Static Analysis Using Spotbugs and PMD

Lab1 – Systems Security

Sumaya Altamimi - 442203026

Table of Contents

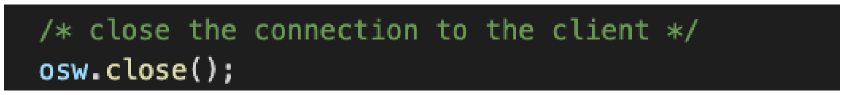
Introduction

For this lab, I will analyze a simple web server written in Java in my macOS. The analysis will be performed first manually and then with two tools, SpotBugs v4.2.0 which is the spiritual successor of FindBugs. It is a tool to find bugs in Java programs. It looks for instances of “bug patterns” or code instances that are likely to be errors. Specifically, it scans bytecode (class files) generated by JDK8 and newer versions.
The second tool that I will use is PMD v6.30.0 which stands for Programming Mistake Detector. There is an add0n to PMD that is called Copy Paste Detector (CPD). Unlike SpotBugs that only works with java, PMD support other languages as well.

As a result, I will build the source for the simple web server in order to analyze it with these tow tools.

Manual Code Review

After analyzing the web server manually, it seems simple, clear, and good. However, there are few things I noticed, and I think it is better to make some changes to better protect the web server. The first code problem is the *unclosed buffer reader ‘br’, and file reader ‘fr’*. The only closed one was the Output Stream Writer ‘osw’ in line 82.

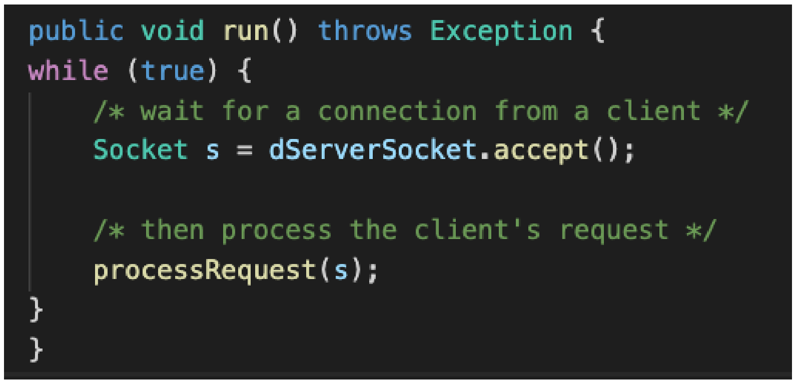


```
/* close the connection to the client */
osw.close();
```

Closing files is important for the following reasons [1]:

- It impacts the performance with too many open files, slowing down the program.
- Changes to files will not go into effect until after the file is closed, so if we edit, leaves open, and reads a file, we won't see the edits.
- It puts the program in the garbage collectors hands.
- Many more...

The second thing that seems suspicious, the while true in the run method, I think this might impact the performance and there should be another way to process the request.



```
public void run() throws Exception {
while (true) {
    /* wait for a connection from a client */
    Socket s = dServerSocket.accept();

    /* then process the client's request */
    processRequest(s);
}
}
```

Static Analysis

Tool choices and versions

The first tool that I used is SpotBugs v4.2.0
The second tool is PMD v6.30.0
Operating system: MacOS

SpotBugs

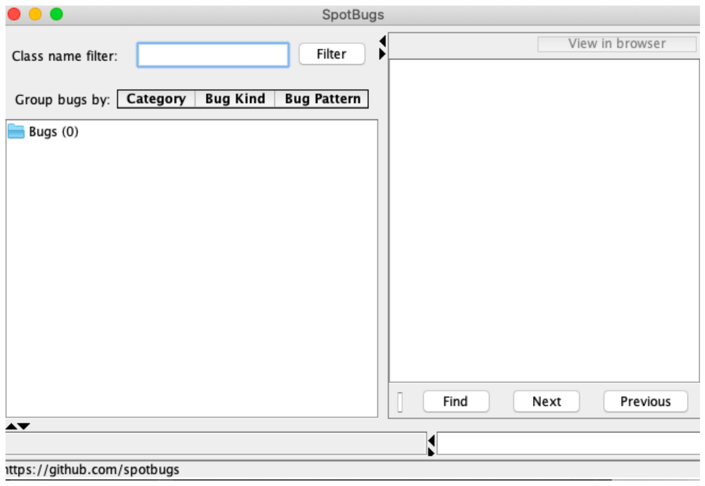
Approach and steps:

Installing and running SpotBugs

1. Install the tool directly from the command line as shown below or install it from GitHub using the following link: https://github.com/spotbugs/spotbugs.

```
brew install findbugs
```

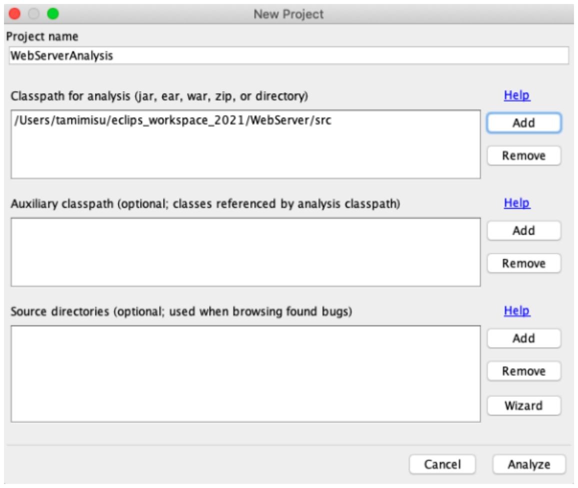
2. Open the tool by writing the tool name '*spotbugs*'. The tool GUI will appear, and looks like this:



Start New project

3. Make a new project from file menu and name it *WebServerAnalysis*.

4. Add the source code files '*webserver.class'* and click on the Analyze button.


New Project

Project name: WebServerAnalysis

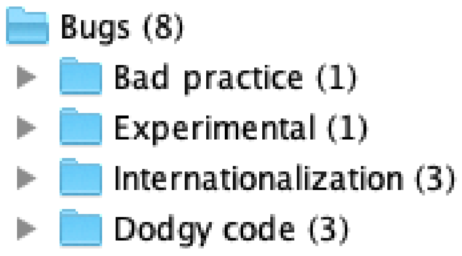
Classpath for analysis (jar, ear, war, zip, or directory): /Users/tamimisu/eclips_workspace_2021/WebServer/src

