

# Android 소스코드 난독화(Allatori) 적용 가이드

v.3.2

Allatori Version 8.7

2023.10



**NSHC**  
S E C U R I T Y

## 목 차

1. 개요 .....	3
2. 지원 사양 .....	3
3. 각 구성요소 설명 .....	3
4. Build 환경 별 라이브러리 설정 방법 .....	5
4.1 Android Studio 환경 .....	5
4.1.1 Kotlin 지원 .....	5
4.2 Eclipse 환경 .....	5
4.3 Ant Build 환경 .....	7
5. 난독화 설정 규칙 가이드.....	8
5.1 설정 파일(allatori.xml) 구조.....	8
allatori.xml .....	8
5.2 input Tag.....	10
5.3 classpath Tag.....	10
5.4 keep-names tag.....	10
5.4.1 class tag .....	11
5.4.2 field tag .....	12
5.4.3 method tag .....	14
5.5 Watermark tag.....	17
5.6 Expiry tag .....	17
5.7 Property tag.....	17
5.7.1 일반적인 속성들 .....	18
5.7.2 String 난독화 속성들 .....	19
5.7.3 흐름제어 난독화 속성들 .....	23
5.7.4 Renaming Properties .....	25
5.7.5 기타 속성들 .....	33
5.7.6 증분 난독화 속성 설정 .....	43
5.8 Ignore-classes 태그 .....	43
6. 어노테이션(Annotations) .....	45
7. 디버깅 가이드 .....	47

7.1 난독화 후 생성되는 파일.....	47
7.2 디버깅 프로세스 가이드.....	47
<b>8. 자주 묻는 질문.....</b>	<b>49</b>
8.1 난독화 적용 시 반드시 예외처리 해야 하는 부분이 있는지요? .....	49
8.2 InnerClass들에 대한 예외 처리 방법이 있는지요? .....	49
8.3 오픈 소스 라이브러리를 예외 처리 방법을 알려주세요.....	49
8.4 난독화 적용 후 발생한 에러에 대해서는 어떻게 해야 하는지요?.....	49
8.4.1 난독화 적용 후 Build 중 에러가 발생한 경우.....	49
8.4.2 난독화 적용 후 Runtime 중 에러가 발생한 경우.....	50
8.5 난독화 적용 후 임의 화면이 깨지는 현상이 발생합니다.....	50
8.6 java.lang.ArrayIndexOutOfBoundsException 오류가 발생합니다. ....	50
8.7 난독화된 로그를 원본으로 되돌릴 수 있는 방법이 있는지요?.....	51
8.8 Android Studio에서 라이브러리를 프로젝트에 allatori를 적용하는 방법이 있는지요?.....	51
8.9 Allatori를 빌드 환경에서 제거하고 싶습니다. ....	52
8.10 난독화 결과물 사이즈에 영향을 주는 요소와 결과물을 최적화 할 수 있는 방법은?.....	52
8.11 소스코드 난독화 적용 후 unsigned app을 생성하고 싶습니다.....	53
8.12 @annotation 예외처리 하는 방법 .....	54
8.13 Proguard rule을 allatori.xml에 변경 적용하는 방법 (예시).....	55
8.14 retro lamda를 사용하는데 gradle 3.x에서 난독화 적용 시 에러 .....	56
8.15 난독화를 적용 후 빌드하면 TransformException 발생과 duplicate entry: o/a.class 에러.....	56
8.16 난독화를 적용할 때 문자열 비교를 위한 equals 함수와 “==”의 차이점 .....	56
8.17 난독화 적용시 Android Studio의 Instant Run 기능 옵션 설정 해제.....	57
8.18 Build Type을 “Release”에서만 난독화가 적용 되도록 gradle 설정 방법.....	57
8.19 특정 클래스를 난독화로부터 완전히 제외시키는 방법 .....	58
8.20 DataBinding 활성화시에 Allatori 규칙 추가 방법 .....	58

## 문서 개정 이력

버전	날짜	내용	작성자	관리자
3.2	2023.09	Allatori v8.7 , v8.6 에서 추가 된 기능 가이드에 내용 추가 <ul style="list-style-type: none"> <li>리네임 옵션 - 'ABC' 추가</li> <li>set-methods-to-public , set-fields-to-public 신규 기능 내용 추가</li> </ul>	NSHC	NSHC
3.1	2022.09	디버그 내용 추가 및 FAQ 개편	NSHC	NSHC
3.0	2022.09	Allatori v8.3 맞춤 가이드 수정 및 member-reorder 내용 추가	NSHC	NSHC
2.9	2022.03	Allatori v8.1 맞춤 가이드 수정 및 프로가드 예시 추가	NSHC	NSHC
2.8	2022.01	Allatori v8.0 맞춤 가이드 수정 및 remove-annotations 추가	NSHC	NSHC
2.7	2021.12	Databinding 가이드 수정 및 일부 예제 수정	NSHC	NSHC
2.6	2021.12	kotlin 지원 가이드 내역 수정 및 문서 포맷 일부 수정	NSHC	NSHC
2.5	2021.07	Allatori 가이드 최신화	NSHC	NSHC
2.4	2020.08	회사 주소록 수정	NSHC	NSHC
2.3	2018.04.24	FAQ 항목 수정 (이너클래스)	NSHC	NSHC
2.2	2018.03.21	FAQ 항목 추가	NSHC	NSHC
2.1	2018.03.17	FAQ 항목 추가	NSHC	NSHC
2.0	2018.01.05	적용가이드 전체 업데이트	NSHC	NSHC
1.9	2016.12.15	예외처리 되는 부분 설명 추가Retrace Tool 관련 내용 추가	NSHC	NSHC
1.8	2016.11.24	Android Studio내에서 Allatori 삭제방법 추가	NSHC	NSHC

버전	날짜	내용	작성자	관리자
1.7	2016.11.17	4.4.4 내용 보강	NSHC	NSHC
1.6	2016.04.21	Trouble Shooting 추가	NSHC	NSHC
1.5	2016.02.18	Eclipse Plugin내용추가	NSHC	NSHC
1.4	2016.01.05	Android Studio library Project 생성가이드 추가	NSHC	NSHC
1.3	2015.11.03	FAQ 추가	NSHC	NSHC
1.2	2015.10.22	난독화 룰 가이드 추가	NSHC	NSHC
1.1	2015.10.13	초안작성	NSHC	NSHC



# 1. 개요

본 문서는 Android 소스코드 난독화 솔루션인 Allatori를 Android 앱에 적용하기 위한 가이드 문서입니다.

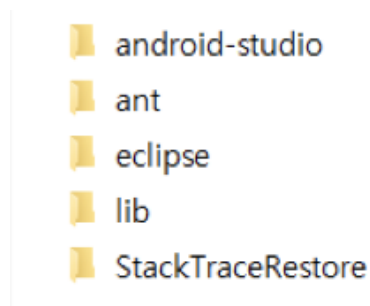
## 2. 지원 사양

- JDK:1.7이상
- Build 환경 : Android Studio, Eclipse Plugin, Ant Build 지원

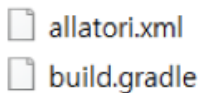
## 3. 각 구성요소 설명

압축 파일은 아래와 같은 폴더들로 구성되어 있으며 고객사 Build 환경에 맞는 파일들을 선택하셔서 적용하시기 바랍니다

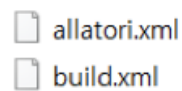
(allatori.xml은 Allatori 설정 파일로서 당사에서 제공하는 기본 rule set이 설정되어 있습니다.)



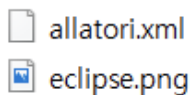
- android-studio



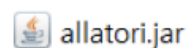
- ant







- eclipse



- lib (Allatori 라이브러리 파일)



- StackTraceRestore (난독화된 로그 복구 유틸리티)

- 
-  Clean.bat
  -  input.txt
  -  log.xml
  -  StackTraceRestore.bat

자세한 사용 방법은 8.7 난독화된 로그를 원본으로 되돌릴 수 있는 방법이 있는지요? (see page 51) 항목 참고



## 4. Build 환경 별 라이브러리 설정 방법

### 4.1 Android Studio 환경

- “rootDir” 디렉토리 안에 ‘allatori’라는 폴더를 만들고 ‘allatori.jar’ 파일을 복사합니다.
- “projectDir”(build.gradle 파일이 위치하는 디렉토리) 안에 ‘allatori.xml’ 파일을 복사합니다.
- 제공되는 ‘build.gradle’ 파일 내 임의 부분을 해당 프로젝트 build.gradle 파일에 추가합니다.

#### 4.1.1 Kotlin 지원

Kotlin 지원을 하기 위해서는 제공드린 압축 파일내 "android-studio" 폴더에 포함된 다음 두 파일의 내용에서 "Kotlin support"로 주석 처리 부분을 모두 해제해야 합니다.

- **allatori.xml**

```
<input>
  <dir out="${classesRoot}-obfuscated" in="${classesRoot}"/>
  <!-- Kotlin support 추가 -->
  <dir in="${kotlinRoot}" out="${kotlinRoot}-obfuscated"/>
</input>
```

- **build.gradle**
  - "// Kotlin support" 아래 주석 해제 (2 부분)

```
...

// Kotlin support
new File("${buildDir}/tmp/kotlin-classes/${variant.name}-obfuscated").deleteDir()

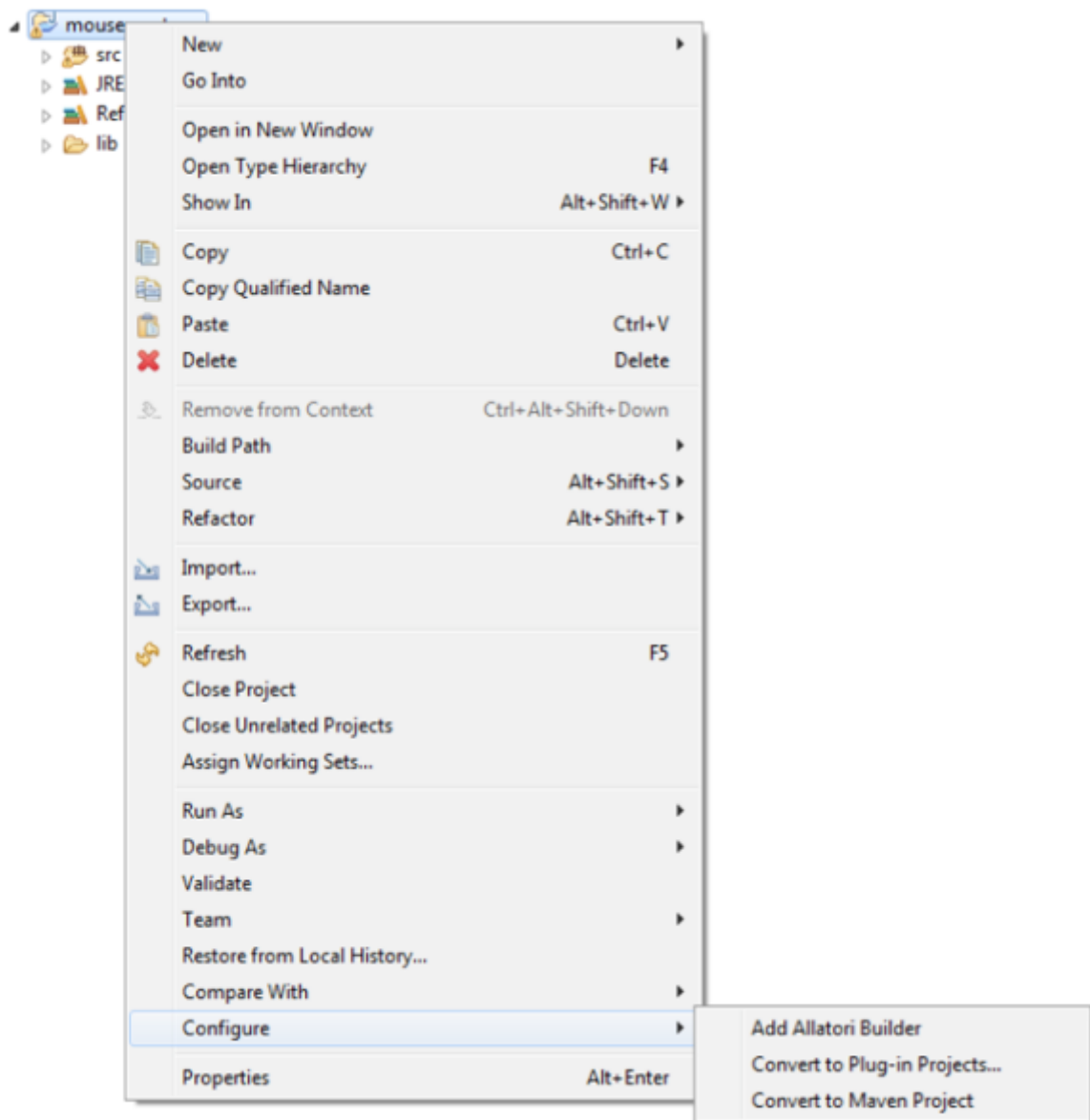
...

// Kotlin support
new File("${buildDir}/tmp/kotlin-classes/${variant.name}").deleteDir()
new File("${buildDir}/tmp/kotlin-classes/${variant.name}-obfuscated").renameTo(new
File("${buildDir}/tmp/kotlin-classes/${variant.name}"))
```

