

PMD

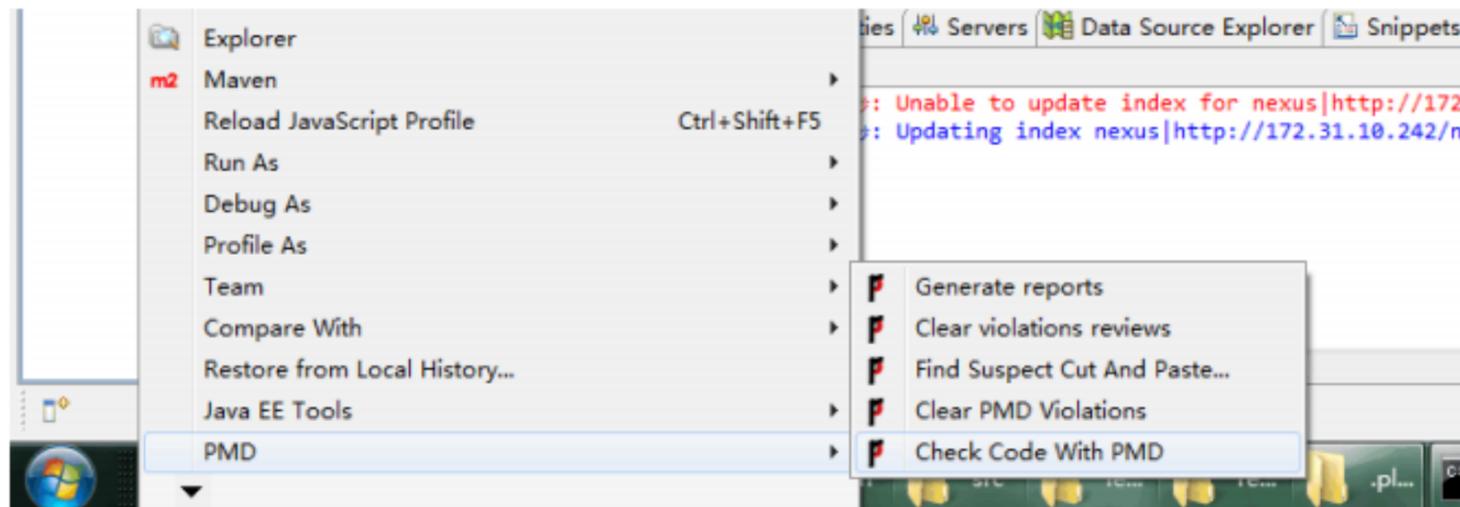
Eclipse-pmd 插件下载：

网上给出的 url 都无法使用，可以去 http://sourceforge.jp/projects/sfnet_pmd/releases/ 手动下载插件，解压后复制到 eclipse 的 plugin 和 features 目录下。重启 eclipse 后，windows —>preferences 下看到 PMD 选项则说明安装成功。

