

.....

Lint4j v.0.9.1

Project Documentation



Table of Contents

1 Lint4j Documentation	
1.1 Overview	1
1.2 Eclipse	2
1.2.1 Eclipse Plugin User Guide	3
1.3 Maven Plugin	4
1.4 Using Lint4j	5
1.5 Detected Problems and Defects	10
1.5.1 Performance	11
1.5.2 Java Language Constructs	14
1.5.3 Java Core API Contracts	16
1.5.4 Architectural problems	18
1.5.5 Immature Code	20
1.5.6 Code portability	21
1.5.7 Serialization	22
1.5.8 Suspicious Code	25
1.5.9 Synchronization and Scalability	27
1.5.10 Readability and Code Size	29
1.5.11 EJB Spec Violations	30
1.6 Conducting Audits	32
1.6.1 Auditing Sun Smart Ticket	33
1.7 Download	39
1.8 Mailing Lists	40
1.9 About us	41

1.1 Overview

News

NEW [Lint4j version 0.9.1](#) was just released. This maintenance release adds an XML formatter and fixes several bugs which are listed in the [changelog](#).

The next release of the [Lint4j Eclipse plugin](#) is planned for end of May. In the mean time, please check out the current version.

Lint4j Summary

Lint4j ("Lint for Java") is a static Java source and byte code analyzer that detects locking and threading issues, performance and scalability problems, and checks complex contracts such as Java serialization by performing type, data flow, and lock graph analysis.

Lint4j was created to help software developers detect defects and security vulnerabilities before writing the first test case. Lint4j saves time during code reviews as well, so developers can focus on getting business logic right. The Ant and Maven plugins enable easy integration into continuous builds with [Cruise Control](#), [AntHill](#) and others.

The checks that are implemented represent the most common problems that were found while implementing products designed for performance and scalability, such as [VisiBroker for Java](#) (the CORBA ORB from Borland, and the basis for the Borland J2EE container), the [WebObjects application server](#) from Apple Computer, the [OpenEJB](#) container, the [OpenORB](#) CORBA implementation, and the [JBoss](#) J2EE application server.

In addition, numerous of the problems described in the following books are detected:

- [Java Pitfalls](#), by Daconta, Monk, Keller, and Bohnenberger, Wiley 2000
- [Effective Java Programming Guide](#), by Joshua Bloch, Addison-Wesley 2001
- [Bug Patterns in Java](#), by Eric Allen, APress 2002
- [Java 2, Performance and idiom guide](#), Craig Larmann, and Rhett Guthrie, Prentice-Hall, 2000
- [The Java Language Specification, 2nd edition](#), by James Gosling, Bill Joy, Guy Steele, Gilad Bracha

1.2 Eclipse

Lint4j Eclipse Plugin

Eclipse is the perfect environment for using Lint4j. Problems can be reported as soon as the source is saved, and the Eclipse problem marker system allows for easy navigation to the offending line in the source file. Lint4j handles Java 1.0 to 1.5 source code. Packages and files can be excluded from the analysis, which is useful for generated code such as parsers, JAXB and web services bindings. The warnings can be individually enabled and disabled for each project, allowing the developer to focus on the audits that matter for his or her project.

The plugin overview is available at the [following link](#) .

Requirements

The plugin requires Eclipse 3.1 or later, running on any JDK 1.4 or later.

Installation

The Plugin is installed using the Eclipse Plugin Manager. Create an update site with the URL <http://www.jutils.com/eclipse-update> or read the [installation instructions](#) .

Screen Shots

Click on each thumbnail for a large picture of the Lint4j features.

1.2.1 **Eclipse Plugin User Guide**

1.3 Maven Plugin

Lint4j Maven Plugin

Lint4j comes with a [Maven Plugin](#) that simplifies generation of reports significantly. This section contains installation and usage instructions.

Requirements

The Lint4j maven plugin has the following requirements:

- JDK or JRE 1.4.x
- Maven 1.x

The plugin has not yet been tested with Maven 2.

Installation

To install the plugin add the following snippet to the [dependencies](#) section of your `project.xml`

```
<dependency>
  <groupId>lint4j</groupId>
  <artifactId>jutils-lint4j-plugin</artifactId>
  <version>1.2</version>
  <type>plugin</type>
</dependency>
```

Then add the following snippet to the `reports` section of the `reports` section of your `project.xml`

```
<report>jutils-lint4j-plugin</report>
```

Add the following line to the project's `maven.properties` file:

```
maven.repo.remote=http://www.ibiblio.org/maven,http://www.jutils.com/maven
```

If you need to connect through a proxy, see the [Maven Properties Page](#) for details.

Plugin Properties

Property Name	Description
<code>maven.lint4j.level</code>	The warning level, default value is "3". Valid range is from 1-5.

Sample Report

Here is a [sample report](#) from the J2EE container project [Geronimo](#).

1.4 Using Lint4j

Using the command line interface

SYNOPSIS

UNIX:

```
lint4j [-J vmoptions] [-v level] -sourcepath path[:path]* [-classpath
path[:path]*] [-exclude packagename]* [-class class[:class]*] packagename+
```

Windows (**Make sure to enclose path arguments in quotes**):

```
lint4j.bat [-J vmoptions] [-home lint4jpath] [-javahome path] [-v level]
-sourcepath path[:path]* [-classpath path[:path]*] [-exclude packagename]*
[-class class[:class]*] [packagename | filename]+
```

Description

The program expects one or more package, class or file names that will be analyzed. The options are as follows:

- -J vmoptions - pass additional options to the Java VM, for example -J \ "-Xms100M -Xmx200M\" for larger heap size. If present, it **must** be the first argument.
- -home path - pass the base directory to the lint4j distribution. Optional if an environment variable exists. (Windows only)
- -javahome path - select a java distribution. Optional, if there is a java executable in the path. (Windows only)
- -v level - specify the verbosity of the emitted warnings. Range is from severe (1) to suggestion (5). Default is 3. Optional.
- -sourcepath path - specify the jar file or directory paths which Lint4j will use to scan for source and class files to analyze, e.g. /Users/Shared/src. Several paths can be specified, separated by the ':' character (';' on Windows), e.g. -sourcepath dir1:dir2. **Classes and packages found using -sourcepath will be part of the analysis of packages (last arguments) and classes as specified with the -class option.**
- -classpath path - specify the paths for all additional classes referenced by the source files, this can be zip files, jar files, or directories. Several paths can be specified, separated by the ':' character (';' on Windows), e.g. -classpath lib/one.jar:lib/two.jar. **There is no analysis performed on classes that are loaded via -classpath.** Optional.
- -exclude packagename - exclude the given package or package prefix in the analysis, e.g. -exclude java.lang excludes just the java.lang package, whereas -exclude java.lang. excludes all packages including and below java.lang. Optional.
- -class class names to be checked, separated by the ':' character. Class names must be fully qualified. Optional.