Auxiliary classpath (optional; classes referenced by analysis classpath)

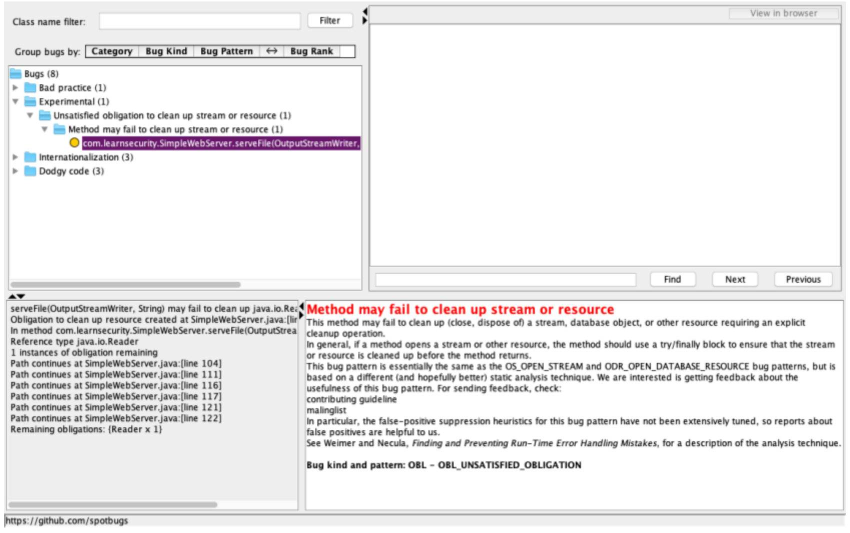
Source directories (optional; used when browsing found bugs)

Analyze the Results

5. The results will appear in the left-hand side as shown below:


Bugs (8)
- Bad practice (1)
- Experimental (1)
- Internationalization (3)
- Dodgy code (3)

6. Click and read each one to enhance your code.


Class name filter:
Filter
Group bugs by: Category | Bug Kind | Bug Pattern | ↔ | Bug Rank
View in browser
Find
Next
Previous

Bugs (8)
- Bad practice (1)
- Experimental (1)
  - Unsatisfied obligation to clean up stream or resource (1)
    - Method may fail to clean up stream or resource (1)
      - com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter,
- Internationalization (3)
- Dodgy code (3)

```
serveFile(OutputStreamWriter, String) may fail to clean up java.io.Rea
Obligation to clean up resource created at SimpleWebServer.java:[lir
In method com.learnsecurity.SimpleWebServer.serveFile(OutputStrea
Reference type java.io.Reader
1 instances of obligation remaining
Path continues at SimpleWebServer.java:[line 104]
Path continues at SimpleWebServer.java:[line 111]
Path continues at SimpleWebServer.java:[line 116]
Path continues at SimpleWebServer.java:[line 117]
Path continues at SimpleWebServer.java:[line 121]
Path continues at SimpleWebServer.java:[line 122]
Remaining obligations: {Reader x 1}
```

Method may fail to clean up stream or resource

This method may fail to clean up (close, dispose of) a stream, database object, or other resource requiring an explicit cleanup operation.

In general, if a method opens a stream or other resource, the method should use a try/finally block to ensure that the stream or resource is cleaned up before the method returns.

This bug pattern is essentially the same as the OS_OPEN_STREAM and ODR_OPEN_DATABASE_RESOURCE bug patterns, but is based on a different (and hopefully better) static analysis technique. We are interested is getting feedback about the usefulness of this bug pattern. For sending feedback, check:

contributing guideline

malinglist

In particular, the false-positive suppression heuristics for this bug pattern have not been extensively tuned, so reports about false positives are helpful to us.

See Weimer and Necula, *Finding and Preventing Run-Time Error Handling Mistakes*, for a description of the analysis technique.

Bug kind and pattern: OBL - OBL_UNSATISFIED_OBLIGATION

https://github.com/spotbugs

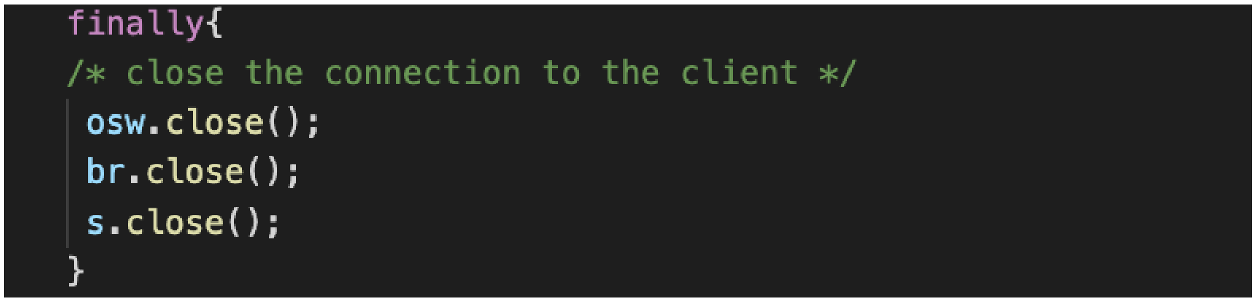
7. If you find bug categories are ambiguous, you can refer to SpotBugs official site[4] ,which include detailed description about each one of them.

SpotBugs Findings:

1.Bad practice: Method may fail to close stream.

As mentioned in Manual Review section, the buffer reader is not closed.

Correction: Close Java.io.reader. Use a try/finally block to ensure that streams are closed before the method returns.

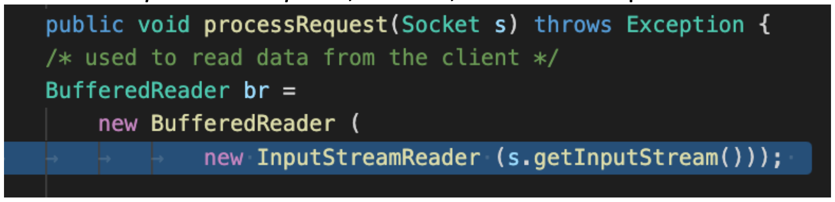


```
finally{
/* close the connection to the client */
 osw.close();
 br.close();
 s.close();
}
```

2.Internationalization: Reliance on default encoding

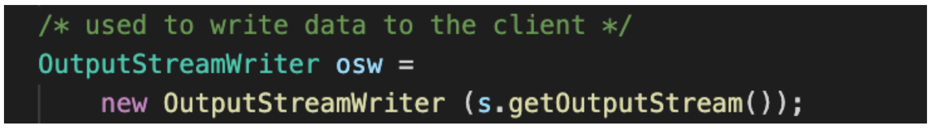
Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. This bug occurs three times.

