DEAD_LOCAL_STORE (2 Warnings)

- MainController.java:50, DictionaryPanel.java:197
- 해당 변수에 값을 할당했지만 사용되지 않는다.

예)

MainController.java:50, DLS_DEAD_LOCAL_STORE, Priority: High

DLS: Dead store to mf in controller.MainController.main(String[])

This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

```
48  public static void main(String args[])
49  {
50    MainFrame mf = new MainFrame();
51  }
```

5) UCF_USELESS_CONTROL_FLOW (1 Warning)

- SoundPlayer.java:53
- control flow가 불필요하게 분기되어 있다.

예)

SoundPlayer.java:53, UCF_USELESS_CONTROL_FLOW, Priority: Normal

UCF: Useless control flow in view.SoundPlayer.display(String)

This method contains a useless control flow statement, where control flow continues onto the same place regardless of whether or not the branch is taken. For example, this is caused by having an empty statement block for an if statement:

```
if (argv.length == 0) {
// TODO: handle this case
}
```

```
52  public void display(String msg) {
53    if (out != null)
54      ;
55  }
```

4. CheckStyle

CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
489	192	969

Summary

Total	High Priority	Normal Priority	Low Priority
489	0	489	0

Details

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

Type	Total	Distribution
ClassDataAbstractionCouplingCheck	3	
CyclomaticComplexityCheck	1	
EmptyLineSeparatorCheck	120	
FileTabCharacterCheck	16	
JavaNCSSCheck	2	
LocalVariableNameCheck	5	
MemberNameCheck	14	
SeparatorWrapCheck	66	
WhitespaceAfterCheck	19	
WhitespaceAroundCheck	243	
Total	489	

- 총 489개의 Warning 발생

6) ClassDataAbstractionCouplingCheck (3 warnings)

- DictionaryPanel.java:22, MainFrame.java:24, PicturePanel.java:14
- 한 클래스에서 data abstraction coupling(데이터 추상 결합: 다른 클래스의 인스턴스를 생성할 때 이를 가리키는 변수가 없음)이 13개(최대 7개까지 허용) 존재한다.

예)

DictionaryPanel.java:22, ClassDataAbstractionCouplingCheck, Priority: Normal

Class Data Abstraction Coupling is 13 (max allowed is 7) classes [BorderLayout, EtchedBorder, FlowLayout, Font, GridLayout, ImageIcon, JLabel, JPanel, KeyAdapter, Runnable, SoundPlayer, TextField, TitledBorder].

This metric measures the number of instantiations of other classes within the given class. This type of coupling is not caused by inheritance or the object oriented paradigm. Generally speaking, any abstract data type with other abstract data types as members has data abstraction coupling; therefore, if a class has a local variable that is an instantiation (object) of another class, there is data abstraction coupling. The higher the DAC, the more complex the data structure (classes) of the system.

```
022 public class DictionaryPanel {
...
056     frame.setLayout(new BorderLayout());

...
```

7) CyclomaticComplexityCheck (1 Warning)

- Board.java:127
- Cyclomatic Complexity(순환 복잡도)가 15(최대 10까지 허용)다.

예)

Board.java:127, CyclomaticComplexityCheck, Priority: Normal

Cyclomatic Complexity is 15 (max allowed is 10).

Checks cyclomatic complexity against a specified limit. The complexity is measured by the number of if, while, do, for, ?:, catch, switch, case statements, and operators && and || (plus one) in the body of a constructor, method, static initializer, or instance initializer. It is a measure of the minimum number of possible paths through the source and therefore the number of required tests. Generally 1-4 is considered good, 5-7 OK, 8-10 consider re-factoring, and 11+ re-factor now!

```
127   protected boolean cycle() {
128
129     if (x > dx && y > dy) {
130       x--;
131       y--;
132       if (x == dx || y == dy)
133         return false;
134       else
135         return true;
136     } else if (x > dx && y < dy) {
137       x--;
138       y++;
```

```
139         if (x == dx || y == dy)
140           return false;
141         else
142           return true;
143       } else if (x < dx && y > dy) {
...
```

