<Software Modeling & Analysis>

정적 분석 대응 보고서

- 그 놈! Clone Checker -

Project Team

T4

Date

2016-06-09

Team Information

201411258 강태준

201411265 김서우

201411321 홍유리

<Contents>

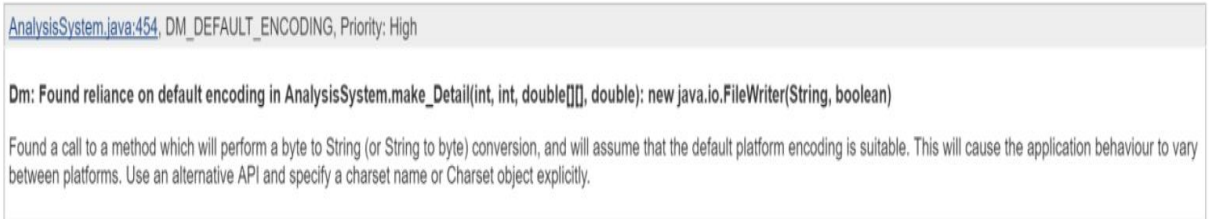
1. SonarQube 대응

분석 결과	가장 많이 위반한 규칙 : Duplicated lines A. AnalysisSytem.java 파일의 전체 659라인 중 238라인이 중복 B. SetupSystem.java 파일의 전체 466라인 중 52라인이 중복
대응	어느 부분이 중복인지 정확히 보이지 않아 대응하기에 모호하였고, 중복의 원인이 함수 호출이나, 유사도 검사를 위한 반복문의 잦은 사용으로 보임.

2. FindBugs 대응

1) High 등급

- AnalysisSystem.java : 454 DM_DEFAULT_ENCODING

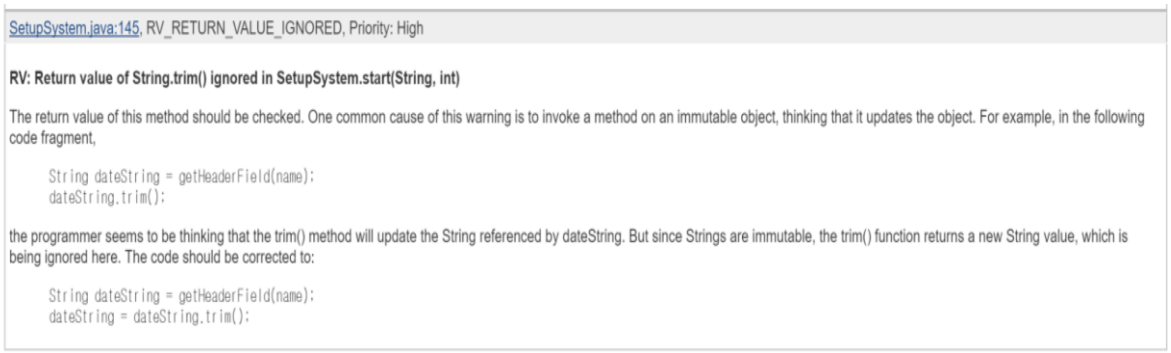

AnalysisSystem.java:454, DM_DEFAULT_ENCODING, Priority: High

Dm: Found reliance on default encoding in AnalysisSystem.make_Detail(int, int, double[][], double): new java.io.FileWriter(String, boolean)

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.

원인 : 플랫폼 마다 동작이 다를 수 있다고 Warning 하는 부분
대응 : 본 프로그램은 구현 환경과, 동작 환경 OS가 Windows이므로 딱히 문제가 없다고 판단

- SetupSystem.java:145 RV_RETURN_VALUE_IGNORED


SetupSystem.java:145, RV_RETURN_VALUE_IGNORED, Priority: High

RV: Return value of String.trim() ignored in SetupSystem.start(String, int)

The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. For example, in the following code fragment,

```
String dateString = getHeaderField(name);
dateString.trim();
```

the programmer seems to be thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() function returns a new String value, which is being ignored here. The code should be corrected to:

```
String dateString = getHeaderField(name);
dateString = dateString.trim();
```