- In method *processRequest*, line 49, the use of InputStreamReader:



```
public void processRequest(Socket s) throws Exception {
/* used to read data from the client */
BufferedReader br =
    new BufferedReader (
            new InputStreamReader (s.getInputStream()));
```

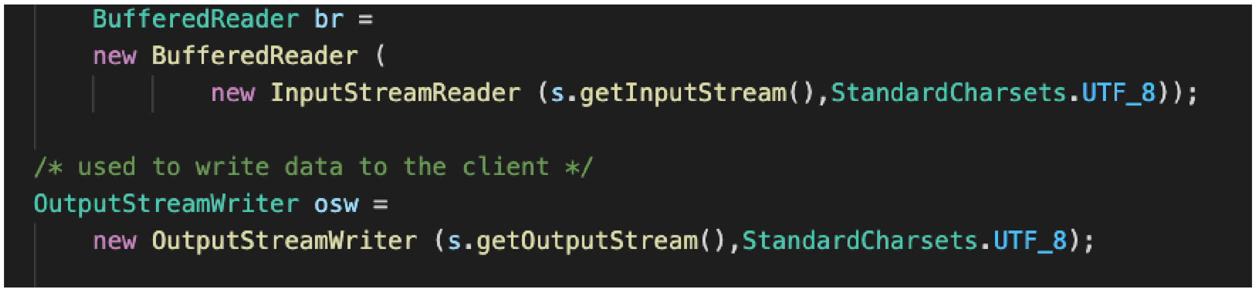
- In the same method *processRequest*, line 53, the use of OutputStreamWriter:



```
/* used to write data to the client */
OutputStreamWriter osw =
    new OutputStreamWriter (s.getOutputStream());
```

In method *serveFile*, line 103, the use of FileReader:

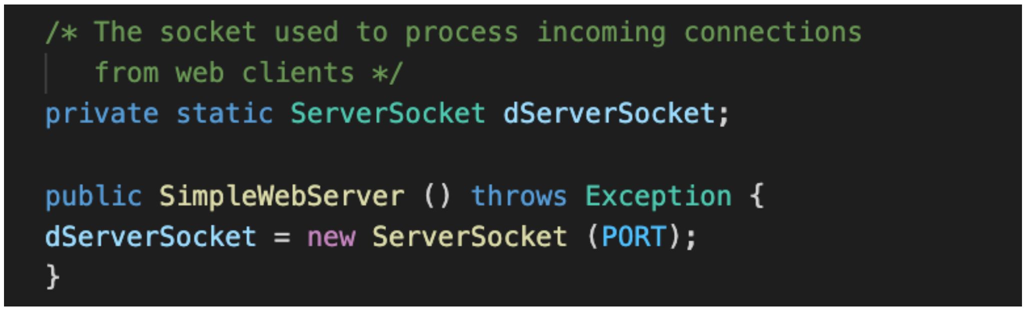
Correction: Use an alternative API and specify a charset name or Charset object explicitly.



```
        BufferedReader br =
        new BufferedReader (
                new InputStreamReader (s.getInputStream(),StandardCharsets.UTF_8));

    /* used to write data to the client */
    OutputStreamWriter osw =
        new OutputStreamWriter (s.getOutputStream(),StandardCharsets.UTF_8);
```

3. Misuse of static fields

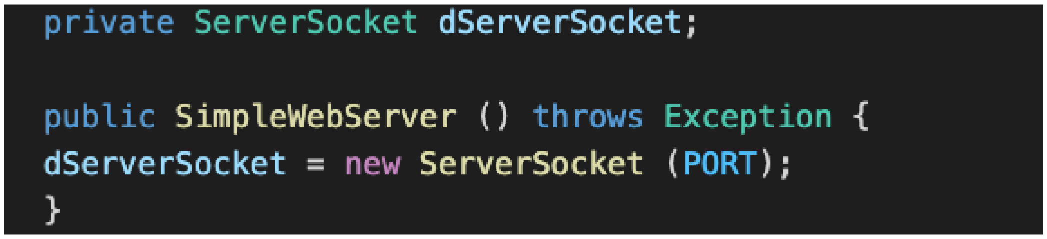
In the *constructor*, line 29, *dServerSocket* makes instance method that write to a static field. Assigning a value to a static field in a constructor could cause unreliable behavior at runtime since it will change the value for all instances of the class.



```
/* The socket used to process incoming connections
    from web clients */
private static ServerSocket dServerSocket;

public SimpleWebServer () throws Exception {
dServerSocket = new ServerSocket (PORT);
}
```

This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

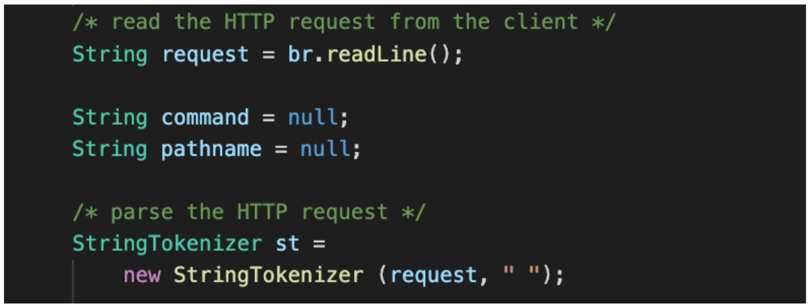
Correction: remove static keyword is one solution. You can either initialize it statically.



```
private ServerSocket dServerSocket;

public SimpleWebServer () throws Exception {
dServerSocket = new ServerSocket (PORT);
}
```