Instead of a package name you can specify a wildcard package name such as java.util.*, which would check the package java.util and all of its subpackages. However, general wildcards are currently not implemented, so referencing the javax and java package by specifying java* doesn't work.

Examples

Here is how Lint4j is invoked on its own source (the backslash is necessary to prevent a Unix shell from expanding the wildcard character '*'):

```
prompt> lint4j -sourcepath src/main -classpath lib/bcel.jar:lib/ant.jar
com.jutils.lint4j.*
```

Invoking Lint4j on the log4j binary distribution (the backslash is necessary to prevent a Unix shell from expanding the wildcard character'*'):

```
prompt> lint4j -sourcepath /Users/Shared/lib/log4j.jar org.apache.*
```

Invoking Lint4j on two packages in the log4j binary distribution:

```
prompt> lint4j -sourcepath /Users/Shared/lib/log4j.jar org.apache.log4j
org.apache.log4j.spi
```

Invoking Lint4j on two files from the BCEL 5.1 distribution:

```
prompt> lint4j -sourcepath . org/apache/bcel/verifier/VerifierFactory.java
org/apache/bcel/verifier/Verifier.java
```

On Windows it looks like this:

```
prompt> lint4j -sourcepath src/main -classpath "lib/bcel.jar;lib/ant.jar"
com.jutils.lint4j.*
```

The distribution comes with an Ant build file that performs checks on the [JDK 1.4 java code base](#), [Tomcat 5](#), [JBoss 3.2](#), [JBoss 4](#), and on several SUN J2EE examples, such as [ecperf](#), the [Adventure Builder](#), [PetStore](#), and the [Smart Ticket](#) applications, plus Unix shell scripts that check the JDK 1.4 java code base, Tomcat 5, and JBoss 3.2, and JBoss 4.

The section about [setting Lint4j to audit PetStore](#) contains more tips to speed up the configuration.

Ant Integration

Lint4j comes with its own Ant task, allowing it to run in the same VM as Ant.

Installation

The Ant plugin is enabled by copying the LINT4J_HOME/jars/lint4j.jar to ANT_HOME/lib, or by declaring a task definition such as the following in the build.xml file.

```
<taskdef name="lint4j" classname="com.jutils.lint4j.ant.Lint4jAntTask">
  <classpath><pathelement location="{distdir}/jars/lint4j.jar"/></classpath>
</taskdef>
```

Lint4j Task Attributes

sourcepath	specify the paths that contain the source files for Lint4j to analyze. Several paths can be specified, separated by the ',' character.	Not set	Yes, unless sourcepathRef is used
sourcepathRef	specify the paths that contain the source files for Lint4j to analyze in terms of an Ant reference	Not set	Yes, unless sourcepath is used
classpath	specify the paths for all additional classes referenced by the source files. Several paths can be specified, separated by the ',' character, e.g. classpath="lib/one.jar,lib/two.jar". Classes specified here will not be analyzed.	Not set	No

classpathRef	specify the paths that contain the source files for Lint4j to analyze in terms of an Ant reference	Not set	No
packages	Analyze the classes in the specified packages. The names must be fully qualified, and separated by a comma only (e.g. no whitespace)	Not set	No
classes	Analyzes the given classes. The names must be fully qualified, and separated by a comma only (e.g. no whitespace)	Not set	No
sourcefiles	Analyzes the specified source or class files. The file names may be relative or absolute.	Not set	No
level	the minimum severity level for emitted warnings. Valid range is from 1 (error) to 5 (notice).	3	No
ignorePackages	the package names that should be skipped in the analysis.	Not set	No
exact	Emit only warnings of the same severity as specified by the <code>level</code> argument	false	No

The attributes `sourcepath`, `classpath` and `sourcefiles` are [path-like structures](#), and can also be set via nested `<sourcepath>`, `<classpath>` and `<sourcefiles>` elements, respectively.

The task supports the nested attribute **formatters** to specify one or more formatters that transform the generated reports into text format.

formatters	specify one or more formatters that transform the generated reports into text format	Generated text output to System.out	No
------------	--	-------------------------------------	----

A **<formatters>** element contains one or more `<formatter>` elements:

toFile	the file name the report is written to	System.out	No
type	the output format. The types "text" and "xml" are currently supported.	No	Yes

Examples

Here is a task snippet from the example ant file that checks the tomcat 5 code base. It checks all classes in `org.apache` and subpackages that can be found in the **sourcepath**, and resolving all types against the contents of **classpath**. The results will be formatted as text, with output going to the console and to the file `"/tmp/tomcat5.out"`

```
<target name="check-tomcat5" description="Perform checks on Tomcat 5">
  <lint4j
    ignorePackages="org.apache.tomcat.util.net.puretls,org.apache.coyote.tomcat3,org.apache.jsp.tomcat33"
    packages="org.apache.*" level="${lint4j.level}" exact="${lint4j.exact}">
    <sourcepath>
      <dirset dir="${tomcat5.src.path}">
        <include name="**/src/share" />
        <include name="**/src/java" />
        <exclude name="**/test/**" />
      </dirset>
      <pathelement path="${tomcat5.src.path}/jakarta-tomcat-connectors/util/java" />
      <pathelement path="${tomcat5.src.path}/jakarta-tomcat-connectors/naming/src" />
    </sourcepath>
    <classpath>
      <fileset dir="/var/tmp/tomcat5-deps/">
```

```
        <include name="**/*.jar" />
    </fileset>
</classpath>
<formatters>
  <formatter type="text" />
  <formatter type="text" toFile="/tmp/tomcat5.out"/>
  <formatter type="xml" toFile="/tmp/tomcat5.xml"/>
</formatters>
</lint4j>
</target>
```

The section about [setting Lint4j to audit Smart Ticket](#) contains more tips to speed up the configuration of the Ant task.

Lint4j needs a lot of memory for large projects. If the lint4j Ant task runs out of memory, invoke Ant like this to increase the maximum memory size on Unix:

```
env ANT_OPTS="-Xmx300M" ant mygoal
```

Emacs JDEE Integration

[Nascif Abousalh-Neto](#) has generously contributed a [JDEE plugin](#) for Lint4j. JDEE is the "Java Development Environment for Emacs", an add-on software package that turns Emacs into a comprehensive system for creating, editing, debugging, and documenting Java applications.

Standard Emacs Integration

To use Lint4j from Emacs, hit Meta-x compile, then enter the correct call to the launch script, for example: `lint4j -sourcepath src/mystuff -classpath lib/my.jar com.mystuff.*` More frequently Lint4j will be invoked from a build system, such as make, or Ant. Ant should be invoked with the `-emacs` flags from Emacs for the source code navigation to work: `ant -emacs` Once Lint4j finished processing the source of the warning can be viewed by hitting return when a warning is selected. Emacs will load the source file and position the cursor at the right line, as seen in the screen shot below.

