

Software Verification

Static Analysis Report

for Team 4

Date

2018-06-04

Team 4

201411259 고수창

201411314 전소영

201412005 이세라

201511304 하지윤

1. Sonarqube	3
1.1 URL	3
1.2 Project Name	3
2. PMD	4
2.1 Rules	4
2.2 Analysis Result	4
3. Findbugs	6
3.1 Rules	6
3.2 Analysis Result	6
4. CheckStyle	7
4.1 Rules	7
4.2 Analysis Result	7
5. Code Scroll	8
5.1 Rules	8
5.2 Analysis Result	9

1. Sonarqube

1.1 URL

- <http://sonarqube.kodev.kr>
- 자세한 결과는 위 주소에서 확인 가능

1.2 Project Name

- CheckStyle: checkstyle_google
- FindBugs: bugfind
- PMD: pmd

The screenshot displays the SonarQube interface for a project named 'softwareModeling'. The main area shows 'Coverage on New Code' at 0.0%. A table lists five test files with their respective coverage metrics:

File	Coverage on New Code	Uncovered Lines on New Code	Uncovered Conditions on New Code
unit_test/src/AccountTest.java	0.0%	18	0
unit_test/src/AtmSystemTest.java	0.0%	99	0
unit_test/src/BankbookTest.java	0.0%	5	0
unit_test/src/BankTest.java	0.0%	36	0
unit_test/src/CardTest.java	0.0%	11	0

Overall Coverage: 0.0%
Lines to Cover: 169
Uncovered Lines: 169
Line Coverage: 0.0%
Conditions to Cover: 0
Uncovered Conditions: 0

Footer: SonarQube™ technology is powered by SonarSource SA. Version 7.0 (build 36138) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

2. PMD

2.1 Rules

Details

Folders	Files	People	Categories	Types	Warnings	Origin	Details	New	Fixed	Normal	Low
Type	Total	Distribution									
AccessorClassGeneration	34										
AccessorMethodGeneration	92										
AssignmentInOperand	2										
AtLeastOneConstructor	5										
AvoidCatchingGenericException	3										
AvoidDuplicateLiterals	2										
AvoidInstantiatingObjectsInLoops	5										
CallSuperInConstructor	10										
CommentDefaultAccessModifier	16										
CommentRequired	184										
ConfusingTernary	2										
DataflowAnomalyAnalysis	14										
DefaultPackage	16										
ImmutableField	29										
LawOfDemeter	46										
LocalVariableCouldBeFinal	107										
LooseCoupling	18										
MethodArgumentCouldBeFinal	60										
NullAssignment	4										
OnlyOneReturn	9										
PositionLiteralsFirstInComparisons	2										
SimpleDateFormatNeedsLocale	1										
TooManyMethods	1										
UseCollectionsEmpty	1										
UseUtilityClass	1										
UselessParentheses	2										
Total	666										

2.2 Analysis Result

- <http://jenkins.kodev.kr/job/softwareModeling/13/pmdResult/> 에서 자세한 전체 사항 확인 가능
- 총 666개의 문제 발생

[Account.java:6](#), ImmutableField, Priority: Normal

Private field 'password' could be made final; it is only initialized in the declaration or constructor.

Identifies private fields whose values never change once they are initialized either in the declaration of the field or by a constructor. This helps in converting existing classes to becoming immutable ones.

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

[Account.java:8](#), MethodArgumentCouldBeFinal, Priority: Normal

Parameter 'accountNum' is not assigned and could be declared final.

A method argument that is never re-assigned within the method can be declared final.

```
public void foo1 (String param) {          // do stuff with param never assigning it
}

public void foo2 (final String param) { // better, do stuff with param never assigning it
}
```

[AtmSystem.java:28](#), NullAssignment, Priority: Normal

Assigning an Object to null is a code smell. Consider refactoring.

Assigning a "null" to a variable (outside of its declaration) is usually bad form. Sometimes, this type of assignment is an indication that the programmer doesn't completely understand what is going on in the code. NOTE: This sort of assignment may be used in some cases to dereference objects and encourage garbage collection.

```
public void bar() {
    Object x = null; // this is OK
    x = new Object();
    // big, complex piece of code here
    x = null; // this is not required
    // big, complex piece of code here
}
```

[AtmSystem.java:29](#), NullAssignment, Priority: Normal

Assigning an Object to null is a code smell. Consider refactoring.

