Software Verification Static Analysis

Static Analysis

- PMD, Findbugs, Checkstyle, Codescroll

Project Team

T6

201311265 김상원

201214150 정성철

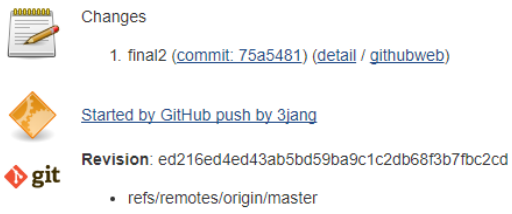
201210908 김성일

Date

2018-06-02

목차

1. 개요



- 상세한 리포트는 기술된 주소를 통하여 확인할 수 있다.
[http://52.231.190.169:9000/dashboard?id=dslab2018]

- 'Reliable ATM' 프로젝트의 빌드 번호 #52번(업로드 시간 2018. 5. 31 오후 6:44) 결과물을 대상으로 정적분석을 진행하였다.

- 정적 분석을 위하여 Sonarqube와 PMD, Findbugs, Checkstyle 플러그인을 사용하였고, 추가로 Suresoft Codescroll Inspector for JAVA를 사용하였다.

2. Sonarqube를 통한 정적 분석

2.1 공통 사항

- 'Sonarqube'의 품질 게이트를 설정하여 일정 수준이상의 정적 분석 결과를 만족시킬 수 있도록 유도하였다.

Metric	Over Leak Period	Operator	Warning	Error
Complexity /class	☐	다음보다 큼	10	15
Complexity /file	☐	다음보다 큼	10	15
Complexity /function	☐	다음보다 큼	8	12
Maintainability Rating on New Code	항상	is worse than	A	B
Reliability Rating on New Code	항상	is worse than	A	B
Security Rating on New Code	항상	is worse than	A	B

➔ Class의 Complexity Number 기준을 경고 10, 에러 15로 설정하였다.

➔ File의 Complexity Number 기준을 경고 10, 에러 15로 설정하였다.

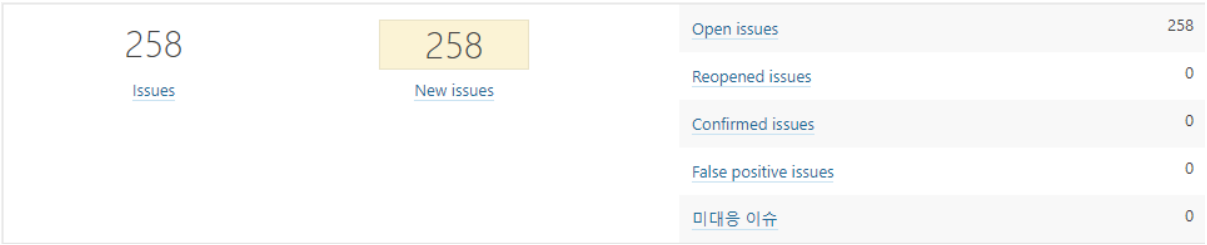
➔ Function의 Complexity Number 기준을 경고 8, 에러 12로 설정하였다.

➔ 유지보수성, 신뢰성, 보안성 등급 기준을 경고 A, 에러 B로 설정하였다.

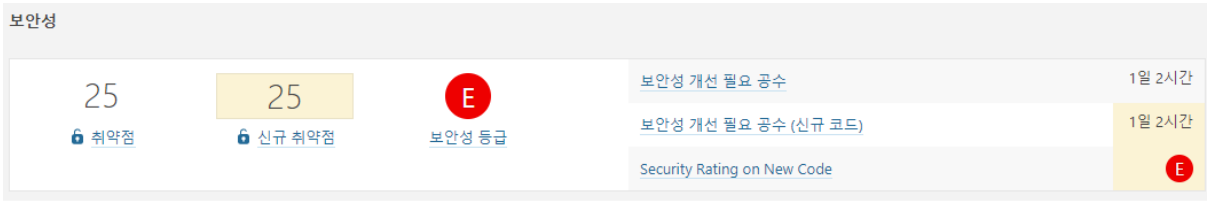
2.2 Sonarqube

- 품질 게이트

➔ Sonarqube가 제공하는 기본 규칙인 'Sonar way'를 적용하여 정적분석을 실행하였고 보안성 및 Complexity 부분에서 품질 게이트에 미달하여 통과하지 못하였다.



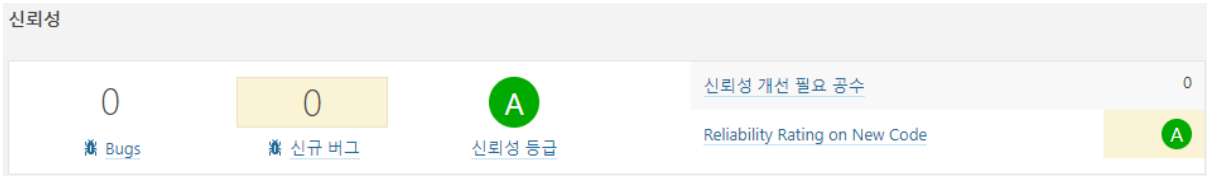
➔ 정적분석 결과 총 258개의 이슈가 발생하였다.



➔ 취약점 부분에서 E등급으로 판정되었고 보안 개선 필요 공수로 1일 2시간이 책정되었다.

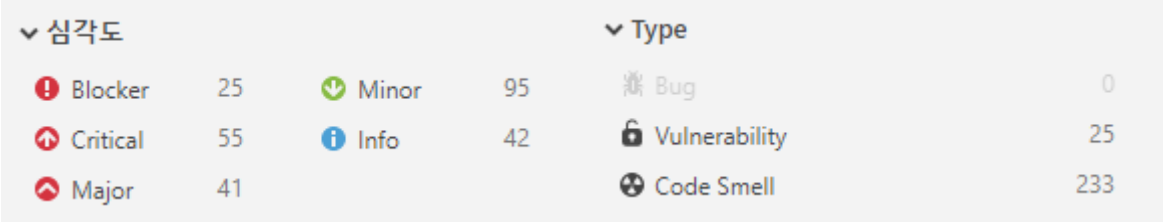
➔ 유지보수성 부분은 총 233개의 코드 악취가 발견되어 등급 B판정을 받아 비교적 양호한 것으로 나타나고 있다.

➔ A등급을 위해 필요 공수로 2일 2시간이 책정되었다.



➔ 신뢰성 부분에서는 0개의 버그가 발견되어 A등급 판정을 받았다.

- 이슈 분석



➔ 발견된 이슈의 종류 분포는 취약점 25개 코드 악취 233개이고 심각도 분포는 Minor 95개, Critical 55개, Major 41개, Blocker 25개 그리고 info 42개이다.

➔ 심각도가 Major, Critical인 이슈들 중 가장 많은 위반이 발견된 주요 3가지 규칙들은 다음과 같다. 자세한 이슈들은 상기된 주소를 통하여 확인할 수 있다.

String literals should not be duplicated

Code Smell Critical design 다음 기간 이후 적용됨 2018년 5월 4일 SonarAnalyzer (Java) 오프셋을 포함한 리니어: 2min +2min per duplicate instance

Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences.

On the other hand, constants can be referenced from many places, but only need to be updated in a single place.

Noncompliant Code Example

With the default threshold of 3:

```
public void run() {
  prepare("action1");                           // Noncompliant - "action1" is duplicated 3 times
  execute("action1");
  release("action1");
}

@SuppressWarning("all")                         // Compliant - annotations are excluded
private void method1() { /* ... */ }
@SuppressWarning("all")
private void method2() { /* ... */ }

public String method3(String a) {
  System.out.println("'" + a + "'");           // Compliant - literal "'" has less than 5 characters and is excluded
  return "";                                    // Compliant - literal "" has less than 5 characters and is excluded
}
```

Compliant Solution

```
private static final String ACTION_1 = "action1";  // Compliant

public void run() {
  prepare(ACTION_1);                            // Compliant
  execute(ACTION_1);
  release(ACTION_1);
}
```

Exceptions

To prevent generating some false-positives, literals having less than 5 characters are excluded.

➔ 해당 이슈는 총 19개의 위반사항이 발견되었다.

➔ 이 이슈와 관련되어 사용 IDE의 문자 인코딩을 UTF-8로 변경할 것을 권고하다.

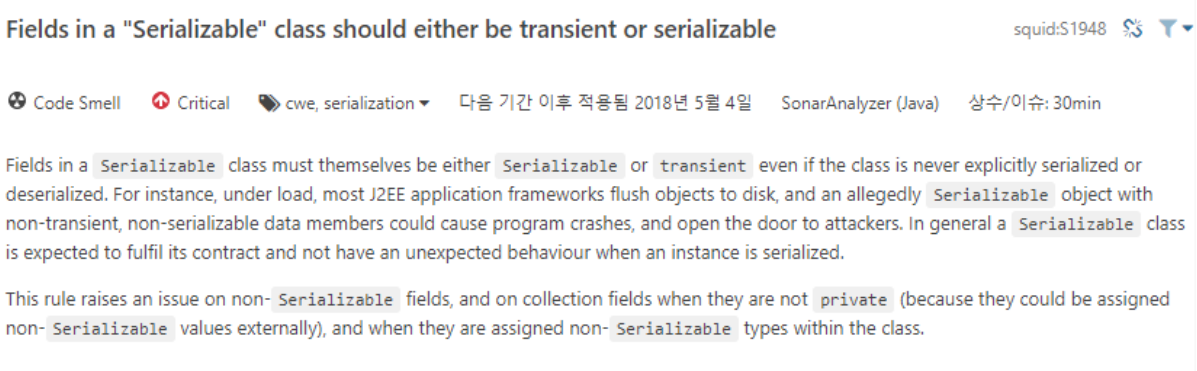
Cognitive Complexity of methods should not be too high

Code Smell Critical brain-overload 다음 기간 이후 적용됨 2018년 5월 4일 SonarAnalyzer (Java) 오프셋을 포함한 리니어: 5min +1min per complexity point over the threshold

Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain.

➔ 해당 이슈는 총 14개의 위반사항이 발견되었다.

➔ 현재 아주 높은 수준의 Complexity number가 측정되므로 빠른 수정이 바람직해 보인다.