The screenshot shows an Emacs editor window titled 'emacs@localhost.localdomain'. The menu bar includes 'File Edit Options Buffers Tools Java Help'. The toolbar contains icons for file operations and editing. The main window displays lint warnings for several Java files in the 'performance' package, followed by the source code for 'ThreadCurrentThread.java'.

```

/home/reich/Software/Javalint/src/testfiles/performance/CheckExit.java:18: (2): The Java VM shutdown method "System.exit()" was detected.
/home/reich/Software/Javalint/src/testfiles/performance/CheckGc.java:18: (3): An explicit call for garbage collection "System.gc()" was detected.
/home/reich/Software/Javalint/src/testfiles/performance/CheckGc.java:19: (3): An explicit call for garbage collection "Runtime.gc()" was detected.
/home/reich/Software/Javalint/src/testfiles/performance/CheckHalt.java:18: (2): The Java VM shutdown method "Runtime.halt()" was detected.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:5: (2): The field "by" of type byte can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:7: (2): The field "bool" of type boolean can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:9: (2): The field "s" of type short can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:11: (2): The field "i" of type int can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:13: (2): The field "l" of type long can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:15: (2): The field "f" of type float can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:17: (2): The field "d" of type double can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:19: (2): The field "c" of type char can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:21: (2): The field "str" of type java.lang.String can be made static, since it is immutable.
/home/reich/Software/Javalint/src/testfiles/performance/ConstantFields.java:23: (2): The field "clz" of type java.lang.Class can be made static, since it is immutable.
-----
-u** *compilation* (Compilation:exit [0])--L15-- 2*-
package performance;
class ThreadCurrentThread {
    void m() {
        Thread.currentThread();
        Thread.currentThread();
    }
    void m2() {
        Thread.currentThread();
    }
}
-----
ThreadCurrentThread.java (Java CVS-1.1 Abbrev)--L7--All-----

```

1.5 Detected Problems and Defects

Build system and IDE Integration

Lint4j has been designed as an embeddable framework for source code (Java 1.4) and byte code analysis. It can be integrated into any kind of IDE or build system. Currently shipping is a commandline interface, an Ant task, and Emacs integration

Detected Problems and Enforced Contracts

- [Architectual problems](#)
- [Contracts defined in the Java core API](#)
- [Java Language Constructs](#)
- [Immature code](#)
- [Performance](#)
- [Code portability](#)
- [The Serialization and Externalization contracts](#)
- [Suspicious coding and likely bugs](#)
- [Java Synchronization and Scalability](#)
- [Coding patterns that impact readability and code size](#)
- [Violations of the EJB specification](#)

1.5.1 Performance

Performance problems

Unnecessary instanceof operator

Not all Java compilers optimize unnecessary instanceof operators away, which results in additional costs at run time. This operation can also lead to dynamically unreachable code.

Solution: Remove the instanceof

Severity level: 2

Using the += operator on java.lang.String

The += operator for the java.lang.String type creates a new StringBuffer instance [using the default constructor](#) (which means an initial capacity of 16) under the cover, and then calling append() for each argument. It is more efficient to explicitly create and use StringBuffer for the same task, especially when there are multiple += operations on the same String variable.

Solution: don't use this operator repeatedly when the data is constructed, use one StringBuffer instead.

Severity level: 3

Using StringBuffer.setLength()

The init() method throws away its internal buffer and allocates a new one of size 16, even if it was created with a larger initial size. This can introduce unexpected performance problems as explained in the [last section](#). In some implementations the original char array is overwritten with a zero value, which takes a lot more time for larger data sets than just throwing it away.

Solution: create a new StringBuffer instead of using StringBuffer.setLength()

Severity level: 3

Repeated calls to java.lang.Thread.currentThread

currentThread() is an expensive method call, because it is a native method and usually a kernel call. The HotSpot client compilers cannot inline such a method call, and have to use the interpreter to call into native code.

Solution: never use more than one call per method-body, and cache the result in a local variable. If possible, cache at a higher level and pass as a method argument.

Severity level: 3

Using String.getBytes() or String.getBytes(String) with ASCII data

Both methods perform quite expensive character conversion, which can be avoided for ASCII data.

Severity level: 3

Calling System.gc() or Runtime.gc()

Sun advises not call these methods [in section 31 of this document](#).

Solution: remove this method call.

Severity level: 3

Using the default constructor of java.lang.StringBuffer

The StringBuffer default constructor creates a buffer that holds 16 characters by default. In many cases a significantly larger size is needed. By choosing to low a limit the class will perform frequent memory allocations and copies.

Solution: use the StringBuffer(int) constructor with an estimate of the final size

Severity level: 4

StringBuffer fields

StringBuffers can grow quite large, and can become a memory sink if the owning class has a long life time.

Solution: null out the reference as soon as possible

Severity level: 4

Using the java.lang.String(java.lang.String) constructor

String instances are immutable in the Java programming language. It is not necessary to create a new String object from an existing String, such as when given as an argument in a method call. This only results in a performance penalty due to object creation, more memory usage, and additional bookkeeping during garbage collection.

Solution: remove the call to this constructor

Severity level: 4

Using the java.lang.Boolean(boolean) constructor

Although object creation is fast in most 1.3 and 1.4 VMs, using Boolean.TRUE and Boolean.FALSE has advantages in the following cases. When a lot of long-lived objects are created, there is a additional cost for the garbage-collector once the objects leave the Eden space, because the data needs to be tracked and copied. Using the constants lowers the load.

Solution: use Boolean.TRUE and Boolean.FALSE, or Boolean.valueOf(boolean)

Severity level: 4

Unnecessary casts of a type to the same or a supertype

Not all Java compilers optimize unnecessary casts away, which results in additional costs at run time.

Solution: Remove the cast.

Severity level: 4

Creation of zero-length arrays, instead of using a constant

Such arrays are immutable, and don't need to be recreated. When a lot of long-lived objects are created, there is a additional cost for the garbage-collector once the objects leave the Eden space, because the data needs to be tracked and copied. Using a constants lowers the load.

Solution: create a constant zero length array, and use it instead of creating new arrays.

Severity level: 5

Assigning the default initialization values to fields

Every field is assigned the value zero or null when a class is instantiated by the virtual machine. Explicitly assigning these values is unnecessary, and can make debugging more painful, because the assignments count as a statement.

Solution: don't use assign the default value to a field

Severity level: 5

1.5.2 Java Language Constructs

Java language related problems

A return in a finally block

A `return` statement in a finally block makes the control flow hard to understand; if the statement returns with a value, the try statement will always return with this value, ignoring a previous return value.

Solution: remove the return statement.

Severity level: 1

A throw in a finally block

A `throw` statement in a finally block makes the control flow hard to understand. It will mask an exception or a successful return from the try block, and make appear business code fail when simply the cleanup code failed.

Solution: refactor the exception handling in the try block

Severity level: 1

Empty catch blocks

Empty catch blocks supposed to handle checked exceptions that are not expected to ever happen should handle this case in a more robust way.