4. Null pointer dereference

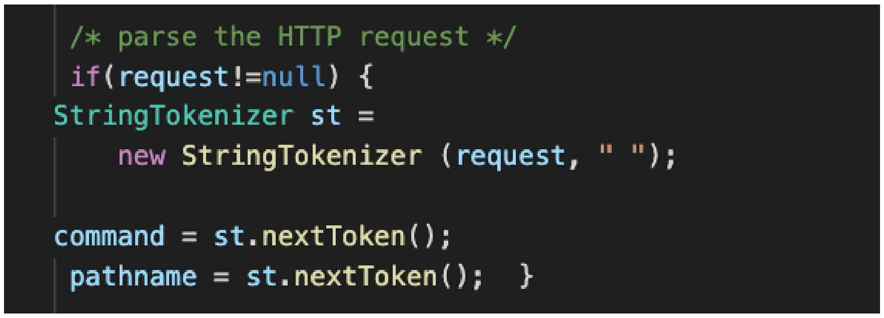
In the *processRequest* method, line 62, *request* variable has dereferenced without nullcheck.



```
    /* read the HTTP request from the client */
    String request = br.readLine();

    String command = null;
    String pathname = null;

    /* parse the HTTP request */
    StringTokenizer st =
        new StringTokenizer (request, " ");
```

The result of invoking readLine() is dereferenced without checking to see if the result is null. If there are no more lines of text to read, readLine() will return null and dereferencing that will generate a null pointer exception.

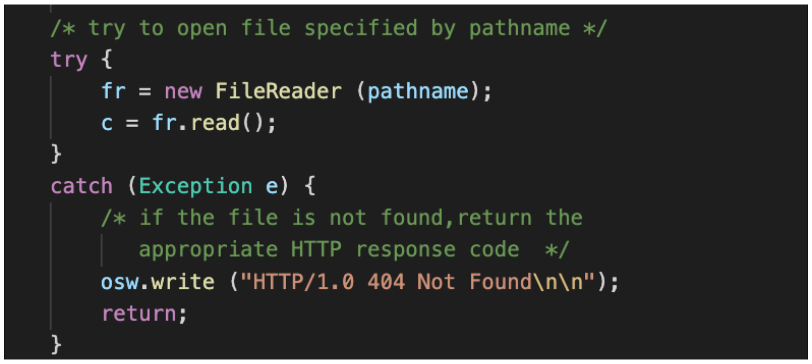
Correction: check the *request* for null before dereferencing it.



```
     /* parse the HTTP request */
     if(request!=null) {
    StringTokenizer st =
        new StringTokenizer (request, " ");

    command = st.nextToken();
     pathname = st.nextToken();   }
```

5. RunTimeException Capture

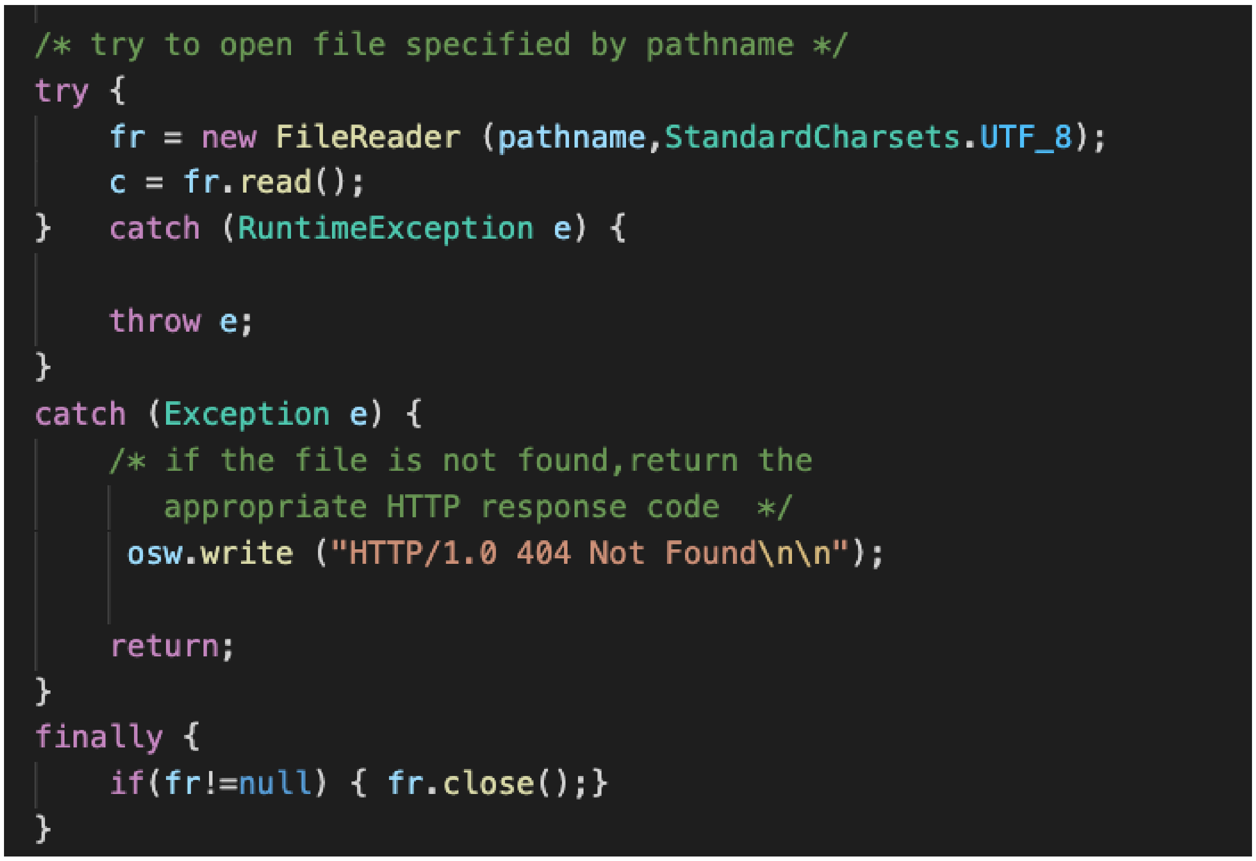
In method *serveFile*, line 106, exception is caught when the exception is not thrown:



```
   /* try to open file specified by pathname */
   try {
       fr = new FileReader (pathname);
       c = fr.read();
   }
   catch (Exception e) {
       /* if the file is not found,return the
          appropriate HTTP response code  */
       osw.write ("HTTP/1.0 404 Not Found\n\n");
       return;
   }
```

The method uses a try-catch block that catches Exception objects, but Exception is not thrown within the try block, and RuntimeException is not explicitly caught. It is a common bug pattern to say try { ... } catch (Exception e) { something } as a shorthand for catching a number of types of exception each of whose catch blocks is identical,but this construct also accidentally catches RuntimeException as well, masking potential bugs.