Fields in a "Serializable" class should either be transient or serializable squid:S1948

Code Smell | Critical | cwe, serialization | 다음 기간 이후 적용됨 2018년 5월 4일 | SonarAnalyzer (Java) | 상수/이슈: 30min

Fields in a `Serializable` class must themselves be either `Serializable` or `transient` even if the class is never explicitly serialized or deserialized. For instance, under load, most J2EE application frameworks flush objects to disk, and an allegedly `Serializable` object with non-transient, non-serializable data members could cause program crashes, and open the door to attackers. In general a `Serializable` class is expected to fulfil its contract and not have an unexpected behaviour when an instance is serialized.

This rule raises an issue on non-`Serializable` fields, and on collection fields when they are not `private` (because they could be assigned non-`Serializable` values externally), and when they are assigned non-`Serializable` types within the class.

➔ 해당 이슈는 총 11개의 위반사항이 발견되었다.

➔

- 코드 중복 분석

	Duplicated lines (%)	Duplicated lines
src/main/java/sv/test/GUI/inputMoneyFrame.java	82.3%	399
src/main/java/sv/test/GUI/inputUSDFrame.java	75.7%	215
src/main/java/sv/test/GUI/insertFrame.java	73.5%	303
src/main/java/sv/test/GUI/inputAccountFrame.java	69.2%	173
src/main/java/sv/test/GUI/inputPasswordFrame.java	65.7%	287
src/main/java/sv/test/GUI/errorPrintFrame.java	54.2%	39
src/main/java/sv/test/GUI/ReceiptFrame.java	46.4%	39
src/main/java/sv/test/GUI/mainFrame.java	27.0%	41
src/main/java/sv/test/GUI/selectCountry.java	26.2%	27
src/main/java/sv/test/GUI/printLogFrame.java	21.3%	17
src/main/java/sv/test/Junit/systemtest.java	0.0%	0
src/main/java/sv/test/OFFER/Offer.java	0.0%	0
src/main/java/sv/test/ATM/MainSystem.java	0.0%	0
src/main/java/sv/test/Junit/Filetest.java	0.0%	0
src/main/java/sv/test/ATM/Account.java	0.0%	0

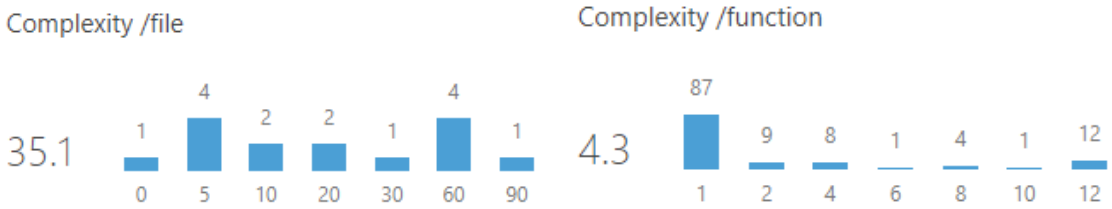
➔ 분석 대상 프로그램의 코드 중복도는 49.7%로 아주 심각한 것으로 나타납니다.

➔ 특히 inputMoneyFrame.java 파일은 전체 코드 중 무려 399라인(82.3%)이 중복되는 것으로 나타나고 있다.

➔ 모든 코드 중복이 GUI관련 부분에서 발견되는 특징이 나타나고 있다.

- Complexity 분석

526 Complexity		
	Complexity /function	4.3
	Complexity /file	35.1
	Complexity /class	29.2
	Cognitive Complexity	709



➔ 분석 대상 프로그램의 전체 Complexity는 526이다.

src/main/java/sv/test/GUI/inputMoneyFrame.java	102.0
src/main/java/sv/test/ATM/MainSystem.java	80.0
src/main/java/sv/test/GUI/inputPasswordFrame.java	76.0
src/main/java/sv/test/GUI/inputUSDFrame.java	66.0
src/main/java/sv/test/GUI/insertFrame.java	60.0
src/main/java/sv/test/GUI/inputAccountFrame.java	40.0
src/main/java/sv/test/ATM/Account.java	23.0
src/main/java/sv/test/GUI/mainFrame.java	22.0
src/main/java/sv/test/GUI/selectCountry.java	15.0
src/main/java/sv/test/Junit/systemtest.java	13.0
src/main/java/sv/test/OFFER/Offer.java	8.0
src/main/java/sv/test/GUI/ReceiptFrame.java	6.0
src/main/java/sv/test/GUI/printLogFrame.java	6.0
src/main/java/sv/test/GUI/errorPrintFrame.java	6.0
src/main/java/sv/test/Junit/Filetest.java	3.0

➔ 파일 Complexity Number를 살펴보면 inputMoneyFram.java에서 무려 102이 기록되었다.

➔ 전반적으로 GUI부분의 Complexity Number가 높지만 MainSystem에서 80점, Account에서 23점이 나오는 등 전반적인 부분에서 아주 높은 점수가 기록되었다.

➔ File Complexity Number가 최소 15이하로 나오도록 품질 게이트 기준을 잡았으므로 총 15개 중 5개를 제외한 10개(약 66%)의 파일에 대한 수정이 필요하다.

src/main/java/sv/test/GUI/inputMoneyFrame.java	14.6
src/main/java/sv/test/GUI/inputUSDFrame.java	11.0
src/main/java/sv/test/GUI/inputPasswordFrame.java	8.4
src/main/java/sv/test/GUI/insertFrame.java	7.5
src/main/java/sv/test/GUI/inputAccountFrame.java	6.7
src/main/java/sv/test/ATM/MainSystem.java	5.7
src/main/java/sv/test/GUI/mainFrame.java	3.7
src/main/java/sv/test/GUI/selectCountry.java	2.5
src/main/java/sv/test/OFFER/Offer.java	2.0
src/main/java/sv/test/Junit/systemtest.java	1.0
src/main/java/sv/test/GUI/ReceiptFrame.java	1.0
src/main/java/sv/test/GUI/printLogFrame.java	1.0
src/main/java/sv/test/Junit/Filetest.java	1.0
src/main/java/sv/test/GUI/errorPrintFrame.java	1.0
src/main/java/sv/test/ATM/Account.java	1.0

➔ 함수 Complexity Number는 inputMoneyFrame.java에서 14.6으로 최고점을 기록했다.

➔ 모든 대상 파일에 대하여 품질 게이트 기준 15미만을 기록하여 양호한 상태이다.

- 코드 Complexity 분석

➔ File, Class, Function 부분 지표에서 독보적으로 가장 높은 Complexity Number를 기록한 inputMoneyFrame.java에 대해 코드 분석을 하였다.

```
if (e.getSource() == b[1]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(1) + "��");
} else if (e.getSource() == b[2]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(2) + "��");
} else if (e.getSource() == b[3]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(3) + "��");
} else if (e.getSource() == b[4]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(4) + "��");
} else if (e.getSource() == b[5]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(5) + "��");
} else if (e.getSource() == b[6]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(6) + "��");
} else if (e.getSource() == b[7]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(7) + "��");
} else if (e.getSource() == b[8]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(8) + "��");
} else if (e.getSource() == b[9]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(9) + "��");
} else if (e.getSource() == b[0]) {
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(0) + "��");
} else if (e.getSource() == bb && money.getText().length() > 1) {
        money.setText(money.getText().substring(0, money.getText().length() - 2) + "��");
```

➔ 끝이 없이 늘어진 if-else문을 볼 수 있다.

```
else if (e.getSource() == b1) {
	if (money.getText().length() > 1) {
		int num = 0;
		for (int k = money.getText().length() - 2; money.getText().charAt(k) == '0'; k--) {
			num++;
		}
		if (num > 4) {
			money.setText(money.getText().substring(0, money.getText().length() - num + 3) + "��");
		} else {
			for (num = 4 - num; num > 0; num--)
				money.setText(money.getText().substring(0, money.getText().length() - 1) + "0��");
		}
	} else
		money.setText("10000��");
else if (e.getSource() == b10) {
	if (money.getText().length() > 1) {
		int num = 0;
		for (int k = money.getText().length() - 2; money.getText().charAt(k) == '0'; k--) {
			num++;
		}
		if (num > 5) {
			money.setText(money.getText().substring(0, money.getText().length() - num + 4) + "��");
		} else {
			for (num = 5 - num; num > 0; num--)
				money.setText(money.getText().substring(0, money.getText().length() - 1) + "0��");
		}
	} else
```

➔ 심지어 동일 if-else문 중간 부분에는 3중 중첩된 if-else가 존재하며 2단계 if문에서 for문을 돌고 이후 3단계 if-else에 돌입하여 다시 for문을 도는 엄청난 depth의 중첩문이 존재한다.

```
	} else {
		switch (menu) {
		case 0: // ◆U◆
			if (money.getText().length() > 5 && money.getText()
					.substring(money.getText().length() - 5, money.getText().length() - 1).equals("0000")) {
				if ((error = main.deposit(
						Integer.parseInt(money.getText().substring(0, money.getText().length() - 1)))) == 0) {
					new ReceiptFrame(main);
				} else {
					new errorPrintFrame(error);
				}
				this.dispose();
			} else {
				state.setText("10000◆◆ ◆◆◆◆◆θ◆ ◆U◆◆◆◆◆Uθ◆.");
				state.setOpaque(true);
				state.setBackground(Color.LIGHT_GRAY);
			}
			break;
		case 1: // ◆◆◆◆◆◆◆U◆
			if (money.getText().length() > 5 && (money.getText()
					.substring(money.getText().length() - 5, money.getText().length() - 1).equals("0000"))) {
				if ((error = main.depositWithoutBank(
						Integer.parseInt(money.getText().substring(0, money.getText().length() - 1)))) == 0) {
					new ReceiptFrame(main);
				} else {
					new errorPrintFrame(error);
				}
				this.dispose();
			} else {
				state.setText("10000◆◆ ◆◆◆◆◆θ◆ ◆U◆◆◆◆◆Uθ◆.");
				state.setOpaque(true);
				state.setBackground(Color.LIGHT_GRAY);
			}
			break;
		case 2: // ◆◆U
			main.bank = Integer.parseInt(money.getText().substring(0, money.getText().length() - 1));
			new inputPasswordFrame(4, main, account);
			this.dispose();
			break;
		case 3: // ◆◆◆
```

➔ 동일 if-else 말미에는 switch문까지 존재한다. Switch문 내부에도 중첩된 if-else가 존재한다.

```
case '\b':
        if (money.getText().length() > 1)
                money.setText(money.getText().substring(0, money.getText().length() - 2) + "��");
        break;
case 'q':
        if (money.getText().length() > 1) {
                int num = 0;
                for (int k = money.getText().length() - 2; money.getText().charAt(k) == '0'; k--) {
                        num++;
                }
                if (num > 4) {
                        money.setText(money.getText().substring(0, money.getText().length() - num + 3) + "��");
                } else {
                        for (num = 4 - num; num > 0; num--)
                                money.setText(money.getText().substring(0, money.getText().length() - 1) + "0��");
                }
        } else
                money.setText("10000��");
        break;
case 'w':
        if (money.getText().length() > 1) {
                int num = 0;
                for (int k = money.getText().length() - 2; money.getText().charAt(k) == '0'; k--) {
                        num++;
                }
                if (num > 5) {
                        money.setText(money.getText().substring(0, money.getText().length() - num + 4) + "��");
                } else {
                        for (num = 5 - num; num > 0; num--)
                                money.setText(money.getText().substring(0, money.getText().length() - 1) + "0��");
                }
        } else
                money.setText("100000��");
```

➔ 키보드 입력을 처리하는 메소드 내부에 존재하는 switch문의 상태이다.