Assigning a "null" to a variable (outside of its declaration) is usually bad form. Sometimes, this type of assignment is an indication that the programmer doesn't completely understand what is going on in the code. NOTE: This sort of assignment may be used in some cases to dereference objects and encourage garbage collection.

```
public void bar() {
    Object x = null; // this is OK
    x = new Object();
    // big, complex piece of code here
    x = null; // this is not required
    // big, complex piece of code here
}
```

3. Findbugs

3.1 Rules

Details

Files	Categories	Types	Warnings	Origin	Details	New	High	Normal
			Type		Total	Distribution		
			DM_DEFAULT_ENCODING		2			
			OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE		1			
			OS_OPEN_STREAM		1			
			SIC_INNER_SHOULD_BE_STATIC		2			
			URF_UNREAD_FIELD		2			
			Total		8			

3.2 Analysis Result

- <http://jenkins.kodev.kr/job/softwareModeling/16/findbugsResult/> 에서 자세한 사항을 확인할 수 있다.
- 총 11개의 문제 발생

<p>Bank.java:23, DM_DEFAULT_ENCODING, Priority: High</p> <p>Dm: Found reliance on default encoding in Bank.fileReader(String): new java.io.FileReader(File)</p> <p>Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. Use an alternative API and specify a charset name or Charset object explicitly.</p>
<p>Bank.java:24, OS_OPEN_STREAM, Priority: Normal</p> <p>OS: Bank.fileReader(String) may fail to close stream</p> <p>The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a <code>finally</code> block to ensure that streams are closed.</p>
<p>Bank.java:74, OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE, Priority: Normal</p> <p>OBL: Bank.transaction(Account, int, String) may fail to clean up java.io.Writer on checked exception</p> <p>This method may fail to clean up (close, dispose of) a stream, database object, or other resource requiring an explicit cleanup operation.</p> <p>In general, if a method opens a stream or other resource, the method should use a try/finally block to ensure that the stream or resource is cleaned up before the method returns.</p> <p>This bug pattern is essentially the same as the <code>OS_OPEN_STREAM</code> and <code>ODR_OPEN_DATABASE_RESOURCE</code> bug patterns, but is based on a different (and hopefully better) static analysis technique. We are interested in getting feedback about the usefulness of this bug pattern. To send feedback, either:</p> <ul style="list-style-type: none">• send email to findbugs@cs.umd.edu• file a bug report: http://findbugs.sourceforge.net/reportingBugs.html <p>In particular, the false-positive suppression heuristics for this bug pattern have not been extensively tuned, so reports about false positives are helpful to us.</p> <p>See Weimer and Neula, <i>Finding and Preventing Run-Time Error Handling Mistakes</i>, for a description of the analysis technique.</p>
<p>Bank.java:74, DM_DEFAULT_ENCODING, Priority: High</p> <p>Dm: Found reliance on default encoding in Bank.transaction(Account, int, String): new java.io.PrintWriter(File, boolean)</p> <p>Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. Use an alternative API and specify a charset name or Charset object explicitly.</p>
<p>Bankbook.java:5, URF_UNREAD_FIELD, Priority: Normal</p> <p>URF: Unread field: Bankbook.account</p> <p>This field is never read. Consider removing it from the class.</p>
<p>Card.java:5, URF_UNREAD_FIELD, Priority: Normal</p> <p>URF: Unread field: Card.account</p> <p>This field is never read. Consider removing it from the class.</p>
<p>View.java:320, SIC_INNER_SHOULD_BE_STATIC, Priority: Normal</p> <p>SIC: Should View\$Btn be a _static_ inner class?</p> <p>This class is an inner class, but does not use its embedded reference to the object which created it. This reference makes the instances of the class larger, and may keep the reference to the creator object alive longer than necessary. If possible, the class should be made static.</p>
<p>View.java:429, SIC_INNER_SHOULD_BE_STATIC, Priority: Normal</p> <p>SIC: Should View\$NextBtn be a _static_ inner class?</p> <p>This class is an inner class, but does not use its embedded reference to the object which created it. This reference makes the instances of the class larger, and may keep the reference to the creator object alive longer than necessary. If possible, the class should be made static.</p>





4. CheckStyle

4.1 Rules

Summary

Total	High Priority	Normal Priority	Low Priority
77	0	77	0

Details

Category	Total	Distribution
Imports	15	
Indentation	2	
Naming	4	
Whitespace	56	
Total	77	

4.2 Analysis Result

- <http://jenkins.kodev.kr/job/softwareModeling/16/checkstyleResult/> 에서 자세한 사항을 확인할 수 있다.
- 총 77개의 문제 발생

Account.java:8 , EmptyLineSeparatorCheck, Priority: Normal 'CTOR_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.
Account.java:14 , EmptyLineSeparatorCheck, Priority: Normal 'METHOD_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.
Account.java:18 , EmptyLineSeparatorCheck, Priority: Normal 'METHOD_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.
Account.java:22 , EmptyLineSeparatorCheck, Priority: Normal 'METHOD_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.
Account.java:26 , EmptyLineSeparatorCheck, Priority: Normal 'METHOD_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.
Account.java:30 , EmptyLineSeparatorCheck, Priority: Normal 'METHOD_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.