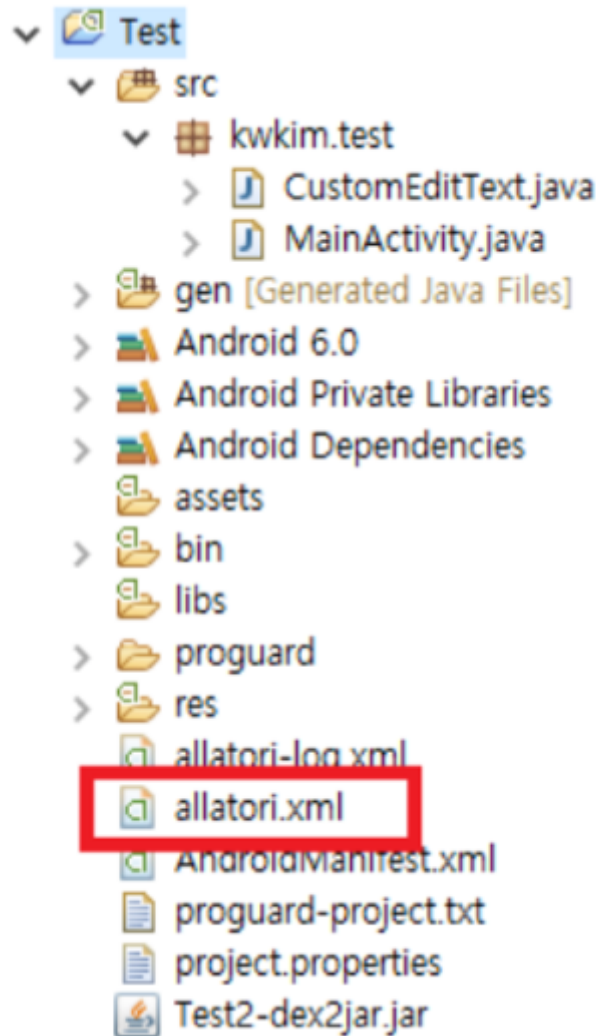
### 4.2 Eclipse 환경

- “allatori.jar” 파일을 Eclipse가 위치하는 곳의 “eclipse\dropins” 디렉토리에 복사한 이후, Eclipse Tool을 재 시작합니다.
- Allatori를 적용하려는 프로젝트를 선택한 후, 오른쪽 마우스를 클릭해서 팝업 메뉴 내 “Configure-> Add Allatori Builder”를 선택합니다.





- 이때 Project의 root폴더에 “allatori.xml”이 자동으로 생성되며 당사에서 제공하는 allatori.xml 내 임의 부분을 추가합니다. (해당 파일은 이미 존재한다면 재생성 되지 않습니다.)



- 프로젝트를 Clean후 Rebuild하면, allatori가 적용됩니다. 이때 난독화를 위한 설정은 프로젝트 root 폴더에 있는 “allatori.xml” 파일을 통해 적용됩니다.

### 4.3 Ant Build 환경

- Ant Build용 예제 allatori.xml 파일을 프로젝트의 폴더에 복사합니다.
- 프로젝트 폴더에 “allatori” 디렉토리를 만들고, “allatori.jar”파일을 생성된 폴더 안에 복사합니다.
- “build.xml”안에 아래와 같이 <target> tag 를 삽입합니다.

```
<target name="-obfuscate" unless="do.not.compile">
  <taskdef name="allatori" classname="com.allatori.ant.ObfuscatorTask" classpath="allatori/allatori.jar"/>
  <delete dir="${out.classes.absolute.dir}-obfuscated"/>
  <allatori config="allatori.xml"/>
  <property name="out.dex.input.absolute.dir" value="${out.classes.absolute.dir}-obfuscated"/>
</target>
```

## 5. 난독화 설정 규칙 가이드

Allatori를 적용하기 위해서는 "allatori.xml"이라는 configuration file에 상세한 난독화 룰에 대한 설정 작업이 필요합니다.

당사에서는 기본 rule set 및 몇가지 규칙들을 allatori.xml에 설정해서 배포하고 있습니다.

기본 설정 내역은 아래와 같으며 항목별 상세 설명은 가이드 문서의 상세부분들을 참고해주시길 바랍니다.

### 5.1 설정 파일(allatori.xml) 구조

Allatori 설정 파일은 다음 구조를 가진 XML 파일입니다.

#### allatori.xml

```
<config>
  <input 개발 환경에 알맞은 샘플 참조>
</input>
  <classpath 개발 환경에 알맞은 샘플 참조>
</classpath>

  <keep-names>
    <class template="public class * instanceof android.app.Activity">
      <method template="public void *(android.view.View)"/>
    </class>
    <class template="public class * instanceof android.app.Application"/>
    <class template="public class * instanceof android.app.Service"/>
    <class template="public class * instanceof android.view.View">
      <method template="public void set*(**)/>
      <method template="get*(**)/>
    </class>
    <class template="public class * instanceof android.content.BroadcastReceiver"
  />
    <class template="public class * instanceof android.content.ContentProvider"/
  >/
    <class template="public class * instanceof
android.app.backup.BackupAgentHelper"/>
    <class template="public class * instanceof android.preference.Preference"/>
    <class template="public class
com.android.vending.licensing.ILicensingService"/>
    <class template="public class
com.google.android.vending.licensing.ILicensingService
  <class template="class * implements android.os.Parcelable">
    <field template="public static final android.os.Parcelable*Creator */>
  </class>
  <!-- 리소스 클래스 예외처리 -->
  <class template="class **.R**">
    <field access="private+"/><method template="private+ *(**)/>
  </class>
  <!-- 외부 라이브러리 예외처리 -->
  <!-- [외부 jar 파일 표시] -->
  <!--
  <class template="class [수정할 부분:jar파일 패키지]">
```

```

        <field access="private+"/>
        <method template="private+ *(*)"/>
    </class>
-->

    <!-- 내부 코드 예외처리 -->
    <!-- 서버연동 클래스 -->
    <!-- Reflection 구현 클래스 -->
</keep-names>

<watermark key="secure-key-to-extract-watermark" value="Customer: John Smith"/>

<expiry date="2017/01/01" string="EXPIRED!"/>

<!-- Configuration properties, all properties are optional -->

<!-- General properties, we recommend to use these two properties -->

<property name="log-file" value="renaming-log.xml"/>
<property name="random-seed" value="type anything here"/>

<!-- String encryption -->

<property name="string-encryption" value="enable"/>
<property name="string-encryption-type" value="fast"/>
<property name="string-encryption-version" value="v4"/>
<property name="string-encryption-ignored-strings" value="patterns.txt"/>

<!-- Control flow obfuscation -->

<property name="control-flow-obfuscation" value="enable"/>
<property name="extensive-flow-obfuscation" value="normal"/>

<!-- Renaming -->

<property name="default-package" value="com.package"/>
<property name="force-default-package" value="enable"/>

<property name="packages-naming" value="abc"/>
<property name="classes-naming" value="compact"/>
<property name="methods-naming" value="compact"/>
<property name="fields-naming" value="compact"/>
<property name="local-variables-naming" value="optimize"/>

<property name="update-resource-names" value="enable"/>
<property name="update-resource-contents" value="enable"/>

<!-- Other -->

<property name="line-numbers" value="obfuscate"/>
<property name="generics" value="remove"/>
<property name="inner-classes" value="remove"/>
<property name="member-reorder" value="enable"/>
<property name="finalize" value="disable"/>
<property name="version-marker" value="anyValidIdentifierName"/>

```

```

<property name="synthesize-methods" value="all"/>
<property name="synthesize-fields" value="all"/>
<property name="remove-toString" value="enable"/>
<property name="remove-calls" value="com.package.Logger.debug"/>
<property name="remove-annotations" value="kotlin.Metadata"/>
<property name="output-jar-compression-level" value="9"/>

<!-- Incremental obfuscation -->
<property name="incremental-obfuscation" value="input-renaming-log.xml"/>
</config>

```

- ① 참고 1. 모든 상대 경로는 allatori.xml 파일 위치에 의해 확인됩니다.  
 참고 2. Ant에서 Allatori를 실행하는 경우 표준 Ant 구문인 `${PropertyName}`을 사용하여 Ant 빌드 파일에 정의된 특성을 참조할 수 있습니다.  
 참고 3. 시스템 속성 및 환경 변수는 각각 `${System.getProperty ( property.name )}` 및 `${System.getenv ( VARIABLE_NAME )}`을 사용하여 참조할 수 있습니다.

## 5.2 input Tag

해당 설정은 당사에서 기본으로 제공하는 allatori.xml 기본 설정 값을 적용 바랍니다.

## 5.3 classpath Tag

해당 설정은 당사에서 기본으로 제공하는 allatori.xml 기본 설정 값을 적용 바랍니다.

## 5.4 keep-names tag

*keep-names*는 난독화 과정에서 이름이 바뀌어서는 안되는 클래스, 메소드, 필드의 이름을 설정하는 데 사용됩니다.

만약 난독화 대상이 라이브러리라면 모든 Public API를 난독화에서 제외시켜야하고,

단독 앱이라면 최소한 main 클래스들의 이름은 난독화를 해서는 안됩니다.

또한 리플렉션을 통해 사용되는 클래스들과 메소드의 이름 또한 난독화에서 제외되어야 합니다.

Allatori의 Annotation 기능을 사용하는 경우 이것은 설정파일의 정의를 무시하고 적용됩니다.

*keep-names* 태그는 갯수에 상관없이 아래 태그를 포함시킬 수 있습니다:

- **field**: 이름이 바뀌어서는 안되는 특정 필드명
- **method**: 이름이 바뀌어서는 안되는 특정 메소드명
- **class**: 이름이 바뀌어서는 안되는 특정 클래스명, 내부에 *field* 태그와 *method* 태그를 포함할 수 있습니다.

이들 태그는 각각 매칭을 위한 특정 규칙이 있으며 'access' 와 'template' 라는 속성을 가지고 있습니다.

아래와 같이 'access' 속성에서 지정된 접근 레벨에 따라 매치 되는 것이 정해지며 자바의 접근제어자 값과 동일합니다.

자세한 예외처리 목록은 [Android 소스코드 난독화\(Allatori\) 적용 가이드#8.1 난독화 적용 시 반드시 예외처리 해야 하는 부분이 있는지요?](#)(see page 1) 항목 참고.

[access 속성 : 공통]

Value	Description
private	private 접근제어자로 설정된 classes, fields or methods 매치됨
private+	private 이상의 접근제어자로 설정된 classes, fields or methods 매치됨
package	package 접근제어자로 설정된 classes, fields or methods 매치됨
package+	package 이상의 접근제어자로 설정된 classes, fields or methods 매치됨
protected	protected 접근제어자로 설정된 classes, fields or methods 매치됨
protected+	protected 이상의 접근제어자로 설정된 classes, fields or methods 매치됨
public	public 접근제어자로 설정된 classes, fields or methods 매치됨

### 5.4.1 class tag

class 태그는 클래스를 매칭하는데 사용됩니다. 태그의 속성은 아래와 같습니다.

Attribute	Value
access	필수*. 매칭 규칙 설정. 가능한 값은 위 (see page 1)의 access 속성 내용 참고.
template	필수*. 매칭 규칙 설정. 아래에 가능한 값의 서식을 설명합니다.
ignore	선택. ("true" or "yes"), 일치하는 클래스는 난독화가 되지만 내부에 포함된 method 태그와 field 태그의 경우 평소대로 예외 처리됩니다. 특정 클래스는 난독화 하지만 특정 내부 메소드 및 필드 값의 이름을 난독화 예외처리 하고자 할 경우 사용됩니다.  ("keep-if-members-match") 중첩된 메서드 또는 필드 태그 중 적어도 하나는 클래스 이름을 유지하기 위해 일치해야 합니다. 앞선 규칙과 일치하는 클래스 이름과 필드, 메소드 모두 예외 처리 됩니다.
stop	선택. ("true" or "yes"), 일치하는 클래스에 대한 어떠한 추가 규칙도 적용을 하지 않습니다.

\* access 또는 template 중 하나는 반드시 사용되어야 함.

template 속성의 서식은 아래와 같습니다.

**[@annotation] [modifiers] (class / interface) 클래스명 [extends 클래스명] [implements 클래스명] [instanceof 클래스명]**

'\*' 기호는 길이에 상관없이 문자열을 매칭시키는 와일드카드 캐릭터 입니다. 이름이 'regex:'로 시작하면 표준 정규식 적용이 가능합니다.

[ template 속성 예시 ]

Value	Description
class *	모든 클래스와 인터페이스 매치.
interface *	모든 인터페이스 매치.
public class *	모든 public 클래스와 인터페이스 매치.
protected+ class *	모든 protected와 public 클래스와 인터페이스 매치.
class *abc*	"abc" 라는 문자열이 이름에 포함된 모든 클래스 매치.
class com.abc.*	com.abc 패키지 아래 하위 패키지의 모든 클래스 매치.
class *.abc.*	"abc" 패키지와 그 하위 패키지의 모든 클래스 매치.
class * extends java.util.Enumeration	java.util.Enumeration 를 상속받은 모든 클래스 매치.
class * extends *.Enumeration	Enumeration 를 상속받은 모든 클래스 매치.
class * instanceof java.io.Serializable	<a href="http://java.io.Serializable">java.io<sup>1</sup>.Serializable</a> 의 인스턴스인 모든 클래스 매치.
class * implements *.MouseListener	MouseListener 를 구현한 모든 클래스 매치.
@java.lang.Deprecated class *	deprecated 정의된 모든 클래스 매치.
class regex:com.package.(foo bar).*	com.package.foo 패키지와 com.package.bar 패키지의 하위 패키지까지 모든 클래스 매치. (정규표현식 사용예제)
class regex:comW.packageW. (foo bar)W..*	위에 것 보다 좀더 명확한 규칙. "W"와 함께 쓰게 되면 이제 '.'은 명확히 "."의 의미를 가지게 됨. '.'만 사용할 경우 문자하나로 인지합니다.