➔ Switch문 안에 2중 중첩 if-else가 존재하며 그 내부에서 for문이 최대 2번 돌게 되어있다.

➔ 해당 파일 내부에 이벤트, 키 처리 부분에 중첩된 분기문과 반복문이 너무 많은 것으로 판단되며 이것이 높은 Complexity Number의 원인으로 생각됩니다.

- 코드 중복 분석

➔ 코드 중복률 82.3%라는 엄청난 지표를 나타낸 inputMoneyFrame.java를 분석해보았다.

```
add(b[0], 1, 6, 1, 1);
add(b[1], 0, 3, 1, 1);
add(b[2], 1, 3, 1, 1);
add(b[3], 2, 3, 1, 1);
add(b[4], 0, 4, 1, 1);
add(b[5], 1, 4, 1, 1);
add(b[6], 2, 4, 1, 1);
add(b[7], 0, 5, 1, 1);
add(b[8], 1, 5, 1, 1);
add(b[9], 2, 5, 1, 1);
```

```
add(b[0], 1, 6, 1, 1);
add(b[1], 0, 3, 1, 1);
add(b[2], 1, 3, 1, 1);
add(b[3], 2, 3, 1, 1);
add(b[4], 0, 4, 1, 1);
add(b[5], 1, 4, 1, 1);
add(b[6], 2, 4, 1, 1);
add(b[7], 0, 5, 1, 1);
add(b[8], 1, 5, 1, 1);
add(b[9], 2, 5, 1, 1);
```

➔ 동일 파일의 다른 부분에 존재하는 코드이다. 완전히 동일한 것을 알 수 있다.

➔ 생성자를 오버로딩한 것인데 높은 코드 중복률에 기여하고 있다.

```
case '0':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(0) + "원");
        break;
case '1':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(1) + "원");
        break;
case '2':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(2) + "원");
        break;
case '3':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(3) + "원");
        break;
case '4':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(4) + "원");
        break;
case '5':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(5) + "원");
        break;
case '6':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(6) + "원");
        break;
case '7':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(7) + "원");
        break;
case '8':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(8) + "원");
        break;
case '9':
        money.setText(money.getText().substring(0, money.getText().length() - 1) + String.valueOf(9) + "원");
```

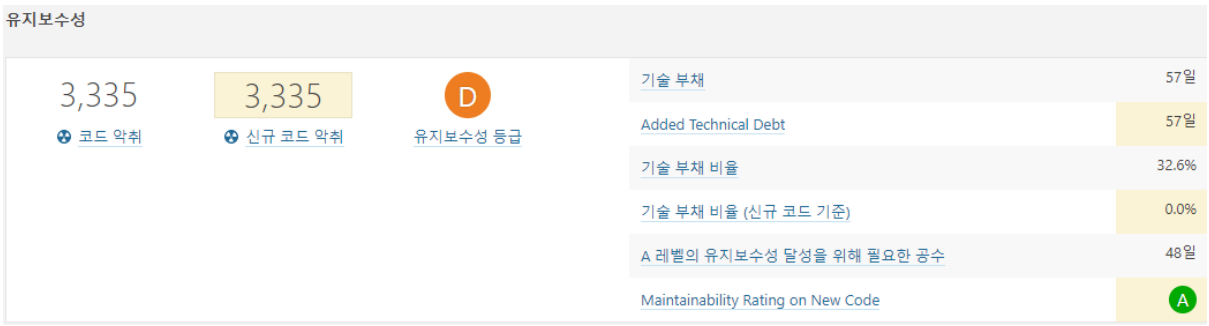
➔ 같은 파일내에 존재하는 case문으로 거의 동일한 형태의 코드가 반복되고 있다.

➔ 'String.valueOf'의 parameter상수를 변수로 바꾸어 중복을 줄이는 것이 좋아 보인다.

➔ 동일 문제가 파일 여러 곳에서 지속적으로 발견되는 것으로 미루어 보아 해당 부분이 높은 코드 중복율의 원인으로 파악됩니다.

➔ 코드 분석 결과를 다른 파일에도 적용하여 전체 프로그램을 개선할 수 있다.

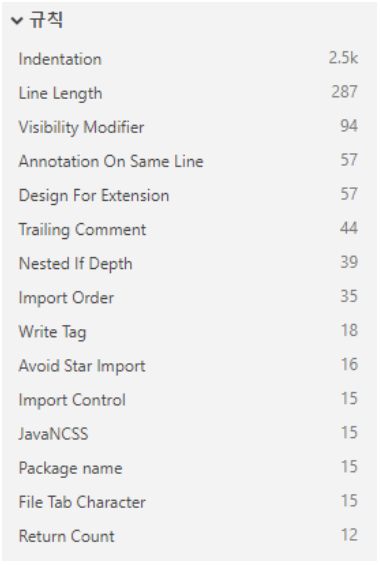
2.3 Checkstyle

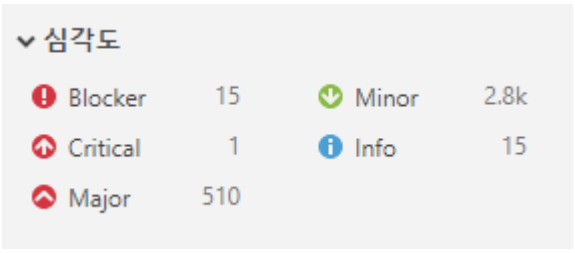


➔ Checkstyle의 코딩 규칙을 적용하여 정적분석을 진행하였고 D등급 판정과 총 3335개의 코드 악취가 발견되었다

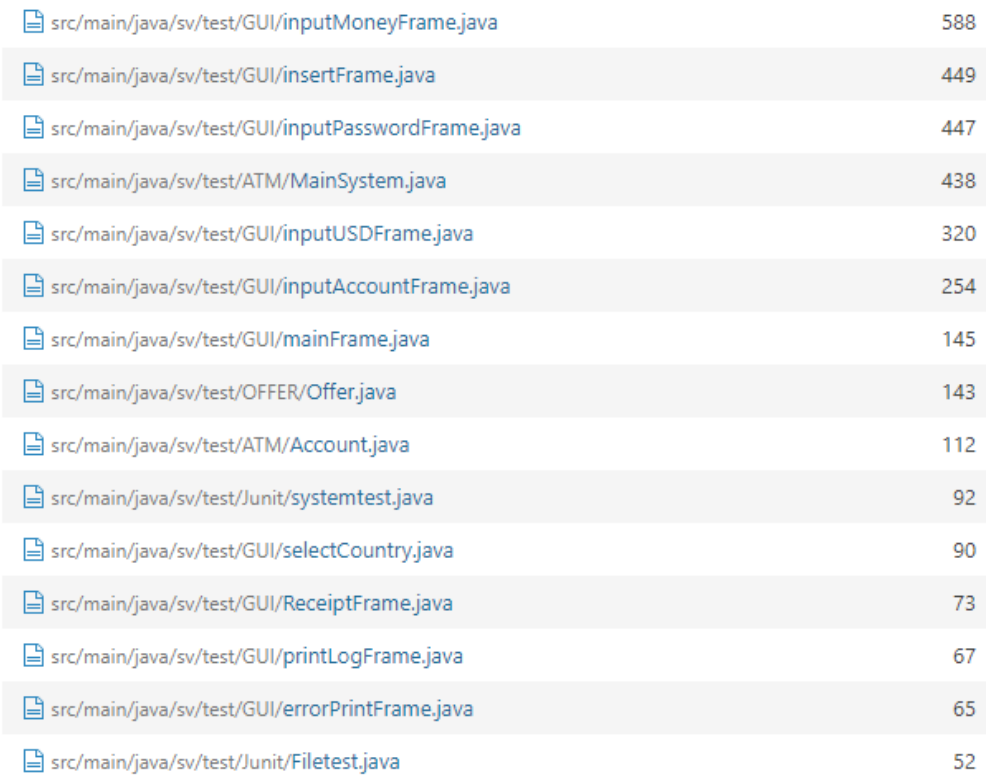
➔ A등급을 위한 필요 공수로 48일이 책정되었다.

- 위반 규칙 및 심각도





➔ Indentation관련 이슈가 약 2500개로 가장 많이 발생하였고 심각도가 Major인 이슈가 무려 510개나 발생하였다.



➔ 파일 별 이슈 개수를 살펴보면 역시 inputMoneyFrame.java에서 588개로 가장 많은 이슈가 발생한 것을 볼 수 있다.

➔ 전반적으로 GUI부분의 이슈가 많으며 MainSystem 또한 이슈가 상당히 많이 발생하였다.

- Critical 이슈

➔ 발견 이슈가 너무 많아 주요 이슈만 다루고, 상세 내용은 상기된 링크를 통해 확인 가능하다.