Correction: Either explicitly catch the specific exceptions that are thrown, or to explicitly catch RuntimeException exception, rethrow it, and then catch all non-Runtime Exceptions.



```java
/* try to open file specified by pathname */
try {
    fr = new FileReader (pathname,StandardCharsets.UTF_8);
    c = fr.read();
}   catch (RuntimeException e) {

    throw e;
}
catch (Exception e) {
    /* if the file is not found,return the
       appropriate HTTP response code  */
     osw.write ("HTTP/1.0 404 Not Found\n\n");

    return;
}
finally {
    if(fr!=null) { fr.close();}
}
```

PMD

QuickStart

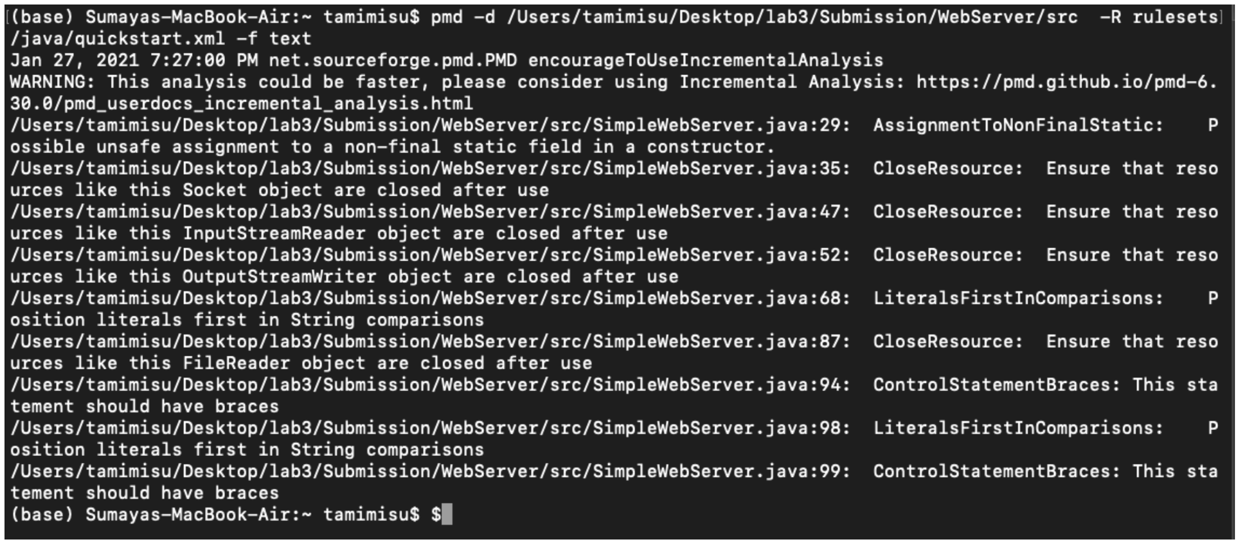
- In terminal, run the command ‘brew install pmd’, or simply go to the official page https://pmd.github.io/ and click download. You can also run the following commands in terminal:

```
-   $ cd $HOME

-   $ curl -OL https://github.com/pmd/pmd/releases/download/pmd_releases%2F6.30.0/pmd-bin-6.30
    .0.zip

-   $ unzip pmd-bin-6.30.0.zip

-   $ alias pmd="$HOME/pmd-bin-6.30.0/bin/run.sh pmd"
- $ pmd -d /usr/src -R rulesets/java/quickstart.xml -f text
```

- Replace the path after -d parameter with your source code path.

Analyze the Results

- Analysis results will be shown in the terminal as shown below:

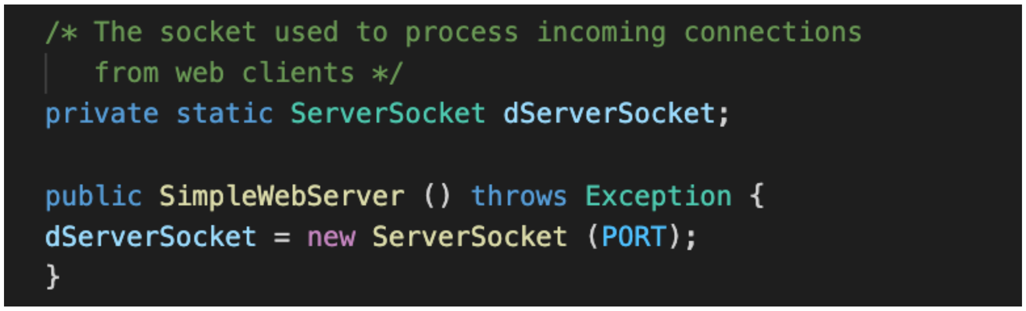


```
(base) Sumayas-MacBook-Air:~ tamimisu$ pmd -d /Users/tamimisu/Desktop/lab3/Submission/WebServer/src  -R rulesets
/java/quickstart.xml -f text
Jan 27, 2021 7:27:00 PM net.sourceforge.pmd.PMD encourageToUseIncrementalAnalysis
WARNING: This analysis could be faster, please consider using Incremental Analysis: https://pmd.github.io/pmd-6.
30.0/pmd_userdocs_incremental_analysis.html
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:29:  AssignmentToNonFinalStatic:    P
ossible unsafe assignment to a non-final static field in a constructor.
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:35:  CloseResource:  Ensure that reso
urces like this Socket object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:47:  CloseResource:  Ensure that reso
urces like this InputStreamReader object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:52:  CloseResource:  Ensure that reso
urces like this OutputStreamWriter object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:68:  LiteralsFirstInComparisons:    P
osition literals first in String comparisons
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:87:  CloseResource:  Ensure that reso
urces like this FileReader object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:94:  ControlStatementBraces: This sta
tement should have braces
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:98:  LiteralsFirstInComparisons:    P
osition literals first in String comparisons
/Users/tamimisu/Desktop/lab3/Submission/WebServer/src/SimpleWebServer.java:99:  ControlStatementBraces: This sta
tement should have braces
(base) Sumayas-MacBook-Air:~ tamimisu$ $
```