### 5.4.2 field tag

field 태그는 field 값을 매칭하기위해 사용합니다.

class 태그와 함께 사용될 경우 class 태그에 매칭된 클래스 내부의 field 값에만 규칙이 적용되며, keep-names 태그 아래에 있을 경우 모든 클래스 내에 있는 field 값으로 매칭됩니다.

<sup>1</sup> <http://java.io>

Attribute	Value
access	필수*. 매칭 규칙 설정. 가능한 값은 위 (see page 1)의 access 속성 내용 참고.
template	필수*. 매칭 규칙 설정. 서식은 아래에 서술합니다.

\* access 또는 template 중 하나는 반드시 사용되어야 함.

field 태그의 template 속성 값 서식:

**[@annotation] [modifiers] [type] 필드이름 [instanceof 클래스명]**

'\*' 기호는 임의의 개수의 문자열을 매칭시키는 와일드카드 캐릭터 입니다. 이름이 'regex:'로 시작하면 표준 정규식 적용이 가능합니다.

#### [ template 속성 예시 ]

Value	Description
*	모든 필드 매치
private *	모든 private 필드 매치.
private+ *	모든 필드 매치.
protected+ *	모든 protected 와 public 필드 매치.
static *	모든 static 필드 매치.
public static *	모든 public static 필드 매치.
public int *	모든 public integer 타입 필드 매치.
java.lang.String *	모든 String 타입 필드 매치.
java.lang.* *	모든 java.lang 패키지 타입 필드 매치.
abc*	이름이 "abc"로 시작하는 모든 필드 매치.
private abc*	이름이 "abc"로 시작하는 모든 private 필드 매치.
* instanceof java.io.Serializable	모든 serializable 필드 매치.
@java.lang.Deprecated *	모든 deprecated 필드 매치.
regex:(a b).*	"a" 또는 "b" 로 시작하는 이름의 모든 필드 매치.



### 5.4.3 method tag

메소드를 매칭하기 위해 method 태그를 이용합니다. 해당 태그가 class 태그 내에서 사용될 경우,

해당 규칙은 class 태그에 의해 매칭된 클래스 내부의 메소드에만 적용되고 keep-names 태그 내부에 있을 경우 모든 클래스의 메소드에 적용됩니다.

Attribute	Value
access	필수*. 매칭 규칙 설정. 가능한 값은 위 (see page 1)의 access 속성 내용 참고.
template	필수*. Sets matching rule. Its format is described below.
parameters	선택. "keep" 이라고 설정할 경우, 메소드의 파라미터의 경우 이름을 바꾸지 않습니다. Public API 메소드에 유용한 옵션입니다.

\* access 또는 template 중 하나는 반드시 사용되어야 함.

method 태그의 template 속성 값 서식:

**[@annotation] [modifiers] [type] 메소드이름(매개변수들)**

'\*' 기호는 메소드 또는 타입이름 매칭에 사용되는 문자입니다. '\*'는 임의의 개수의 문자를 매칭하고 '\*\*' 는 임의의 개수의 매개변수를 매치합니다.

'regex:'로 시작하면 표준 정규식 적용이 가능합니다.

[ template 속성 예시 ]

Value	Description
*(**)	모든 메소드 매치.
private *(**)	모든 private 메소드 매치.
private+ *(**)	모든 메소드 매치.
protected+ *(**)	모든 protected 메소드와 public 메소드 매치.
private+ *(*)	매개변수가 하나인 모든 메소드 매치.
private+ *(*,*)	매개변수가 두개인 모든 메소드 매치.
private+ *(java.lang.String)	매개변수 타입이 String인 모든 메소드 매치.
private+ *(java.lang.String,**)	첫번째 매개변수가 String인 모든 메소드 매치.
private+ *(java.lang.*)	매개변수가 java.lang 패키지내의 타입인 모든 메소드 매치.
public get*(**)	이름이 get으로 시작하는 모든 public 메소드 매치.

Value	Description
public *abc*(**)	이름에 'abc'가 포함된 모든 public 메소드 매치.
private+ int *(**)	리턴 타입이 int인 모든 메소드 매치.
@java.lang.Deprecated *(**)	모든 deprecated 선언된 메소드 매치.
public regex:(g s)et.*(**)	모든 public getter/setter 메소드 매치.

전체 사용 예제:

```

<keep-names>
  <!-- Stops applying further rules to classes in the com.company.abc package,
    therefore all classes, methods and fields in this package will be
renamed -->
  <class template="class com.company.abc.*" stop="true"/>
  <!-- Further rules instruct Allatori not to rename matched elements -->

  <!-- Matches classes with the name "Main" in any package -->
  <class template="class *.Main"/>

  <!-- Matches classes with the name ending with "Bean" -->
  <class template="class *Bean">
    <!-- Matches all fields -->
    <field access="private+"/>
    <!-- Matches public integer fields -->
    <field template="public int *"/>
    <!-- Matches all static fields -->
    <field template="static *"/>
    <!-- Matches protected and public String fields -->
    <field template="protected+ java.lang.String *"/>
    <!-- Matches all methods -->
    <method template="private+ *(**)/>
    <!-- Matches all getter methods -->
    <method template="private+ get*(**)/>
    <!-- Matches all methods with String argument,
      parameter names of these methods will not be changed -->
    <method template="private+ *(java.lang.String)" parameters="keep"/>
  </class>

  <!-- Matches serialization members -->
  <class template="class * instanceof java.io.Serializable">
    <field template="static final long serialVersionUID"/>
    <method template="void writeObject(java.io.ObjectOutputStream)"/>
    <method template="void readObject(java.io.ObjectInputStream)"/>
    <method template="java.lang.Object writeReplace()"/>
    <method template="java.lang.Object readResolve()"/>
  </class>

  <!-- Matches applets -->
  <class template="class * instanceof java.applet.Applet"/>

```

```
<!-- Matches servlets -->  
<class template="class * instanceof javax.servlet.Servlet"/>  
<!-- Matches midlets -->  
<class template="class * instanceof javax.microedition.midlet.MIDlet"/>  
</keep-names>
```



## 5.5 Watermark tag

watermark 태그는 워터마킹을 위해 키와 값을 설정하여 사용합니다.

Attribute	Value
key	필수. 이 값은 스테가노그래피 기술을 사용하여 애플리케이션에 워터마크를 삽입하고자 할 때에 키로 사용됩니다.
value	워터마크 삽입시 필수. 애플리케이션 jar 안에 삽입될 임의의 문자열. 이것은 카피라이트, 고객명, 회사명 또는 고유하게 식별할 수 있는 임의의 어떤 정보라도 될 수 있습니다. 워터마크는 소프트웨어의 소유주를 확인하거나 또는 소프트웨어가 해적판인지를 식별가능하게 합니다.

사용 예제:

```
<watermark key="secure-key-to-extract-watermark" value="Customer: John Smith"/>
```

워터마크는 이미 난독화 되었거나 난독화 되지 않은 jar 파일에서도 사용가능합니다.

워터마크를 적용하는 완벽한 사용 예제는 당사에서 제공한 Allatori 배포본의 튜토리얼에서 확인하실 수 있습니다.

## 5.6 Expiry tag

expiry 태그는 어플리케이션의 만료 날짜를 설정하는 것에 사용합니다.

만료 날짜 체크는 쉽게 지울 수 없도록 메인 메소드만이 아닌 다양한 메소드들에 삽입됩니다.

이 기능은 메인 메서드가없는 라이브러리의 난독화시도 사용할 수 있습니다. 태그 사용시 필요한 두가지 속성값은 아래와 같습니다.

Attribute	Value
date	필수. 만료 날짜 포맷 (yyyy/mm/dd).
string	필수. 어플리케이션 실행시 만료날짜가 되어 예외 발생시 던져줄 임의의 메시지 내용

사용 예제:

```
<expiry date="2017/01/01" string="EXPIRED!"/>
```

## 5.7 Property tag

property 태그는 앞서 설명한 것과 다른 난독화시 속성을 설정할 때 사용합니다.

이 태그는 아래와 같이 name과 value 속성을 가지고 있습니다.

```
<property name="속성 이름" value="속성 값"/>
```

## 5.7.1 일반적인 속성들

### 5.7.1.1 log-file

Value	Description
파일이름	Allatori는 난독화 로그를 value로 설정한 파일명의 파일에 기록합니다. 만약 이 값이 설정되지 않을 경우 로그파일은 생성되지 않습니다. 로그 파일의 경로는 설정 파일의 위치를 기반한 상대경로로 설정됩니다.

로그 파일은 난독화된 이름 및 라인 넘버의 원본 값 사이의 매핑을 보존하며 난독화된 stack trace를 원래 stack trace로 복구하는데 사용합니다.

사용 예제:

```
<property name="log-file" value="log.xml"/>
<property name="log-file" value="logs/file.xml"/>
```

Stack trace 도구에서 로그파일을 사용하여 본래의 이름을 복구하는 방법은 아래와 같습니다.

```
//java -cp Allatori라이브러리 com.allatori.StackTrace2 [매핑로그파일명] [난독화된 StackTrace 저장파일명] [난독화 복구결과가 저장될 파일명]
java -cp allatori.jar com.allatori.StackTrace2 log.xml input.txt output.txt
```

### 5.7.1.2 random-seed

Value	Description
임의의 문자열	랜덤 시드 생성 초기화를 위한 문자열

기본값은 현재시간의 milliseconds 값.

기본값 적용시, 동일한 소스로 Allatori가 적용 될 때마다 서로 다른 난독화 결과물이 생성됩니다.

즉, 클래스, 필드, 메소드들은 이전 결과물과 다른 이름을 가지게 됩니다.

이것은 난독화된 애플리케이션의 두 가지 버전을 분석하기 매우 어렵게 만들기 위해 수행됩니다.

만약 귀사에서 Allatori를 여러번 동작시킬 때마다 같은 결과물이 나오는 것이 필요하시다면 이 랜덤 시드 속성 값을 활성화 해야합니다.

당사에서는 보안을 위해 어플리케이션이 정식 배포될 때마다 랜덤 시드값을 바꾸기를 권장합니다.

사용 예제:

```
<property name="random-seed" value="임의의 문자열 입력"/>
```

## 5.7.2 String 난독화 속성들

### 5.7.2.1 string-encryption

Value	Description
enable	(기본값) 암호화 된 값으로 안전하게 변경할 수 있는 모든 문자열이 암호화됩니다. Allatori는 런타임시 문자열 복호화 메소드를 삽입하여 정상적으로 실행되게 만들어줍니다.
disable	문자열 암호화를 비활성화 합니다.
maximum	모든 문자열을 암호화 합니다. 한계점은 아래 내용을 참고해주시길 바랍니다.
maximum-with-warnings	모든 문자열을 암호화 합니다. '==' 를 사용하여 문자열을 비교를 사용하는 모든 문자열 사용에 대하여 경고문을 발생하며, 이에 따라 이러한 구문에 대하여 equals() 를 호출하여 비교문을 수행하도록 변경할 수 있습니다.

사용 예제:

```
<property name="string-encryption" value="enable"/>
```

가끔 문자열은 이와 같은 방법으로 비교문을 수행합니다.

```
String myString = "Hello";
...
public boolean test() {
    return myString == "Hello";
}
```

문자열 비교에 equals 대신 '=='를 사용하는 나쁜 습관에도 불구하고 위 예제의 test() 메소드는 true 값을 리턴합니다. JVM은 같은 클래스내의 문자열 객체는 재사용 가능하도록 캐싱하기 때문입니다. 그러나 문자열 난독화 이후 test() 메소드는 아래와 비슷하게 됩니다.

```
public boolean test() {
    return myString == new String("Hello");
    // "Hello" 문자열은 예제에 대하여 명확한 이해를 돕기위해 난독화 하지 않았습니다.
}
```

위와 같은 경우 test() 메소드는 객체를 비교하게 되고 객체는 다르므로 false를 리턴하게 됩니다.

만약 문자열 암호화 설정시 'enable'로 값을 설정한다면, Allatori는 '==' 를 사용하는 문자열은 암호화를 진행하지 않을 것이며 어플리케이션은 올바르게 동작할 것입니다.

만약 귀사에서 항상 equals 메소드를 사용하여 문자열 비교를 수행한다면 string-encryption 속성을 'maximum'으로 설정해 주시길 바랍니다.

문자열 암호화는 어노테이션 기능 또는 apply2class를 사용하여 특정 클래스들에 대하여 활성/비활성 을 설정할 수 있습니다.

'apply2class' 은 class 태그의 template 과 동일한 사용 포맷을 가지고 있습니다. 아래 예제를 참고하시길 바랍니다.

```

<!-- com.abc 패키지 안의 클래스들의 문자열 암호화를 비활성화 시킬때 -->
<property name="string-encryption" value="disable" apply2class="class com.abc.*"/
>
<!-- 다른 모든 클래스의 문자열 암호화를 활성화 시킬 때 -->
<property name="string-encryption" value="enable"/>

```

### 5.7.2.2 string-encryption-type

Value	Description
fast	(기본) 매우 빠른 문자열 암호화를 사용.
strong	강력하고 교묘한 암호화 알고리즘 사용. 그러나 좀더 느립니다.
custom (package.EncryptClassName.encryptMethodName, package.DecryptClassName.decryptMethodName)	명시한 커스텀 문자열 암호화/복호화 메소드를 사용함. 자세한 내용은 아래 예제 참고.