PMD 使用：

1.检查代码

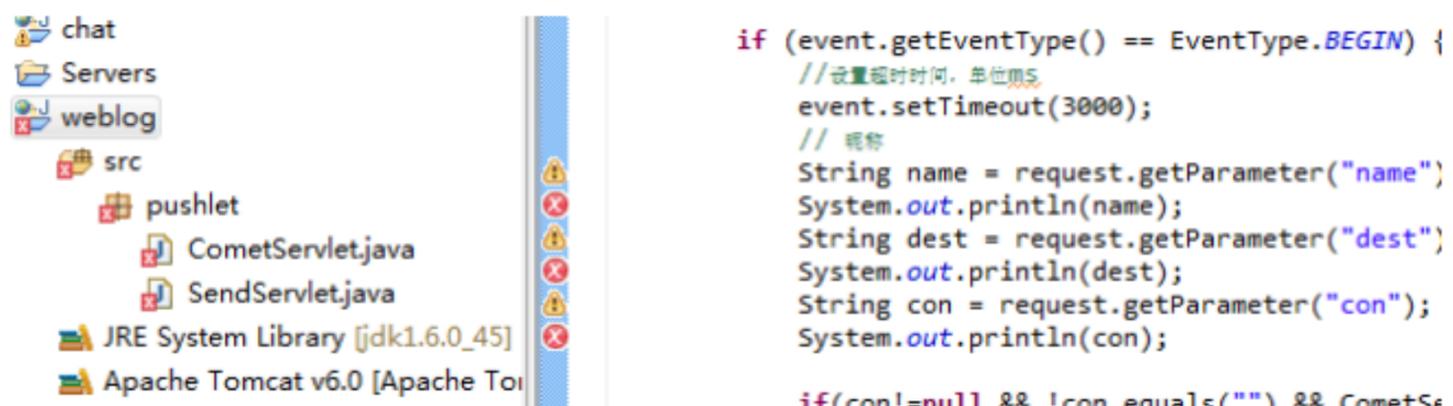
1) 右键项目， PMD —>Check Code With PMD



2) 在 PMD 视图下，可以看到检查结果。每个代码文件的违反规则的地方都被列出，右上角的五色圆形按钮，可以按照违规等级过滤列出的信息。从左到右依次为 error high, error, warning high, warning, information 。

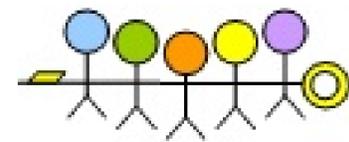
| Element | # Violations | # Violations/... | # Violations/... | Project |
|----------------------------|--------------|------------------|------------------|---------|
| pushlet | 35 | 486.1 / 1000 | 7.00 | weblog |
| CometServlet.java | 16 | 516.1 / 1000 | 16.00 | weblog |
| SystemPrintln | 2 | 64.5 / 1000 | 2.00 | weblog |
| EmptyIfStmt | 3 | 96.8 / 1000 | 3.00 | weblog |
| MethodArgumentCouldBeFinal | 1 | 32.3 / 1000 | 1.00 | weblog |
| ConfusingTernary | 1 | 32.3 / 1000 | 1.00 | weblog |
| UnusedImports | 4 | 129.0 / 1000 | 4.00 | weblog |
| DataflowAnomalyAnalysis | 2 | 64.5 / 1000 | 2.00 | weblog |