PMD findings

1. Assignment To Non Final Static

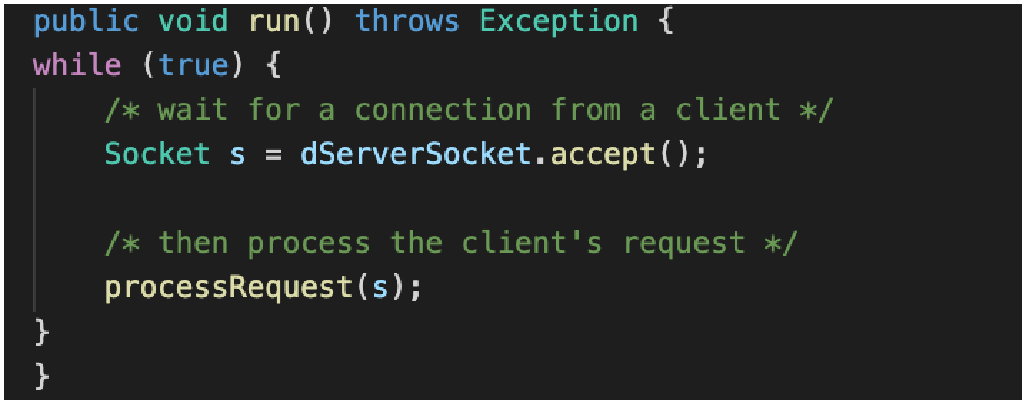
In line 29, Possible unsafe assignment to a non-final static field in a constructor.



```
/* The socket used to process incoming connections
   from web clients */
private static ServerSocket dServerSocket;

public SimpleWebServer () throws Exception {
dServerSocket = new ServerSocket (PORT);
}
```

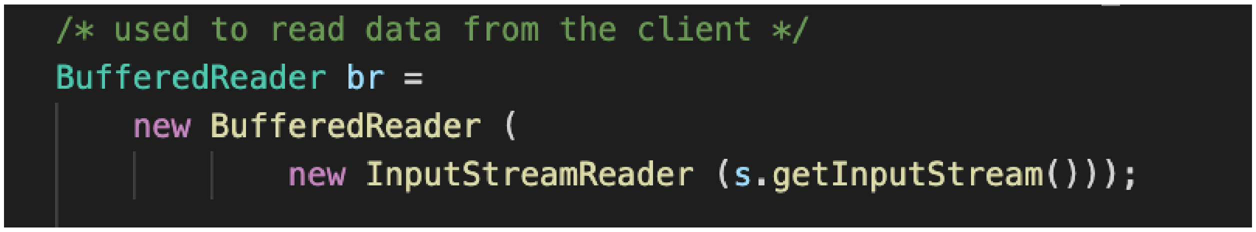
2. Close Resource

In line 35, Ensure that resources like this Socket object are closed after use.



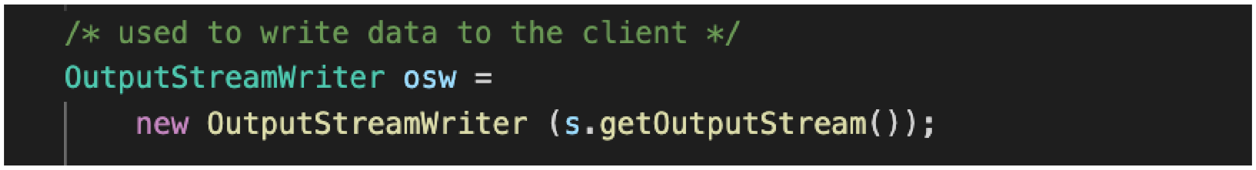
```
public void run() throws Exception {
while (true) {
    /* wait for a connection from a client */
    Socket s = dServerSocket.accept();

    /* then process the client's request */
    processRequest(s);
}
}
```

In line 47, Ensure that resources like this InputStreamReader object are closed after use.



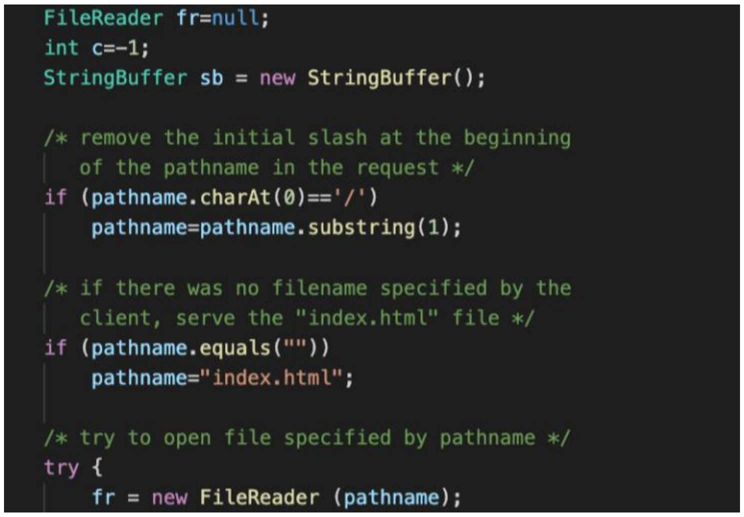
```
/* used to read data from the client */
BufferedReader br =
    new BufferedReader (
            new InputStreamReader (s.getInputStream()));
```

In line 52, Ensure that resources like this OutputStreamWriter object are closed after use.



```
/* used to write data to the client */
OutputStreamWriter osw =
    new OutputStreamWriter (s.getOutputStream());
```

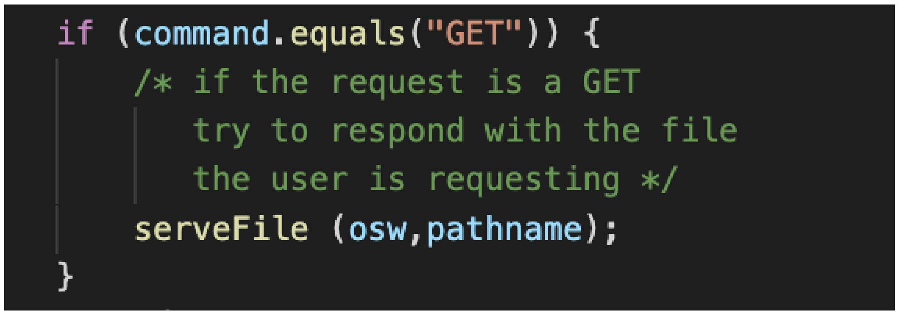
In line 87, Ensure that resources like this FileReader object are closed after use.



```
FileReader fr=null;
int c=-1;
StringBuffer sb = new StringBuffer();

/* remove the initial slash at the beginning
   of the pathname in the request */
if (pathname.charAt(0)=='/')
    pathname=pathname.substring(1);

/* if there was no filename specified by the
   client, serve the "index.html" file */
if (pathname.equals(""))
    pathname="index.html";

/* try to open file specified by pathname */
try {
    fr = new FileReader (pathname);
```