이 속성을 사용하기 위해서는 *"string-encryption"*는 반드시 활성화 되어 있어야 합니다.

사용 예제:

```

<property name="string-encryption-type" value="strong"/>

```

문자열 암호화는 어노테이션 기능 또는 apply2class 을 사용하여 특정 클래스들에 대하여 활성화/비활성 을 설정할 수 있습니다.

'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다. 아래 예제를 참고 하시길 바랍니다.

```

<!-- Setting strong string encryption type for classes in com.abc package -->
<property name="string-encryption-type" value="strong" apply2class="class com.abc.*"/
>
<!-- Setting fast string encryption type for all other classes -->
<property name="string-encryption-type" value="fast"/>

```

#### \* custom 속성 사용 방법

암호화 메소드는 난독화 중에만 필요하고 어플리케이션 동작 중에는 필요하지 않습니다. 암호화 메소드는 배포판에 포함 되지 않은 별도의 jar 파일에 있을 수 있습니다.

복호화 메소드는 런타임에 필요하며 애플리케이션의 모든 클래스에 배치 할 수 있습니다. 커스텀 문자열 암호화를 Allatori의 문자열 암호화와 조합하여 사용할 수 있습니다.

```
<property name="string-encryption-type"
value="custom(package.EncryptClassName.encryptMethodName,
package.DecryptClassName.decryptMethodName)"
apply2class="class com.some.package.*"/>
```

그리고 여러개의 문자열 암호화 메소드를 사용할 수도 있습니다.

```
<property name="string-encryption-type"
value="custom(EncryptClassName1.encryptMethodName1,
DecryptClassName1.decryptMethodName1)"
apply2class="class com.some.package.*"/>
<property name="string-encryption-type"
value="custom(EncryptClassName2.encryptMethodName2,
DecryptClassName2.decryptMethodName2)"
apply2class="class com.some.other.package.*"/>

<!-- 위 규칙에 맞지 않은 클래스들을 위한 암호화/복호화 메소드들 -->
<property name="string-encryption-type"
value="custom(EncryptClassName3.encryptMethodName3,
DecryptClassName3.decryptMethodName3)"/>
```

커스텀 문자열 암호화 기능은 런타임시에 문자열을 치환하므로, 기본 암호화 대신 사용할 수 있습니다.

암호화 메소드는 난독화 프로세스 중 모든 문자열(문자열 암호화 속성으로 maximum 설정시)에 대해 호출되므로 이를 사용하여 모든 문자열을 기록하는데 사용할 수 있습니다.

그리고 복호화 메소드는 런타임시에 문자열을 각기 사용된 암호화한 버전으로 치환합니다.

### 5.7.2.3 string-encryption-version

Value	Description
v4	(기본) 신규 문자열 암호화 알고리즘을 사용
v3	3.X 버전의 Allatori의 문자열 암호화 기능을 이용

\* 이 특성을 적용 받고자 한다면 문자열 암호화(string-encryption)를 활성화 해야만 합니다.

사용 예제:

```
<property name="string-encryption-version" value="v3"/>
```

### 5.7.2.4 string-encryption-ignored-strings

Value	Description
filename	난독화로 부터 제외되어야 하는 문자열 형태를 담은 문서 파일명. 명시된 규칙의 문자열은 암호화 되지 않을 것입니다.



템플릿 문자열을 담고 있는 파일을 지정합니다.

사용 방법:

```
<property name="string-encryption-ignored-strings" value="patterns.txt"/>
```

문서내 한 줄당 새로운 규칙으로 인식됩니다. '\*' 은 임의의 개수의 임의의 문자를 매치를 의미하며 "regex:"로 시작하면 정규표현식의 사용이 가능합니다.

파일 내용의 예제는 아래와 같습니다.

#### patterns.txt 파일 내용

Copyright\*

All Rights Reserved\*

\*CompanyName\*

regex:Wd+



## 5.7.3 흐름제어 난독화 속성들

### 5.7.3.1 control-flow-obfuscation

Value	Description
enable	(기본) 모든 메소드의 코드에 대하여 적용. 런타임시 어플리케이션의 동작을 바꾸지 않지만 디컴파일 프로세스를 무척 힘들게 만듭니다. 일반적으로, 흐름제어 난독화는 어플리케이션을 더 작고 빠르게 만듭니다.
disable	흐름제어 난독화 비활성화

사용 방법:

```
<property name="control-flow-obfuscation" value="enable"/>
```

흐름제어 난독화는 특정 클래스에 대하여 어노테이션 기능과 "apply2class" 속성을 사용하여 활성화/비활성화 시킬 수 있습니다.

'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다. 아래 예제를 참고 하시길 바랍니다.

예제:

```
<!-- com.abc 패키지 내의 모든 클래스들에 대하여 흐름제어 난독화 비활성화 -->
<property name="control-flow-obfuscation" value="disable" apply2class="class
com.abc.*"/>
<!-- 이외 모든 클래스에 대하여 흐름제어 난독화 활성화 -->
<property name="control-flow-obfuscation" value="enable"/>
```

### 5.7.3.2 extensive-flow-obfuscation

Value	Description
normal	(기본) 난독화된 어플리케이션이 더 크고 느려지게 만드는 흐름제어 난독화 기술사용에 대한 설정. 그러나 최저한으로 코드변경을 수행합니다.
disable	확장된 흐름제어 난독화 비활성화.
maximum	최대한의 흐름제어 난독화 기술을 사용합니다. 어플리케이션은 좀더 크고 느려집니다.

"control-flow-obfuscation" 를 활성화 해야 해당 속성을 사용할 수 있습니다.

적용 예제:

```
<property name="extensive-flow-obfuscation" value="maximum"/>
```

광범위한 흐름제어 난독화는 특정 클래스에 대하여 어노테이션 기능과 "apply2class" 속성을 사용하여 활성화/비활성화 시킬 수 있습니다.

'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다. 아래 예제를 참고 하시길 바랍니다.

예제:

```
<!-- com.abc 패키지 내의 모든 클래스들에 대하여 "maximum" 값을 설정 -->  
<property name="extensive-flow-obfuscation" value="maximum" apply2class="class  
com.abc.*"/>  
<!-- 이외 모든 클래스에 대하여 "normal" 속성 설정 -->  
<property name="extensive-flow-obfuscation" value="normal"/>
```



## 5.7.4 Renaming Properties

### 5.7.4.1 default-package

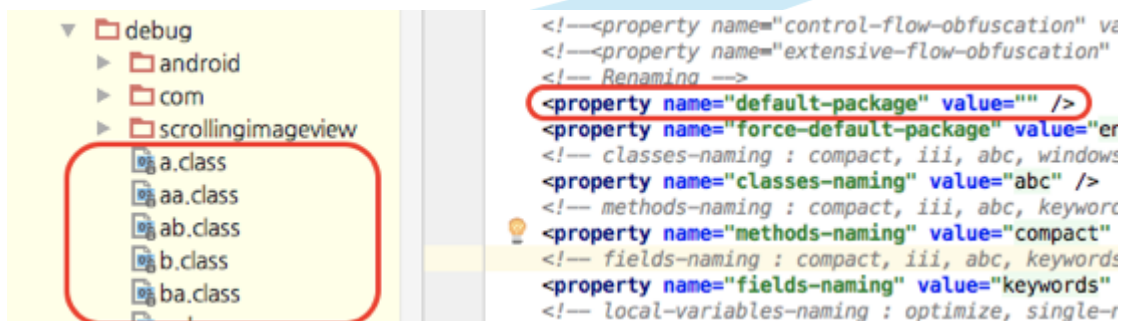
Value	Description
package name	새로운 것 또는 이미 있는 것으로 패키지 이름 전체 패스

만약 어떤 패키지 내에 모든 클래스의 이름이 바뀔 경우 Allatori는 디폴트 패키지에 설정된 값으로 해당 클래스들을 옮깁니다.

이름이 바뀐 모든 클래스들이 완벽히 디폴트 패키지로 이동하길 원한다면, `force-default-package`<sup>2</sup> 속성을 활성화 해야만 합니다.

디폴트 패키지 값으로 ""로 사용할 경우 jar 결과물의 크기를 줄일 수 있습니다.

적용 예제:



```
<property name="default-package" value=""/>
```

```
<property name="default-package" value="com.company.product"/>
```

### 5.7.4.2 force-default-package

Value	Description
disable	(기본값) 모든 클래스의 이름이 변경된 패키지의 클래스만 기본 패키지로 이동됩니다.
enable	이름이 바뀐 클래스 모두 디폴트 패키지로 이동합니다.

\* default-package 값을 설정해야지 해당 옵션이 활성화 됩니다.

적용 예제:

```
<property name="force-default-package" value="enable"/>
```

<sup>2</sup> <http://www.allatori.com/doc.html#property-force-default-package>

### 5.7.4.3 packages-naming

Value	Description
abc	(기본값) 패키지의 이름이 모두 'a', 'b', 'c', 'd', ..., 'aa', 'ab' 등 과 같이 변경됩니다. 모두 소문자로만 구성됩니다.
ACB	패키지의 이름이 'A', 'B', 'C', 'D', ..., 'AA', 'AB' 등 과 같이 모두 대문자로만 구성됩니다.
123	패키지의 이름이 '1', '2', '3', ..., '00', '01', 등과 같이 숫자값으로 변경됩니다.
keep	패키지 이름이 원래 값을 유지합니다.
custom(filename.txt)	파일에 명시된 사용자 지정 규칙으로 패키지 이름을 변경합니다. 규칙이 명시된 파일명을 인자로 받습니다. 파일에 0과 1이 두개의 값이 한줄당 하나씩 있다면, 패키지 이름 변경 규칙은 '0', '1', '00', '01', '10', '11', '000' 등과 같습니다.

적용 예제:

```
<property name="packages-naming" value="abc"/>
<property name="packages-naming" value="custom(filename.txt)"/>
```

### 5.7.4.4 classes-naming

Value	Description
compact	(기본값) 가능한 하나의 문자로 이름을 치환하여 난독화 결과물의 사이즈가 작아질 수 있도록 합니다. 클래스는 a.class 또는 A.class 와 같이 대소문자가 섞인 이름을 가질 수도 있습니다. Windows 파일 시스템은 대소문자가 다른 파일이 있을 경우 일부 클래스의 압축을 푸는 것이 까다로울 것입니다. ( Windows에서 압축을 풀면 a.class가 A.class를 덮어 씁니다.) 그러나 Jar 파일은 대소 문자가 혼합된 파일 이름을 허용하기에 윈도우를 포함한 모든 플랫폼에서 대소문자가 섞인 파일 이름이 잘 동작 합니다.
iii	모든 클래스의 이름이 같은 길이의 다른 대소문자를 가지는 iihi, ihil, ihli 등 과 같이 변경됩니다. 다른 치환 옵션에 비해 결과물의 사이즈가 커질 수 있습니다.
abc	'a', 'b', 'c', 'd', ..., 'aa', 'ab' 등과 같이 오직 소문자 문자로만 이루어지도록 이름이 변경됩니다.
ABC	클래스의 이름이 'A', 'B', 'C', 'D', ..., 'AA', 'AB' 등 과 같이 모두 대문자로만 구성됩니다.

Value	Description
123	'1', '2', '3', ..., '00', '01', 등과 같이 숫자값으로 치환됩니다.
windows	<p>윈도우에서 클래스 이름으로 금지된 'con', 'prn', 'aux', 'nul' 등과 같은 값을 클래스 이름으로 사용합니다.</p> <p>jar 파일에서 con.class는 얼마든지 괜찮습니다. 그러나 이러한 클래스 파일은 윈도우 시스템에서는 압축을 해제할 수 없습니다.</p> <p>클래스에는 대소문자만 다른 대소문자가 혼합된 이름이 있을 수도 있습니다.</p> <p>이러한 파일 이름을 가진 jar 파일은 Windows를 포함한 모든 플랫폼에서 잘 작동합니다.</p> <p>이 옵션은 compact 또는 abc 이름 지정에 비해 결과물인 jar 파일을 더 크게 만듭니다.</p>
custom(filename.txt)	<p>파일에 명시된 사용자 지정 규칙으로 패키지 이름을 변경합니다. 규칙이 명시된 파일명을 인자로 받습니다.</p> <p>파일에 0과 1이 두개의 값이 한줄당 하나씩 있다면, 이름 변경 규칙은 '0', '1', '00', '01', '10', '11', '000' 등과 같습니다.</p>
unique	<p>모든 클래스의 이름은 고유한 이름을 가지게 됩니다.</p> <p>서로다른 패키지에서 서로 중복되는 클래스명은 존재하지 않습니다.</p> <p>다른 네이밍 옵션과 함께 사용가능합니다.</p>
keep-\$-sign	<p>자바 이너클래스 이름의 기호를 바꾸지 않고 그대로 있도록 합니다.</p> <p>예를 들어 Foo와 Foo\$bar는 이름 치환 후 a와 a\$b 와 같이 변경됩니다.</p> <p>기본값 설정시 Allatori는 Foo를 a로 Foo\$Bar를 b로 변경합니다.</p> <p>다른 클래스 네이밍 옵션과 함께 사용할 수 있습니다.</p>

사용 예제:

```
<property name="classes-naming" value="abc"/>
```

#### 5.7.4.5 methods-naming

Value	Description
compact	(기본 값) 이름들을 가능한 문자 하나로 치환하여 만들어지는 jar 파일의 사이즈를 줄입니다.
iii	<p>이름이 같은 길이와 서로 다른 다른 대소문자를 가지는 iii, iiii, iiii 등 과 같이 변경됩니다.</p> <p>다른 치환 옵션에 비해 jar 결과물의 사이즈가 커질 수 있습니다.</p>
abc	메소드명이 'a', 'b', 'c', 'd', ..., 'aa', 'ab' 등과 같이 오직 소문자 문자로만 이루어지도록 이름이 변경됩니다.