8) EmptyLineSeparatorCheck (120 Warnings)

- DictionaryController.java:12, 13, 24, 27, 44, GameController.java:13, 14, 15, 16, 17, MainController.java:12, 13, 14, 32, 36, 40, 44, PictureController.java:10, 11, 18, 21, 24, 27, 31, Alphabet.java:6, 16, Contents.java:6, Database.java:13, 14, 187, Dictionary.java:16, Picture.java:13, Word.java:9, 10, 11, 12, 13, AlphabetBoard.java:2, 13, 14, 15, 15, 16, 17, 70, Board.java:33, 34, 35, 35, 36, 37, 38, 39, 39, 40, 40, 41, 42, 43, 44, DictionaryPanel.java:24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 46, GamePanel.java:17, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31, 32, 35, 36, 39, 40, MainFrame.java:27, 28, 29, 30, 31, 32, 35, 36, 37, 38, 39, 40, 41, PicturePanel.java:16, 17, 18, 19, 22, 23, 28, 29, 32

- 앞에 공백 라인을 넣도록 한다.

예)

DictionaryController.java:12, EmptyLineSeparatorCheck, Priority: Normal

'VARIABLE_DEF' should be separated from previous statement.

Checks for empty line separators after header, package, all import declarations, fields, constructors, methods, nested classes, static initializers and instance initializers.

```
11      private Word word;
12      private Dictionary dic;
13      private Database db;
14
...
20
21      public Word getWord(){
22        return word;
23      }
24      public Dictionary getDictionary(){
25        return dic;
26      }
27      public void searchbyInitial(char alphabet) {
28          word = db.selectRandomWordbyAlphabet(alphabet);
29      }
...
```

9) FileTabCharacterCheck (16 Warnings)

- DictionaryController.java:16, GameController.java:12, MainController.java:11, PictureController.java:14, Alphabet.java:9, Database.java:12, Dictionary.java:10, Picture.java:1, Word.java:1, AlphabetBoard.java:1, Board.java:3, DictionaryPanel.java:2, GamePanel.java:1, MainFrame.java:2, PicturePanel.java:1, SoundPlayer.java:14

- 자바 파일 내부에 tab 문자가 있다. tab을 모두 space로 치환해서 사용한다.

예)

DictionaryController.java:16, FileTabCharacterCheck, Priority: Normal

File contains tab characters (this is the first instance).

Checks that there are no tab characters ('₩t') in the source code.

Rationale:

- Developers should not need to configure the tab width of their text editors in order to be able to read source code.
- From the Apache jakarta coding standards: In a distributed development environment, when the commit messages get sent to a mailing list, they are almost impossible to read if you use tabs.

```
15      public DictionaryController() {
16        word = new Word();
17        dic = new Dictionary();
18        db = new Database();
19      }
```

10) JavaNCSSCheck (2 Warnings)

- DictionaryPanel.java:120, PicturePanel.java:38

- 주석이 아닌 라인수(Non Commenting Source Statements,NCSS)가 50보다 많다.

예)

DictionaryPanel.java:120, JavaNCSSCheck, Priority: Normal

NCSS for this method is 133 (max allowed is 50).

Determines complexity of methods, classes and files by counting the Non Commenting Source Statements (NCSS). This check adheres to the specification for the JavaNCSS-Tool written by **Chr. Clemens Lee**.
Roughly said the NCSS metric is calculated by counting the source lines which are not comments, (nearly) equivalent to counting the semicolons and opening curly braces.
The NCSS for a class is summarized from the NCSS of all its methods, the NCSS of its nested classes and the number of member variable declarations.
The NCSS for a file is summarized from the ncss of all its top level classes, the number of imports and the package declaration.

Rationale: Too large methods and classes are hard to read and costly to maintain. A large NCSS number often means that a method or class has too many responsibilities and/or functionalities which should be decomposed into smaller units.

```
120   public void viewDictionary(MainController mainControl, JPanel main, M
ainFrame frame) {
121     d = Toolkit.getDefaultToolkit().getScreenSize();
122     ratio = ((double) (d.getWidth() + d.getHeight()) * 0.0005);
...
```

11) LocalVariableNameCheck (5 Warnings)

- DictionaryPanel.java:84, 124, 125, 126, 127
- local, non-final 변수 이름이 format property의 format spec.과 다르다. '_'를 제거한다.