Solution: add a Runtime exception to the catch block (e.g. `throw new IllegalStateException();`), or use `assert false;`

Severity level: 2

Catching Error or Throwable

Errors shouldn't be caught by user code without rethrowing, as [recommended by Sun](#); they indicate a severe problem that can't be handled by user code, e.g. out of memory.

Solution: Revisit exception handling, and

Severity level: 2

Switch statements without default branch

Switch statements without a default branch will become a liability once the value range is expanded.

Solution: add a default branch that throws a runtime exception, such as `java.lang.IllegalStateException`, or use `assert false;`

Severity level: 3

Case statements that may fall through

Case statements fall through if not terminated by a `return`, `break` or a `throw` statement, as explained in [section 4.11 of the JLS](#). This often occurs because of a forgotten `break` statement.

Solution: add a control statement, if necessary

Severity level: 3

Pathological Switch statements

A `switch` statement with less than three branches is better expressed as an `if` statement, and may be a result of incomplete coding, or a branch that has been commented out.

Solution: replace by an `if` statement.

Severity level: 3

1.5.3 Java Core API Contracts

Contracts on `java.lang.Object`

Explicit calls to `finalize()`

`finalize()` must never be called directly, it is to be invoked by the Java VM only. See the [Sun API documentation](#) for details.

Solution: remove the call, and consider refactoring the `finalize` method by moving out code that releases resources into a separate method.

Severity level: 1

Overriding `clone` without implementing `Cloneable`

The contract for the `clone` method requires that the implementing class implement the `java.lang.Cloneable` interface, but it doesn't.

Solution: make the class implement `java.lang.Cloneable`

Severity level: 1

Clone implementation uses Constructor

The `clone` method is not marked as `final`, and the implementation uses a constructor call instead of calling `super.clone()` for the object allocation. If subclasses redefine the `clone` method then the `clone` contract will be violated.

Solution: Use `super.clone()` for the object allocation, or don't use `clone` at all to copy an object.

Severity level: 1

Usage of `finalize()`

Objects should not rely on the `finalize()` method to free resource, for several reasons:

- prolongs garbage collection, and delays release of memory
- holds on to resources for too long, if used as the only means to free resources
- relies on the garbage collector to kick in before possibly limited resources are exhausted (e.g. file descriptors)
- `finalize` is not guaranteed to be called (during a VM shutdown, for example)

Solution: consider refactoring the `finalize` method by moving out code that releases resources into a separate method.

Severity level: 2

Empty `finalize()` method

When overriding `finalize`, superclasses should be given the opportunity to clean up as well, by calling `super.finalize()`.

Solution: remove the overriding method

Severity level: 2

Using equals() on Arrays

The `equals()` method on array types tests identity only, in most cases the intent is to test if both arrays contain the same element.

Solution: Use the method `equals` defined in [java.util.Arrays](#) for this purpose.

Severity level: 2

The methods hashCode and equals are not both overridden in a type

The methods `equals` and `hashCode` generally should be defined in the same class to comply with the requirements stated in the [hashCode documentation](#) . If the implementation doesn't fulfill the contract it can't be used as a key in HashMaps.

Solution: implement the contract.

Severity level: 3

1.5.4 Architectural problems

Architectural problems

Local variables that shadow accessible fields

A local variable shadows an accessible and compatible field, which might be a programming error.

Solution: consider using the field or renaming the local variable

Severity level: 1

Field declarations that shadow accessible fields in a super type

A field declared in class shadows an accessible and compatible one in a super class, which might be a programming error.

Solution: remove the field, if appropriate

Severity level: 1

Class implementing an interface already implemented by a superclass

This class declares an interface in its "implements" declaration which is already implemented by a super class. Methods required by that interface might be implemented in the class, inadvertently overriding methods of the super class.

Solution: remove the interface from the "implements" section and check if methods required by the interface are necessary on the offending class. Caveat: serializable classes without a serialVersionUID will change their UID value.

Severity level: 2

Interface declaring a method already declared by an extended interface

An interface declares a method that is already declared by one of the interfaces it extends.

Solution: remove the method from the interface declaration

Severity level: 3

Abstract class without abstract method

This abstract class doesn't define any abstract method, and the abstract keyword was probably only used to prevent instantiation.

Solution: use a private constructor instead, or a protect one if the class has subclasses.

Severity level: 3

Abstract methods overrides abstract method

An abstract methods overrides an abstract method declared in a superclass.

Solution: remove the method in the subclass.

Severity level: 3

Empty method implementation in abstract class

An empty method implementation in an abstract class was detected. It may be better to declare the method abstract, and have concrete subclasses implement the appropriate behavior.

Solution: Consider declaring the method abstract.

Severity level: 3

Abstract class without abstract method

This abstract class doesn't define any abstract method, and the abstract keyword was probably only used to prevent instantiation.

Solution: use a private constructor instead, or a protect one if the class has subclasses.

Severity level: 3

Abstract class without any method

This abstract class doesn't define any method, and could be replaced by a marker interface. Subclasses are then free to use any superclass.

Solution: use an interface instead

Severity level: 3

A class using any of its subclasses

A type referencing one of its derived types breaks object encapsulation. This is not an issue if the class implements a class cluster.

Solution: revisit the inheritance hierarchy and prefer to use polymorphism

Severity level: 3

Poor exception propagation

Poor exception propagation by throwing general purpose exceptions such as `java.lang.Throwable`, `java.lang.Exception`, `java.lang.Error`, or `java.lang.RuntimeException`.

Solution: replace with more specific exceptions from the Java libraries, or create your own.

Severity level: 4

1.5.5 Immature Code

Code Maturity problems

Lint4j checks for calls to the following methods that are used to debug problems during the test phase of a software. These calls are typically removed before the release, but it is easy to forget one.

Call to `java.lang.Throwable.printStackTrace` or `java.lang.Thread.dumpStack`

These debug statements should be removed or replaced with a better suited logging mechanism.

Solution: remove in a production environment

Severity level: 4

Using `System.err` or `System.out`

Using `System.err` or `System.out` indicate possible debug code and should be removed or replaced with a better suited logging mechanism.

Solution: remove in a production environment

Severity level: 4

Using `Vector` or `Hashtable`

These two classes have been superceded by the `Collections` classes. In addition, `Vector` and `Hashtable` are synchronized, and may introduce scalability problems when used in a static context.

Solution: replace `Vector` with `ArrayList`, and `Hashtable` with `HashMap`, or a more suitable collection

Severity level: 3

Empty blocks in `try/finally/if/for/while/do/synchronized` constructs

Empty blocks in `try`, `finally`, `if`, `for`, `while`, `while`, or `synchronized` constructs indicate immature code (e.g. a piece of code that has been commented out) with a negative performance impact.

Solution: ensure that code hasn't been accidentally commented out.