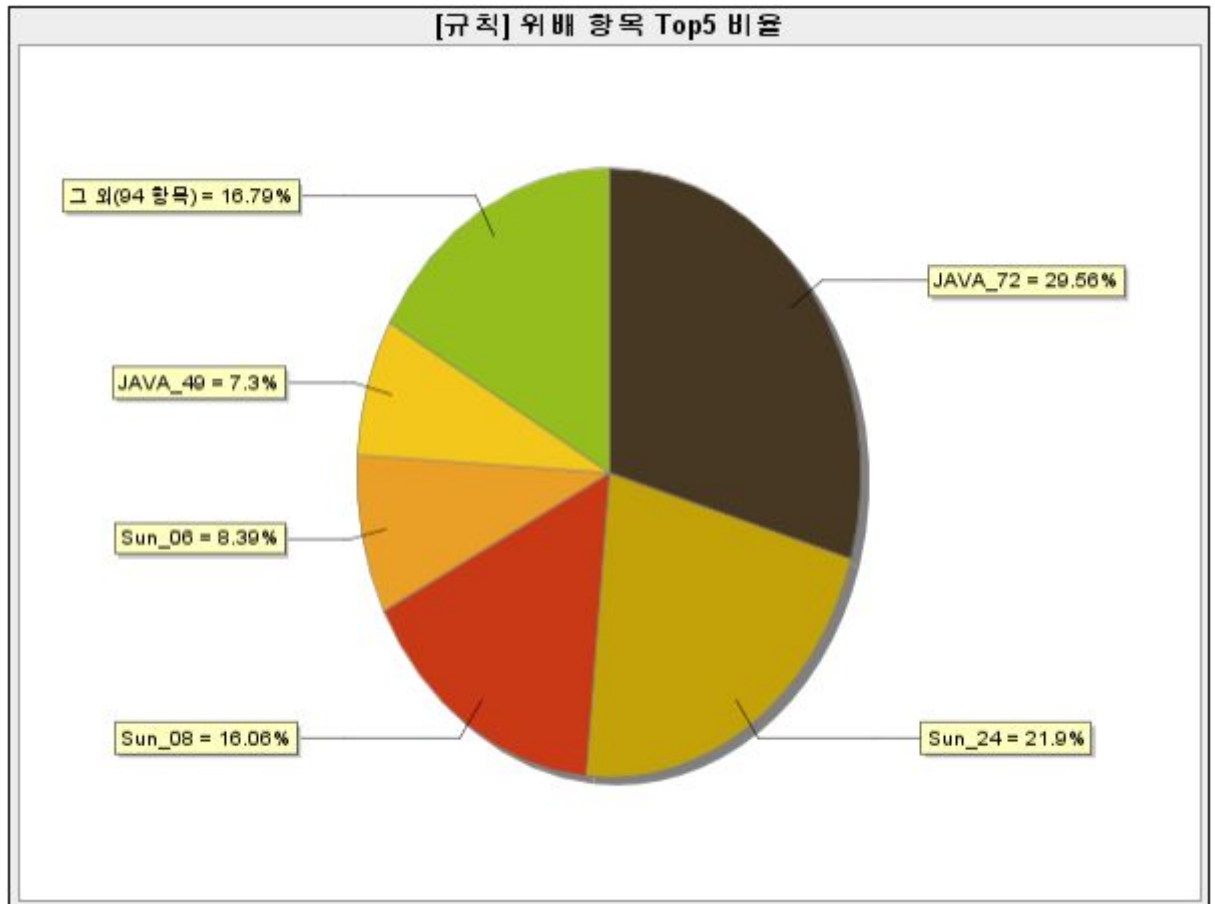
<p>BankTest.java:46, <code>WhitespaceAroundCheck</code>, Priority: Normal</p> <p>WhitespaceAround: '+' is not followed by whitespace. Empty blocks may only be represented as {} when not part of a multi-block statement (4.1.3)</p> <p>Checks that a token is surrounded by whitespace. Empty constructor, method, class, enum, interface, loop bodies (blocks), lambdas of the form</p> <pre>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 Runnable noop = () -> {} // empty lambda public @interface Beta {} // empty annotation type</pre> <p>may optionally be exempted from the policy using the <code>allowEmptyMethods</code>, <code>allowEmptyConstructors</code>, <code>allowEmptyTypes</code>, <code>allowEmptyLoops</code> and <code>allowEmptyLambdas</code> properties.</p> <p>This check does not flag as violation double brace initialization like:</p> <pre>new Properties() {{ setProperty("key", "value"); }};</pre>
<p>BankbookTest.java:3, <code>CustomImportOrderCheck</code>, Priority: Normal</p> <p>Import statement for 'org.junit.Assert.assertEquals' is in the wrong order. Should be in the 'STATIC' group, expecting not assigned imports on this line.</p> <p>Checks that the groups of import declarations appear in the order specified by the user. If there is an import but its group is not specified in the configuration such an import should be placed at the end of the import list.</p> <p>Examples section contains examples that work with default formatter configurations of Eclipse, IntelliJ IDEA and NetBeans</p>
<p>CardTest.java:3, <code>AvoidStarImportCheck</code>, Priority: Normal</p> <p>Using the '*' form of import should be avoided - org.junit.Assert.*.</p> <p>Checks that there are no import statements that use the * notation.</p> <p>Rationale: Importing all classes from a package or static members from a class leads to tight coupling between packages or classes and might lead to problems when a new version of a library introduces name clashes.</p>
<p>CardTest.java:3, <code>CustomImportOrderCheck</code>, Priority: Normal</p> <p>Import statement for 'org.junit.Assert.*' is in the wrong order. Should be in the 'STATIC' group, expecting not assigned imports on this line.</p> <p>Checks that the groups of import declarations appear in the order specified by the user. If there is an import but its group is not specified in the configuration such an import should be placed at the end of the import list.</p> <p>Examples section contains examples that work with default formatter configurations of Eclipse, IntelliJ IDEA and NetBeans</p>

