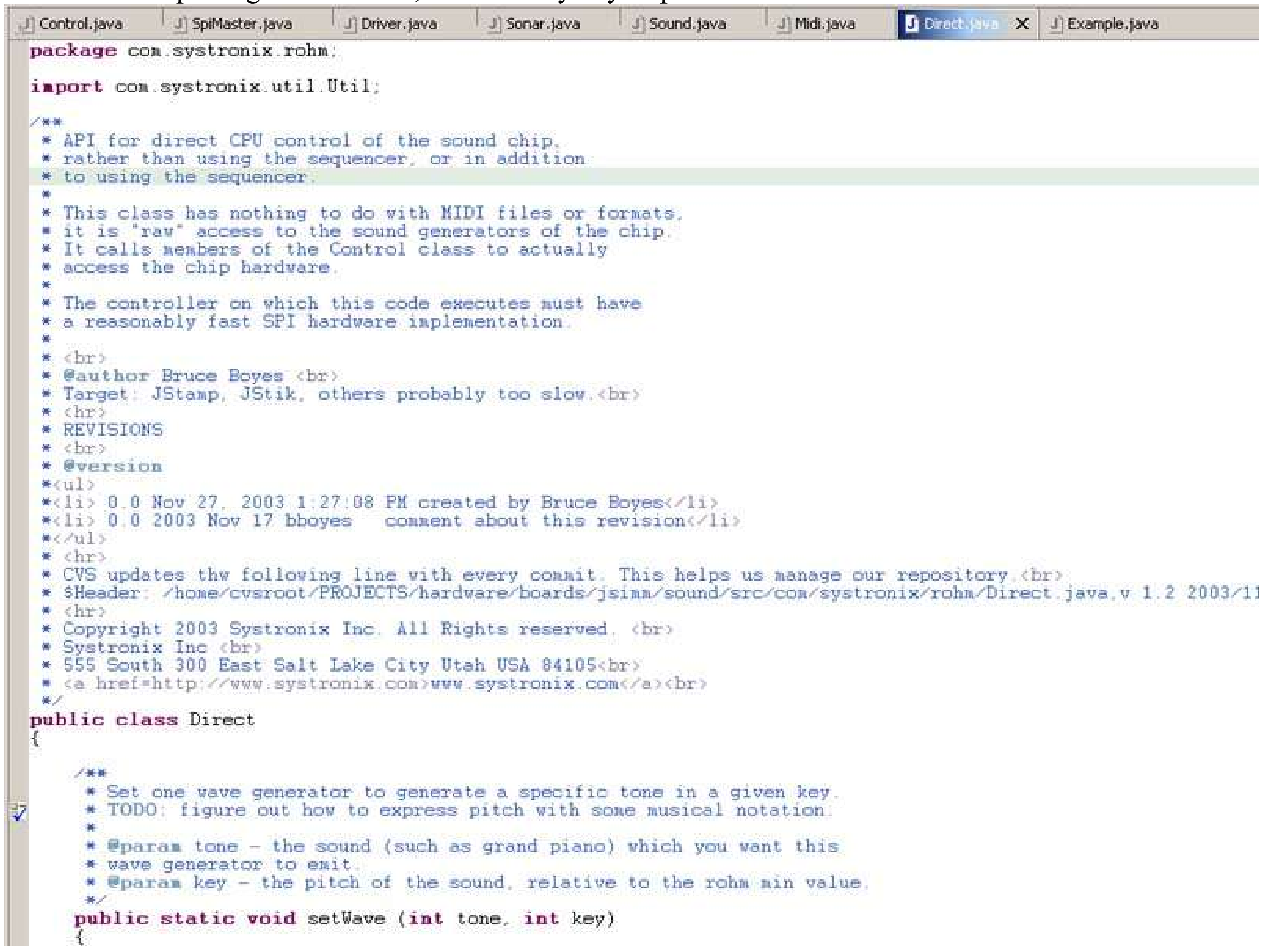
Ben Holt
Suggestions for Coding Standards
2nd Draft bboyes 30 Nov 2003

Section 1: Header

1. 1. All java files should have a header as seen in this screenshot. It identifies the company and copyright, with intended package, filename, target, creation date, creator's name, and a section for revision history that is EASY to add to.
2. Each Revision history line should take up only one line, if multiple things are changed, document each one. If more space is needed, reference a function and include the specifics there. Revision history is an AT A GLANCE understanding. Long diatribes do not belong here.
3. The other labels are of use when something goes wrong (a file gets misnamed, or moved) and for computer searchability.
4. Next is the package declaration, followed by any imports.



```
package com.systronix.rohm;

import com.systronix.util.Util;

/**
 * API for direct CPU control of the sound chip.
 * rather than using the sequencer, or in addition
 * to using the sequencer.
 *
 * This class has nothing to do with MIDI files or formats.
 * it is "raw" access to the sound generators of the chip.
 * It calls members of the Control class to actually
 * access the chip hardware.
 *
 * The controller on which this code executes must have
 * a reasonably fast SPI hardware implementation.
 *
 * <br>
 * @author Bruce Boyes <br>
 * Target: JStamp, JStik, others probably too slow.<br>
 * <hr>
 * REVISIONS
 * <br>
 * @version
 *<ul>
 *<li> 0.0 Nov 27, 2003 1:27:08 PM created by Bruce Boyes</li>
 *<li> 0.0 2003 Nov 17 bboyes   comment about this revision</li>
 *</ul>
 * <hr>
 * CVS updates thw following line with every commit. This helps us manage our repository.<br>
 * $Header: /home/cvsroot/PROJECTS/hardware/boards/jsimm/sound/src/com/systronix/rohm/Direct.java,v 1.2 2003/11
 * <hr>
 * Copyright 2003 Systronix Inc. All Rights reserved. <br>
 * Systronix Inc <br>
 * 555 South 300 East Salt Lake City Utah USA 84105<br>
 * <a href=http://www.systronix.com>www.systronix.com</a><br>
 */
public class Direct
{

    /**
     * Set one wave generator to generate a specific tone in a given key.
     * TODO: figure out how to express pitch with some musical notation.
     *
     * @param tone - the sound (such as grand piano) which you want this
     * wave generator to emit.
     * @param key - the pitch of the sound, relative to the rohm min value.
     */
    public static void setWave (int tone, int key)
    {
```