Value	Description
ABC	메소드명이 'A', 'B', 'C', 'D', ..., 'AA', 'AB' 등 과 같이 모두 대문자로만 구성됩니다.
123	메소드명이 '1', '2', '3', ..., '00', '01', 등과 같이 숫자값으로 치환됩니다.
keywords	자바에서의 예약어('if', 'for', 'int', 등)를 메소드 이름으로 사용합니다. 이런 이름들은 class 파일 포맷에서는 괜찮지만 많은 디컴파일러에게는 혼동을 일으킵니다. 그러나 compact 옵션과 비교하여 jar 파일 사이즈가 더욱 커집니다.
real	일반적으로, 몇몇 메소드들은 설정 규칙에 따라 이름을 치환하지 않습니다. Allatori는 이러한 메서드의 이름을 가져와서 이름이 변경된 메서드에 제공함으로써 새 이름과 원래 이름의 차이가 명확하지 않습니다. 다른 메소드 이름 지정 옵션과 결합 할 수 있습니다. (이름이 부족하면 두 번째 옵션이 사용됩니다).
custom(filename.txt)	파일에 명시된 사용자 지정 규칙으로 패키지 이름을 변경합니다. 규칙이 명시된 파일명을 인자로 받습니다. 파일에 0과 1이 두개의 값이 한줄당 하나씩 있다면, 이름 변경 규칙은 '0', '1', '00', '01', '10', '11', '000' 등과 같습니다.
unique	unique-renaming 속성을 바로 적용하는 옵션입니다. 다른 네이밍 옵션과 함께 사용가능합니다. 만약 두개의 메소드가 동일한 이름과 시그니처를 가지고 있다면 이들 메소드는 동일한 새로운 이름을 가질 것입니다. 만약 어떤 두 메소드가 서로 다른 이름과 시그니처를 가진다면 이들 메소드는 서로 다른 이름으로 치환됩니다. 이것은 <a href="#">5.7.6 순차적 증분 난독화</a> (see page 1)의 견고함을 보장합니다.

사용 예제:

```
<property name="methods-naming" value="abc"/>
```

#### 5.7.4.6 fields-naming

Value	Description
compact	(기본 값) 이름들을 가능한 문자 하나로 치환하여 만들어지는 jar 파일의 사이즈를 줄입니다.
iii	이름이 같은 길이와 서로 다른 다른 대소문자를 가지는 iiiii, iiiii, iiiii 등 과 같이 변경됩니다. 다른 치환 옵션에 비해 jar 결과물의 사이즈가 커질 수 있습니다.

Value	Description
abc	'a', 'b', 'c', 'd', ..., 'aa', 'ab' 등과 같이 오직 소문자 문자로만 이루어지도록 이름이 변경됩니다.
ABC	필드명이 'A', 'B', 'C', 'D', ..., 'AA', 'AB' 등 과 같이 모두 대문자로만 구성됩니다.
123	'1', '2', '3', ..., '00', '01', 등과 같이 숫자값으로 치환됩니다.
keywords	자바에서의 예약어('if', 'for', 'int' 등)를 메소드 이름으로 사용합니다. 런 이름들은 class 파일 포맷에서는 괜찮지만 많은 디컴파일러에게는 혼동을 일으킵니다. 그러나 compact 옵션과 비교하여 jar 파일 사이즈가 더욱 커집니다.
real	일반적으로, 몇몇 필드들은 설정 규칙에 따라 이름을 치환하지 않습니다. Allatori는 이러한 메서드의 이름을 가져와서 이름이 변경된 메서드에 제공하기에 새 이름과 원래 이름의 차이가 명확하지 않습니다. 다른 메소드 이름 지정 옵션과 결합 할 수 있습니다 (이름이 부족하면 두 번째 옵션이 사용됩니다).
custom(file name.txt)	파일에 명시된 사용자 지정 규칙으로 패키지 이름을 변경합니다. 규칙이 명시된 파일명을 인자로 받습니다. 파일에 0과 1이 두개의 값이 한줄당 하나씩 있다면, 이름 변경 규칙은 '0', '1', '00', '01', '10', '11', '000' 등과 같습니다.
unique	모든 필드 이름은 고유한 이름을 가지게 됩니다. 다른 네이밍 옵션과 함께 사용가능합니다.

사용법:

```
<property name="fields-naming" value="iii"/>
```

#### 5.7.4.7 classes-naming-prefix

Value	Description
any string	모든 치환되는 클래스 이름앞에 명시한 문자열을 새로운 이름 앞에 추가 합니다.

사용법:

```
<property name="classes-naming-prefix" value="c_"/>
```

앞 첨자로서 "MainClass\$"를 사용할 수 있습니다.

```
<property name="classes-naming-prefix" value="MainClass$"/>
```

일부 디컴파일러는 치환된 클래스를 MainClass의 이너클래스로 인식할 것입니다.



#### 5.7.4.8 methods-naming-prefix

Value	Description
any string	모든 치환되는 메소드 이름앞에 명시한 문자열을 새로운 이름 앞에 추가 합니다.

사용법:

```
<property name="methods-naming-prefix" value="m_"/>
```

#### 5.7.4.9 fields-naming-prefix

Value	Description
any string	모든 치환되는 필드 이름앞에 명시한 문자열을 새로운 이름 앞에 추가 합니다.

사용법:

```
<property name="fields-naming-prefix" value="f_"/>
```

#### 5.7.4.10 local-variables-naming

Value	Description
optimize	(기본값) Allatori는 메소드내의 로컬 변수의 수를 줄이기 위한 최적화를 수행합니다. 이름이 바뀐 로컬 변수들은 같은 이름을 가집니다. (single-name 옵션 설정시) 이것은 기본값으로 요구되어지는 옵션입니다.
single-name	대부분의 모든 로컬 변수들이 같은 이름으로 치환합니다. 이것은 자바 가상 머신은 괜찮지만 대부분의 디컴파일러에 혼동을 줄 수 있습니다.
abc	로컬 변수들은 'a', 'b', 'c', 'd', 등 유니크한 이름으로 치환됩니다.
remove	로컬 변수의 원본 이름을 삭제합니다. 결과물인 jar 파일의 사이즈를 줄일 수 있습니다.
keep-parameters	모든 다른 로컬 변수값은 변할지라도 매개변수의 이름은 변경되지 않습니다. 이것은 공용 API로 사용되는 메소드들에 유용합니다. keep-names 섹션의 method 태그를 사용하여 지정된 메소드만 매개변수를 유지할 수도 있습니다.

Value	Description
keep	모든 로컬 변수의 이름을 변경하지 않습니다. 이 옵션은 권장하지 않습니다.

사용 예시:

```
<property name="local-variables-naming" value="single-name"/>

<!-- single-name 와 optimize 옵션의 기본값인 문자는 'a' 입니다. 여러분은 아래와 같이 변경할 수 있습니다. -->

<property name="local-variables-naming" value="optimize:임의의 어떤 값"/>

<property name="local-variables-naming" value="optimize:int"/>

<property name="local-variables-naming" value="single-name:4"/>
```

#### 5.7.4.11 skip-renaming

Value	Description
disable	(기본값) keep-names 의 규칙에 따라 클래스 메소드 필드의 이름 치환을 진행합니다.
enable	모든 클래스, 메소드, 필드 값을 바꾸지 않습니다. 로컬 변수 치환은 <b>local-variables-naming</b> 속성에 따라 따로 처리 됩니다. 문자열 암호화, 흐름 난독화 등은 일반적으로 설정파일에 설정된 값에 따라 적용됩니다.

사용 예시:

```
<property name="skip-renaming" value="enable"/>
```

#### 5.7.4.12 update-resource-names

Value	Description
disable	(기본값) 리소스 파일 이름을 바꾸지 않습니다.
enable	리소스 파일은 클래스 이름의 변경 사항을 반영하도록 이름이 변경됩니다. 만약 리소스 파일명이 클래스 이름에 기반하고 그 클래스의 이름이 변경된다면 리소스 파일의 이름도 변경될 것입니다.

사용 예시:

```
<property name="update-resource-names" value="enable"/>
```

#### 5.7.4.13 update-resource-contents

Value	Description
disable	(기본 값) 리소스 파일 내용이 바뀌지 않습니다.
enable	Resource contents will be updated to reflect changes in class names. 리소스 내용은 클래스 이름의 변경 사항을 반영하도록 업데이트 됩니다.
enable:ENCODING	지정한 캐릭터 인코딩으로 리소스 내용은 클래스 이름의 변경 사항을 반영하도록 업데이트 됩니다. 기본값은 UTF-8 입니다.

사용 예시:

##### update-resource- contents

```
<property name="update-resource-contents" value="enable"/>

<property name="update-resource-contents" value="enable:UTF-8"/>

<!-- apply2file 특성 값 설정을 통해 특정 파일들에만 적용할 수 있습니다. -->

<property name="update-resource-contents" value="enable" apply2file="*.xml"/>
```

NSHC  
SECURITY

## 5.7.5 기타 속성들

### 5.7.5.1 line-numbers

Value	Description
obfuscate	<p>(기본값) 디버그 정보는 난독화되어 추가 변환 없이는 사용할 수 없습니다. Allatori는 특별한 유틸리티를 가지고 있습니다.</p> <p>Allatori에는 난독화 된 기능을 사용하여 원래 스택 추적을 재구성 할 수있는 특수 유틸리티가 있습니다.</p> <p>아래와 같이 스택 트레이스가 발생하였을 때에</p> <pre>java.lang.NullPointerException     at com.company.c.a(m:61)     at com.company.b.b(w:94)     at com.company.b.a(w:83)     at com.company.a.a(n:75)</pre> <p>아래와 같이 "Allatori Stack Trace Utility" 를 사용하여 복구할 수 있습니다.</p> <pre>java.lang.NullPointerException     at com.company.Util.createTestException(Util.java:38)     at com.company.TraceTest.testNullObject(TraceTest.java:53)     at com.company.TraceTest.allTraceTests(TraceTest.java:14)     at com.company.Main.runTest(Main.java:27)</pre> <p>이 옵션을 활성화 할 경우 jar 사이즈를 더 줄일 수 있습니다.</p>
remove	<p>이 옵션은 응용 프로그램의 크기가 정말 중요한 경우 최적화를 위해 사용할 수 있습니다. 스택 트레이스는 아래와 같이 보일 것입니다.</p> <pre>java.lang.NullPointerException     at com.company.c.a(Unknown Source)     at com.company.b.b(Unknown Source)     at com.company.b.a(Unknown Source)     at com.company.a.a(Unknown Source)</pre>

Value	Description
keep	<p>수정하는 것 없이 디버그 정보를 유지합니다. 디버그시 이 옵션은 활성화 하는게 좋습니다. 그렇지 않을 경우 다른 옵션을 선택하는 것이 보안에 더 유리합니다.</p> <p>아래와 같이 스택 트레이스가 보여집니다.</p> <pre>java.lang.NullPointerException     at com.company.c.a(Util.java:38)     at com.company.b.b(TraceTest.java:53)     at com.company.b.a(TraceTest.java:14)     at com.company.a.a(Main.java:27)</pre>

사용법:

```
<property name="line-numbers" value="obfuscate"/>
```

### 5.7.5.2 generics

Value	Description
keep	(기본값) 리플렉션을 사용하여 제네릭 형식을 결정하거나 난독화 된 jar를 라이브러리로 사용하여 다른 클래스를 컴파일 해야하는 경우 제네릭 형식을 유지해야 합니다.
remove	<p>제네릭 타입 정보를 지웁니다. 예를 들어 Vector&lt;String&gt; 를 Vector 와 같이 표시하여 제네릭으로 표시된 타입값을 제거합니다.</p> <p>이것은 성능에 영향을 주지 않으며 결과 파일(jar)의 크기를 더욱 작게 만들어 주므로 권장되는 옵션값입니다.</p>

사용법:

```
<property name="generics" value="remove"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

아래 예제를 참고 하세요.

```
<!-- com.abc 패키지 내의 클래스들의 제네릭 타입 정보를 모두 유지 -->
<property name="generics" value="keep" apply2class="class com.abc.*"/>
<!-- 모든 다른 클래스의 제네릭 타입 정보를 지웁니다. -->
<property name="generics" value="remove"/>
```

### 5.7.5.3 inner-classes

Value	Description
keep	(기본값) 자바 컴파일러는 이너 클래스 이름에 이너 클래스라는 정보를 나타내는 속성을 추가 합니다. 이 속성을 클래스 난독화시에 유지 시킵니다.
remove	정보 속성이 제거되고 클래스 계층 구조를 복원하기가 더 어려워집니다. 성능에 영향을 미치지 않고 jar 파일을 더 작게 만들기 때문에 권장되는 옵션입니다.

사용법:

```
<property name="inner-classes" value="remove"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

사용 예제 :

```
<!-- com.abc 패키지내 클래스의 이너클래스 정보 유지 -->
<property name="inner-classes" value="keep" apply2class="class com.abc.*"/>
<!-- 그외 모든 클래스의 이너클래스 정보 삭제 -->
<property name="inner-classes" value="remove"/>
```

### 5.7.5.4 member-reorder

Value	Description
enable	(기본값) 일반적으로, 개발자는 연관된 메소드와 필드를 차례로 소스파일에 위치시킵니다. 이러한 순서는 컴파일 후에도 유지됩니다. Allatori는 이러한 필드와 메소드의 위치를 흐트러트립니다.
random	enable과 동일 합니다.
alphabetic	필드와 메소드들은 알파벳의 순서로 정렬되어 배치됩니다.
reverse-alphabetic	필드와 메소드들은 알파벳의 역순서로 정렬되어 배치됩니다.
disable	멤버값에 대한 재정렬을 비활성화 합니다.