5. Code Scroll

5.1 Rules

규칙 모음	포함된 규칙 수	위배 수	무시된 위배 수	설명
Sun_Code_Conventions_for_Java	27	159	0	파일 이름, 파일 구성, 들여쓰기, 주석, 선언, 문장, 이름 규칙, 프로그래밍 practice 등 Sun에서 따르기를 권고하는 표준 코딩 가이드라인
CODESCROLL_JAVA_RULES	72	115	0	일반적인 자바프로그램에서 지켜야 할 규칙과 권고들

5.2 Analysis Result



규칙	규칙 모음	위배 수	무시된 위배 수	규칙 설명
JAVA_72	CODESCROLL_JAVA_RULES	81	0	메서드의 설명 주석이 있어야 함
Sun_24	Sun_Code_Conventions_for_Java	60	0	숫자 상수 사용 금지
Sun_08	Sun_Code_Conventions_for_Java	44	0	블록의 시작 부분에서만 선언 검사
Sun_06	Sun_Code_Conventions_for_Java	23	0	소스 라인 길이 검사
JAVA_49	CODESCROLL_JAVA_RULES	20	0	static, local, anonymous가 아닌 내부 클래스 사용 금지
Sun_17	Sun_Code_Conventions_for_Java	11	0	메서드 내부의 지역 변수 선언부와 실행 문장 사이를 빈 줄로 구분
Sun_03	Sun_Code_Conventions_for_Java	7	0	소스 파일 시작의 C 스타일 주석 검사

Sun_04	Sun_Code_Conventions_for_Java	7	0	파일에 package 선언이 있는지 검사
JAVA_71	CODESCROLL_JAVA_RULES	7	0	클래스의 설명 주석이 있어야 함
Sun_09	Sun_Code_Conventions_for_Java	5	0	필드 hide 제한
JAVA_70	CODESCROLL_JAVA_RULES	5	0	변수 hiding 금지
JAVA_44	CODESCROLL_JAVA_RULES	2	0	printStackTrace 메서드 사용 금지
Sun_10	Sun_Code_Conventions_for_Java	1	0	지역 변수 선언 시 초기화 검사
Sun_26	Sun_Code_Conventions_for_Java	1	0	괄호 안의 수식에 연산자 혼용 금지

- 심각도

심각도	위배 수	무시된 위배 수	포함된 규칙 수	위배 규칙 수	*SCR	설명
매우높음	2	0	20	1	95%	프로그램에 매우 심각한 영향을 미치므로 오류 수정이 필요함
높음	113	0	46	4	91.3%	프로그램에 심각한 영향을 줄 수 있으므로 오류 수정이 강력히 권장됨
낮음	7	0	10	3	70%	프로그램에 영향을 줄 수 있으므로 오류 수정이 권장됨
매우낮음	152	0	23	6	73.91%	프로그램에 큰 영향은 없지만 오류 수정이 권장됨
기타	0	0	0	0	0%	심각도가 정의되지 않음

- View.java에서 가장 많은 위배 항목 발견

소스	위배 수	무시된 위배 수	위배 규칙 수	*RCR	**RVD
View.java	151	0	9	90.91%	0.31
Bank.java	51	0	12	87.88%	0.43
AtmSystem.java	45	0	9	90.91%	0.29
Account.java	11	0	4	95.96%	0.34
Card.java	7	0	4	95.96%	0.36
Bankbook.java	5	0	4	95.96%	0.55
Terminate.java	4	0	4	95.96%	0.66