

# JUnit Test Report

<i>Test Name</i>	<i>Test Description</i>	<i>Success/Fail (runtime)</i>	<i>Cause of Failure</i>
<i>testSellExisting</i>	Test that an event cannot be sold with the same name as an existing one	Success (0.001 s)	N/A
<i>testBuy</i>	Test that the event must exist to be bought	Success (0.001 s)	N/A
<i>testMain</i>	Test main	Success (0.007 s)	N/A
<i>testSell</i>	Test that the event can be sold	Success (0.000 s)	N/A
<i>testDeleteExisting</i>	Test the user exists to delete	Success (0.002 s)	N/A
<i>testAddPaddingCredit</i>	Tests that credit adds padding	Success (0.000 s)	N/A
<i>testAddPaddingNothing</i>	Test that nothing happens for padding	Success (0.000 s)	N/A
<i>testLoopRunsOnce</i>	Test that add padding loop runs one time	Success (0.000 s)	N/A
<i>testLoopRunsZero</i>	Test that add padding loop runs zero time	Success (0.000 s)	N/A
<i>testAddCredit</i>	Test that add credit passes	Success (0.001 s)	N/A
<i>testAddPadding</i>	Test that other tickets adds padding	Success (0.000 s)	N/A
<i>testBuyExisting</i>	Test that the transaction buy succeeds	Success (0.001 s)	N/A
<i>testReadDTF</i>	Test that there is a merged daily transaction to read	Success (0.000 s)	N/A
<i>testCreate</i>	Test that the user can be created	Success (0.002 s)	N/A
<i>testDelete</i>	Test that a non-existing user cannot be deleted	Success (0.002 s)	N/A
<i>testLoopRunsTwice</i>	Test that add padding loop runs two times	Success (0.001 s)	N/A
<i>testAddCreditMax</i>	Test that add credit fails on maximum amount	Success (0.010 s)	N/A
<i>testRefund</i>	Test that the buyer is refunded money from seller	Success (0.001 s)	N/A
<i>testOldFiles1</i>	Test that the old users file exists	Success (0.000 s)	N/A

<i>testOldFiles2</i>	Test that the old tickets file exists	Success (0.000 s)	N/A
<i>testCreateExisting</i>	Test that the user cannot be created if it exists	Success (0.004s)	N/A
<i>testNewFile1</i>	Test that the new current users file is made	Success (0.000s)	N/A
<i>testNewFile2</i>	Test that the new available tickets file is made	Success (0.000s)	N/A

\*All previous failures were because of human input error.

## FindBugs Report

---

Bug Number	Reason for Fix/Not Fix
1	We didn't fix this bug because we decided to use this convention as a group.
2	We did not want to create another variable just so we could append in a better format.
3	We don't know why this is a bug. The stream is closed whenever it's opened.
4	We did not want to create another variable just so we could append in a better format.
5	We did not want to create another variable just so we could append in a better format.

### Bug 1

Bug: The class name backEnd doesn't start with an upper case letter

Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

Rank: Of Concern (16), confidence: Normal

Pattern: NM\_CLASS\_NAMING\_CONVENTION

Type: Nm, Category: BAD\_PRACTICE (Bad practice)

XML output:

```
<BugInstance type="NM_CLASS_NAMING_CONVENTION" priority="2" rank="16" abbrev="Nm"
category="BAD_PRACTICE" first="1">
```

```
<Class classname="backEnd">
```

```
<SourceLine classname="backEnd" start="6" end="395" sourcefile="backEnd.java"
sourcepath="backEnd.java"/>

</Class>

</BugInstance>
```

---

## Bug 2

Bug: backEnd.delete(String) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad

String s = "";

for (int i = 0; i < field.length; ++i) {

    s = s + field[i];

}

// This is better

StringBuffer buf = new StringBuffer();

for (int i = 0; i < field.length; ++i) {

    buf.append(field[i]);

}

String s = buf.toString();
```

Rank: Of Concern (18), confidence: Normal

Pattern: SBSC\_USE\_STRINGBUFFER\_CONCATENATION

Type: SBSC, Category: PERFORMANCE (Performance)

XML output:

```
<BugInstance type="SBSC_USE_STRINGBUFFER_CONCATENATION" priority="2" rank="18"
abbrev="SBSC" category="PERFORMANCE" first="1">

  <Class classname="backEnd">
```

```
<SourceLine classname="backEnd" sourcefile="backEnd.java" sourcepath="backEnd.java"/>
</Class>

<Method classname="backEnd" name="delete" signature="(Ljava/lang/String;)V" isStatic="true">

  <SourceLine classname="backEnd" start="165" end="195" startBytecode="0" endBytecode="483"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>

</Method>

<SourceLine classname="backEnd" start="177" end="177" startBytecode="72" endBytecode="72"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>

</BugInstance>
```

---

### Bug 3

Bug: backEnd.refund(String) may fail to close 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.

Rank: Of Concern (16), confidence: Normal

Pattern: OS\_OPEN\_STREAM

Type: OS, Category: BAD\_PRACTICE (Bad practice)

XML output:

```
<BugInstance type="OS_OPEN_STREAM" priority="2" rank="16" abbrev="OS"
category="BAD_PRACTICE" first="6">

  <Class classname="backEnd">

    <SourceLine classname="backEnd" sourcefile="backEnd.java" sourcepath="backEnd.java"/>

  </Class>

  <Method classname="backEnd" name="refund" signature="(Ljava/lang/String;)V" isStatic="true">

    <SourceLine classname="backEnd" start="280" end="308" startBytecode="0" endBytecode="129"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>

  </Method>

  <Type descriptor="Ljava/io/Reader;" role="TYPE_CLOSEIT">

    <SourceLine classname="java.io.Reader"/>

  </Type descriptor="Ljava/io/Reader;" role="TYPE_CLOSEIT">

  </Method>

</BugInstance>
```

</Type>

<SourceLine classname="backEnd" start="284" end="284" startBytecode="26" endBytecode="26" sourcefile="backEnd.java" sourcepath="backEnd.java"/>

<SourceLine classname="backEnd" start="284" end="284" startBytecode="26" endBytecode="26" sourcefile="backEnd.java" sourcepath="backEnd.java"/>

</BugInstance>

---

#### Bug 4

Bug: backEnd.addcredit(String) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
```

```
String s = "";
```

```
for (int i = 0; i < field.length; ++i) {
```

```
    s = s + field[i];
```

```
}
```

```
// This is better
```

```
StringBuffer buf = new StringBuffer();
```

```
for (int i = 0; i < field.length; ++i) {
```

```
    buf.append(field[i]);
```

```
}
```

```
String s = buf.toString();
```

Rank: Of Concern (18), confidence: Normal

Pattern: SBSC\_USE\_STRINGBUFFER\_CONCATENATION

Type: SBSC, Category: PERFORMANCE (Performance)

XML output:

```
<BugInstance type="SBSC_USE_STRINGBUFFER_CONCATENATION" priority="2" rank="18"
abbrev="SBSC" category="PERFORMANCE" first="1">

  <Class classname="backEnd">

    <SourceLine classname="backEnd" sourcefile="backEnd.java" sourcepath="backEnd.java"/>

  </Class>

  <Method classname="backEnd" name="addcredit" signature="(Ljava/lang/String;)V" isStatic="true">

    <SourceLine classname="backEnd" start="316" end="364" startBytecode="0" endBytecode="820"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>

  </Method>

  <SourceLine classname="backEnd" start="340" end="340" startBytecode="129" endBytecode="129"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>

</BugInstance>
```

---

## Bug 5

Bug: backEnd.addPadding(String, int, double) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
```

```
String s = "";
```

```
for (int i = 0; i < field.length; ++i) {
```

```
    s = s + field[i];
```

```
}
```

```
// This is better
```

```
StringBuffer buf = new StringBuffer();
```

```
for (int i = 0; i < field.length; ++i) {
```

```
    buf.append(field[i]);
```

```
}
```

```
String s = buf.toString();
```

Rank: Of Concern (18), confidence: Normal

Pattern: SBSC\_USE\_STRINGBUFFER\_CONCATENATION

Type: SBSC, Category: PERFORMANCE (Performance)

XML output:

```
<BugInstance type="SBSC_USE_STRINGBUFFER_CONCATENATION" priority="2" rank="18"
abbrev="SBSC" category="PERFORMANCE" first="6">
```

```
<Class classname="backEnd">
```

```
<SourceLine classname="backEnd" sourcefile="backEnd.java" sourcepath="backEnd.java"/>
```

```
</Class>
```

```
<Method classname="backEnd" name="addPadding"
signature="(Ljava/lang/String;ID)Ljava/lang/String;" isStatic="true">
```

```
<SourceLine classname="backEnd" start="366" end="395" startBytecode="0" endBytecode="526"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>
```

```
</Method>
```

```
<SourceLine classname="backEnd" start="380" end="380" startBytecode="92" endBytecode="92"
sourcefile="backEnd.java" sourcepath="backEnd.java"/>
```

```
</BugInstance>
```

We used eclemma to check our code coverage. Eclemma gave us a 99.2% coverage.

backEndTest (Apr 5, 2015 7:39:46 PM)					
Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions	
part5	99.2 %	1,088	9	1,097	
src	99.2 %	1,088	9	1,097	
(default package)	99.2 %	1,088	9	1,097	
backEnd.java	99.0 %	860	9	869	
backEnd	99.0 %	860	9	869	
addcredit(String)	99.3 %	147	1	148	
buy(String)	99.1 %	107	1	108	
copyOldFiles()	98.7 %	74	1	75	
create(String)	98.3 %	59	1	60	
delete(String)	98.6 %	73	1	74	
main(String[])	96.0 %	24	1	25	
readDTF()	98.6 %	71	1	72	
refund(String)	99.2 %	131	1	132	
sell(String)	98.3 %	59	1	60	
addPadding(String, int, double)	100.0 %	112	0	112	
backEndTest.java	100.0 %	228	0	228	

All of the missed instructions were catch methods that didn't run because there was no exception to catch.