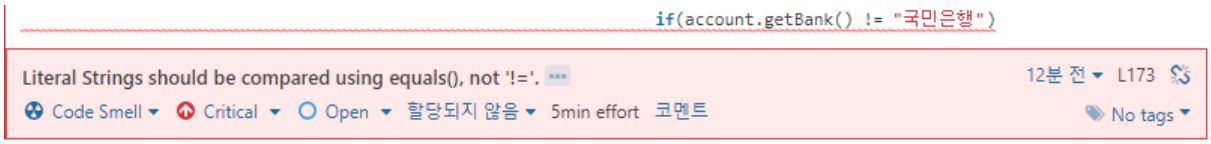
String Literal Equality

Code Smell Critical 태그 없음 다음 기간 이후 적용됨 2018년 5월 4일 Checkstyle (Java) 상수/이슈: 5min

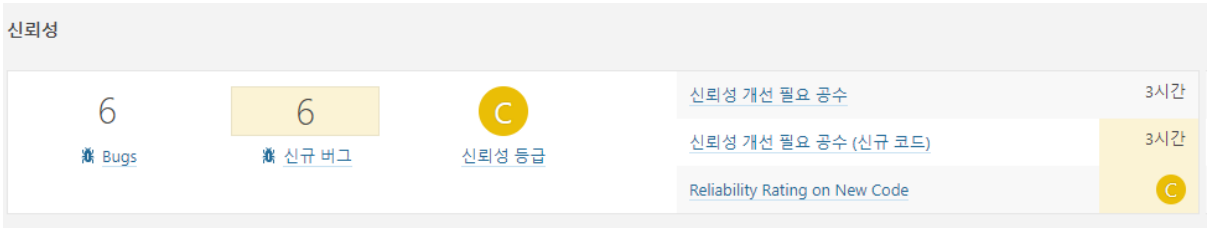
Checks that string literals are not used with == or !=.

➔ String 문자열을 equals()가 아닌 '!='을 사용하여 비교하였다. 함수 equals()는 속 내용을 비교하지만 '=='이나 '!='는 주소 값을 비교하기 때문에 잘못된 결과가 나올 수 있다.

➔ 실제 코드 모습



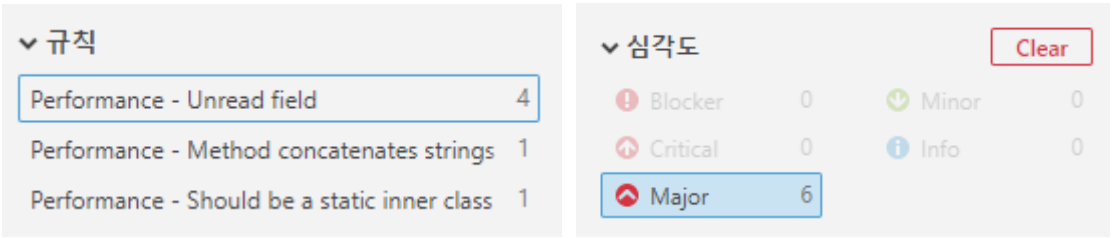
2.4 Findbugs



➔ Findbugs 코딩 규칙을 적용하여 정적분석을 진행하였고 신뢰성 C등급과 총 6개의 버그가 발견되었다.

➔ 신뢰성 개선을 위한 공수로 3시간이 책정되었다.

- 위반 규칙 및 심각도



➔ 총 6개의 버그가 발견되었는데 모드 Major 등급의 이슈로 판정되었다.

➔ 위반한 규칙 3가지 모두 성능과 관련된 이슈로 나타나고 있다.

- 위반 규칙

Performance - Unread field

Bug Major performance 다음 기간 이후 적용됨 2018년 5월 4일 FindBugs (Java) 상수/이슈: 30min

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

➔ 해당 이슈는 총 4개의 위반사항이 발견되었다.

➔ 절대 실행되지 않는 코드가 있으므로 로직에 대한 체크가 필요하다.

Performance - Method concatenates strings using + in a loop

Bug Major performance 다음 기간 이후 적용됨 2018년 5월 4일 FindBugs (Java) 상수/이슈: 30min

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();
```

➔ 해당 이슈는 총 1개의 위반사항이 발견되었다.

➔ 문자열을 '+'를 통해 연결하는 것은 느리기 때문에 'StringBuffer' 를 활용하는 것이 더 나은 성능을 보인다.

Performance - Should be a static inner class

findbugs:SIC_INNER_SHOULD_BE_STATIC

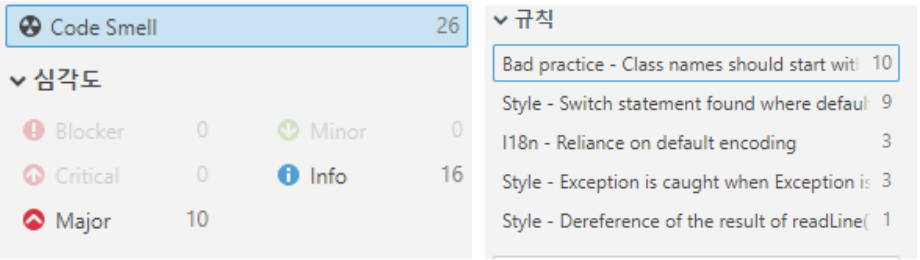
Bug Major performance 다음 기간 이후 적용됨 2018년 5월 4일 FindBugs (Java) 상수/이슈: 30min

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.

설명 추가

➔ 해당 이슈는 총 1개의 위반사항이 발견되었다.

- Findbugs 유지보수성 규칙 위반 현황



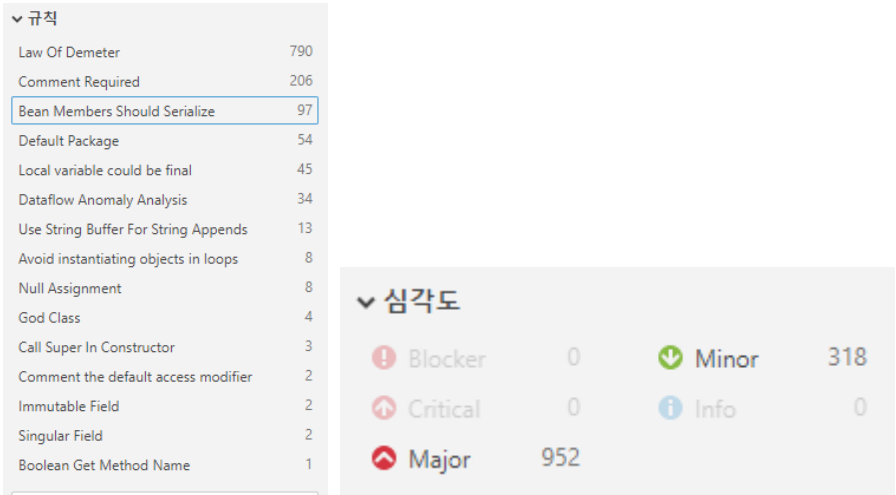
➔ Findbugs 코딩 규칙 중 유지보수성 관련 부분에서 총 26개의 이슈가 발생하였고, 그 중 총 10개의 Major등급 이슈가 발생하였다.

2.5 PMD

➔ PMD 코딩 규칙을 적용하여 정적분석을 진행하였고 유지보수성 D등급과 총 1270개의 코드 악취가 발견되었다.

➔ A등급을 위한 필요 공수로 54일이 책정되었다.

- 위반 규칙 및 심각도



➔ 1207개의 이슈 중 Major 952개, Minor 318개의 심각도 분포를 나타내고 있다.

src/main/java/sv/test/GUI/inputMoneyFrame.java	355
src/main/java/sv/test/GUI/inputUSDFrame.java	252
src/main/java/sv/test/ATM/MainSystem.java	175
src/main/java/sv/test/GUI/inputPasswordFrame.java	111
src/main/java/sv/test/GUI/insertFrame.java	76
src/main/java/sv/test/GUI/inputAccountFrame.java	51
src/main/java/sv/test/OFFER/Offer.java	44
src/main/java/sv/test/Junit/systemtest.java	43
src/main/java/sv/test/GUI/selectCountry.java	35
src/main/java/sv/test/ATM/Account.java	34
src/main/java/sv/test/GUI/ReceiptFrame.java	22
src/main/java/sv/test/GUI/printLogFrame.java	22
src/main/java/sv/test/GUI/errorPrintFrame.java	20
src/main/java/sv/test/GUI/mainFrame.java	19
src/main/java/sv/test/Junit/Filetest.java	11

➔ 파일 별 개수를 보면 GUI관련 inputMoneyFrame.java에서 제일 많은 이슈가 발생하였다.

➔ Major 등급 이슈 중 가장 많은 위반 사항이 발견된 주요 3개 규칙은 다음과 같다.

Law Of Demeter pmd:LawOfDemeter

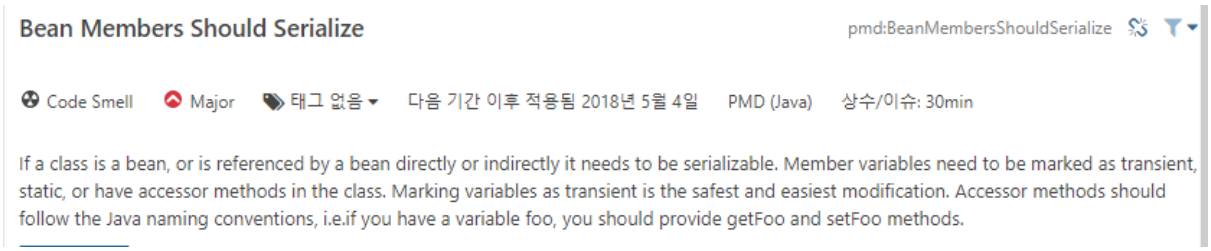
Code Smell Major 태그 없음 다음 기간 이후 적용됨 2018년 5월 4일 PMD (Java) 상수/이슈: 30min

The Law of Demeter is a simple rule, that says "only talk to friends". It helps to reduce coupling between classes or objects. See also the references: Andrew Hunt, David Thomas, and Ward Cunningham. The Pragmatic Programmer. From Journeyman to Master. Addison-Wesley Longman, Amsterdam, October 1999.; K.J. Lieberherr and I.M. Holland. Assuring good style for object-oriented programs. Software, IEEE, 6(5):38–48, 1989.; http://www.ccs.neu.edu/home/lieber/LoD.html; http://en.wikipedia.org/wiki/Law_of_Demeter

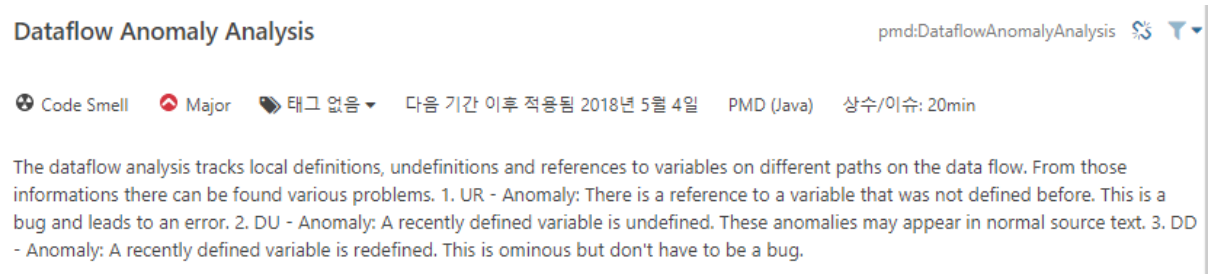
Example:

```
public class Foo {
    /**
     * This example will result in two violations.
     */
    public void example(Bar b) {
        // this method call is ok, as b is a parameter of "example"
        C c = b.getC();

        // this method call is a violation, as we are using c, which we got from B.
        // We should ask b directly instead, e.g. "b.doItOnC();"
        c.doIt();

        // this is also a violation, just expressed differently as a method chain without temporary variables.
        b.getC().doIt();

        // a constructor call, not a method call.
        D d = new D();
        // this method call is ok, because we have create the new instance of D locally.
        d.doSomethingElse();
    }
}
```

➔ 해당 이슈는 총 790개의 위반사항이 발견되었다.



➔ 해당 이슈는 총 97개의 위반사항이 발견되었다.



➔ 해당 이슈는 총 34개의 위반사항이 발견되었다.

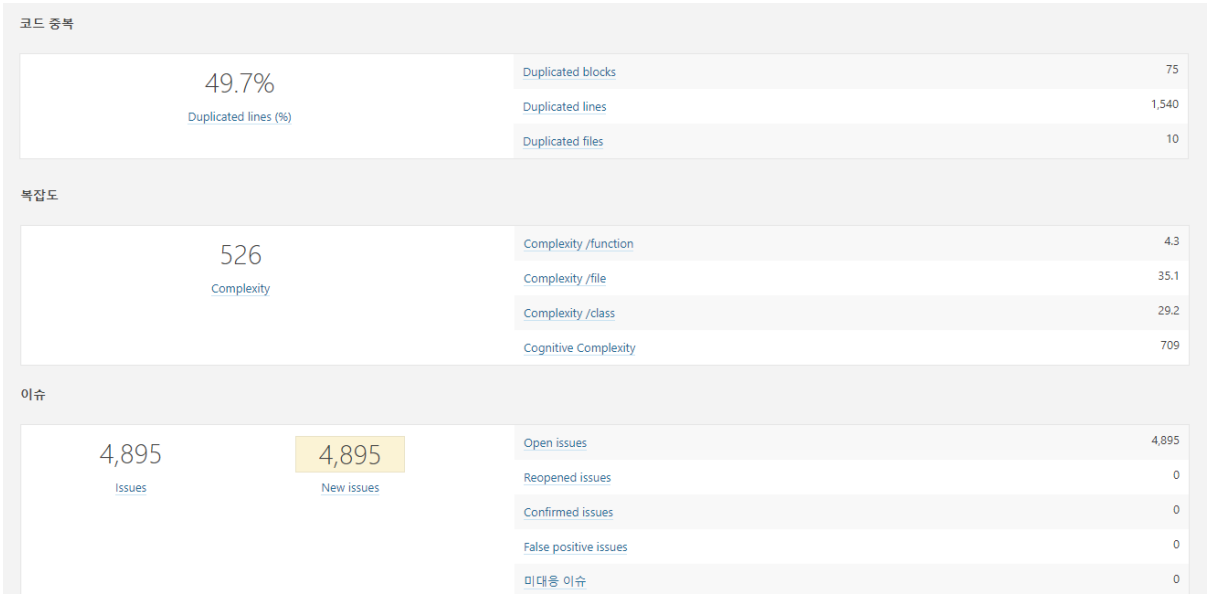
➔ Law of Demeter 규칙의 경우 잘 지키면 coupling을 낮출 수 있어 유용하다.

3. Sonarqube 종합 결과

3.1 테스트 결과

➔ 진행한 모든 정적 분석을 종합한 결과 총 6개의 버그와 25개의 취약점 그리고 4864개의 코드 악취가 발견되었다.

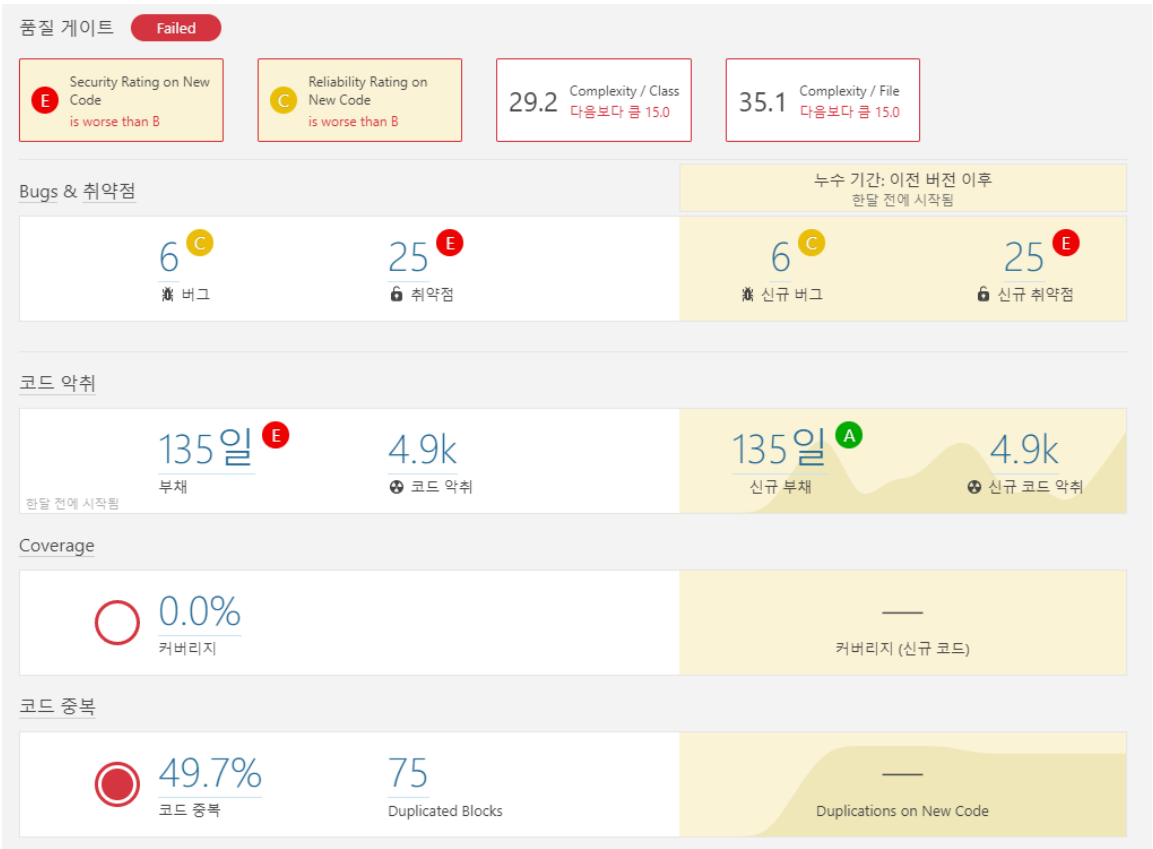
➔ 종합 결과 신뢰성 등급은 C등급으로 보안성 등급은 E등급으로 판정되었고, 유지보수성 등급 또한 E등급으로 판정되었다.



➔ 또한 중복된 코드가 1540라인(49.7%) 발견되었고, 복잡도가 526으로 측정되었다.

➔ 최종 발생한 이슈는 총 4895개이다.

3.2 품질 게이트



➔ 적용한 품질 게이트 기준에 보안성, 신뢰성, Complexity 부분에서 미달하여 품질 게이트를 달성하지 못하였다.

3.3 Unit test coverage

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
DSLAB2018_T6-master	10.6 %	1,310	10,996	12,306
src/main/java	10.6 %	1,310	10,996	12,306
GUI	0.0 %	0	9,571	9,571
ATM	42.9 %	801	1,064	1,865
OFFER	32.6 %	163	337	500
Junit	93.5 %	346	24	370

➔ 유닛 테스트의 Coverage가 종합 10.6%로 매우 낮은 수치를 기록하고 있다.

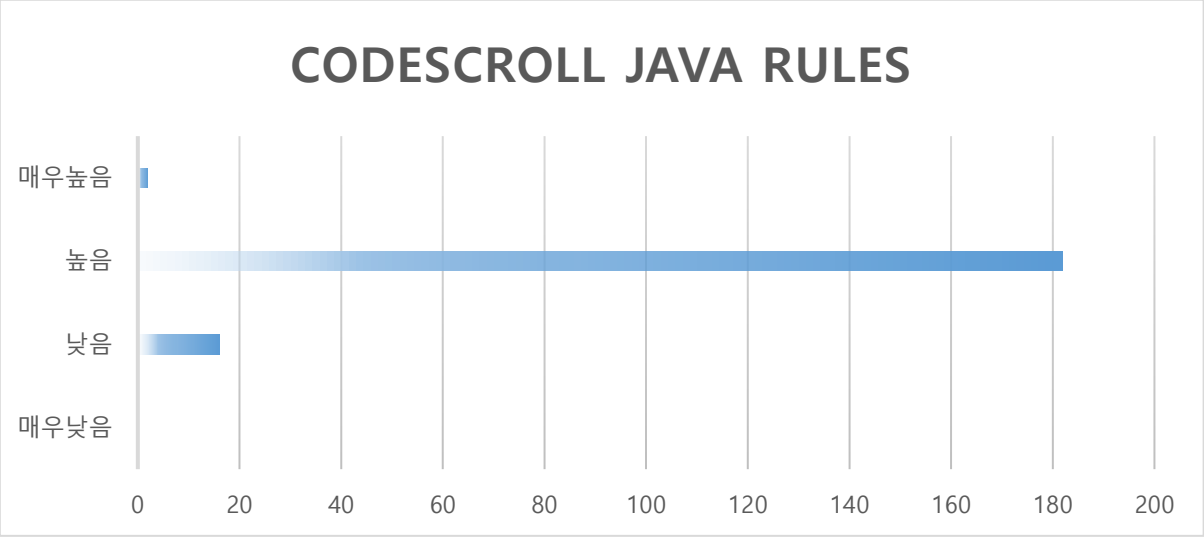
4. Codescroll inspector for JAVA

4.1 Codescroll inspector를 이용한 정적 분석

진단 메시지	규칙	심각도	소속	개수
체인형 메서드 호출이 올바른 형태가 아님	VNA04-J	낮음	inputUSDFrame.actionPerformed(ActionEvent)	92
괄호를 사용하지 않고 여러 연산자를 사용하여 수식을혼용함.	Sun_26	낮음	insertFrame.actionPerformed(ActionEvent)	6
숫자 상수를 사용함(for loop의 counter로 -1이나 0, 1을 사용하는 경우 제외)	Sun_24	매우 낮음	selectCountry.keyTyped(KeyEvent)	626
non-static 필드가 public으로 선언.	Sun_22	매우 낮음	MainSystem	4
클래스의 이름(selectCountry)이 규칙(^[A-Z]([a-z]|[A-Z]|[0-9])*)에 맞지 않음.	Sun_18	매우 낮음	GUI	13
지역 변수 선언문과 실행 문장 사이에 빈 줄이 없음.	Sun_17	매우 낮음	selectCountry.selectCountry(MainSystem)	20
Switch 문에 default case가 존재하지 않음.	Sun_13	매우 낮음	insertFrame.keyTyped(KeyEvent)	11
if 문에서 중괄호가 사용되지 않음.	Sun_11	매우 낮음	MainSystem.transfer(int, Account)	16
지역 변수(error) 선언 시 초기화를 하지 않음.	Sun_10	낮음	insertFrame.actionPerformed(ActionEvent)	20
identifier(bank)가 이미 필드로 선언되어 있어 hide가 발생함	Sun_09	낮음	MainSystem.payUtilityBill(Account)	10
블록의 시작 부분이 아닌 곳에 선언함	Sun_08	매우 낮음	selectCountry.selectCountry(MainSystem)	32
한 줄에 여러 식별자를 선언함	Sun_07	매우 낮음	insertFrame	10
라인의 길이가 (80)을 넘음	Sun_06	매우 낮음	Offer.Offer(int)	161
해당 소스 파일이 C 스타일 주석으로 시작하지 않음	Sun_03	매우 낮음	GUI	15
mutable 클래스가 신뢰할 수 없는 코드에 객체를 안전하게 전달하기 위한 복사 기능을 제공	OBJ04-J	낮음	GUI	18
non-static, non-final인 필드가 private으로 선언되지 않음.	OBJ01-J	높음	selectCountry	58
생성자 내에서 final이나 private이 아닌 메서드를 호출함.	MET05-J	높음	inputPasswordFrame.inputPasswordFrame(int, MainSystem)	221
메서드의 설명 주석이 없거나, 설명 주석 내용에 모든 태그가 포함되어있지 않음	JAVA_72	높음	Offer	122
클래스의 설명 주석이 없거나, 설명 주석 내용에 모든 태그가 포함되어있지 않음	JAVA_71	높음	OFFER	15
identifier(bank)가 이미 필드로 선언되어 있어 hide가 발생함	JAVA_70	높음	MainSystem.payUtilityBill(Account)	10
*를 이용한 import문을 사용함.	JAVA_68	낮음	GUI	16
condition에 직접적인 assignment operator를 사용함.	JAVA_66	매우 높음	MainSystem.exchange(int, String)	1
반복문 안에서 String concatenation 연산자를 사용함.	JAVA_64	높음	Offer.readDatabase(Account)	13
반복문 안에서 synchronized 메서드(addKeyListener)를 호출함.	JAVA_62	높음	inputPasswordFrame.inputPasswordFrame(int, MainSystem)	8
Switch 문에 default case가 존재하지 않음.	JAVA_61	높음	inputPasswordFrame.selectmenu()	11
static, local, anonymous가 아닌 내부 클래스가 사용됨.	JAVA_49	높음	inputAccountFrame	3
String 비교에 "==" 또는 "!=" 을 사용함. equals 메서드를 사용하는것을 권장한다.	JAVA_07	매우 높음	MainSystem.transfer(int, Account)	1
(Exception)을 catch함.	ERR08-J	높음	Offer.checkValid(Account)	3
적절한 로그를 사용하지 않음.	ERR02-J	높음	Offer.checkValid(Account)	2

➔ codescroll_java_rules cert_secure_codeing_standard sun_code_conventions_for_java 3개의 룰을 위배사항으로 지정하여 codescroll inspector사용하여 정적분석을 진행하여 총 1538개의 위반사항이 발견되었다.

4.2 Codescroll_java_rules 위배 현황



➔총 200개의 위반사항이 발견되었으며 java_72 규칙이 122회로 가장 많이 위배되었다.

4.2.1 규칙 위반 상세 사항

➔ 위배 사항 중 중요 사항에 대해서 본 문서에서 심층 분석하였다. 기타 상세 위배 사항은 별첨 report 파일을 통해 확인할 수 있다.

규칙	심각도	진단 메시지
JAVA_07	매우 높음	String 비교에 "==" 또는 "!=" 을 사용함. equals 메서드를 사용하는 것을 권장한다.