Severity level: 3

1.5.6 Code portability

Code Portability problems

Call to `System.exit` or `Runtime.halt`

Using these methods to communicate successful return status constitutes a dependency on the underlying operating system. Sometimes these calls are used when a fatal error is detected, which makes problems extremely hard to track down.

Solution: avoid these calls, if possible, by using `assert` or throwing an exception.

Severity level: 2

Operating System specific methods

The methods `System.getenv()` and `Runtime.exec()` introduce dependencies on the operating system and its configuration. For example, a program used by `Runtime.exec()` may not even exist on a different platform, or may exist in a different version with potentially different parameters. Similarly, environment variables accessed by `System.getenv()` may or may not exist, and their values may be operating system dependent (e.g. file paths), or may or may not be case sensitive.

Solution: carefully evaluate if these calls are really necessary

Severity level: 4

Hardcoded newline characters

The character sequence for a new line is operating system specific. The characters `\r` and `\n` in string and character literals should therefore be avoided in favor of using `System.getProperty("line.separator")` to ensure portability across operating systems.

Solution: use `System.getProperty("line.separator")`

Severity level: 5

1.5.7 Serialization

Serialization and Externalization problems

The first non-serializable super-class doesn't have an accessible zero-argument constructor

The first non-serializable super-class doesn't have an accessible zero-argument constructor, as required by [section 1.10](#) of the Java Serialization specification.

Solution: add a zero-argument constructor that is accessible to the first serializable subclass.

Severity level: 1

An Externalizable class doesn't have a public zero-argument constructor

An Externalizable class doesn't have a public zero-argument constructor, as required by [section 1.11](#) of the Java Serialization specification.

Solution: add a public zero-argument constructor.

Severity level: 1

Serializable class containing a non-transient field of a non-serializable type

Serializable class containing a non-transient field of a non-serializable type can make serialization fail if the type is an interface, and is an error if the type is a class.

Severity level: 1

Wrong definition of serialVersionUID

The Java Serialization specification mandates the following declaration for `serialVersionUID` in [section 4.6](#):
`private static final long serialVersionUID = 3487495895819393L;`

Solution: correct the declaration.

Severity level: 1

Wrong definition of serialPersistentFields

The Java Serialization specification mandates the following declaration for `serialPersistentFields` in [section 1.5](#):
`private static final ObjectStreamField[] serialPersistentFields =
{new ObjectStreamField("next", List.class)};`

Solution: correct the declaration.

Severity level: 1

Non-static inner classes that implement Serializable w/o a serialVersionUID field

The Java Serialization specification strongly recommends in [section 4.6](#) and [section 1.10](#) that inner classes implementation `java.io.Serializable` use a `serialVersionUID` field.

Solution: correct the declaration.

Severity level: 1

Non-static inner classes where the enclosing class is not serializable

Serialization will fail for these classes, because the inner class has a hidden reference to the enclosing class, which will fail serialization. Please refer to the Java Serialization specification [section 1.10](#) for a detailed explanation.

Solution: correct the declaration.

Severity level: 1

Using serialVersionUID or serialPersistentFields in a class that is not serializable

Using `serialVersionUID` or `serialPersistentFields` in a class that is not serializable is confusing at best, but more likely a programming error. These fields are useless if the class implements `java.io.Externalizable`.

Solution: remove the field or make the class implement `java.io.Serializable`

Severity level: 1

Using serialization-related methods in a class that is not serializable

Using `readObject`, `readResolve`, `writeObject`, or `writeReplace` in a class that is not serializable is confusing at best, but more likely a programming error.

Solution: remove the method or make the class implement `java.io.Serializable`

Severity level: 1

Using serialization methods in a class that is externalizable

Using `readObject` or `writeObject` in an `Externalizable` class is useless, they will never be invoked.

Solution: remove the method.

Severity level: 1

Wrong method signatures for custom serialization

The method signature of one serialization-related method was not declared as per the Java Serialization specification. The correct signatures are defined in [section 2.3-5](#) and [section 1.4-1.6](#) of the Java Serialization specification.

Solution: correct the declaration.

Severity level: 1

Fields defined in serialPersistentFields that don't exist

This is an obvious oversight while refactoring a class.

Solution: remove the entry from `serialPersistentFields`.

Severity level: 1

Detect inherited readExternal() and writeExternal() methods when the subclass has additional serializable fields

Each subclass of an `Externalizable` class that has additional state must implement `readExternal()` and `writeExternal()` for proper externalization.

Solution: implement `readExternal()` and `writeExternal()`.

Severity level: 1

1.5.8 Suspicious Code

Suspicious code or possible bugs

Assignment has no effect

A statement such as `x=x` has been detected. This is usually a scoping error, and most likely to happen in constructors when the parameter names match the field names.

Solution: replace with `this.x=x`, if appropriate.

Severity level: 1

Accidental Assignment

An assignment to a boolean variable has been detected in an expression, e.g. `if (haveSource=true)` that was probably intended to be an equality check instead: `if (haveSource==true)`.

Solution: change the assignment to a comparison, if appropriate

Severity level: 1

Method with constructor name

A method has the same name as the class, and may have been confused with constructor.

Solution: change to a constructor by removing the return type, if appropriate

Severity level: 1

Using equals() or == on Arrays

Both the `equals()` method and the `==` operator check for object identity only. In most cases however, the intention is to check if the arrays have the same content.

Solution: use `java.util.Arrays.equals()` to compare array content

Severity level: 1

Comparing strings with the == operator

Strings, like any other object, should be compared using the `equals()` method. In most cases, it is a programming error if they are not; the `==` operator can return false even for identical Strings, [these situations](#) are described in the JLS. Only Strings that are computable at compile time can be compared reliably with the `==` operator, Strings whose value is computed at run-time must be uniquified using the `intern()` method to achieve the same result. In Java JDK 1.3 and later, the method `String.equals()` tests for reference equality before testing the more expensive by-value equality, so there is no performance improvement when the `==` operator is used, it just makes the code harder to maintain and to get correct.

Solution: use the `equals()` method to compare Strings

Severity level: 2

Comparing objects with the == operator

This is harmless if classes explicitly document this behavior, such as `java.lang.Class`, but can be a

programming error that is hard to spot. Lint4j omits reporting many classes from the Java API, as well as ignoring comparisons with `null` and `this`, commonly used in `equals()` methods, to reduce the number of false positives.

Solution: prefer using the `equals()` method. It is a good idea to encapsulate the implementation decision that objects of a class can be compared using the `==` operator by testing for reference equality first in the `equals` method, avoiding a possibly costly by-value comparison.

Severity level: 4

Equality comparisons with floating point types

Floating point types are inherently imprecise. Using the operators `==` or `!=` might not yield the expected result, and are especially dangerous in a loop.

Solution: consider using an interval

Severity level: 2

Immutable field could be static

Final fields that have object scope and are immutable should be elevated to class scope to reduce memory footprint.