사용법:

```
<property name="member-reorder" value="enable"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

아래 예제를 참고 하시길 바랍니다.

```
<!-- com.abc 패키지의 클래스 멤버들을 재정렬 합니다. -->

<property name="member-reorder" value="random" apply2class="class com.abc.*"/>

<!-- 그외 모든 클래스의 멤버를 재정렬 하지 않습니다. -->

<property name="member-reorder" value="disable"/>
```

### 5.7.5.5 finalize

Value	Description
disable	(기본값) Class의 finalizing을 비활성화 합니다.
enable	subclass가 없는 Class 들을 명시적으로 final로 선업합니다. 이 기능은 단독앱의 난독화를 위해서만 사용되어야 합니다. 기능 활성화시 좀더 빠르게 앱이 동작하도록 만들어 줍니다.

사용법:

```
<property name="finalize" value="enable"/>
```

### 5.7.5.6 version-marker

Value	Description
자바에서 허용되는 식별자 이름 형식의 문자열	Allatori는 이름이 변경된 메서드 및 필드의 이름으로 주어진 식별자를 사용합니다. 이것은 난독화 된 클래스 파일을 표시합니다. 제품의 데모 버전을 표시하는 데 사용할 수 있습니다. 예를 들어 Allatori 데모 버전은 "ALLATORI_DEMO" 문자열을 표시됩니다. Allatori 의 데모 버전은 난독화 된 jar를 표시하고 이 속성 값인 "ALLATORI_DEMO_"를 모든 항목에 추가합니다.

사용법:

```
<property name="version-marker" value="THIS_IS_DEMO_VERSION"/>
```

### 5.7.5.7 synthesize-methods

Value	Description
private	(기본값) 모든 private 메소드는 합성으로 표시됩니다.
all	모든 메소드는 합성으로 표시됩니다.
package	모든 package 식별자 메소드는 합성으로 표시됩니다.
protected	모든 protected 메소드는 합성으로 표시됩니다.
public	모든 public 메소드는 합성으로 표시됩니다.
disable	메소드를 합성으로 표시하지 않습니다.

몇몇 디컴파일러는 합성 메소드를 출력하지 않습니다. 이러한 점을 이용하여 메소드들을 합성 메소드로 표시되게 합니다.

사용법:

```
<property name="synthesize-methods" value="all"/>

<!-- 이 속성은 하나 이상을 사용할 수 있습니다: -->
<property name="synthesize-methods" value="private"/>
<property name="synthesize-methods" value="package"/>
<property name="synthesize-methods" value="protected"/>

<!-- "apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다
. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다. -->
<property name="synthesize-methods" value="all" apply2class="class com.abc.*"/>
<property name="synthesize-methods" value="private"/>
```

### 5.7.5.8 synthesize-fields

Value	Description
disable	(기본값) 모든 필드를 합성으로 표시하지 않습니다.
all	모든 필드를 합성으로 표시됩니다.
private	모든 private 필드들이 합성으로 표시됩니다.
package	모든 package 식별자 필드들이 합성으로 표시됩니다.



Value	Description
protected	모든 protected 필드들이 합성으로 표시됩니다.
public	모든 public 필드들이 합성으로 표시됩니다.

몇몇 디컴파일러들은 합성 필드를 출력하지 않습니다.  
사용법:

```
<property name="synthesize-fields" value="all"/>

<!-- 이 속성은 하나 이상을 사용할 수 있습니다: -->
<property name="synthesize-fields" value="private"/>
<property name="synthesize-fields" value="package"/>
<property name="synthesize-fields" value="protected"/>

<!-- "apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다
. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다. -->
<property name="synthesize-fields" value="all" apply2class="class com.abc.*"/>
<property name="synthesize-fields" value="private"/>
```

#### 5.7.5.9 set-methods-to-public

Value	Description
disable	(기본값) 알라토리는 메소드를 public 으로 표시하지 않습니다.
all	모든 메소드를 public 으로 표시하게 합니다. private 메소드를 public 으로 표시하게 될 경우 어플리케이션의 기능이 중단될 수 있으니주의해야 합니다.
private	모든 private 메소드를 public 으로 표시하게 합니다. private 메소드를 public 으로 표시하게 될 경우 어플리케이션의 기능이 중단될 수 있으니 주의해야 합니다.
package	모든 package 공개 메소드를 public 으로 표시합니다.
protected	모든 protected 메소드를 public으로 표시 합니다.

예시:

```
<property name="set-methods-to-public" value="all"/>
```

해당 프로퍼티는 아래와 같이 하나 이상 사용가능합니다.

```
<property name="set-methods-to-public" value="package"/>
<property name="set-methods-to-public" value="protected"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

아래 예제를 참고 하시길 바랍니다.

```
<property name="set-methods-to-public" value="all" apply2class="class com.abc.*"/>
<property name="set-methods-to-public" value="protected"/>
```

### 5.7.5.10 set-fields-to-public

Value	Description
disable	(기본값) 알라토리는 필드를 public 으로 표시하지 않습니다.
all	모든 필드를 public 으로 표시하게 합니다.
private	모든 private 필드를 public 으로 표시하게 합니다.
package	모든 package 공개 필드를 public 으로 표시합니다.
protected	모든 protected 필드를 public으로 표시 합니다.

예시:

```
<property name="set-fields-to-public" value="all"/>
```

해당 프로퍼티는 아래와 같이 하나 이상 사용가능합니다.

```
<property name="set-fields-to-public" value="package"/>
<property name="set-fields-to-public" value="protected"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

아래 예제를 참고 하시길 바랍니다.

```
<property name="set-fields-to-public" value="all" apply2class="class com.abc.*"/>
<property name="set-fields-to-public" value="protected"/>
```

### 5.7.5.11 remove-toString

Value	Description
disabl e	(기본값) toString 메소드를 지우지 않습니다.
enabl e	난독화된 클래스 안의 toString() 메소드를 삭제 합니다. toString 메소드는 클래스의 몇몇 정보를 드러낼수 있습니다. 이러한 것은 디버깅시에만 종종 사용합니다. 그러므로 이러한 정보를 삭제할 수 있습니다.

사용법:

```
<property name="remove-toString" value="enable"/>
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

예시:

```
<property name="remove-toString" value="enable" apply2class="class com.abc.*"/>
<property name="remove-toString" value="enable" apply2class="class com.xyz.*"/>
```

### 5.7.5.12 remove-calls

Value	Description
ClassName.method Name	특정 메소드 호출을 지울 수 있습니다. 디버그 로깅 호출을 삭제하는데 유용합니다. ClassName과 methodName 은 '*' 을 사용하여 매칭되는 여러개의 클래스 및 메소드들 을 포함할 수 있습니다. 메서드에 반환 값이 있는 경우 반환 값은 null(0, false)로 바꾸고 Allatori는 난독화 중에 경고를 출력합니다.

사용법:

```
<property name="remove-calls" value="android.util.Log.d"/>
<property name="remove-calls" value="android.util.Log.*"/>
```

```
// if a call to methodOne() is removed, then the line
int i = methodOne();
// would be changed to
int i = 0;

// if a call to methodTwo() is removed, then the line
Object o = methodTwo();
// would be changed to
Object o = null;
```

"apply2class" 속성을 사용하여 명시적으로 클래스를 지정하여 속성을 설정할 수 있습니다. 'apply2class' 은 앞선 'template' 속성과 동일한 사용 포맷을 가지고 있습니다.

예시:

```
<!-- com.abc 패키지 아래 모든 클래스에서 호출되는 Logger.debug 을 삭제 -->
<property name="remove-calls" value="com.package.Logger.debug" apply2class="class
com.abc.*"/>
```

### 5.7.5.13 remove-annotations

Value	Description
AnnotationClassName	Allatori는 AnnotationClassName 주석을 제거합니다. 불필요한/디버그 주석을 제거하는데 사용할 수 있습니다. AnnotationClassName은 여러 클래스와 일치하도록 *를 사용할 수 있습니다.

예제:

```
<property name="remove-annotations" value="kotlin.Metadata"/>
<property name="remove-annotations" value="com.package.annotations.*"/>
```

apply2class 속성을 사용하여 지정된 클래스에 적용할 수 있습니다. apply2class 속성은 class 태그의 template 속성과 같은 형식을 가집니다.

다음은 예입니다.

```
<!-- Removing com.package.Annotation annotations from classes in com.abc package -->
<property name="remove-annotations" value="com.package.Annotation" apply2class="class
com.abc.*"/>
```

### 5.7.5.14 output-jar-compression-level

Value	Description
0-9	jar 파일 압축 레벨을 설정합니다. 0은 압축을 하지 않고, 9은 가장 높은 값입니다. 해당 값은 <code>ZipOutputStream.setLevel(int)</code> <sup>3</sup> 에서 정의 되는 값과 동일합니다.

사용법:

```
<property name="output-jar-compression-level" value="9"/>
```

### 5.7.5.15 output-jar-duplicate-name-entries

Value	Description
remove	(기본값) 만약 입력되는 jar 가 폴더안에 여러개의 같은 이름의 파일들이 포함된다면 Allatori는 오직 하나의 파일만을 출력 jar 파일에 포함 시킵니다.
keep	Allatori는 모든 중복된 이름의 jar 항목을 출력 jar에 기록합니다. Zip (jar) 파일 형식은 중복 이름 항목을 허용하지만 표준 Java API는 이러한 항목 작성을 허용하지 않습니다. 이를 극복하기 위해 <code>java.util.zip.ZipOutputStream.names</code> 비공개 필드에 액세스해야하므로 Allatori를 실행할 때 <code>--illegal-access = permit</code> 옵션으로 JVM을 실행해야 할 수도 있습니다.

사용법:

```
<property name="output-jar-duplicate-name-entries" value="keep"/>
```

<sup>3</sup> <https://docs.oracle.com/javase/8/docs/api/java/util/zip/ZipOutputStream.html#setLevel-int->

## 5.7.6 증분 난독화 속성 설정

### 5.7.6.1 incremental-obfuscation

Value	Description
이전에 생성된 log 파일 이름	<p>이전 Allatori 동작으로 생성된 log 파일 이름을 설정합니다. 상대 경로는 설정 파일 위치에 의해 확인됩니다.</p> <p>증분 난독화는 귀사의 어플리케이션에 패치 또는 애드온을 만들 필요가 있을때 사용합니다. 이 경우 새로운 클래스, 메소드, 필드의 이름이 이전의 난독화된 버전의 것과 일치하는 지를 보증할 필요가 있습니다.</p> <p>이전에 생성된 난독화 로그 파일을 다음 버전의 난독화를 위해 사용함으로써 완벽하게 두개의 릴리즈가 호환 되므로, 패치 또는 애드온은 이전의 어플리케이션에 오류없이 통합될 수 있습니다.</p> <p><u>그러므로 증분 난독화를 사용할 때에는 일부만 배포 될지라도 반드시 모든 클래스들을 난독화 프로세스에 포함해야만 합니다.</u></p>

사용법:

```
<property name="incremental-obfuscation" value="input-renaming-log.xml"/>
```

### 5.7.6.2 unique-renaming

Value	Description
disable	(기본값) 유니크한 이름으로의 치환을 비활성화 함.
enable	<p>만약 어떤 두개의 메소드들이 동일한 이름과 시그니처를 가진다면, 이들 메소드는 동일한 새로운 이름으로 치환될 것입니다.</p> <p>만약 어떠한 두개의 메소드들이 다른 이름과 시그니처를 가지고 있다면 서로 다른 이름으로 치환 될 것입니다.</p> <p>이것은 후속 증분 난독화에서의 일괄성을 보증합니다.</p>

사용법:

```
<property name="unique-renaming" value="enable"/>
```

apply2class 속성을 사용하여 지정된 클래스에 적용할 수 있습니다. apply2class 속성은 class 태그의 template 속성과 같은 형식을 가집니다.

예:

```
<property name="unique-renaming" value="enable" apply2class="class com.abc.*"/>
```

## 5.8 Ignore-classes 태그

ignore-classes 태그는 난독화 프로세스로부터 특정 클래스들을 완벽히 예외 처리하고자 할 때에 사용됩니다.

이들 클래스 들은 변경된 것이 없는 이전 상태 그대로 결과물이 복사됩니다.

무시 처리된 클래스들은 그들 자신의 이름으로 다른 클래스에서 참조될 것입니다.

무시를 통해 예외처리한 클래스에서 참조되는 클래스/메소드의 이름은 바꿀 수 없습니다.

이 태그는 keep-name 에서 사용하는 class 태그를 포함하며 적용 규칙도 동일 합니다.