```
newLog = newLog + newAccount.getLog();
newAccount.setLog(newLog);
if (!offer[bank].updateDatabase(newAccount)) {
    return 2;
}
offer[bank].readDatabase(account);
if(account.getBank() != "신한은행")
    takeCharge(account);
account.setBalance(account.getBalance() - money);
newLog = new Date() + " " + money + " ••" + newAccount.ge
        + newAccount.getAccountNumber() + "\t( ••: " + ac
newLog = newLog + account.getLog();
account.setLog(newLog);
if (offer[bank].updateDatabase(account)) {
    return 0;
```

규칙	심각도	진단 메시지
JAVA_49	높음	static, local, anonymous가 아닌 내부 클래스가 사용됨.

```
}

class myKeyListener implements KeyListener {


    public void keyPressed(KeyEvent arg0) {
        // TODO Auto-generated method stub

    }


    public void keyReleased(KeyEvent arg0) {
        // TODO Auto-generated method stub

    }
```

규칙	심각도	진단 메시지
JAVA_61	높음	Switch 문에 default case가 존재하지 않음.

```
public void selectmenu() {
    int error;
    switch (menu) {
    case 0: // withdraw
        if ((error = main.withdraw(main.bank)) == 0) {
            new ReceiptFrame(main);
        } else {
            new errorPrintFrame(error);
        }
        this.dispose();
        break;
    case 1: // exchange
        if ((error = main.exchange(main.bank, main.card)) == 0) {
            new ReceiptFrame(main);
            this.dispose();
        } else {
            new errorPrintFrame(error);
```

규칙	심각도	진단 메시지
JAVA_62	높음	반복문 안에서 synchronized 메서드(addKeyListener)를 호출함.

```
add(money, 0, 1, 5, 1);
b = new JButton[10];
for (int i = 0; i < 10; i++) {
    b[i] = new JButton(String.valueOf(i));
    b[i].addActionListener(this);
    b[i].addKeyListener(this);
}
add(b[0], 1, 6, 1, 1);
add(b[1], 0, 3, 1, 1);
add(b[2], 1, 3, 1, 1);
add(b[3], 2, 3, 1, 1);
```

규칙	심각도	진단 메시지
JAVA_64	높음	반복문 안에서 String concatenation 연산자를 사용함.

```
    direc = "./▯▯▯/" + currentAccount.getBank() + "/" +currentAccount.getAccountNumber() + "/▯▯▯.txt";
    this.input = new BufferedReader(new FileReader(direc));
    String log = "";
    String tempLog;
    while((tempLog = this.input.readLine()) != null) {
        log = log + tempLog + "\n";
    }
    currentAccount.setLog(log);
    this.input.close();
} catch(Exception e) {
    System.out.println("error 1 : [requestInformationFromServer] load information failed");
    return 1;
```

규칙	심각도	진단 메시지
JAVA_66	매우 높음	condition에 직접적인 assignment operator를 사용함.

```
            break;
        }
        if (this.account.getBalance() < (won = money * exchangeRate[i])) {
            return 1;
        } else {
            account.setBalance(account.getBalance() - won);
```

규칙	심각도	진단 메시지
JAVA_68	낮음	*를 이용한 import문을 사용함.

```
import ATM.MainSystem;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
```

규칙	심각도	진단 메시지
JAVA_70	높음	identifier(bank)가 이미 필드로 선언되어 있어 hide가 발생함

```
    public int payUtilityBill(Ac
        String[] list_bank = { "
        int bank;
        for(bank = 0; bank < list
            if(list_bank[bank].ec
                break;
        }
```

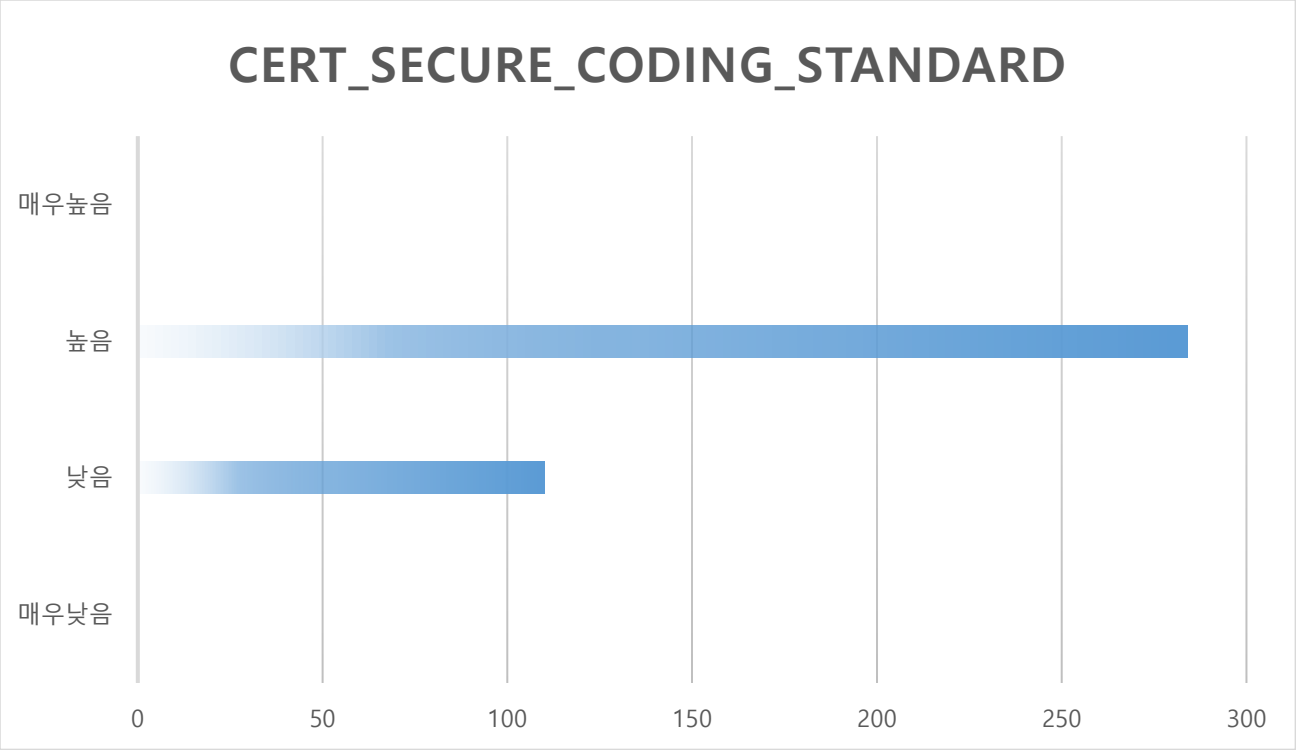
규칙	심각도	진단 메시지
JAVA_71	높음	클래스의 설명 주석이 없거나, 설명 주석 내용에 모든 태그가 포함되어 있지 않음

