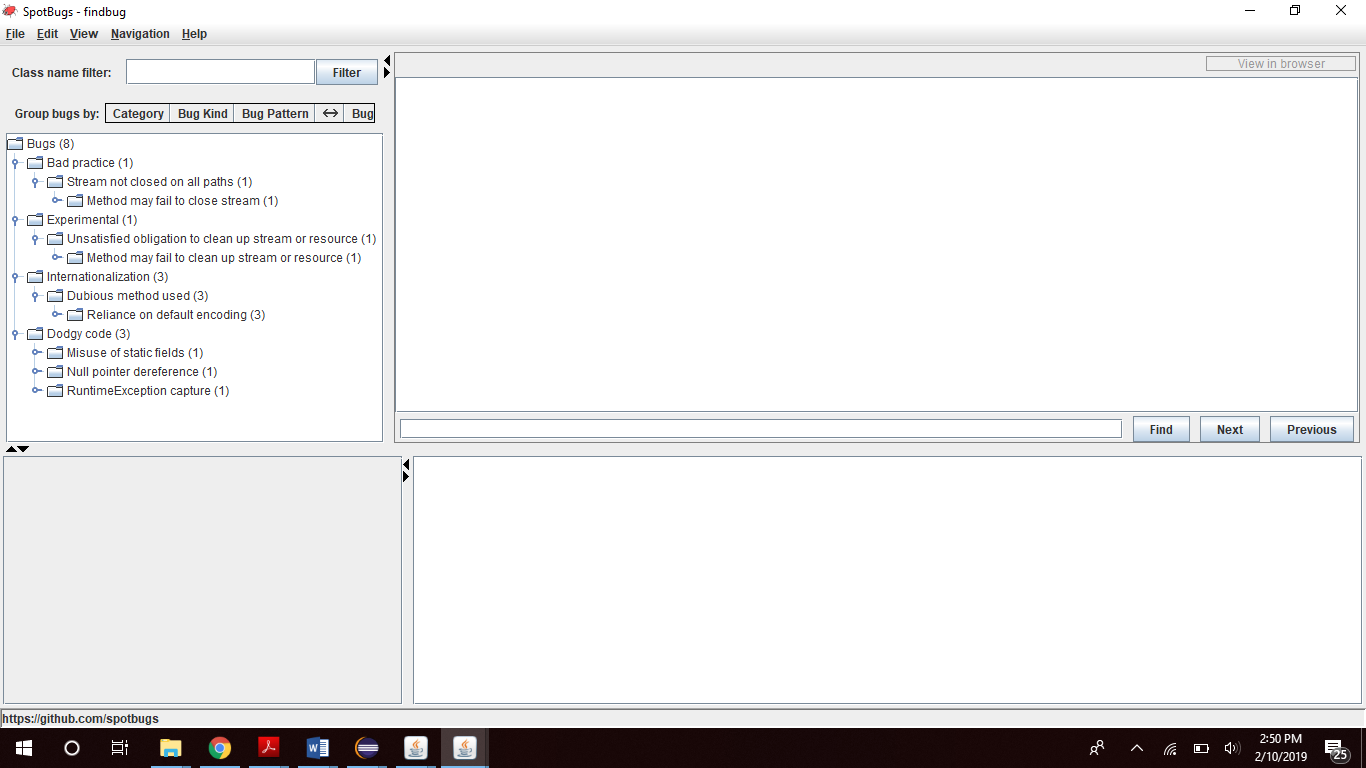
**Results:**

**Tool used: Spotbugs**

**Total bugs: 8**



1. Bad Practice(1)->Stream not closed on all paths->Method may fail to close stream->SimpleWebServer.serveFile(OutputStreamWriter, String) may fail to close stream

[**Method may fail to close stream**](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#OS_OPEN_STREAM)

The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method.  This may result in a file descriptor leak.  It is generally a good idea to use a finally block to ensure that streams are closed.

**Bug kind and pattern: OS - OS\_OPEN\_STREAM**

1. Experimental(2)->Unsatisfied obligation to clean up stream or resource->Method may fail to clean up stream or resource->SimpleWebServer.serveFile(OutputStreamWriter, String) may fail to clean up java.io.Reader

# [Method may fail to clean up stream or resource](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#OBL_UNSATISFIED_OBLIGATION)

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](https://github.com/spotbugs/spotbugs/blob/master/CONTRIBUTING.md)
* [malinglist](https://github.com/spotbugs/discuss/issues?q=)

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**

1. Internationalization(3)->Dubious method used->Reliance on default encoding

Found reliance on default encoding in SimpleWebServer.processRequest(Socket): new java.io.InputStreamReader(InputStream)

# [Reliance on default encoding](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#DM_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. Use an alternative API and specify a charset name or Charset object explicitly.