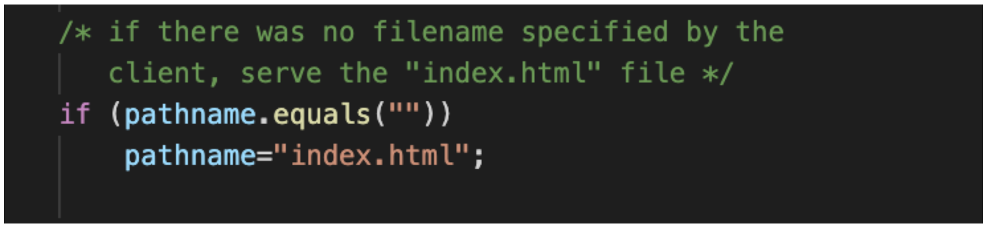
3. Literals First In Comparisons

In line 68, Position literals first in String comparisons.



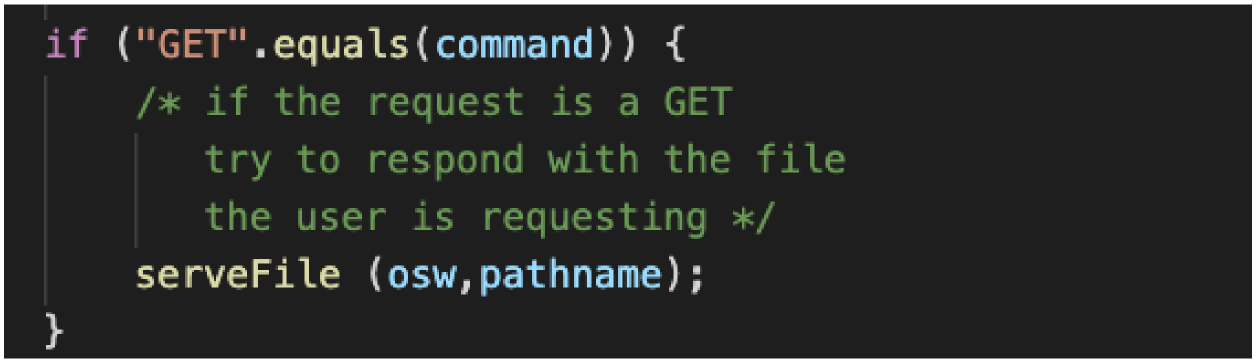
```
if (command.equals("GET")) {
    /* if the request is a GET
       try to respond with the file
       the user is requesting */
    serveFile (osw,pathname);
}
```

In line 98, Position literals first in String comparisons.

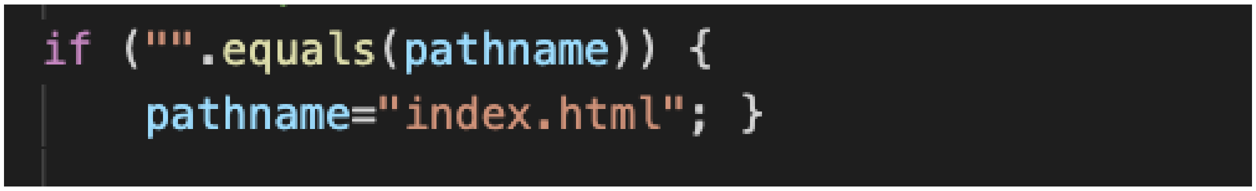


```
/* if there was no filename specified by the
   client, serve the "index.html" file */
if (pathname.equals(""))
    pathname="index.html";
```

<u>Correction</u>: If the variable is null, we won't get a null pointer exception. So, these will be like:



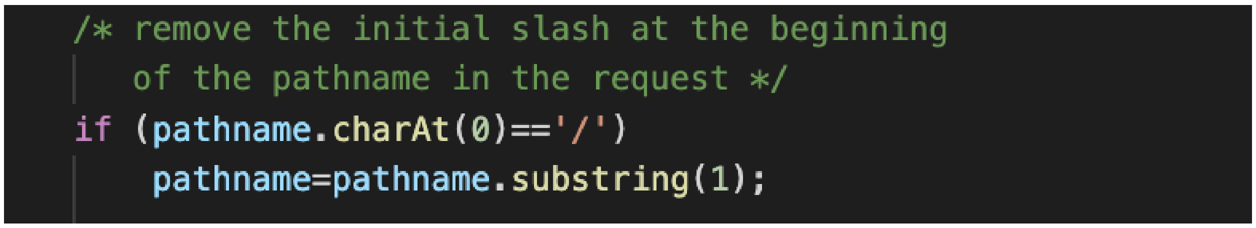
```
if ("GET".equals(command)) {
    /* if the request is a GET
       try to respond with the file
       the user is requesting */
    serveFile (osw,pathname);
}
```



```
if ("".equals(pathname)) {
    pathname="index.html"; }
```

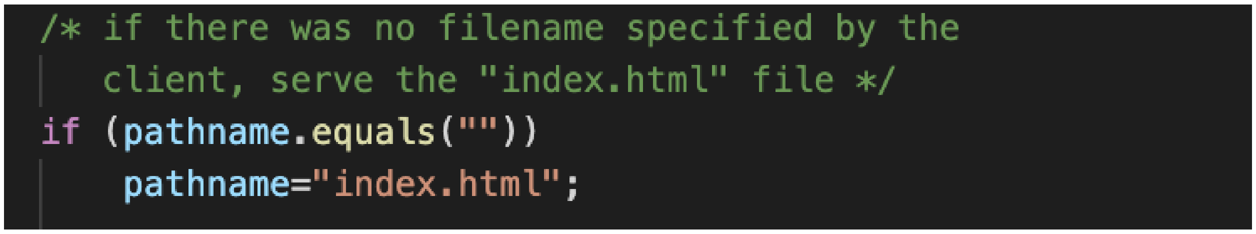
4. Control Statement Braces

In line 94, This statement should have braces.