```
import java.io.BufferedReader;

public class Offer {
    String company;
    private BufferedReader input;
    private PrintWriter output;
```

규칙	심각도	진단 메시지
JAVA_72	높음	메서드의 설명 주석이 없거나, 설명 주석 내용에 모든 태그가 포함되어 있지 않음

```
    public boolean updateDatabase(Account newAccount) {
        try {
            String direc;
            boolean card = newAccount.getBank().substring(2, 4).equal
```

4.3 cert_secure_coding_standard 위배 현황



➔총 394개의 위반사항이 발견되었으며 MET05-J규칙이 211회로 가장 많이 위배되었다.

4.3.1 규칙 위반 상세 사항

규칙	심각도	진단 메시지
VNA04-J	낮음	체인형 메서드 호출이 올바른 형태가 아님

```
public void actionPerformed(ActionEvent e) {
    // TODO Auto-generated method stub
    if (e.getSource() == b[1]) {
        money.setText(money.getText().substring(0, money.getText().length() - 3) + String.valueOf(1) + str);
    } else if (e.getSource() == b[2]) {
        money.setText(money.getText().substring(0, money.getText().length() - 3) + String.valueOf(2) + str);
    } else if (e.getSource() == b[3]) {
```

규칙	심각도	진단 메시지
OBJ01-J	높음	non-static, non-final인 필드가 private으로 선언되지 않음.

```
public class selectCountry extends JFrame implements ActionListener, KeyListener {
    MainSystem main;
    JButton[] country;
    GridBagLayout gbl;
    GridBagConstraints gbc;
```

규칙	심각도	진단 메시지
OBJ04-J	낮음	mutable 클래스가 신뢰할 수 없는 코드에 객체를 안전하게 전달하기 위한 복사 기능을 제공하지 않음.

```
public class selectCountry extends JFrame implements ActionListener, KeyListener {
    MainSystem main;
    JButton[] country;
    GridBagLayout gbl;
    GridBagConstraints gbc;
```

규칙	심각도	진단 메시지
ERR02-J	높음	적절한 로그를 사용하지 않음.

```
        }
        newAccount.setBalance(newAccount.getBalance() + money);
        String tempLog = new Date() + " " + money + " ₩" + account.getBank() + " "
                + account.getAccountNumber() + "\t( ₩ : " + newAccount.getBalance() + " )\n";
        System.out.println(tempLog);
        tempLog = tempLog + newAccount.getLog();
        newAccount.setLog(tempLog);
        if (!offer[temp_bank].updateDatabase(newAccount)) {
            return 2;
        }
```

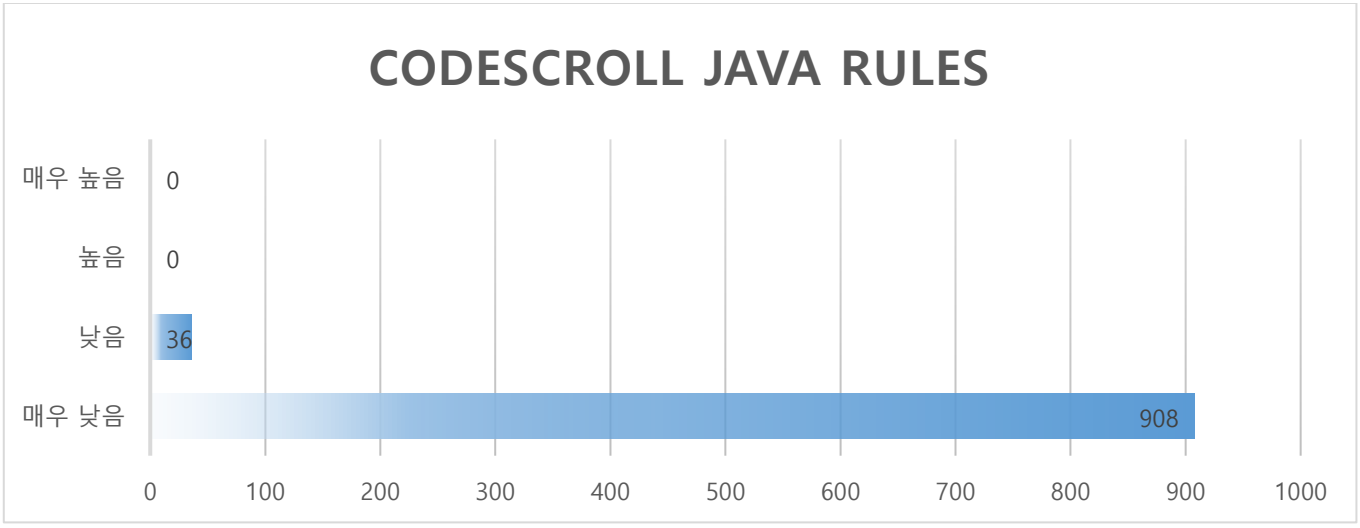
규칙	심각도	진단 메시지
MET05-J	높음	생성자 내에서 final이나 private이 아닌 메서드를 호출함.

```
        add(bb, 0, 6, 1, 1);
        add(reset, 2, 6, 1, 1);
        this.setSize(500, 500);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

규칙	심각도	진단 메시지
ERR08-J	높음	(Exception)을 catch함.

```
public boolean checkValid(Account account) {
    try {
        String direc = "./ddd/" + account.getBank() + "/" +account
        System.out.println(direc);
        this.input = new BufferedReader(new FileReader(direc));
        this.input.close();
        System.out.println("valid input");
    } catch(Exception e) {
        System.out.println("invalid input");
        return false;
    }
    return true;
```

4.4 sun_code_conventions_for_java 위배 현황



➔총 944개의 위반사항이 발견되었으며 Sun_24 규칙이 가장 많이 위배되었다.

4.4.1 규칙 위반 상세 사항

규칙	심각도	진단 메시지
Sun_03	매우 낮음	해당 소스 파일이 C 스타일 주석으로 시작하지 않음

```
1  package GUI;
2
3  import ATM.MainSystem;
4  import java.awt.*;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.KeyEvent;
8  import java.awt.event.KeyListener;
9
```

규칙	심각도	진단 메시지
Sun_06	매우 낮음	라인의 길이가 (80)을 넘음

```
                    break;
                case 4: // ****
                    if (money.getText().length() > 5 && (money.getText()
                            .substring(money.getText().length() - 5, money.getText().length() - 1).equals("0000"))) {
                        main.bank = Integer.parseInt(money.getText().substring(0, money.getText().length() - 1));
                        new inputPasswordFrame(2, main);
                        this.dispose();
                    } else {
```

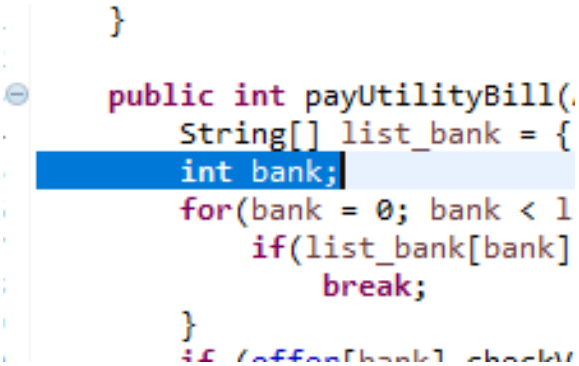
규칙	심각도	진단 메시지
Sun_07	매우 낮음	한 줄에 여러 식별자를 선언함

```
public class insertFrame extends JFrame implements ActionListener, KeyListener {
    JTextField tf;
    JButton ok, b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, bb, reset;
    JLabel state;
    JComboBox<String> combo;
    MainSystem main, temp;
    int menu;
    GridBagLayout gbl;
    GridBagConstraints gbc;

    public insertFrame(int menu) {
        super("insert");
```

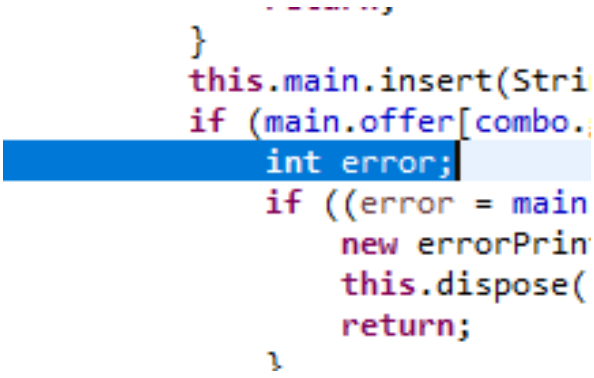
규칙	심각도	진단 메시지
Sun_08	매우 낮음	블록의 시작 부분이 아닌 곳에 선언함

```
            country[i].addKeyListener(this);
        }
        JLabel cc = new JLabel("Select Country");
        cc.addKeyListener(this);
        add(cc, 0, 0, 3, 1);
        add(country[0], 0, 1, 1, 1);
        add(country[1], 2, 1, 1, 1);
        add(country[2], 0, 2, 1, 1);
        add(country[3], 2, 2, 1, 1);
        this.setSize(500, 500);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_
```

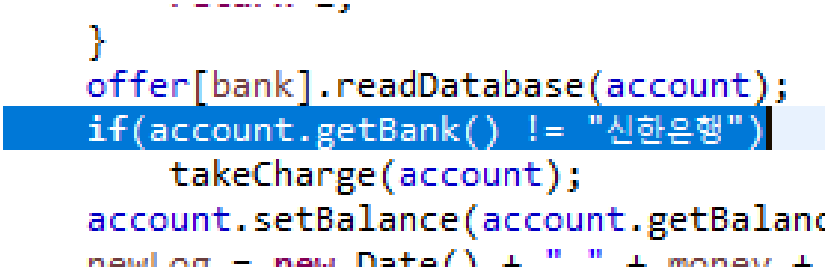
규칙	심각도	진단 메시지
Sun_09	낮음	identifier(bank)가 이미 필드로 선언되어 있어 hide가 발생함



규칙	심각도	진단 메시지
Sun_10	낮음	지역 변수(error) 선언 시 초기화를 하지 않음.