3) 在 package explorer 和代码文件中都会有标记

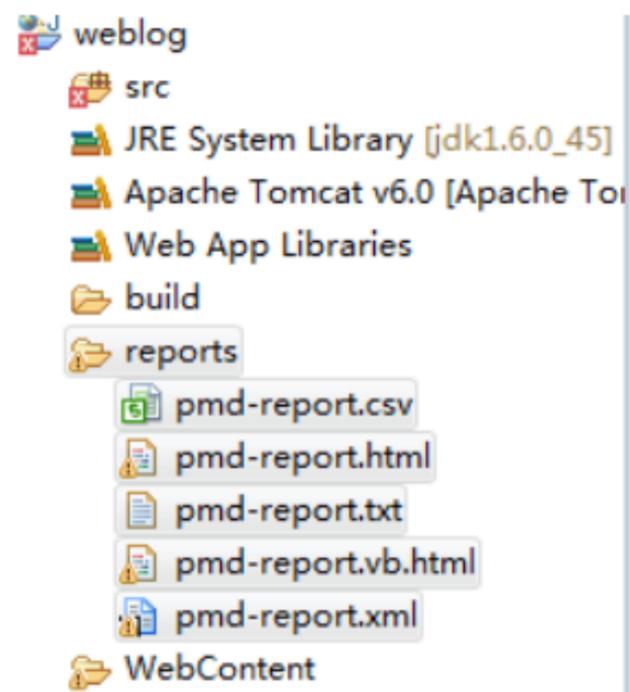


2.生成检查报告

1) 检查后，右键项目， PMD —>Generate Reports。在项目目录下会生成 reports 文件夹，存



放检查报告。

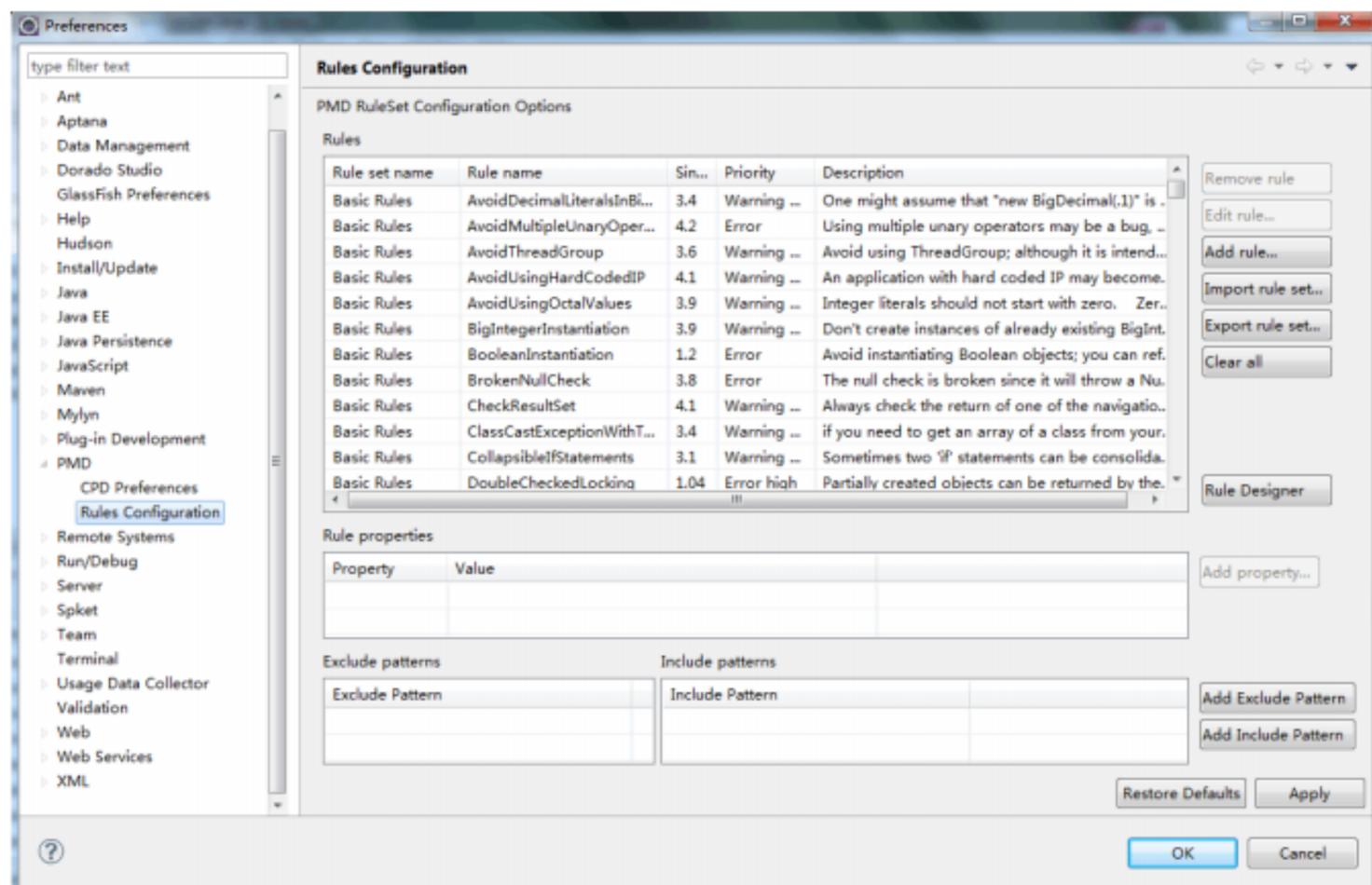


3.清除违规标记

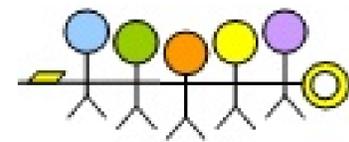
1) 右键项目，PMD → Clear PMD Violations

4.编辑检查规则

1) Window → Preferences，左侧选择 PMD → Rules Configuration。



在 Rules 下已显示出 PMD 自带的检查规则。点击右侧 Add rule 按钮，进入规则制定界面，如下所示。



PMD Plugin

RuleSet name : Rule Reference

Since :

Rule name : XPath rule

Rule implementation class :

Message :

Priority : Uses Type Resolution Uses DFA

Description :

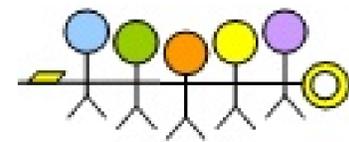
External Info URL :

Examples :

XPath :

检查规则在 XPath 项配置。

2)Window —>preferences—>PMD , 点击 Rule Designer , 可以设计自己的规则。



The screenshot shows the PMD Rule Designer interface. The 'Source code' pane contains the following Java code:

```
public class A {  
    public static void main(String[] args) {  
        System.out.println();  
    }  
}
```