예)

DictionaryPanel.java:84, LocalVariableNameCheck, Priority: Normal

Name 'word_icon' must match pattern '^[a-z][a-zA-Z0-9]*$'.

No description available. Please upgrade to latest checkstyle version.

```
084     ImageIcon word_icon = new ImageIcon(mainControl.getDictionaryContro
ller()
...
124     int animal_Count = 0;
125     int plant_Count = 0;
126     int tool_Count = 0;
127     int nation_Count = 0;
```

12) MemberNameCheck (14 Warnings)

- Contents.java:5, 6, AlphabetBoard.java:12, Board.java:32, 33, 34, DictionaryPanel.java:34, GamePanel.java:21, 22, 23, 24, 25, 26, 31

- instance 변수 이름이 format property의 format spec.과 다르다. '_'를 제거한다.

예)

Contents.java:5, MemberNameCheck, Priority: Normal

Name 'sound_URL' must match pattern '^[a-z][a-zA-Z0-9]*$'.

No description available. Please upgrade to latest checkstyle version.

```
05      private String sound_URL;
06      private String image_URL;
```

13) SeparatorWrapCheck (66 Warnings)

- Board.java:69, DictionaryPanel.java:85, 129, 242, 245, 246, 256, 257, 267, 268, 278, 279, GamePanel.java:72, 128, 131, 149, 150, 153, 156, 169, 171, 174, 176, 192, 193, 196, 199, 211, 213, 232, 233, 234, MainFrame.java:71, 79, 83, 88, 159, 161, 164, 165, 166, 167, 169, 175, 176, 177, 178, 180, 182, 183, 184, 186, 193, 219, PicturePanel.java:93, 95, 102, 103, 105, 106, 157, 159, 165, 166, 168, 169

- '.'를 이전 줄로 올린다.

예)

Board.java:69, SeparatorWrapCheck, Priority: Normal

'.' should be on the previous line.

Checks line wrapping with separators.

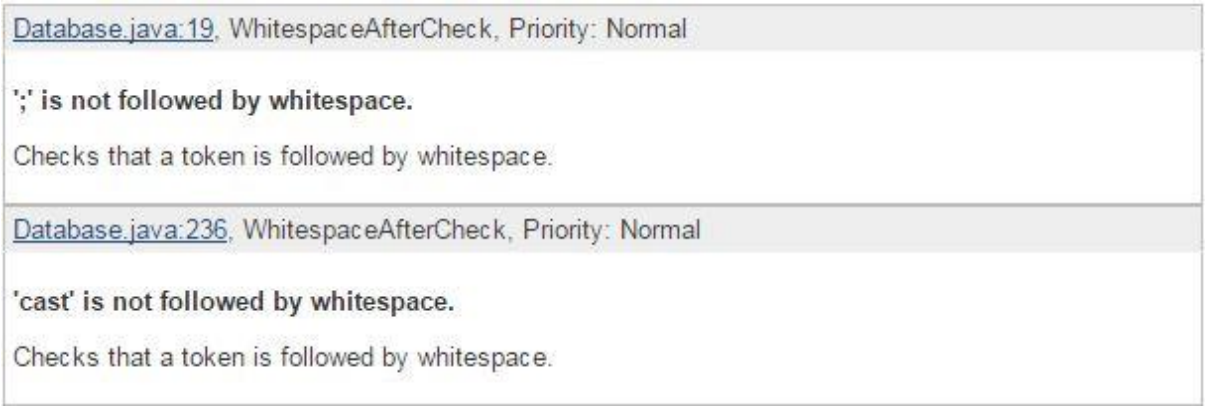
```
068     ImageIcon ii = new ImageIcon(mainControl.getGameController().getWor
d()
069             .getImageURL());
```

14) WhitespaceAfterCheck (19 Warnings)

- Database.java:19, 236, AlphabetBoard.java:34, 37, 40, 40, 40, 46, 46, 46, 49, 49, 49, 53, 53, 53, 66, 66, MainFrame.java:64

- 해당 token 뒤에 공백을 넣는다.