원인 : String 변수를 trim 한 후에 String 변수에 다시 배정을 안해서 발생한 Warning
대응 : trim 한 후에 다시 String 변수에 배정
[기존 코드]

```
144                 temp_str = temp.replaceAll(";", "");
145                 temp_str.trim();
146                 temp_str = temp_str.substring(temp_str.lastIndexOf(" ")+1);
147                 int exist_flag=0;//없으면0 있으면 1
148                 for(String s: sc[index].type_list){
149                     if(s.equals(temp_str)){
```

[수정 코드]

```
144                 temp_str = temp.replaceAll(";", "");
145                 temp_str = temp_str.trim();
146                 temp_str = temp_str.substring(temp_str.lastIndexOf(" ")+1);
147                 int exist_flag=0;//없으면0 있으면 1
148                 for(String s: sc[index].type_list){
```

2) Normal 등급

- 분석 결과

AnalysisSystem.java:85, SF_SWITCH_NO_DEFAULT, Priority: Normal

SF: Switch statement found in AnalysisSystem.analyzeVar(SourceCode, SourceCode) 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.

AnalysisSystem.java:225, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: AnalysisSystem.analyzeFunc(SourceCode, SourceCode) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

GUI.java:346, DLS_DEAD_LOCAL_STORE, Priority: Normal

DLS: Dead store to e1 in GUI$6.actionPerformed(ActionEvent)

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.

GUI.java:392, DM_EXIT, Priority: Normal

Dm: GUI$7$1.actionPerformed(ActionEvent) invokes System.exit(...), which shuts down the entire virtual machine

Invoking System.exit shuts down the entire Java virtual machine. This should only been done when it is appropriate. Such calls make it hard or impossible for your code to be invoked by other code. Consider throwing a RuntimeException instead.

Loop.java:6, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: Loop.loop_type

This field is never read. Consider removing it from the class.

Loop.java:7, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: Loop.loop_body

This field is never read. Consider removing it from the class.

MainController.java:13, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: MainController.folder_path

This field is never read. Consider removing it from the class.

SetupSystem.java:30, NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE, Priority: Normal

NP: Possible null pointer dereference in SetupSystem.folder_list(String) due to return value of called method

The return value from a method is dereferenced without a null check, and the return value of that method is one that should generally be checked for null. This may lead to a `NullPointerException` when the code is executed.

SetupSystem.java:83, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: SetupSystem.setting_files(String, int) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

SetupSystem.java:271, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: SetupSystem.start(String, int) concatenates strings using + in a loop

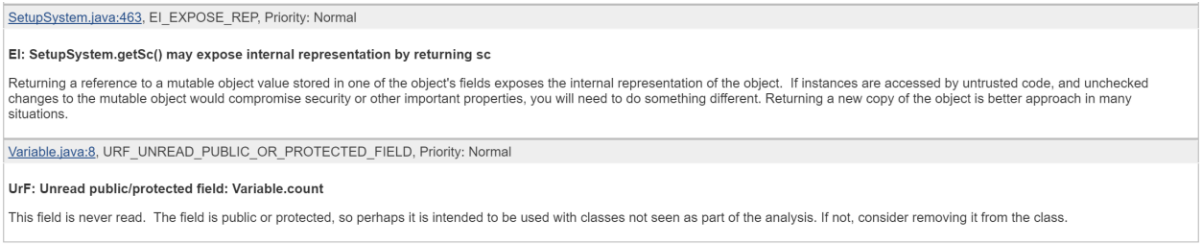
The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```



SetupSystem.java:463, EI_EXPOSE_REP, Priority: Normal

EI: SetupSystem.getSc() may expose internal representation by returning sc

Returning a reference to a mutable object value stored in one of the object's fields exposes the internal representation of the object. If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Returning a new copy of the object is better approach in many situations.

Variable.java:8, URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD, Priority: Normal

UrF: Unread public/protected field: Variable.count

This field is never read. The field is public or protected, so perhaps it is intended to be used with classes not seen as part of the analysis. If not, consider removing it from the class.

