Errors:

Resource		Description
SimpleWebServer.java	null	Catch a list of specific exception subtypes instead.
SimpleWebServer.java	null	Make this line start after 2 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 4 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 4 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 4 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 4 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 4 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 6 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 6 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 6 spaces to indent the code consistently.
SimpleWebServer.java	null	Make this line start after 6 spaces to indent the code consistently.
SimpleWebServer.java	null	Move the "" string literal on the left side of this string comparison.
SimpleWebServer.java	null	Move the "GET" string literal on the left side of this string comparison.
SimpleWebServer.java	null	Move the array designator from the variable to the type.
SimpleWebServer.java	null	Move the declaration of "sb" closer to the code that uses it.
SimpleWebServer.java	null	Move this "catch" on the same line that the previous closing curly brace.
SimpleWebServer.java	null	Move this "else" on the same line that the previous closing curly brace.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java	null	Move this left curly brace to the beginning of next line of code.

SimpleWebServer.java null Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java null Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java null Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java null Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java null Move this left curly brace to the beginning of next line of code.
SimpleWebServer.java null Remove this use of "java.io.FileReader"
SimpleWebServer.java null Remove this use of constructor "FileReader(String)"
SimpleWebServer.java null Remove this use of constructor "InputStreamReader(InputStream)"
SimpleWebServer.java null Remove this use of constructor "OutputStreamWriter(OutputStream)"
SimpleWebServer.java null Replace all tab characters in this file by sequences of white-spaces.
SimpleWebServer.java null Use 'java.io.Writer' here; it is a more general type than 'OutputStreamWriter'.
SimpleWebServer.java null Define and throw a dedicated exception instead of using a generic one.
SimpleWebServer.java null Define and throw a dedicated exception instead of using a generic one.
SimpleWebServer.java null Either log or rethrow this exception.
SimpleWebServer.java null Remove this assignment of "dServerSocket". [+1 location]
SimpleWebServer.java null Replace the synchronized class "StringBuffer" by an unsynchronized one such as "StringBuilder".
SimpleWebServer.java null Explicitly import the specific classes needed.
SimpleWebServer.java null Explicitly import the specific classes needed.
SimpleWebServer.java null Explicitly import the specific classes needed.
SimpleWebServer.java null Missing curly brace.
SimpleWebServer.java null Missing curly brace.
SimpleWebServer.java null Add an end condition to this loop.
SimpleWebServer.java null Add or update the header of this file.
SimpleWebServer.java null Remove this throws clause.
SimpleWebServer.java null Use try-with-resources or close this "FileReader" in a "finally" clause.

Exception handlers should preserve the original exceptions

(java:S1166)
Code smellMajor

When handling a caught exception, the original exception's message and stack trace should be logged or passed forward.

Noncompliant Code Example

```
try {
  /* ... */
} catch (Exception e) {   // Noncompliant - exception is lost
  LOGGER.info("context");
}

try {
  /* ... */
} catch (Exception e) {  // Noncompliant - exception is lost (only message is
preserved)
  LOGGER.info(e.getMessage());
}

try {
  /* ... */
} catch (Exception e) {  // Noncompliant - original exception is lost
  throw new RuntimeException("context");
}
```

Compliant Solution

```
try {
  /* ... */
} catch (Exception e) {
  LOGGER.info(e);  // exception is logged
}

try {
  /* ... */
} catch (Exception e) {
  throw new RuntimeException(e);   // exception stack trace is propagated
}

try {
  /* ... */
} catch (RuntimeException e) {
  doSomething();
  throw e;  // original exception passed forward
} catch (Exception e) {
  throw new RuntimeException(e);  // Conversion into unchecked exception is also
allowed
}
```

Exceptions

`InterruptedException`, `NumberFormatException`, `DateTimeParseException`, `ParseException` and `MalformedURLException` exceptions are arguably used to indicate nonexceptional outcomes. Similarly, handling `NoSuchMethodException` is often required when dealing with the Java reflection API.

Because they are part of Java, developers have no choice but to deal with them. This rule does not verify that those particular exceptions are correctly handled.

```
int myInteger;
try {
  myInteger = Integer.parseInt(myString);
} catch (NumberFormatException e) {
  // It is perfectly acceptable to not handle "e" here
  myInteger = 0;
}
```

Furthermore, no issue will be raised if the exception message is logged with additional information, as it shows that the developer added some context to the error message.

```
try {
  /* ... */
} catch (Exception e) {
  String message = "Exception raised while authenticating user: " +
e.getMessage();
  LOGGER.warn(message); // Compliant - exception message logged with some
contextual information
}
```

See

- OWASP Top 10 2017 Category A10 - Insufficient Logging & Monitoring
- CERT, ERR00-J. - Do not suppress or ignore checked exceptions
- MITRE, CWE-778 - Insufficient Logging

Wildcard imports should not be used (java:S2208)

Code smellCritical

Blindly importing all the classes in a package clutters the class namespace and could lead to conflicts between classes in different packages with the same name. On the other hand, specifically listing the necessary classes avoids that problem and makes clear which versions were wanted.