The 'XPath Query (if any):' pane contains the following query:

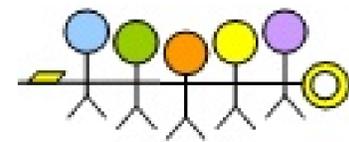
```
//Name[  
  starts-with(@Image, 'System.out.print')  
  or  
  starts-with(@Image, 'System.err.print')  
]
```

The 'Abstract Syntax Tree / XPath / Symbol Table' pane shows a tree structure for the code. The 'Data Flow Analysis' pane is empty. The 'AST Node: VariableDeclarationId args' pane shows the scope information for the 'args' parameter:

```
AST Node: VariableDeclarationId args  
  Scope: SourceFileScope  
    Class name declaration: Class A  
  Scope: ClassScope  
    Method name declaration: Method main, line 3, params = 1  
  Scope: MethodScope  
    Variable name declaration: Variable: image = 'args', line = 3
```

输入 Source Code 和 XPath Query , 点击 Go , 可以查看 PMD 根据源代码生成的抽象语法数 (AST) 和匹配结果。

PS : 想要熟练配置自己的规则 , 需要对 XPath 和 PMD 工作原理有一定的了解。 可参考 [PMD 使用说明 .doc](#) 中相关内容。



CheckStyle

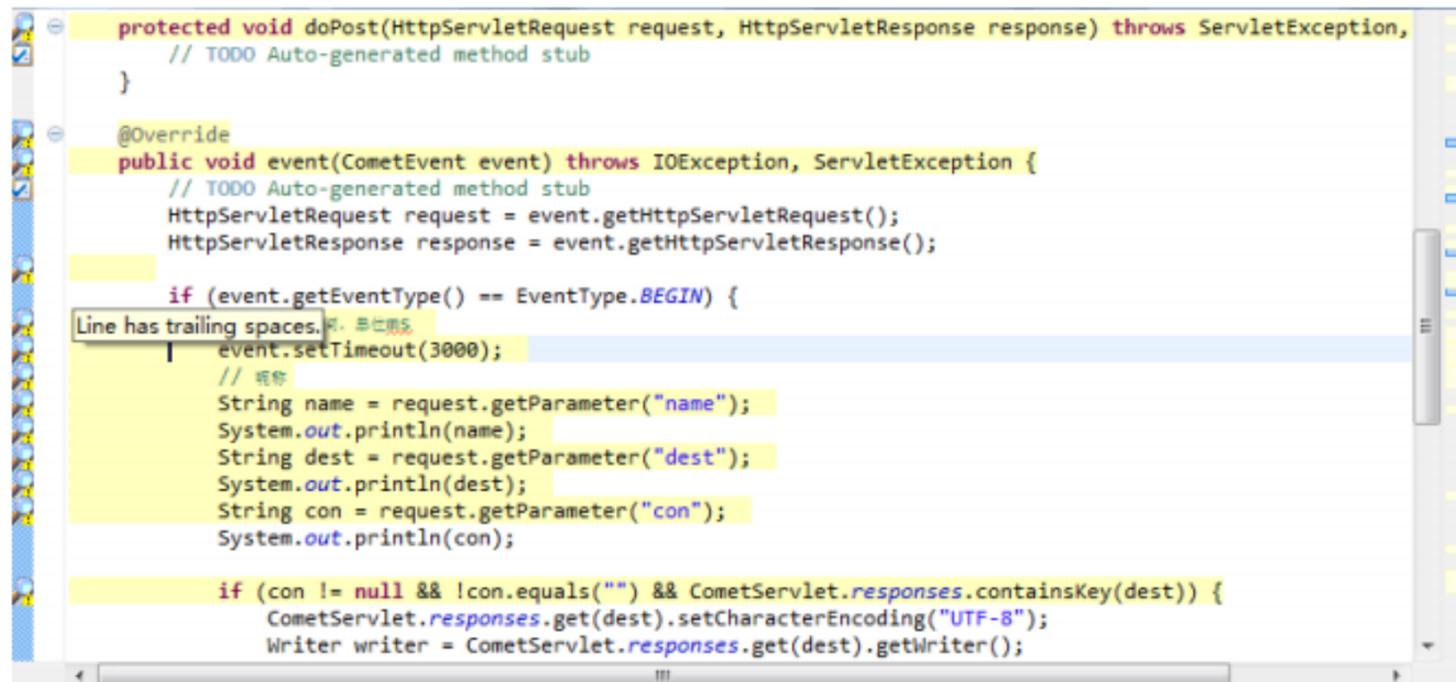
Eclipse-CheckStyle 插件下载：

<http://sourceforge.net/projects/eclipse-cs/files/> 手动下载插件，解压后复制到 eclipse 的 plugin 和 features 目录下。重启 eclipse 后，windows—>preferences 下看到 checkstyle 选项则说明安装成功。