Solution: declare the field static

Severity level: 2

1.5.9 Synchronization and Scalability

Thread-related problems

Arithmetic operations on volatile types

Arithmetic operations on volatile types can result in inconsistent values, and represent programming errors in most cases. It is a common misconception that the atomicity guarantee for volatile types as defined in [Section 17.7 JLS](#) implies that the field can safely be used for arithmetic calculation, but that's not true. In fact, changes to such a field can get lost, and the value can become inconsistent in terms of business rules. This is known as the [lost update problem](#).

Solution: use synchronization instead.

Severity level: 1

Usage of volatile long and double types

Although the The Java Language Specification guarantees atomic access to volatile long and double field in [Section 17.7](#), many Java VM including the SUN VM including the 1.4.0 releases violated the specification. If you are curious, [test your VM](#).

Solution: Use synchronization, or consider using an int or float.

Severity level: 1

Excessively deep lock nesting

This code segments holds three or more locks at the same time, which is a bad "code smell". In most cases, one lock should be sufficient.

Solution: revisit if you really need that many locks

Severity level: 1

Possible deadlocks because of wrong lock ordering

One of the necessary requirements for a deadlock is that one thread of execution attempts to acquire locks in a different order than another thread. This situation can happen in this class due to the order in which synchronized statements are grouped. [This article](#) gives a good overview of this problem.

Solution: correct the order

Severity level: 1

Calls to Object.wait(), notify(), notifyAll() w/o lock

When calling one of the wait(), notify(), or notifyAll() methods, the calling thread must own the lock on the callee, otherwise an exception will occur. Refer to the API documentation of [these methods](#) for details.

Solution: wrap the call with a synchronized statement

Severity level: 1

Calls to Object.wait() with more than one lock held

When calling one of the `wait()` methods the calling threads will be suspended. If it holds other locks during that time, it can block other threads from making progress in the best case, or cause a deadlock in the worst case. Refer to the API documentation of [these methods](#) for details.

Solution: revisit why more than lock must be held during the `wait()` call.

Severity level: 1

Calls to native methods while holding locks

Native methods incur an expensive context switch, especially in Sun's client VM, so it's a good idea to keep them out of synchronized blocks.

Solution: move the call out of the synchronized block, if possible.

Severity level: 2

Inconsistent use of locks when accessing fields

Frequent causes of this kind of error are the [broken double-checked locking idiom](#), or the wrong belief that read access doesn't have to be synchronized. [Chapter 18](#) of the Java Language Specification is the definitive source to understanding the Java Threading model.

Solution: use synchronization each time the field is accessed

Severity level: 3

Using Hashtable in a static field

The class `java.util.Hashtable` is threadsafe by default. Having an instance in class scope can severely impact scalability in a multi-threaded environment. The same is true for all subclasses of `Hashtable`, most notably [`java.util.Properties`](#).

Solution: use a `HashMap`

Severity level: 2

Objects used for locking that are not declared final

Declaring fields that are used as locks `final` ensures that accidental reassignments to a such a field is impossible, which would result in concurrent access of a protected path.

Solution: declare the field `final`

Severity level: 4

1.5.10 Readability and Code Size

Readability and Code Size

Unnecessary return statement

The last statement in a void method was a return statement, which should be omitted.

Solution: remove the statement.

Severity level: 4

Negated if statement

An `if` statement with two branches has a `not` operator as the top-level expression. The statement becomes easier to understand and slightly faster by switching the two branches and removing the negation.

Solution: remove the `not` operator and switch the two branches.

Severity level: 4

1.5.11 EJB Spec Violations

Verifying EJB 2.1 compliance

The checks implemented verify that EJBs comply with the [EJB 2.1](#) specification, which is unfortunately not available as HTML document.

Static fields

Section 25.1.2 does not allow modifiable static fields, because the EJB tier may be deployed in a distributed environment.

Severity level: 2

Thread synchronization

Section 25.1.2 does not allow beans to use thread synchronization, because the EJB tier may be deployed in a distributed environment.

Severity level: 1

Native code

Section 25.1.2 does not allow native code in EJBs.

Severity level: 1

Field is not serializable

All Enterprise Beans extend the Serializable interface, which dictates that all non-transient fields must be serializable, which is not the case for the mentioned field.

Severity level: 1

EJB use prohibited class

Section 25.1.2 prohibits the use of the following classes:

- java.io.File
- java.io.FileInputStream
- java.io.FileOutputStream
- java.io.FileReader
- java.io.FileWriter
- java.net.Socket
- java.net.ServerSocket
- java.io.RandomAccessFile
- java.lang.ClassLoader
- java.lang.Thread
- java.lang.ThreadGroup

Severity level: 1

EJB uses forbidden method call

Section 25.1.2 prohibits the use of:

- AWT and Swing for input and output
- java.io for input and output
- java.io.FileOutputStream
- java.io.FileReader
- java.io.FileWriter
- java.net.Socket
- java.net.ServerSocket
- java.io.RandomAccessFile
- java.lang.ClassLoader
- java.lang.Thread
- java.lang.ThreadGroup

Severity level: 1

Missing default constructor

Sections 7.11.2, 10.6.2, 12.2.2, and 15.7.2 state that the bean must have a public constructor that takes no arguments.

Severity level: 1

Initialization in constructor

Initialization should be performed in `setEntityContext` or `setSessionContext`.

Severity level: 1

Wrong class modifiers for session

Section 7.11.2 requires that a session beans class modifier must be `public`, and must not be `final` nor `abstract`.

Severity level: 1

EJB implements finalize()

Sections 7.11.2, 10.6.2, 12.2.2, and 15.7.2 state that an EJB must not implement the `finalize()` method.

Severity level: 1

Session Bean ejbCreate signature

Sections 7.11.2 states that `ejbCreate` methods must return `void`, and must not be `final` nor `static`.

Severity level: 1

Use of RemoteException

Sections 7.11.2, 10.6.2, and 12.2.2 recommend to use `javax.ejb.EJBException` instead of `RemoteException`.

Severity level: 1

1.6 Conducting Audits

Auditing code with Lint4j

This document provides links to several sample reports generated by Lint4j from well-known server-side projects that are widely deployed.

Document	Description
Setting up Lint4j for command line usage	This document describes how to set up Lint4j for a new project if you want to use the command line interface.
Auditing Sun Blueprint Smart Ticket	This document contains an audit of the Sun Blueprint Smart Ticket application, using the Lint4j Ant integration.

1.6.1 Auditing Sun Smart Ticket

Auditing Sun's Smart Ticket Blueprint Application

This document exemplifies how to conduct a source code audit with Lint4j.