**Bug kind and pattern: Dm - DM\_DEFAULT\_ENCODING**

Found reliance on default encoding in SimpleWebServer.processRequest(Socket): new java.io.OutputStreamWriter(OutputStream)

# [Reliance on default encoding](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#DM_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. Use an alternative API and specify a charset name or Charset object explicitly.

**Bug kind and pattern: Dm - DM\_DEFAULT\_ENCODING**

Found reliance on default encoding in SimpleWebServer.serveFile(OutputStreamWriter, String): new java.io.FileReader(String)

# [Reliance on default encoding](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#DM_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. Use an alternative API and specify a charset name or Charset object explicitly.

**Bug kind and pattern: Dm - DM\_DEFAULT\_ENCODING**

1. Dodgy code(3)->Misuse of static field->write to static field from from instance method

Write to static field SimpleWebServer.dServerSocket from instance method new SimpleWebServer()

# [Write to static field from instance method](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD)

This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

**Bug kind and pattern: ST - ST\_WRITE\_TO\_STATIC\_FROM\_INSTANCE\_METHOD**

Null pointer dereference->Dereference of the result of readLine() without nullcheck

Dereference of the result of readLine() without nullcheck in SimpleWebServer.processRequest(Socket)

# [Dereference of the result of readLine() without nullcheck](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#NP_DEREFERENCE_OF_READLINE_VALUE)

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.

**Bug kind and pattern: NP - NP\_DEREFERENCE\_OF\_READLINE\_VALUE**

RunTimeException capture->Exception is caught when Exception is not thrown

Exception is caught when Exception is not thrown in SimpleWebServer.serveFile(OutputStreamWriter, String)

[**Exception is caught when Exception is not thrown**](https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html#REC_CATCH_EXCEPTION)

This 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.

A better approach is to either explicitly catch the specific exceptions that are thrown, or to explicitly catch RuntimeException exception, rethrow it, and then catch all non-Runtime Exceptions, as shown below:

try {

...

} catch (RuntimeException e) {

throw e;