- 대응 방안(예정)

1) Switch 문에서 Default의 경우가 없음 => case default : break; 추가

2) String에 + 연산을 반복문을 통하여 진행하여 발생하는 Warning => StringBuffer의 append()함수를 이용하여 변수를 계산한 뒤에 최종적인 값을 String에 대입

3) Unread 되었던 부분 => 삭제를 하거나 read 하는 부분이 있도록 수정

3. PMD 대응

- 분석 결과

AnalysisSystem.java:4, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.util.ArrayList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:3, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.awt.FileDialog'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:7, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JScrollBar'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:8, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JScrollPane'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:9, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JTextArea'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:15, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.BufferedWriter'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:16, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.FileWriter'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:24, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.ScrollPaneConstants'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:26, DontImportJavaLang, Priority: Normal

Avoid importing anything from the package java.lang.

Avoid importing anything from the package 'java.lang'. These classes are automatically imported (JLS 7.5.3).

```
import java.lang.String;        // this is unnecessary

public class Foo {}

// --- in another source code file...

import java.lang.*;     // this is bad

public class Foo {}
```

SetupSystem.java:7, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.StringReader'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

SetupSystem.java:8, DontImportJavaLang, Priority: Normal

Avoid importing anything from the package java.lang.

Avoid importing anything from the package 'java.lang'. These classes are automatically imported (JLS 7.5.3).

```
import java.lang.String;        // this is unnecessary

public class Foo {}

// --- in another source code file...

import java.lang.*;     // this is bad

public class Foo {}
```

SetupSystem.java:9, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.util.LinkedList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

SourceCode.java:2, UnusedImports, Priority: Normal

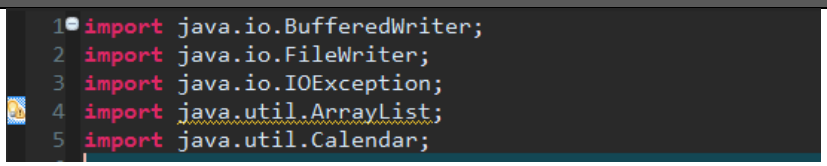
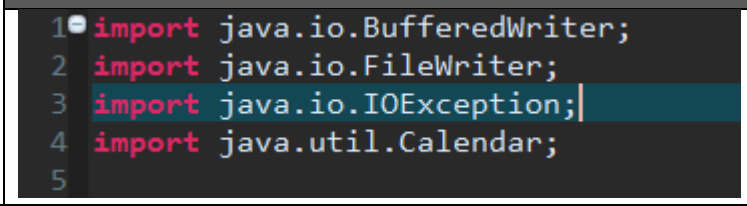
Avoid unused imports such as 'java.util.LinkedList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

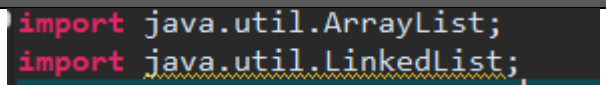
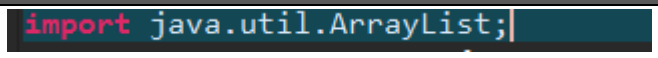
```
// this is bad
import java.io.File;
public class Foo {}
```