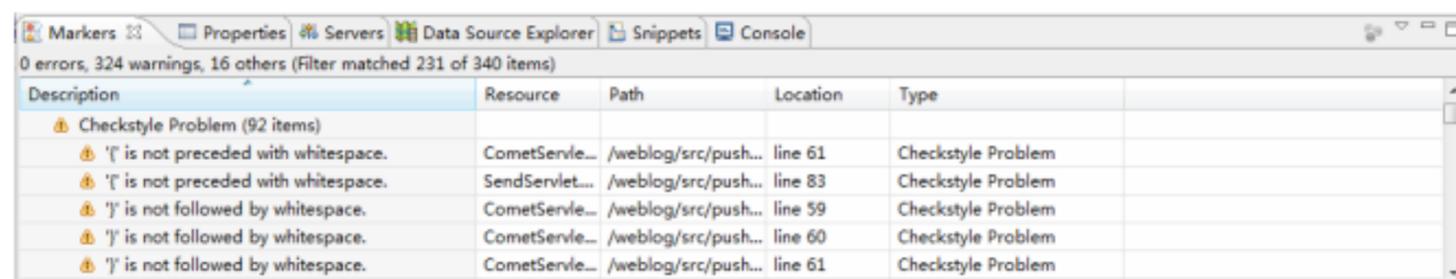
Checkstyle 使用：

1.代码检查

1) 右键项目， checkstyle—>check code with checkstyle.



违规处高亮显示。

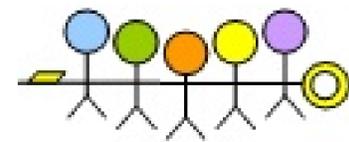


违规信息和其他 warning、error 信息一起显示在 Markers 标签下。

2) 激活 checkstyle，自动检查代码。右键项目， checkstyle—>activate checkstyle。在写代码时，可以实时提示违反规则的代码。

2.定制规则

1) Window —>Preferences，在左侧选择 checkstyle。



| Check Configuration | Location | Type | Default |
|----------------------|------------------------|------------------------|---------|
| Sun Checks | sun_checks.xml | Built-In Configuration | ✓ |
| Sun Checks (Eclipse) | sun_checks_eclipse.xml | Built-In Configuration | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Buttons: New..., Properties..., Configure..., Copy..., Remove, Set as Default

已有的两个规则不能改变，点击 New... 自定义规则。
新建后，在如下界面编辑各项规则。

Internal Configuration "My Rules"
Edit checkstyle configuration.

Known modules

Input filter text here

- Annotations
- Annotation Use Style
- Missing Deprecated
- Missing Override
- Package Annotation
- Suppress Warnings
- Javadoc Comments
- Naming Conventions
- Headers
- Imports
- Size Violations
- Whitespace
- Regexp
- Modifiers
- Blocks

Configured modules for group "Annotations"

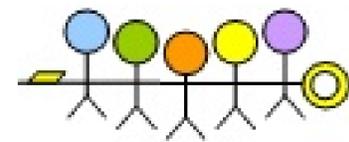
| Enabled | Module | Severity | Comment |
|---------|--------|----------|---------|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Description:
No description available.

Open module editor(s) on add action

Buttons: Add... ->, <- Remove, Open..., ?