The eclipse code to generate this header is as follows:

```
${package_declaration}

/**
 * TODO: Brief summary here - one or two sentence overview.
 *
 * <br>
 * DESCRIPTION
 * <br>
 * TODO: Detailed explanation of how this works,
 * what it subclasses or what should sublcass it, etc.
 *
 * <br>
 * Target: Which Java systems - PC, JStamp, TStik, etc?<br>
 * <hr>
 * REVISIONS
 * <br>
 * @version
 *<ul>
 *<li> 0.0 ${date} ${time} created by ${user}</li>
 *<li> 0.0 ${year} Nov 17 bboyes         comment about this revision</li>
 *</ul>
 * <hr>
 * CVS updates the following line with every commit.
 * This helps us manage our repository.<br>
 * $$Header: /home/cvsroot/PROJECTS/templates/eclipse/codetemplates_bboyes.xml,v 1.3
 2003/11/18 23:24:04 bab Exp $$
 * <hr>
 * Copyright ${year} Systronix Inc. All Rights reserved. <br>
 * Systronix Inc <br>
 * 555 South 300 East Salt Lake City Utah USA 84105<br>
 * <a href=http://www.systronix.com>www.systronix.com</a><br>
 */
${type_declaration}
```

Section 2: Class Declaration

1. The first line of a class should be a summary. If one to two sentences are not sufficient, maybe your class is trying to do too much.
2. List any important contracts that the class expects. If something needs to be listed in the forname field, it should be mentioned here.
3. Write a full user explanation of what the class does, any notes that the author might have. This can be as long as the programmer desires.
4. An author tag should be automatically generated for accountability.
5. The class declaration should have a newline after the declaration and before the opening brace.
6. Classes will be named starting with a capitol letter and no spaces or underscores. Instead each word will begin with a capitol letter (like class BusinessAccount). This is consistant with the java standards.

The eclipse generated code is here to be placed under types:

```
/**
 * TODO: Summary - text up to the first period becomes the javadoc summary.<br>
 * Then include detail about how this works, including boundary conditions,
 * assumptions and any known 'gotchas'.
 */
```

Section 3: Class Variables

1. ALL class variables should be declared immediately after the class declaration.
2. No class variables should be sprinkled in the rest of the class body.
3. They should be arranged by access (private, public, protected, default) and then further subdivided by keywords (static vs non-static) and then subdivided by type.
4. Each enumeration should be grouped together as well.
5. Every variable will have a one line summary. A longer description may follow, but a one-line summary is mandatory.
6. Enumerated types will be all capitol letters with an underscore between words. This is consistant with java standards.
7. Variable names will begin with a lowercase letter and capitol letters to represent new words. No underscores or spaces are permitted (like BusinessAccount myBankAccount). This is consistant with java standards.

Section 4: Constructors and Methods

1. All Constructors will be placed immediately after the Class Variables
2. The default constructor will be placed first.
3. The default constructor must appear in all classes, EVEN if it is not used (you may declare it private and eliminate access)
4. Constructors should be organized in ascending number of parameters.
5. Methods must appear after the constructors.
6. Methods must be in alphabetical order.
7. If a method takes an enumerated type, it must specify that in the contract (ie if an int is required but you expect YourClass.ENUMERATED_TYPE_X).
8. All variables must be declared at the BEGINNING of their scope. Method variables will be decalared at the beginning of the function and sorted the same way as specified as class variables.

The eclipse code is as follows:

```
/**
 * TODO: Summary - text up to the first period becomes the javadoc summary.<br>
 * Then include detail about how this works, including boundary conditions,
 * assumptions and any known 'gotchas'.
 * ${tags}
 */

For Overriding methods the Eclipse code is:

/**
 * TODO: Summary - text up to the first period becomes the javadoc summary.<br>
 * Then include detail about how this works, including boundary conditions,
 * assumptions and any known 'gotchas'.
 * ${see_to_overridden}
 */
```

Section 5: Exception Handling

1. No type of class Exception will be caught: The specific Exception (like IOException) should be caught instead.
2. Each specific type of exception will be caught separately. No blanket Exception catching.
3. The only deviation from blanket exception handling is for a last-ditch Safe-System Shutdown. This must be noted as such.
4. No empty catch blocks. NO EXCEPTIONS.
5. No quiet dismissal of exceptions. If a catch block exists, it must do something about the exception.
6. No throwing class Exception or class RuntimeException. These MUST be subclassed before they are used. This eliminates blanket Exception catching.
7. If you create a thread, you are responsible for catching any Exceptions that that specific thread may cause by itself. Because threads are in their own environment and no Exceptions your system thread creates could be caught by a user thread, all problems MUST be dealt with in a reasonable manner. Bad circumstances in your system thread should NOT cause the user program to crash unless there is a global problem.

Section 6: Syntax

1. All function declarations must have a newline separating the declaration and the opening bracket of the function.
2. Every opening bracket will be on its own line, lining up with its closing bracket below. Only comments will be permitted on the same line.
3. No System.out.print() instructions will be left in release code other than a version identifier if needed. All other print statements should be removed OR turned into subclasses of exception. This eliminates random messages to the console that the user cannot do anything about.