Creating the Ant file

The first step is to construct the source path. Change to the base directory of the project, and then look for the first package of the source code, in this case "com" (aka com.sun.j2ee).

```
/Users/Shared/smart_ticket2.0] % find . -type d -name src
./src
./src/app/client/midp/src
./src/app/server/ejb/src
./src/app/server/web/src
./src/app/shared/src
```

Then we need to find all archives that Smart Ticket compiles against:

```
/Users/Shared/smart_ticket2.0] % find . -name \*.zip -o -name \*.jar
./smart_ticket-client.jar
./src/tools/ant/lib/ant.jar
./src/tools/ant/lib/crimson.jar
./src/tools/ant/lib/jaxp.jar
```

Based on this output we configure the Lint4j Ant task as follows:

```
<lint4j packages="com.sun.j2me.*">
  <sourcepath>
    <dirset dir="${ticket.src.path}">
      <include name="**/src" />
    </dirset>
  </sourcepath>
  <classpath>
    <fileset dir="${adventure.src.path}">
      <include name="**/*.jar" />
    </fileset>
  </classpath>
</lint4j>
```

After the first run Lint4j reports missing classes that reduce the accuracy of the report, so the class path is augmented as follows:

```
<classpath>
  <fileset dir="${adventure.src.path}">
    <include name="**/*.jar" />
  </fileset>
  <pathelement path="${servlet.jar.path}" />
  <pathelement path="${ejb.jar.path}" />
  <pathelement path="${jms.jar.path}" />
  <pathelement path="${mail.jar.path}" />
</classpath>
```

Now we are ready to perform the audit.

Most severe warnings

First, let's have a look at the most severe problems that Lint4j reports for Smart Ticket 2.0.

```

ant -Dlint4j.level=1 -emacs check-ticket
Buildfile: build.xml

check-ticket:
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/MovieRating.java:151:
(1): Statement has no effect, possible scoping problem
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/TheaterSchedule.java:118:
(1): The local variable "showTimes" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.shared.midp.model.TheaterSchedule.MovieSchedule
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/SmartTicketFacadeBean.java:234:
(1): The local variable "account" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.server.ejb.SmartTicketFacadeBean
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/SmartTicketFacadeBean.java:451:
(1): The local variable "account" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.server.ejb.SmartTicketFacadeBean
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/rating/MovieRatingData.java:128:
(1): Statement has no effect, possible scoping problem
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/AccountPopulator.java:1
(1): The local variable "account" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.server.web.admin.populate.AccountPopulator
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/MoviePopulator.java:153
(1): The local variable "movie" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.server.web.admin.populate.MoviePopulator
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/TheaterPopulator.java:1
(1): The local variable "theater" shadows an accessible field with the same name and
compatible type in class
com.sun.j2me.blueprints.smartticket.server.web.admin.populate.TheaterPopulator
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/TheaterSchedulePopulator
(1): The local variable "theaterSchedule" shadows an accessible field with the same name
and compatible type in class
com.sun.j2me.blueprints.smartticket.server.web.admin.populate.TheaterSchedulePopulator
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/LocalModel.java:68:
(1): Statement has no effect, possible scoping problem

```

Looking at the source code for the three scoping problems that were reported, the problems become obvious:

```

public void setLastViewingDate(long
viewingDate) {
    this.lastViewingDate =
lastViewingDate;
    [...]
}

```

```

public void setLastViewingDate(long
viewingDate) {
    this.lastViewingDate = viewingDate;
    [...]
}

```

```

protected static ProgressObserver
progressObserver;
public static void
setProgressObserver(ProgressObserver
progressObserver) {
    progressObserver = progressObserver;
}

```

```

protected static ProgressObserver
progressObserver;
public static void
setProgressObserver(ProgressObserver
progressObserver) {
    this.progressObserver =
progressObserver;
}

```

```
public void setLastViewingDate(long
viewingDate) {
    this.lastViewingDate =
lastViewingDate;
    [...]
}
```

```
public void setLastViewingDate(long
viewingDate) {
    this.lastViewingDate = viewingDate;
    [...]
}
```

These are three severe programming errors that could have been eliminated by Lint4j even before compiling the source files.

Next are several cases where a local variable shadows a field. The implementation appears to be correct, but it is confusing to a reader, which is not a good idea for any application, especially not for a sample application. An example of this kind of problem is below.

```
private AccountLocal account = null;
[.]
if (account == null || !account.getUserName().equals(userName)
    || !account.getPassword().equals(password)) {
    try {
        AccountLocal account = (AccountLocal) accountHome.findByPrimaryKey(userName);
        if (account.getPassword().equals(password)) {
            this.account = account;
        }
    }
}
```

Potentially severe issues

Next comes warning level 3, since no issues are reported for this application for level 2.

```
ant -Dlint4j.level=3 -Dlint4j.exact=true -emacs check-ticket
Buildfile: build.xml

check-ticket:
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/Movie.java:51:
(3): The method "equals" is overridden, but not the method "hashCode"
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/MovieRating.java:52:
(3): The method "equals" is overridden, but not the method "hashCode"
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/RecommendationRecipient.java:52:
(3): The method "equals" is overridden, but not the method "hashCode"
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/Theater.java:51:
(3): The method "equals" is overridden, but not the method "hashCode"
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/midp/SmartTicketBD.java:99:
(3): This catch block silently ignores the exception "javax.ejb.RemoveException".
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/midp/SmartTicketServlet.java:88:
(3): This finally statement is empty and should be removed because it prevents the JIT
from optimizing the code block.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/midp/SmartTicketSessionListener.java:6:
(3): This catch block silently ignores the exception
"com.sun.j2me.blueprints.smartticket.shared.midp.ApplicationException".
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/PopulateServlet.java:17:
(3): This catch block silently ignores the exception
"com.sun.j2me.blueprints.smartticket.server.web.admin.populate.PopulateException".
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/PopulateServlet.java:22:
(3): The serialization specification strongly discourages non-static serializable inner
classes such as
"com.sun.j2me.blueprints.smartticket.server.web.admin.populate.PopulateServlet.ParsingDoneException"
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:89:
(3): The method StringBuffer.setLength() should be avoided in favor of creating a new
StringBuffer.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:167:
(3): The method StringBuffer.setLength() should be avoided in favor of creating a new
StringBuffer.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:189:
(3): The method StringBuffer.setLength() should be avoided in favor of creating a new
StringBuffer.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:278:
(3): This catch block silently ignores the exception "java.lang.NumberFormatException".
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/LocalModel.java:308:
```

```
(3): This catch block silently ignores the exception
"com.sun.j2me.blueprints.smartticket.shared.midp.ApplicationException".
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/MessageHandler.java:53:
(3): This class is declared abstract but does not have an abstract method.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/RemoteModelRequestHandler.java:
(3): This class is declared abstract but does not have an abstract method.
```

The four cases of a missing definition of the hashCode method seem to be harmless in the scope of this application, because these classes are not used as a key in a Map.

The empty finally block detected in SmartTicketServlet shows that something was supposed to happen here...

```
finally {
    // FIXME
}
```

as is the case for the ignored exception in SmartTicketSessionListener

```
} catch (ApplicationException ae) {
    // XXX
}
```

... where we also detect that the body of the method sessionDestroyed was commented out completely.