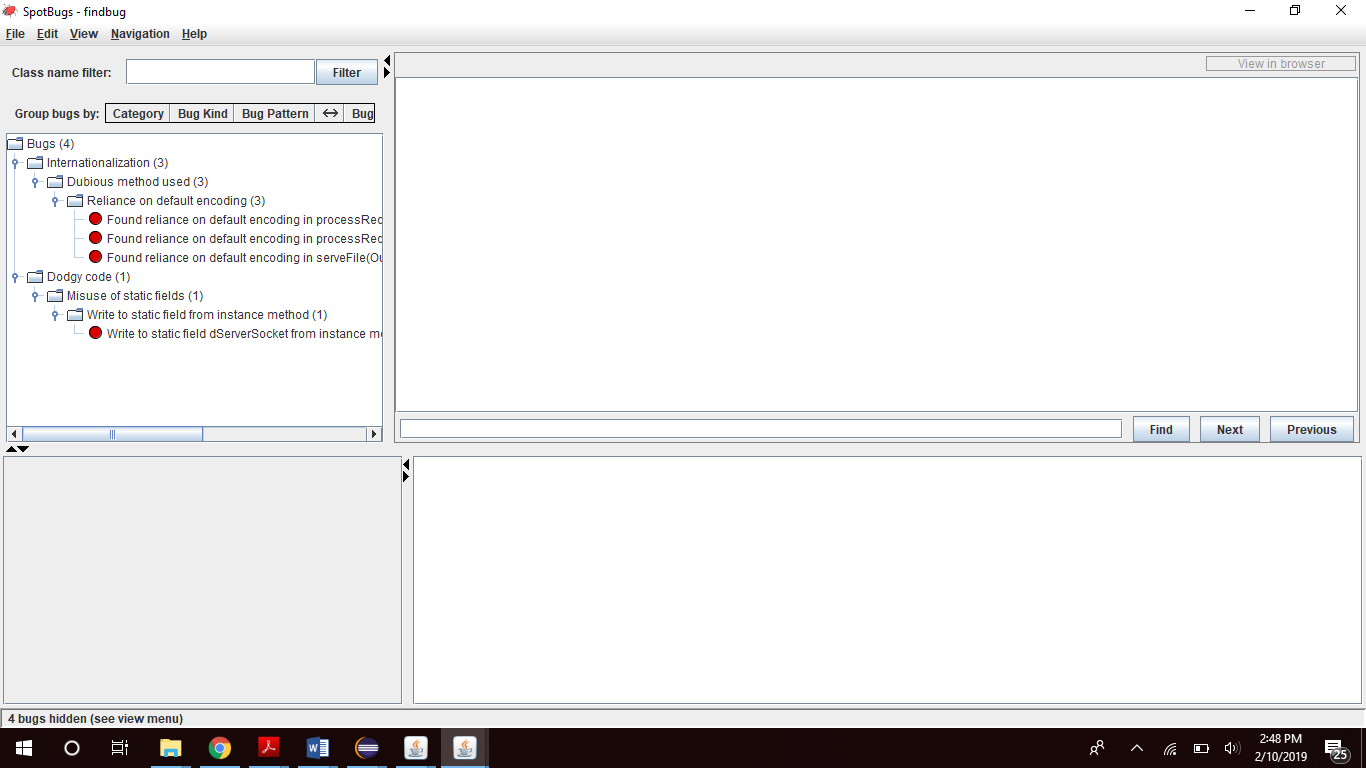
} catch (Exception e) {

... deal with all non-runtime exceptions ...

}

**Bug kind and pattern: REC - REC\_CATCH\_EXCEPTION**

* Changing priority to High Priority, bugs found(4):



**Tool Used: SonarLint**

1. **public** SimpleWebServer () **throws** Exception

# Generic exceptions should never be thrown (squid:S00112)

 Code smell  Major

Using such generic exceptions as Error, RuntimeException, Throwable, and Exception prevents calling methods from handling true, system-generated exceptions differently than application-generated errors.

1. *dServerSocket* = **new** ServerSocket (***PORT***);

# Static fields should not be updated in constructors (squid:S3010)

 Code smell  Major

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.

Instead remove the field's static modifier, or initialize it statically.

1. **while** (**true**) {

/\* wait for a connection from a client \*/

Socket s = *dServerSocket*.accept();

/\* then process the client's request \*/

processRequest(s);

}

# Loops should not be infinite (squid:S2189)

 Bug  Blocker

An infinite loop is one that will never end while the program is running, i.e., you have to kill the program to get out of the loop. Whether it is by meeting the loop's end condition or via a break, every loop should have an end condition.

1. StringBuffer sb = **new** StringBuffer();

# Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used (squid:S1149)

 Code smell  Major

Early classes of the Java API, such as Vector, Hashtable and StringBuffer, were synchronized to make them thread-safe. Unfortunately, synchronization has a big negative impact on performance, even when using these collections from a single thread.

It is better to use their new unsynchronized replacements:

* ArrayList or LinkedList instead of Vector
* Deque instead of Stack
* HashMap instead of Hashtable
* StringBuilder instead of StringBuffer

**Example of a Bug found by both the tools:**

Spotbugs:

**Bug**: Write to static field SimpleWebServer.dServerSocket from instance method new SimpleWebServer()

This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

**Rank**: Of Concern (15), **confidence**: High  
**Pattern**: ST\_WRITE\_TO\_STATIC\_FROM\_INSTANCE\_METHOD   
**Type**: ST, **Category**: STYLE (Dodgy code)

SonarLint:

# Static fields should not be updated in constructors (squid:S3010)

 Code smell  Major

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.

Instead remove the field's static modifier, or initialize it statically.

**Example of a bug missed by one while detected by other:**

Spotbugs:

**Bug**: Dereference of the result of readLine() without nullcheck in SimpleWebServer.processRequest(Socket)

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.

**Rank**: Of Concern (15), **confidence**: Normal  
**Pattern**: NP\_DEREFERENCE\_OF\_READLINE\_VALUE   
**Type**: NP, **Category**: STYLE (Dodgy code)

SonarLint:

Missed by SonarLint

* Manally detected errors
* Both of the errors which were detected manually were detected by Spotbugs as well while SonarLint missed one of those, it didn’t find the error: Dereference of the result of readLine() without nullcheck in SimpleWebServer.processRequest(Socket)