규칙	심각도	진단 메시지
Sun_11	매우 낮음	if 문에서 중괄호가 사용되지 않음.



규칙	심각도	진단 메시지
Sun_13	매우 낮음	Switch 문에 default case가 존재하지 않음.

```
    public void selectmenu() {
        int error;
        switch (menu) {
        case 0: // withdraw
            if ((error = main.withdraw(main.bank)) == 0) {
                new ReceiptFrame(main);
            } else {
                new errorPrintFrame(error);
            }
            this.dispose();
            break;
        case 1: // exchange
            if ((error = main.exchange(main.bank, main.card)) == 0) {
                new ReceiptFrame(main);
                this.dispose();
            } else {
                new errorPrintFrame(error);
```

규칙	심각도	진단 메시지
Sun_17	매우 낮음	지역 변수 선언문과 실행 문장 사이에 빈 줄이 없음.

```
country[3] = new JButton("CNY");
for (int i = 0; i < 4; i++) {
    country[i].addActionListener(this);
    country[i].addKeyListener(this);
}
JLabel cc = new JLabel("Select Country");
cc.addKeyListener(this);
add(cc, 0, 0, 3, 1);
add(country[0], 0, 1, 1, 1);
add(country[1], 2, 1, 1, 1);
add(country[2], 0, 2, 1, 1);
add(country[3], 2, 2, 1, 1);
this.setSize(500, 500);
```

규칙	심각도	진단 메시지
Sun_18	매우 낮음	클래스의 이름(selectCountry)이 규칙(^[A-Z]([a-z]|[A-Z]|[0-9])*)에 맞지 않음.

```
import javax.swing.*;

public class selectCountry extends JFrame implements ActionListener, KeyListener {
    MainSystem main;
    JButton[] country;
    GridBagLayout gbl;
    GridBagConstraints gbc;

    public selectCountry(MainSystem main) {
        super("selectCountry");
        gbl = new GridBagLayout();
        gbc = new GridBagConstraints();
```

규칙	심각도	진단 메시지
Sun_22	매우 낮음	non-static 필드가 public으로 선언.

```
public class MainSystem {
    private Account account;
    public String card;
    public Offer[] offer;
    public int bank;
    private int exchangeRate;
    private int wrongPasswordTimes;
    private int errorType;
```

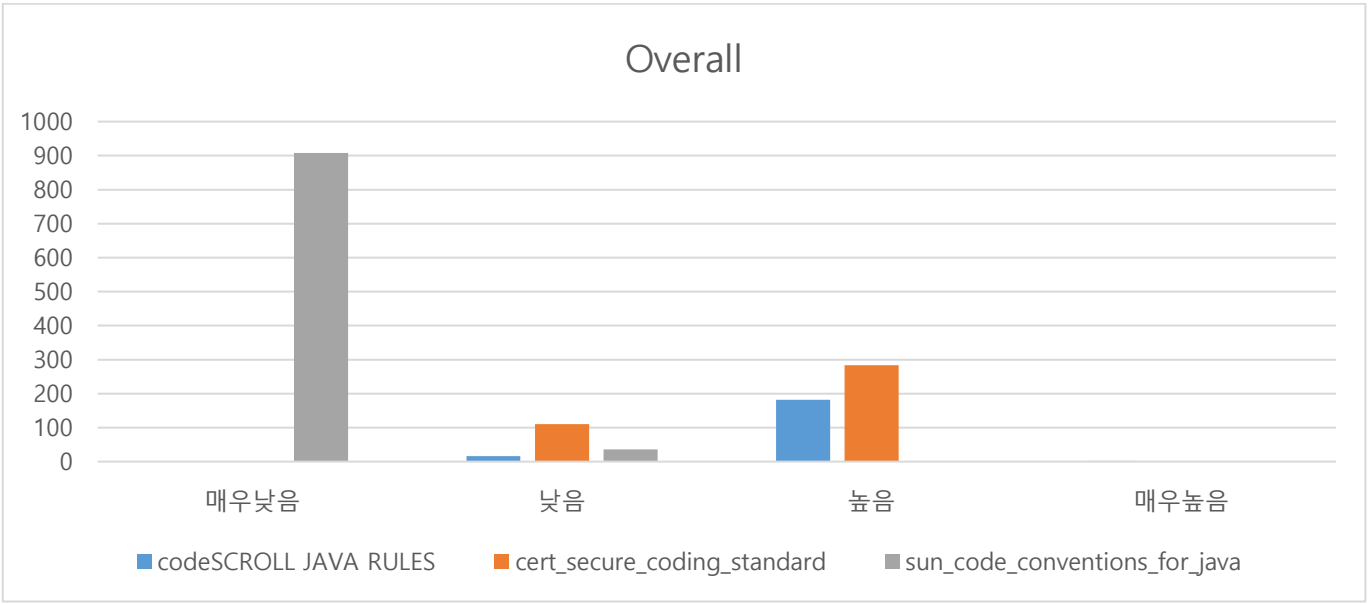
규칙	심각도	진단 메시지
Sun_24	매우 낮음	숫자 상수를 사용함(for loop의 counter로 -1이나 0, 1을 사용하는 경우 제외)

```
    }
    break;
case 4: // ****
    if (money.getText().length() > 5 && (money.getText()
            .substring(money.getText().length() - 5, money.getText().length() - 1).equals("0000"))) {
        main.bank = Integer.parseInt(money.getText().substring(0, money.getText().length() - 1));
        new inputPasswordFrame(2, main);
        this.dispose();
    } else {
```

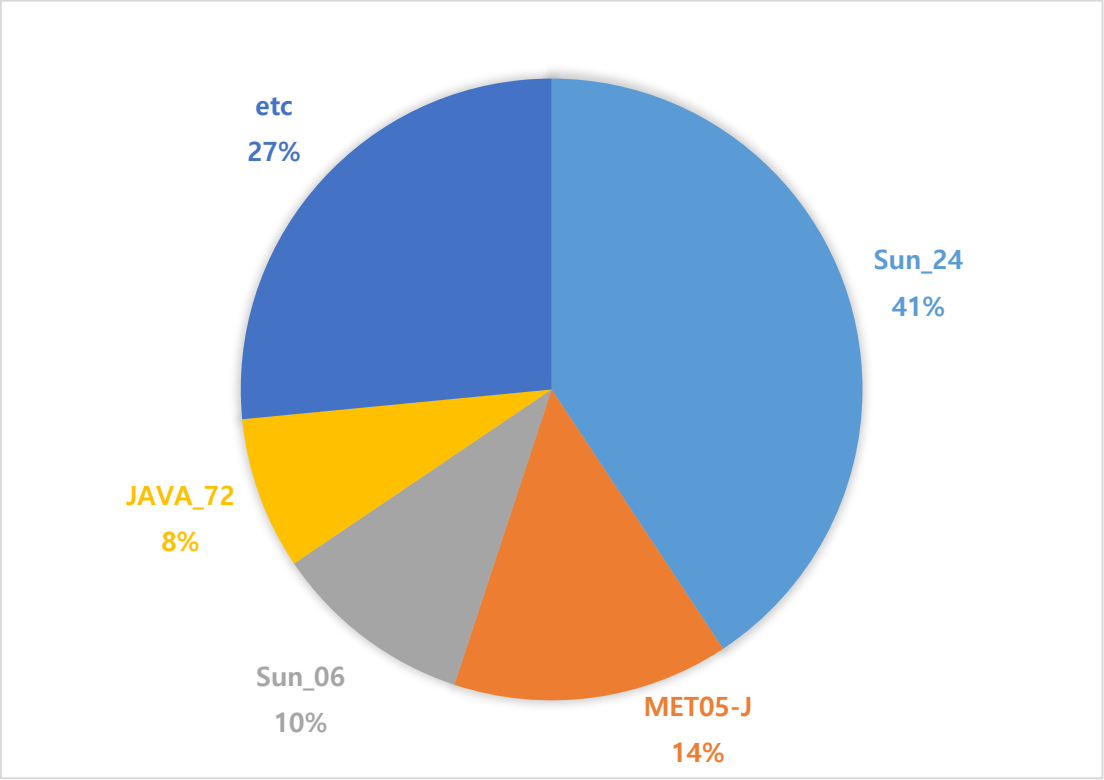
규칙	심각도	진단 메시지
Sun_26	낮음	괄호를 사용하지 않고 여러 연산자를 사용하여 수식을 혼용함.

```
        } else {
            new errorPrintFrame(3);
            this.dispose();
        }
    } else if (e.getSource() == bb && tf.getText().length() > 0) {
        tf.setText(tf.getText().substring(0, tf.getText().length() - 1));
    } else if (e.getSource() == reset) {
        tf.setText("");
    }
}
```

5. codescroll 결과



➔ 'sun_code_conventions_for_java'의 규칙이 가장 많이 위배되었으나 심각도는 매우 낮음과 낮음이며, 심각도가 높은 규칙의 위배 사항은 'codescroll_java_rules' 와 'cert_secure_coding_standard'에서 나타났다.



➔ Sun_24 규칙이 가장 많이 위배되었으며, 그 뒤로는 MET05-J, Sun_06, Java_72 순서로 위배되었다. Sun_24& Sun_06의 심각도는 매우 낮음 인 반면, Java_72와 MET05-J의 심각도는 높이이므로 이후 코드를 수정할 때 매우 높음의 심각도를 가진 Java_07 & Java_66 과 많이 위배한 Java_72 & MET05-J부터 수정해야 할 것으로 보인다.