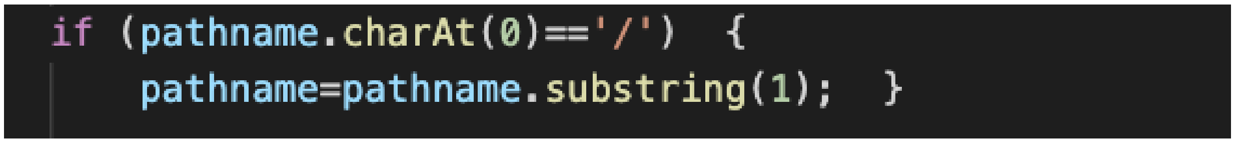
```
/* remove the initial slash at the beginning
   of the pathname in the request */
if (pathname.charAt(0)=='/')
    pathname=pathname.substring(1);
```

In line 99, This statement should have braces.

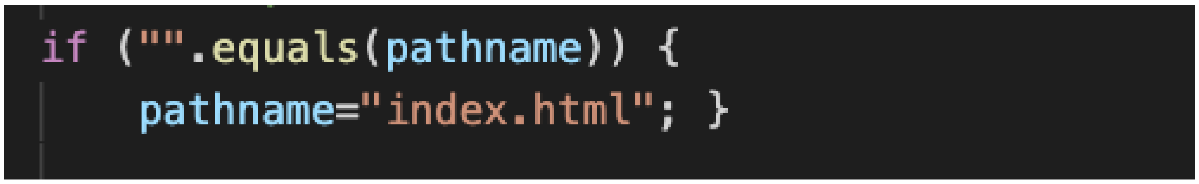


```
/* if there was no filename specified by the
   client, serve the "index.html" file */
if (pathname.equals(""))
    pathname="index.html";
```

<u>Correction</u>: It is better to have braces for if statements, like this:



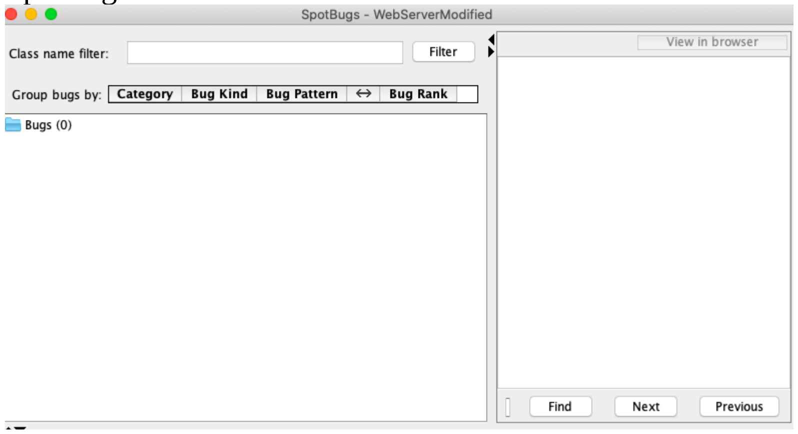
```
if (pathname.charAt(0)=='/')  {
    pathname=pathname.substring(1);  }
```



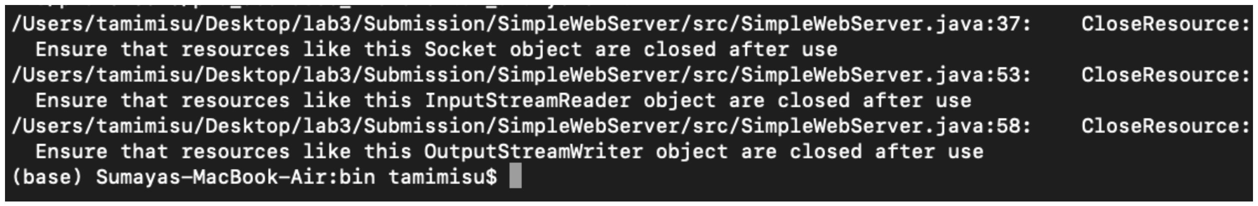
```
if ("".equals(pathname)) {
    pathname="index.html"; }
```

Code After Modification

After modifying all problems mentioned by the two tools, there is no bugs as shown below using SpotBugs tool:



However, in the case of PMD there is still three unavoidable alarms for closing the resources: buffer reader, socket, and output stream writer as shown below:



```
/Users/tamimisu/Desktop/lab3/Submission/SimpleWebServer/src/SimpleWebServer.java:37:    CloseResource:
  Ensure that resources like this Socket object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/SimpleWebServer/src/SimpleWebServer.java:53:    CloseResource:
  Ensure that resources like this InputStreamReader object are closed after use
/Users/tamimisu/Desktop/lab3/Submission/SimpleWebServer/src/SimpleWebServer.java:58:    CloseResource:
  Ensure that resources like this OutputStreamWriter object are closed after use
(base) Sumayas-MacBook-Air:bin tamimisu$
```

I assumed these as false alarms or (false negative) alarms. The modified code can be found in attachments as described in the attachment section.

Tool Comparison

After the Simple Web Server analysis experience and research on these tools, I found SpotBugs more efficient, clear, user friendly and more accurate. Also, for larger projects, SpotBugs will be my first solution. In addition to that, the GUI in SpotBugs, and it is available resources and easy configuration makes it preferable. These are basic requirements in today's technology. For the bugs description as well, I found SpotBugs more informative. However, it is worth to mention that PMD has a lot of features that I didn't use and a lot of documentations if someone wants to dive deeper in this. Below are some points to compare between the two tools.

Factor	SpotBugs	PMD
Does the tool analyze source or binary as input?	Binary	Source Code
Which category of tools is it?	Type checking Style checking Program understanding Program verification Property checking Bug finding Security review	Type checking Style checking Program understanding Program verification Property checking Bug finding Security review
Example of a finding that is reported by one tool but not the other.	RunTimeException Capture	Control Statement Braces
Example of a finding reported by both tools.	Close Resource	
RunTimeException Capture	True negative	False positive
Control Statement Braces	False positive	True negative

Attachments

Along with this report, I included the original source code java file, the modified version of the code in SimpleWebServer folder, and the exported results from each tool after analysis.

References

1. https://stackoverflow.com/questions/25070854/why-should-i-close-files-in-python/25070998
2. https://github.com/spotbugs/spotbugs
3. https://spotbugs.github.io/#using-spotbugs
4. https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html