사용법:

```
<ignore-classes>  
  <class template="class com.company.abc.*"/>  
  <class template="class com.company.xyz.SomeClass"/>  
</ignore-classes>
```



NSHC  
S E C U R I T Y

## 6. 어노테이션(Annotations)

어노테이션 클래스들은 allatori-annotations.jar 에 있습니다. 모든 Allatori 어노테이션들은 난독화 프로세스 중에 제거됩니다.

그러므로 이 jar 파일은 런타임시에 필요하지 않습니다. 어노테이션은 난독화 설정을 좀더 쉽고 정확하게 설정하는 용도로 사용됩니다.

com.allatori.annotations 패키지에 있는 모든 사용 가능한 어노테이션 값은 아래와 같습니다.

Annotation	Applicable to	Description
Rename	classes, methods and fields	Allatori에게 주석이 달린 요소의 이름을 변경하도록 지시합니다. 설정 파일 설정을 무시합니다.
DoNotRename	classes, methods and fields	Allatori에게 주석이 달린 요소의 이름을 변경하지 않도록 지시합니다. 설정 파일 설정을 무시합니다.
StringEncryption	classes	가능한 값: StringEncryption.ENABLE StringEncryption.DISABLE StringEncryption.MAXIMUM StringEncryption.MAXIMUM_WITH_WARNINGS 기존 <b>string-encryption</b> 설정을 무시합니다. 예제: <code>@StringEncryption(StringEncryption.ENABLE)</code>
StringEncryptionType	classes	가능한 값: StringEncryptionType.FAST StringEncryptionType.STRONG 기존 <b>string-encryption-type</b> 설정을 무시합니다. 예제: <code>@StringEncryptionType(StringEncryptionType.STRONG)</code>
ControlFlowObfuscation	classes and methods	가능한 값: ControlFlowObfuscation.ENABLE ControlFlowObfuscation.DISABLE 기존 <b>control-flow-obfuscation</b> 설정을 무시합니다. 예제: <code>@ControlFlowObfuscation(ControlFlowObfuscation.ENABLE)</code>



Annotation	Applicable to	Description
ExtensiveFlowObfuscation	classes and methods	<p>가능한 값:</p> <p>ExtensiveFlowObfuscation.DISABLE</p> <p>ExtensiveFlowObfuscation.NORMAL</p> <p>ExtensiveFlowObfuscation.MAXIMUM</p> <p>기존 <b>extensive-flow-obfuscation</b> 설정을 무시합니다.</p> <p>예제:</p> <p>@ExtensiveFlowObfuscation(ExtensiveFlowObfuscation.MAXIMUM)</p>



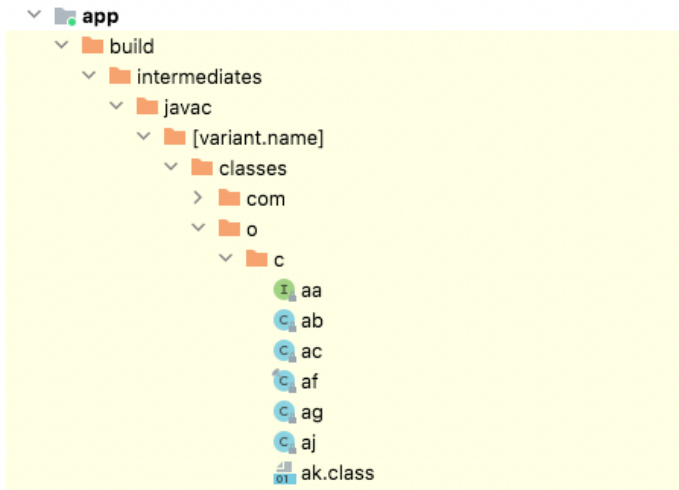
**NSHC**  
S E C U R I T Y

## 7. 디버깅 가이드

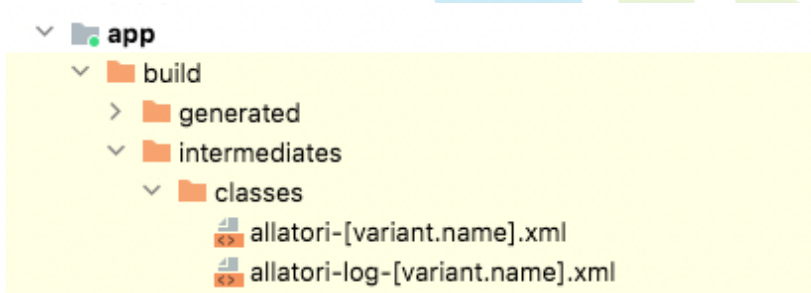
### 7.1 난독화 후 생성되는 파일

당사에서 제공하는 기본 설정값에 의하면 난독화 적용 후 Build를 한 결과 아래와 같은 경로에 특정 파일이 생성됩니다.

#### 1) 난독화 결과물



#### 2) 난독화 로그 파일



### 7.2 디버깅 프로세스 가이드

#### 1) 난독화 적용 후 발생하는 로그정보의 line number를 유지합니다.

Allatori.xml 에서 아래와 같이 추가합니다.

```
<property name = "line-numbers" value = "keep"/>
```

위 내용으로 설정하면 StackTrace에서 표시되는 라인 값을 유지합니다. 이 옵션은 응용 프로그램의 내부 테스트에 유용할 수 있습니다.

적용시 스택 추적은 다음과 같이 식별 가능하게 변경됩니다.

```
java.lang.NullPointerException
    com.company.ca (Util.java:38)
    com.company.bb (TraceTest.java:53)
    com.company.ba (TraceTest.java:14)에서
```

```
com.company.aa (Main.java:27)에서
```

2) 난독화 설정 변경시 프로젝트를 Clean후 Rebuild하여 Allatori를 다시 적용합니다.

3) Logcat에서 발생한 에러를 확인 후, allatori-log-[variant.name].xml에서 변경된 이름을 식별합니다.

allatori-log-[variant.name].xml에서는 난독화 관련 통계가 제공되며,

old와 new 속성 값을 통해 난독화 과정 중 달라진 이름에 대한 정보를 제공합니다.

4) 해당하는 클래스, 메소드, 필드를 예외처리합니다.

그 외 추가적인 조치 방법은 8 (see page 1).4 난독화 적용 후 발생한 에러에 대해서는 어떻게 해야하는지요? (see page 49) 항목 참고



NSHC  
S E C U R I T Y

## 8. 자주 묻는 질문

### 8.1 난독화 적용 시 반드시 예외처리 해야 하는 부분이 있는지요?

소스코드 난독화 적용 시 정상적인 앱 실행을 위해 반드시 "예외처리"가 필요한 부분들이 있습니다.

이는 모든 Android용 소스코드 난독화 솔루션에 해당하는 부분이며,

만약 이 부분에 예외처리가 되지 않을 시에는 앱 실행 시 크래시가 발생하거나 아예 실행이 되지 않을 수 도 있습니다.

[정상적인 앱 실행을 위해 예외처리가 필요한 부분]

- 최초 앱 실행 시 진입하는 Intro
- 3rd Party library I/F 혹은 서로 연결되는 지점
- NDK (Native library) I/F
- Java Reflection을 사용해 호출되는 class 나 method
- AIDL(서비스 간 통신을 위한 I/F) 관련 클래스들
- 서버와 통신하는 클래스들
- Activity를 상속받는 클래스 중 View를 전달받는 method들
- 화면 처리를 위해 View클래스를 상속받는 클래스들
- 리소스 세팅하는 클래스들
- Context 혹은 Activity를 상속받는 클래스 중 View를 전달받는 method들
- 자바스크립트를 사용하기 위해 annotation type지정된 method들
- Data 전달을 위해 Parcelable을 상속받는 클래스들

### 8.2 InnerClass들에 대한 예외 처리 방법이 있는지요?

InnerClass들은 그 이름에 \$기호가 들어가기에 다음과 같은 형태로 예외처리가 가능합니다.

상위클래스 이름 : Pragment.class

이너클래스 이름 : AndroidBrige.class

상위 클래스와 내부 클래스 사이에 \*(와일드카드)로 설정하여 예외처리 가능합니다.

```
<class template="class com.android.Pragment*AndroidBridge">
```

### 8.3 오픈 소스 라이브러리를 예외 처리 방법을 알려주세요.

오픈 소스 (ex. butterknife) 예외 처리를 위해서는 allatori.xml에 아래와 같은 룰을 추가하시면 됩니다.

```
<class template="public class * instanceof butterknife.*"><field access="private+"/><method template="private+ *(*)"/></class>
```

```
<class template="public class butterknife.*"><field access="private+"/><method template="private+ *(*)"/></class>
```

### 8.4 난독화 적용 후 발생한 에러에 대해서는 어떻게 해야 하는지요?

#### 8.4.1 난독화 적용 후 Build 중 에러가 발생한 경우

Build를 다시 실행 하기전에, 우선 아래와 같은 상황인지 확인해 보시기 바랍니다.

- (1) Allatori와 ProGuard 혹은 DexGuard를 중복 사용하는 경우 에러가 발생할 수 있습니다. 반드시 ProGuard나 DexGuard를 삭제한 이후 진행하시기 바랍니다.
- (2) 사용중인 JDK 버전이 1.7 이상인지 확인하십시오. 최소지원 JDK버전은 1.7입니다.
- (3) JDK1.6에서 빌드된 android 라이브러리가 포함될 경우 문제가 될 수 있습니다.
- (4) JDK 1.6으로 구축한 상태에서 JDK 1.7에서 빌드 된 라이브러리를 프로젝트에 import해서 사용하고 있는 경우

그 외 발생할 수 있는 일반적인 에러와 해결 방법은 다음과 같습니다.

- (1) "Process 'command' .../Home/bin/java" finished with non-zero exit value 255.
  - 원인: xml 또는 gradle에서 잘못된 문법으로 작성된 코드로 인해 발생함.
  - 조치: 올바른 문법으로 수정.
- (2) Build Output의 compileDebugKotlin 중 Unresolved reference: YourClass.
  - 원인: 코틀린 컴파일 중 타 패키지/모듈에 있는 난독화된 코틀린 클래스의 이름을 인식하지 못해 발생함.
  - 조치: 해당 코틀린 클래스의 이름을 예외조치에 추가.
- (3) Build Output의 kaptDebugKotlin 중 static import only from classes and interfaces.
  - 원인: 자바 컴파일 중 난독화된 코틀린 annotation 클래스의 이름을 인식하지 못해 발생함.
  - 조치: 해당 annotation 클래스를 예외조치에 추가.
- (4) Build Output의 compileDebugJavaWithJavac 중 error.
  - 원인: 자바 컴파일 중 난독화된 자바 클래스의 이름을 인식하지 못해 발생함.
  - 조치: 해당 자바 클래스의 이름을 예외조치에 추가.
- (5) Build Output의 mergeDexDebug 중 error.
  - 원인: 멀티모듈을 난독화 할 때 클래스 패키지 path가 중복되어 발생함.
  - 조치: path가 중복되지 않게 "o.a", "o.b" ... 로 xml에서 default-package를 변경.

#### 8.4.2 난독화 적용 후 Runtime 중 에러가 발생한 경우

- (1) FATAL EXCEPTION: ... : java.lang.ClassCastException : ...
  - 원인: 난독화된 클래스로 인해 클래스 형변환(Cast)이 실패하여 발생함.
  - 조치: 해당 클래스 예외처리 진행.
- (2) FATAL EXCEPTION: ... : java.lang.NoSuchMethodError : ...
  - 원인: 기존 난독화 클래스에 난독화가 2중으로 중첩되어 메소드를 찾지 못해 발생함.
  - 조치: 프로젝트 Clean 후, Rebuild.

#### 8.5 난독화 적용 후 임의 화면이 깨지는 현상이 발생합니다.

별도의 로그 없이 화면이 깨지는 현상이 발생할 경우에는 아래와 같이 조치해 주시기 바랍니다.

1. 해당 현상이 발생하는 부분에서 사용하고 있는 외부 라이브러리 예외처리 추가
2. 서버 연동 부분이 있다면 해당 부분 예외처리 추가  
(ex. addJavascriptInterface()를 사용하는 클래스 목록을 확인한 후 해당 클래스를 예외 처리함)

#### 8.6 java.lang.ArrayIndexOutOfBoundsException 오류가 발생합니다.

해당 에러의 원인은 Android Studio 2.1.1 버전의 bug 중 하나로, 해당 버전에서 난독화를 적용하면 발생하는 이슈입니다.

이는 Android Studio 업데이트를 통해 문제 해결이 가능합니다.

즉 Android Studio를 2.1.2 이상으로 업데이트 하시기 바랍니다.

## 8.7 난독화된 로그를 원본으로 되돌릴 수 있는 방법이 있는지요?

Allatori는 난독화된 로그를 원본으로 되돌릴 수 있는 툴을 별도로 제공합니다.

당사에서 제공하는 파일 내 StackTraceRestore 폴더를 참고하시면 되면 각 파일 별 설명은 아래와 같습니다.



- Clean.bat : 생성된 out.txt 파일을 삭제합니다.
- input.txt : 난독화되어 분석이 어려운 로그 파일
- log.xml : 난독화 mapping 정보를 담고 있는 xml 파일
- StackTraceRestore.bat : 원본 로그를 생성하는 파일 (out.txt 파일 생성) 해당 bat 파일을 실행하시면 원복 로그로의 복구가 가능합니다.