Noteworthy issues

Next comes warning level 4.

```
ant -Dlint4j.level=4 -Dlint4j.exact=true -emacs check-ticket
Buildfile: build.xml

check-ticket:
[.]
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:161:
(4): The type "java.util.Vector" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:171:
(4): Dont hardcode newline characters, use System.getProperty("line.separator") instead.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:171:
(4): Dont hardcode newline characters, use System.getProperty("line.separator") instead.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:250:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:252:
(4): Dont hardcode newline characters, use System.getProperty("line.separator") instead.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/IndexedResourceBundle.java:258:
(4): Don't hardcode newline characters, use System.getProperty("line.separator")
instead.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/Movie.java:111:
(4): Object types such as "com.sun.j2me.blueprints.smartticket.shared.midp.model.Movie"
should be compared using the equals() method!
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/MovieRating.java:255:
(4): Object types such as
"com.sun.j2me.blueprints.smartticket.shared.midp.model.MovieRating" should be compared
using the equals() method!
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/RecommendationRecipient.java:100:
(4): Object types such as
"com.sun.j2me.blueprints.smartticket.shared.midp.model.RecommendationRecipient" should
be compared using the equals() method!
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/SeatingPlan.java:140:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/SeatingPlan.java:146:
(4): Don't hardcode newline characters, use System.getProperty("line.separator")
instead.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/SyncAnchor.java:108:
```



```

(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/shared/src/com/sun/j2me/blueprints/smartticket/shared/midp/model/Theater.java:106:
(4): Object types such as
"com.sun.j2me.blueprints.smartticket.shared.midp.model.Theater" should be compared using
the equals() method!
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/SmartTicketFacadeBean.java:234:
(4): The cast to type
"com.sun.j2me.blueprints.smartticket.server.ejb.account.AccountLocal" is unnecessary,
since the operand already is of the same type.
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/SmartTicketFacadeBean.java:451:
(4): The cast to type
"com.sun.j2me.blueprints.smartticket.server.ejb.account.AccountLocal" is unnecessary,
since the operand already is of the same type.
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/show/Seating.java:174:
(4): The cast to type "byte" is unnecessary, since the operand already is of the same
type.
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/show/Seating.java:179:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/server/ejb/src/com/sun/j2me/blueprints/smartticket/server/ejb/reservation/ReservationBean.java:120:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/MovieSchedulePopulator.java:120:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/MovieSchedulePopulator.java:120:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:59:
(4): Make sure to null out the reference to type java.lang.StringBuffer as soon as
possible to avoid holding on to too much memory or system resources
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:59:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/server/web/src/com/sun/j2me/blueprints/smartticket/server/web/admin/populate/XMLDBHandler.java:174:
(4): This "if" statement is easier to understand if the negetion is eliminated, and the
two branches are switched
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/util/ApplicationUtilities.java:45:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/util/ApplicationUtilities.java:179:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/LocalModel.java:96:
(4): The type "java.util.Vector" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/ModelFacade.java:357:
(4): This "if" statement is easier to understand if the negetion is eliminated, and the
two branches are switched
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/ModelFacade.java:417:
(4): It is strongly recommended to create a StringBuffer with a reasonable initial size
instead of the default size of 16.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/Preferences.java:72:
(4): The type "java.util.Hashtable" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/RemoteModelProxy.java:64:
(4): The type "java.util.Hashtable" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/SynchronizationAgent.java:83:
(4): The type "java.util.Vector" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/SynchronizationAgent.java:102:
(4): The type "java.util.Hashtable" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/SynchronizationAgent.java:141:
(4): The type "java.util.Hashtable" should be avoided and replaced with its counterpart
from the Collections classes if possible.
src/app/client/midp/src/com/sun/j2me/blueprints/smartticket/client/midp/model/SynchronizationAgent.java:181:
(4): The type "java.util.Hashtable" should be avoided and replaced with its counterpart
from the Collections classes if possible.

```

At this warning level 114 unnecessary return statements at the end of a void method were logged, which increase code size for no reason. In addition, 20 debug statements (System.err.println and printStackTrace)

were detected. These warnings are omitted in the listing above because of large number of occurrences.

There are 8 cases where Vector and Hashtable, both synchronized objects, could be replaced with the unsynchronized collections ArrayList and HashMap.

This concludes our quick code review of the Smart Ticket application.

1.7 Download

System Requirements

Lint4j is written in pure Java, and runs on any platform with JDK or JRE 1.4 or later. Lint4j has been developed and tested extensively on MacOSX 10.2 and 10.3, and on Red Hat Linux 8 and 9 using Sun JDK 1.4.1, 1.4.2, IBM Java 1.4.1, and JRockit 8.1. Windows support has been tested on Windows XP.

License

The use of this software is subject to these [licensing terms](#).

Download

Please subscribe to our [mailing lists](#) for new releases and site updates.

[lint4j-0.9.1.tar.gz](#)

[lint4j-0.9.1.zip](#)

[The latest documentation in PDF format](#)

Installation

Unpack the tar.gz version of lint4j as follows:

```
cd yourfavoritedirectory
guntar zxf lint4j-0.9.1.tar.gz
```

Unpack the zip file on Windows with WinZip. If you want to use the commandline program in addition to the Ant task, you need to right-click on the "My Computer" icon, click the "Advanced" tab, select "Environment Variables". Double-click the "Path" entry in the section "System variables". In the dialog box, append a semicolon followed by the absolute path to the **lint4j-0.9.1/bin** directory in the "Variable value" text field. For example:

```
%\SystemRoot%\system32;%SystemRoot%;%\SystemRoot%\WBem;D:\lint4j-0.9.1\bin
```

The distribution main directory contains:

- [CHANGELOG](#) : The list of new features in this version of Lint4j
- [LICENSE](#) : The license for using Lint4j
- bin - this directory contains the main shell script `lint4j` to launch lint4j. Please refer to the [user guide](#) for details on how to invoke it.
- examples - this directory contains an Ant build file that performs checks on the [JDK 1.4 java code base](#), [Tomcat 5](#), [JBoss 3.2](#), [JBoss 4](#), and on several SUN J2EE examples, such as [ecperf](#), the [Adventure Builder](#), [PetStore](#), and the [Smart Ticket](#) applications, plus Unix shell scripts that check the JDK 1.4 java code base, Tomcat 5, and JBoss 3.2, and JBoss 4.
- jars - this directory contains the application jar file

1.8 Mailing Lists

Lint4j Mailing Lists

The mailing list "lint4j_announce" is used to inform about new releases and site updates. The list is closed; only the administrator can send messages, and your email address is not visible to others.

Subscribe

Send an email to majordomo@jutils.com with "subscribe lint4j_announce@jutils.com" as the **body**

Unsubscribe

Send an email to majordomo@jutils.com with "unsubscribe lint4j_announce@jutils.com" as the **body**

1.9 About us

Contact us

For bug reports, source code licenses, custom development or support please [contact us](#) .

Thank you for using lint4j!