Noncompliant Code Example

```
import java.sql.*; // Noncompliant
import java.util.*; // Noncompliant

private Date date; // Date class exists in java.sql and java.util. Which one is
this?
```

Compliant Solution

```
import java.sql.Date;
import java.util.List;
import java.util.ArrayList;

private Date date;
```

Exceptions

Static imports are ignored by this rule. E.G.

```
import static java.lang.Math.*;
```

Loops should not be infinite (java:S2189)

BugBlocker

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.

Noncompliant Code Example

```
for (;;) {  // Noncompliant; end condition omitted
  // ...
}
```

```
int j;
while (true) { // Noncompliant; end condition omitted
  j++;
}

int k;
boolean b = true;
while (b) { // Noncompliant; b never written to in loop
  k++;
}
```

Compliant Solution

```
int j;
while (true) { // reachable end condition added
  j++;
  if (j  == Integer.MIN_VALUE) {  // true at Integer.MAX_VALUE +1
    break;
  }
}

int k;
boolean b = true;
while (b) {
  k++;
  b = k < Integer.MAX_VALUE;
}
```

See

- CERT, MSC01-J. - Do not use an empty infinite loop

Variables should not be declared before they are relevant

(java:S1941)
Code smellMinor

For the sake of clarity, variables should be declared as close to where they're used as possible. This is particularly true when considering methods that contain early returns and the potential to throw exceptions. In these cases, it is not only pointless, but also confusing to declare a variable that may never be used because conditions for an early return are met first.

Noncompliant Code Example

```
public boolean isConditionMet(int a, int b) {
  int difference = a - b;
  MyClass foo = new MyClass(a);  // Noncompliant; not used before early return

  if (difference < 0) {
    return false;
  }

  // ...

  if (foo.doTheThing()) {
    return true;
  }
  return false;
}
```

Compliant Solution

```
public boolean isConditionMet(int a, int b) {
  int difference = a - b;

  if (difference < 0) {
    return false;
  }

  // ...

  MyClass foo = new MyClass(a);
  if (foo.doTheThing()) {
    return true;
  }
  return false;
}
```

Array designators "[]" should be on the type, not the variable
(java:S1197)
Code smellMinor

Array designators should always be located on the type for better code readability. Otherwise, developers must look both at the type and the variable name to know whether or not a variable is an array.

Noncompliant Code Example

```
int matrix[][];   // Noncompliant
int[] matrix[];   // Noncompliant
```

Compliant Solution

```
int[][] matrix;   // Compliant
```

Strings literals should be placed on the left side when checking for equality (java:S1132)
Code smellMinor

It is preferable to place string literals on the left-hand side of an `equals()` or `equalsIgnoreCase()` method call.

This prevents null pointer exceptions from being raised, as a string literal can never be null by definition.

Noncompliant Code Example

```
String myString = null;

System.out.println("Equal? " + myString.equals("foo"));                        //
Noncompliant; will raise a NPE
System.out.println("Equal? " + (myString != null && myString.equals("foo")));  //
Noncompliant; null check could be removed
```

Compliant Solution

```
System.out.println("Equal?" + "foo".equals(myString));
```

Source code should be indented consistently (java:S1120)
Code smellMinor

```
  }
}
```

Proper indentation is a simple and effective way to improve the code's readability. Consistent indentation among the developers within a team also reduces the differences that are committed to source control systems, making code reviews easier.

This rule raises an issue when indentation does not match the configured value. Only the first line of a badly indented section is reported.

Noncompliant Code Example

With an indent size of 2:

```
class Foo {
  public int a;
   public int b;   // Noncompliant, expected to start at column 4

...

  public void doSomething() {
    if(something) {
          doSomethingElse();  // Noncompliant, expected to start at column 6
  }   // Noncompliant, expected to start at column 4
  }
}
```

Compliant Solution

```
class Foo {
  public int a;
  public int b;

...

  public void doSomething() {
    if(something) {
        doSomethingElse();
    }
  }
}
```

"Exception" should not be caught when not required by called methods (java:S2221)

Code smellMinor

Catching `Exception` seems like an efficient way to handle multiple possible exceptions. Unfortunately, it traps all exception types, both checked and runtime exceptions, thereby casting too broad a net. Indeed, was it really the intention of developers to also catch runtime exceptions? To prevent any misunderstanding, if both checked and runtime exceptions are really expected to be caught, they should be explicitly listed in the `catch` clause.

This rule raises an issue if `Exception` is caught when it is not explicitly thrown by a method in the `try` block.

Noncompliant Code Example

```
try {
  // do something that might throw an UnsupportedDataTypeException or
UnsupportedEncodingException
} catch (Exception e) { // Noncompliant
  // log exception ...
}
```

Compliant Solution

```
try {
  // do something
} catch
(UnsupportedEncodingException|UnsupportedDataTypeException|RuntimeException e) {
  // log exception ...
}
```

or if runtime exceptions should not be caught:

```
try {
  // do something
} catch (UnsupportedEncodingException|UnsupportedDataTypeException e) {
  // log exception ...
}
```

See

- MITRE, CWE-396 - Declaration of Catch for Generic Exception