### [Retrace 명령 문법]

최신:

```
java -cp Allatori라이브러리 com.allatori.StackTrace2 [매핑로그파일명] [난독화된 StackTrace
저장 파일명] [난독화 복구결과가 저장될 파일명]
```

과거:

```
java -Xms128m -Xmx512m -cp ${AllatoriLib} com.allatori.StackTrace2 ${log} $
${StackTrace} ${Output}
```



- \${AllatoriLib} : Allatori라이브러리 (allatori.jar).
- \${log} : 난독화 수행 중 생성되는 파일(로그 매핑 정보 담고 있음)로, 따로 설정을 변경하지 않을 경우 Project Home에 생성되며 해당 로그 파일의 위치와 이름은 프로젝트 설정에 따라 달라질 수 있습니다.
- Eclipse : allatori.xml 내 property-name="log-file"로 설정됨
- Android Studio : Build Script 내 runAllatori 함수 중 logFile: "allatori-log-\${VARIANT.NAME}.xml"로 설정됨
- \${StackTrace} : StackTrace로그를 저장한파일 (난독화된 로그 파일)
- \${Output} : Retrace결과를 저장할 파일 (난독화된 로그를 원복한 파일)

## 8.8 Android Studio에서 라이브러리를 프로젝트에 allatori를 적용하는 방법이 있는지요?

[4.1 Android Studio 환경]과 동일하며, 아래의 내용을 참조하여 libraryVariants 에서 runAllatori()를 호출하시면 됩니다.

```
android {
    ...
    //apk빌드 시에는 applicationVariants로, 라이브러리 빌드 시에는 libraryVariants로 변경함.
```

```

libraryVariants.all { variant ->
    variant.javaCompile.doLast {
        runAllatori(variant)
    }
}
}

def runAllatori(variant) {
    copy {
        from "$projectDir/allatori.xml"
        into "$buildDir/intermediates/classes/"
        expand(classesRoot: variant.javaCompile.destinationDir,
            kotlinRoot: "${buildDir}/tmp/kotlin-classes/${variant.name}",
            androidJar: "${android.sdkDirectory}/platforms/${
                android.compileSdkVersion}/android.jar",
            classpathJars: variant.javaCompile.classpath.getAsPath(),
            logFile: "allatorilogl${variant.name}.xml")
        rename('allatori.xml', "allatoril${variant.name}.xml")
    }
    new File("${variant.javaCompile.destinationDir}-obfuscated").deleteDir()
    javaexec {
        main = 'com.allatori.Obfuscate'
        classpath = files("$rootDir/allatori/allatori.jar")
        args "$buildDir/intermediates/classes/allatoril${variant.name}.xml"
    }
    new File("${variant.javaCompile.destinationDir}").deleteDir()
    new File("${variant.javaCompile.destinationDir}-obfuscated").renameTo(new File("${
        variant.javaCompile.destinationDir}"))
}
}

```

## 8.9 Allatori를 빌드 환경에서 제거하고 싶습니다.

- **Android Studio**에서 제거하는 방법

### 1) Allatori 난독화 적용 일시 해지 방법

Allatori가 적용된 모든 모듈의 build.gradle 파일에서 난독화를 진행하는 함수인 *runAllatori(variant)* 호출 부분을 주석 처리 합니다.

클린 후 재빌드시 Allatori 적용이 해제된다.

### 2) Allatori 완전 삭제 방법

- “rootDir” 디렉토리 안에 allatori/ 디렉토리를 삭제합니다.
- ‘build.gradle’ 파일에 추가된 내용(runAllatori(variant) 함수 추가 관련 내용)을 삭제합니다.
- “projectDir”(build.gradle 파일이 위치하고 있는 디렉토리) 안에 ‘allatori.xml’ 파일을 삭제합니다.
- **Eclipse**에서 제거하는 방법

해당 프로젝트 목록에서 오른쪽 마우스를 클릭 후, “Configure-> Remove Allatori Builder”를 선택합니다.

## 8.10 난독화 결과물 사이즈에 영향을 주는 요소와 결과물을 최적화 할 수 있는 방법은?

### 1. 난독화 결과물 사이즈에 영향을 주는 요소

동일한 어플의 난독화 output 파일 사이즈가 다른 이유는 아래와 같습니다.

### (1) random-seed 설정 값

기본 값은 milli seconds 단위의 현재 시간이며 만약 특정 random-seed를 지정하고 싶다면 다음과 같이 설정할 수 있습니다.

```
<property name="random-seed" value="any text here"/>
```

이와 같이 random-seed를 특정 값으로 지정한 경우 난독화 결과 사이즈는 매번 동일할 것입니다.

### (2) JVM Version

Allatori는 output jar파일 생성을 위해 "java.util.jar.JarOutputStream"를 사용하는데 이 구현 방식이 자바 버전에 따라 다릅니다.

그로 인해 같은 input일지라도 output 파일 사이즈가 다른 경우가 발생합니다.

### (3) 그 외

- 제어흐름 난독화 및 난독화 시 치환 이름 규칙 중 특정 옵션들은 결과물의 사이즈에 영향을 끼칩니다.

## 2. 난독화 결과물을 최적화하는 방법

### (1) String Encryption 보안 레벨 조정

String Encryption의 버전을 "v3"로 설정하시기 바랍니다.

```
<property name="string-encryption-version" value="v3"/>
```

"v3"를 사용할 경우 String 알고리즘의 복잡도는 조금 낮아지지만 output 사이즈는 줄일 수 있습니다.

### (2) 제어흐름 난독화 옵션 disable

```
<property name="extensive-flow-obfuscation" value="disable"/>
```

대신 이 옵션을 사용할 경우 보안 레벨은 낮아지니 유의하시기 바랍니다.

❗ 일반적으로 최적화를 위해서는 *Optimizing*(사이즈축소) + *Shrinking*(필요없는 클래스삭제)가 지원되어야 눈에 띄는 효과를 얻을 수 있습니다.  
 Allatori의 경우에는 *Optimizing*만을 지원하며 *Shrinking*기능은 지원하지 않습니다.  
 다만, *remove* 옵션을 가지는 속성 값들을 적절히 적용하여 결과물의 사이즈를 추가로 줄이는 것을 시도하실 수 있습니다.

## 8.11 소스코드 난독화 적용 후 unsigned app을 생성하고 싶습니다.

- Signature를 APK에서 직접 지우는 방법

APK파일 안에 서명 값은 다음과 같은 파일로 저장됩니다.

METAINF/MANIFEST.MF

METAINF/CERT.SF

METAINF/CERT.RSA

다음의 명령을 이용하여, 서명 값을 삭제하면 됩니다.

```
$zip ld myapplication.apk METAINF/*
```



- Ant Build를 사용해서 지우는 방법

Ant Build시에 signing키를 명시하지 않고 "ant release"를 사용하여 APK를 만들면 됩니다.

local.properties 라는 파일 안에 다음과 같은 내용들이 있습니다. 이 부분을 전부 삭제하고 빌드하면 됩니다.

```
key.store          = /home/user/.android/debug.keystore
key.store.password = android
key.alias          = AndroiddebugKey
key.alias.password = android
```

## 8.12 @annotation 예외처리 하는 방법

@annotation을 사용하는 클래스를 하기와 예외처리 해야합니다.

아래는 Json에서 어노테이션을 사용하는 클래스를 찾아 예외처리를 한 예시입니다.

```
public class ExtendableBean {
    public String name;
    private Map<String, String> properties;

    @JsonAnyGetter
    public Map<String, String> getProperties(){
        Return properties;
    }
}
```

```
<class template = "class com.android.ExtendableBean">
    <method template = "@JsonAnyGetter public *(**)"/>
</class>
<-- 또는 -->
<class template = "class com.android.ExtendableBean">
    <method template = "public getProperties(**)"/>
</class>
<-- 또는 프로젝트내 일괄 적용을 원할 경우 아래 태그만 추가 -->
<method template = "@JsonAnyGetter public *(**)"/>
```

## 8.13 Proguard rule을 allatori.xml에 변경 적용하는 방법 (예시)

```
# Gson specific classes [프로가드 예시]

-dontwarn com.google.gson.**
-keep class com.google.gson.** {*; }
-keepclasseswithmembers class * {
    @retrofit2.http.* <methods>;
}

#Gson specific classes [Allatori 예시]

<class template="class com.google.gson.**">
    <field access="private+"/>
    <method template="private+ *(**)" />
</class>

<class template="class *" ignore="keep-if-members-match">
    <method template="@retrofit2.http.* *(**)" />
</class>

#특정 패키지 이하 클래스와 내부 필드, 메소드 전체
-keep class com.company.** {
    *;
}

<class template="class com.company.**"> <field access="private+"/><method template="private+ *(**)" /> </class>

#특정 이름의 메소드 기준
-keepclassmembers class * {
    public static void Scanrisk(int, java.lang.String, java.lang.String,
    java.lang.String,java.lang.String);
}

<method template="public static void Scanrisk(**)" />

#특정 클래스와 내부 public 필드와 메소드
-keep class com.signcompany.passlibrary.util.PassResult {
    public <fields>;
    public <methods>;
}

<class template="class com.signcompany.passlibrary.util.PassResult"> <field
access="public"/><method template="public *(**)" /> </class>

#특정 클래스 상속 과 특정 메소드 예외처리
-keepclassmembers class * extends java.lang.Enum {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}
```

```
<class template="class * extends java.lang.Enum">
  <method template="values()"/>
  <method template="valueOf(java.lang.String)"/>
</class>
```

## 8.14 retro lamda를 사용하는데 gradle 3.x에서 난독화 적용 시 에러

Retro lamda와 Desugar 간에 충돌이 발생할 수 있습니다.

gradle 3.0부터 제공하는 Desugar 기능을 아래와 같이 설정한 후 적용하시면 됩니다.

1. Java 1.8 옵션 제거
2. Desugar 옵션 false 처리
  - Gradle 설정 파일에서 android.enableDesugar = false 추가
3. sync
4. 다시 retro lamda 추가
5. 빌드

## 8.15 난독화를 적용 후 빌드하면 TransformException 발생과 duplicate entry: o/a.class 에러

난독화를 적용하는 모듈이 2개 이상일 때 난독화된 클래스 패키지 path가 중복되어 발생 할 수 있습니다.

Path가 중복되지 않도록 아래와 같이 설정하여야 합니다.

```
<!-- Application에서 설정된 allatori.xml -->
<property name="default-package" value="o.a"/>

<!-- Third party library(모듈)에서 설정된 allatori.xml -->
<property name="default-package" value="o.b"/>
```

## 8.16 난독화를 적용할 때 문자열 비교를 위한 equals 함수와 "==" 의 차이점

경우에 따라 문자열을 다음과 같이 비교합니다.

```
String myString = "Hello";

...

public boolean test () {

    return myString == "Hello";

}
```

equals 메소드 대신 == 연산자를 사용하여 문자열을 비교하는 것은 좋지 않지만

JVM이 String 객체를 캐시하여 동일한 클래스 내에서 다시 사용할 수 있기 때문에 위 예의 메소드는 true를 반환합니다.

그러나 문자열 암호화 후에 메소드는 다음과 같습니다.

```
public boolean test () {
    return myString == 새 문자열 ( "Hello");

    //이 예제를 명확하게 설명하기 위해 "Hello"문자열을 암호화 내용으로 보여주지 않았습니다.
}
```

이 버전의 메서드에서 사용된 == 연산자는 비교된 객체(Object)가 다르기 때문에 false를 반환합니다.

따라서 문자열을 비교 시 객체의 내용을 비교할 수 있도록 equals 메소드 사용을 권고드립니다.

하지만 써드 파티 코드 등 문자열 비교 시 “==” 사용이 있을 수 있다고 예상된다면,

```
<property name="string-encryption" value="enable "/>
```

로 설정하시기 바랍니다.

## 8.17 난독화 적용시 Android Studio의 Instant Run 기능 옵션 설정 해제

난독화를 적용할 때 해당 옵션은 해제하는 것을 권장합니다.

아래는 해제 하는 방법에 대한 내용입니다.

1. File -> Settings 로 이동.
2. 검색에서 Instant 입력 시 맨 하단에 Instant Run 클릭
3. 해제 설정 후 확인.

## 8.18 Build Type을 “Release”에서만 난독화가 적용 되도록 gradle 설정 방법

Gradle에서 설정파일을 아래와 같이 수정하면 release에서만 난독화가 적용됩니다.

```
#AS-IS
applicationVariants.all { variant ->
    variant.javaCompile.doLast {
        runAllatori(variant)
    }
}

#TO-BE
applicationVariants.all { variant ->
    variant.javaCompile.doLast {
```

```

    if (variant.buildType.name.equals("release")) {
        runAllatori(variant)
    }
}
}

```

## 8.19 특정 클래스를 난독화로부터 완전히 제외시키는 방법

Ignore-classes 태그는 난독화 과정에서 해당 클래스의 난독화 적용을 완전히 제외하는 데 사용됩니다.

예:

```

<ignore-classes>

    <class template = "class com.company.abc. *"/>
    <class template = "class com.company.xyz.SomeClass"/>

</ignore-classes>

```

## 8.20 DataBinding 활성화시에 Allatori 규칙 추가 방법

아래와 같이 "패키지명"에 개발 앱의 패키지명을 넣고 규칙을 추가합니다.

```

<!-- 외부 라이브러리 예외처리-->
<class template="class * instanceof androidx.**"/>
<class template="class androidx.**" <field access="private+"/><method template="private+ *(**)" /> </class>
<class template="class * extends androidx.**" <field access="private+"/><method template="private+ *(**)" /> </class>

<!-- 내부 코드 일괄처리를 위한 설정 -->
<class template="class 패키지명.databinding.**" <field access="private+"/><method template="private+ *(**)" /> </class>

```