JUnit Test Report

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

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

<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>