Buttons: OK, Cancel



FindBugs

Eclipse-FindBugs 插件下载：

http://sourceforge.jp/projects/sfnet_findbugs/releases/

FindBugs 使用：

使用方法与前两个很相似。

比较

检查规则：

三者都提供了很多检查规则，且可以生成检查报告。并且利用这些报告，能够提取里面涉及的规则，在其他地方使用。

PMD 的自定义规则更加灵活，可使用 XPath 定义各种规则。

checkstyle 是在原来较为严格的规则基础上做定制和修改，无法定义新规则。

findbugs 只能在原来的基础上做定制，无法修改规则，findbugs 若要定义新规则，需要引入 jar 包。

检查对象：

PMD 对 .java 源代码进行检查，将源代码解析成抽象语法树 (AST)，检查源代码中潜在的问题。主要包括：空 try/catch/finally/switch 语句块，未使用的局部变量、参数和 private 方法，空 if/while 语句，过于复杂的表达式，如不必要的 if 语句等，复杂类。

CheckStyle 对源代码进行检查。主要包括：Javadoc注释，命名规范，多余没用的 Imports，Size 度量，如过长的方法，缺少必要的空格 Whitespace，重复代码。

FindBugs 检查 .class 文件，基于 Bug Patterns 概念，查找 javabytecode (.class 文件) 中的潜在 bug。主要检查 bytecode 中的 bug patterns，如 NullPoint 空指针检查、没有合理关闭资源、字符串相同判断错 (== 而不是 equals) 等。

使用目的：

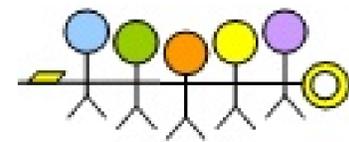
PMD：使代码更简洁明了，方法、变量命名更加规范，增强可读性。

CheckStyle：规范代码的格式，统一代码风格。

FindBugs：找出代码中存在的明显的可能导致 bug 的缺陷。

以下是分析示例，对同一段代码使用 3 种工具进行检查，体现各自的侧重点。

PMD

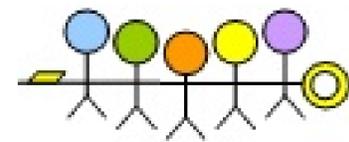


```
import java.io.File;
import java.util.Iterator;
import org.dom4j.*;
import org.dom4j.io.*;

public class Dom4JXMLReader {
    public static void main(String[] args) {
        String s = "abcd";
        s.replace("bc", "cb");
        for(int i=0;i<100;){}
        try {
            File f = new File("sample.xml");
            SAXReader reader = new SAXReader();
            Document doc=reader.read(f);
            Element root = doc.getRootElement();
            Element foo;
            for (Iterator<?> i = root.elementIterator("user"); i.hasNext();) {
                foo = (Element) i.next();
                System.out.println("name: " + foo.elementText("name"));
            }
        } catch (Exception e) {
        }
    }
}
```

提示有：变量前可加 final 修饰符，有空的块，有 System.out.print 语句等。

CheckStyle

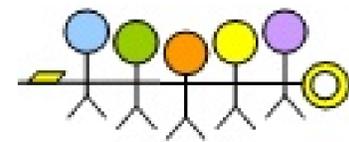


```
import java.io.File;
import java.util.Iterator;
import org.dom4j.*;
import org.dom4j.io.*;

public class Dom4JXMLReader {
    public static void main(String[] args) {
        String s = "abcd";
        s.replace("bc", "cb");
        for(int i=0;i<100;) {}
        try {
            File f = new File("sample.xml");
            SAXReader reader = new SAXReader();
            Document doc=reader.read(f);
            Element root = doc.getRootElement();
            Element foo;
            for (Iterator<?> i = root.elementIterator("user"); i.hasNext();) {
                foo = (Element) i.next();
                System.out.println("name: " + foo.elementText("name"));
            }
        } catch (Exception e) {
        }
    }
}
```

提示有：import 中有“.*”，缺少 Javadoc Comment，一些符号前后缺少空格，一些常量是 magic number，语句后有多余空格，一行的代码长度超长，空的块等。

FindBugs



```
import java.io.File;
import java.util.Iterator;
import org.dom4j.*;
import org.dom4j.io.*;

public class Dom4JXMLReader {
    public static void main(String[] args) {
        String s = "abcd";
        s.replace("bc", "cb");
        for(int i=0;i<100;){}
        try {
            File f = new File("sample.xml");
            SAXReader reader = new SAXReader();
            Document doc=reader.read(f);
            Element root = doc.getRootElement();
            Element foo;
            for (Iterator<?> i = root.elementIterator("user"); i.hasNext();){
                foo = (Element) i.next();
                System.out.println("name: " + foo.elementText("name"));
            }
        } catch (Exception e) {
        }
    }
}
```

提示有：s.replace(“bc”, “cb”) 的返回值被忽略，存在明显的无限循环。