- 대응

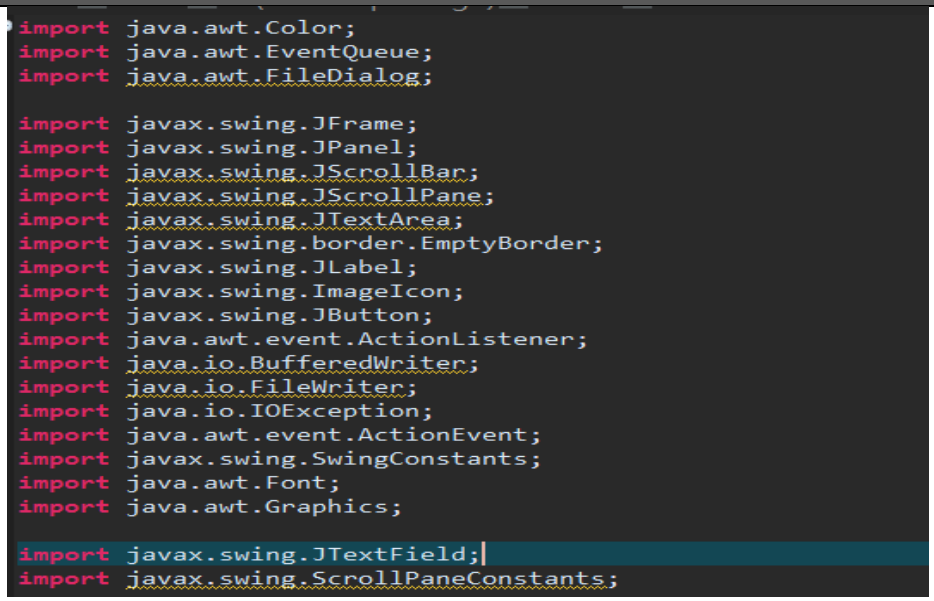
1) AnalysisSystem.java

line number	기존 코드
4	 수정 코드 

2) SourceCode.java

line number	기존 코드
2	 수정 코드 

3) GUI.java

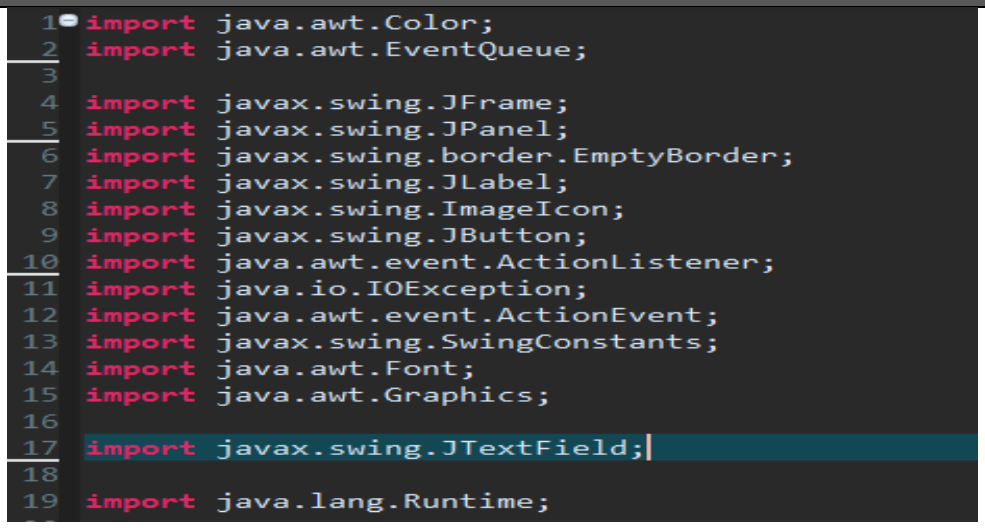
line number	기존 코드
3,7,8,9, 15, 16,24	

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.FileDialog;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.awt.event.ActionEvent;
import javax.swing.SwingConstants;
import java.awt.Font;
import java.awt.Graphics;

import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;
```

수정 코드



```
1  import java.awt.Color;
2  import java.awt.EventQueue;
3
4  import javax.swing.JFrame;
5  import javax.swing.JPanel;
6  import javax.swing.border.EmptyBorder;
7  import javax.swing.JLabel;
8  import javax.swing.ImageIcon;
9  import javax.swing.JButton;
10 import java.awt.event.ActionListener;
11 import java.io.IOException;
12 import java.awt.event.ActionEvent;
13 import javax.swing.SwingConstants;
14 import java.awt.Font;
15 import java.awt.Graphics;
16
17 import javax.swing.JTextField;
18
19 import java.lang.Runtime;
```

4) SetupSystem.java

line number	기존 코드

7,9	`import java.io.StringReader;`
	`import java.lang.String;`
	`import java.util.LinkedList;`
	수정 코드
	`import java.lang.String;`
	`import java.util.regex.Matcher;`