예)




```
019      }catch(Exception e){e.printStackTrace();}
...
236      randNum = (int)(randomvalue*wordarr.size());
```

(공백이 잇는 것처럼 보이지만 ' = '를 제외하고는 공백이 전혀 없음)

15) WhitespaceAroundCheck (243 Warnings)

- DictionaryController.java:21, 24, 36, 36, 39, 39, MainController.java:14, 14, 16, 23, 23, 23, 23, 33, 33, 37, 37, 41, 41, 44, PictureController.java:31, Alphabet.java:9, 20, Database.java:16, 17, 17, 19, 19, 19, 19, 19, 19, 22, 23, 23, 25, 25, 25, 25, 26, 26, 26, 26, 38, 38, 38, 38, 42, 52, 52, 52, 62, 62, 62, 62, 63, 63, 66, 74, 74, 74, 82, 82, 82, 82, 83, 83, 87, 87, 87, 100, 100, 103, 115, 115, 115, 124, 124, 127, 137, 137, 137, 145, 145, 145, 145, 146, 146, 150, 150, 150, 159, 159, 159, 159, 160, 160, 164, 164, 164, 172, 172, 172, 172, 173, 173, 176, 182, 182, 182, 189, 189, 189, 189, 190, 190, 193, 206, 206, 206, 218, 218, 218, 218, 219, 219, 222, 232, 232, 232, 236, 236, 242, 242, 244, 244, 244, Picture.java:5, Word.java:6, 29, 29, 41, 41, 45, 45, 53, 53, 61, 61, 61, 61, 61, AlphabetBoard.java:34, 34, 37, 37, 37, 37, 37, 37, 40, 40, 40, 40, 46, 46, 46, 46, 46, 46, 49, 49, 49, 49, 49, 49, 50, 50, 50, 53, 53, 53, 53, 53, 53, 57, 57, 61, 66, 66, 66, 66, 73, 73, 73, 73, 73, 73, 73, 73, 74, 74, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 77, 77, 79, 79, 81, 81, 81, 81, 81, 81, 81, 83, 83, 85, 85, 85, 85, 85, 85, 85, 85, 85, 87, 87, 89, 89, 89, 89, 121, 121, 121
- 해당 token 앞뒤에 공백을 넣는다.

예)

DictionaryController.java:24, WhitespaceAroundCheck, Priority: Normal

'{' is not preceded with whitespace.

Checks that a token is surrounded by whitespace. Empty constructor, method, class, enum, interface, loop bodies (blocks) of the form

public MyClass() {} // empty constructor public void func() {} // empty method public interface Foo {} // empty interface public class Foo {} // empty class public enum Foo {} // empty enum MyClass c = new MyClass() {}; // empty anonymous class while (i = 1) {} // empty while loop for (int i = 1; i > 1; i++) {} // empty for loop do {} while (i = 1); // empty do-while loop public @interface Beta {} // empty annotation type

may optionally be exempted from the policy using the allowEmptyMethods, allowEmptyConstructors , allowEmptyTypes and allowEmptyLoops properties.

DictionaryController.java:36, WhitespaceAroundCheck, Priority: Normal

'if' is not followed by whitespace.

Checks that a token is surrounded by whitespace. Empty constructor, method, class, enum, interface, loop bodies (blocks) of the form

public MyClass() {} // empty constructor public void func() {} // empty method public interface Foo {} // empty interface public class Foo {} // empty class public enum Foo {} // empty enum MyClass c = new MyClass() {}; // empty anonymous class while (i = 1) {} // empty while loop for (int i = 1; i > 1; i++) {} // empty for loop do {} while (i = 1); // empty do-while loop public @interface Beta {} // empty annotation type

may optionally be exempted from the policy using the allowEmptyMethods, allowEmptyConstructors , allowEmptyTypes and allowEmptyLoops properties.

```
24      public Dictionary getDictionary(){
...
36          if(text.length() == 1){
```