

Good Comments

EJUG – Good Code Lightning Talks

March 23, 2011

Warren Blanchet – wskb.ca

[CC-BY-3.0](https://creativecommons.org/licenses/by/3.0/)

Why Comments?

Maintenance

- 10 Generations of programmers work on your code before it is rewritten
- 50% - 60% of change time is spent on understanding the existing code
- You have to communicate with these people

Inaccuracy vs. Distance



Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

Comment Taxonomy

- Marker
- Summary
- Explanation: intent + rationale
- API = Summary + reuse manual

Marker

License and copyright

- Audience is coder, sometimes
- Stick in header - ignorable when not relevant

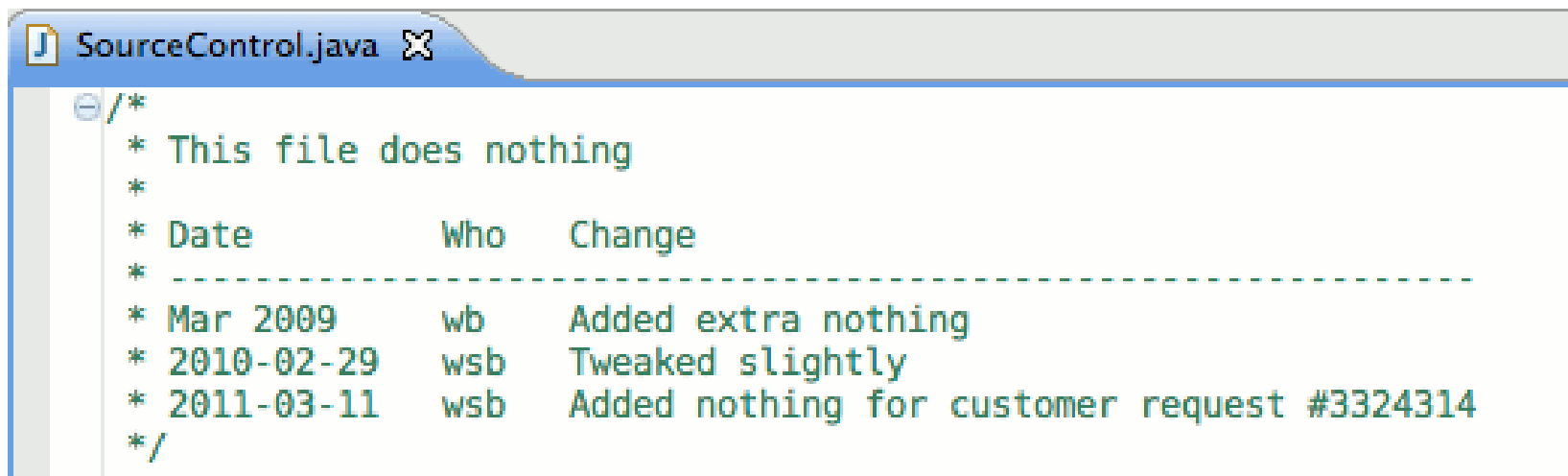
```
LicenseAndCopyright.java ✕
/*
 * Copyright (C) 2011 by Warren S.K. Blanchet
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
package ca.wskb.example;
public class LicenseAndCopyright {
```

```
LicenseAndCopyright.java ✕
+ * Copyright (C) 2011 by Warren S.K. Blanchet
package ca.wskb.example;
public class LicenseAndCopyright {
```

Version Control

```
// public void foo() {  
//  
//     System.out.println(System.getProperty( "otherproperty" ));  
// }  
}
```

Version Control



The image shows a screenshot of a code editor window titled "SourceControl.java". The code content is a version control log. It starts with a comment block: "/* This file does nothing". This is followed by a table with three columns: "Date", "Who", and "Change". The table is separated from the rest of the code by a dashed line. The table contains three rows of data: "Mar 2009" by "wb" with the change "Added extra nothing"; "2010-02-29" by "wsb" with the change "Tweaked slightly"; and "2011-03-11" by "wsb" with the change "Added nothing for customer request #3324314". The code block ends with "*/".

```
/*
 * This file does nothing
 *
 * Date          Who    Change
 * -----
 * Mar 2009      wb     Added extra nothing
 * 2010-02-29    wsb    Tweaked slightly
 * 2011-03-11    wsb    Added nothing for customer request #3324314
 */
```

Version Control

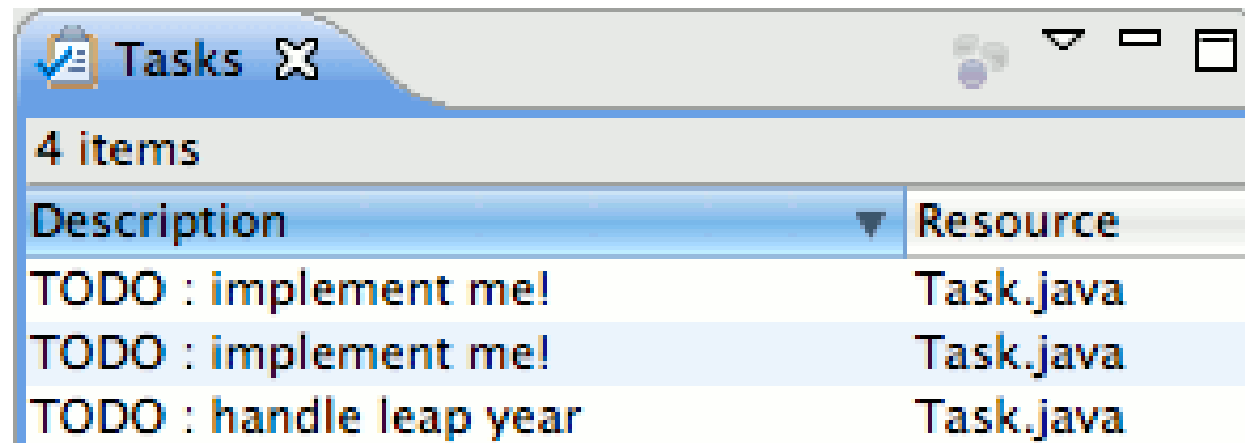
```
public void bar() {  
    // #4323432 - 2011-02-29 - wb - added  
    // #MG34DF3 - 2011-03-11 - wsb - changed from bad value  
    String property = System.getProperty( "someproperty" );  
  
    // #MG34DF3 - 2011-03-11 - wsb - Changed from err to out  
    System.out.println(property);  
}
```

Version Control

- Did you know?
 - There are tools for version control
 - Many are free
- Advantages
 - Up to date
 - Accurate
 - Better tool support
 - Only visible when relevant

Task

```
public Date calculateImportantDateThing() {  
    // TODO: handle leap year
```



The screenshot shows a window titled 'Tasks' with a close button and a search icon. The window contains a list of 4 items. The list has two columns: 'Description' and 'Resource'. The first three items are 'TODO : implement me!' and the last item is 'TODO : handle leap year'. All items are associated with the resource 'Task.java'.

Description	Resource
TODO : implement me!	Task.java
TODO : implement me!	Task.java
TODO : handle leap year	Task.java

Task

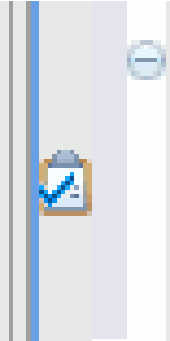
```
public void doImportantThing() {  
    // TODO: implement me!  
}
```

```
public int calculateImportantThing() {  
    throw new UnsupportedOperationException("not yet implemented");  
}
```

Task

```
public int deriveImportantValue() {  
    // TODO: implement me!  
    throw new UnsupportedOperationException("not yet implemented");  
}
```

Task

A snippet of code from a code editor. On the left, there is a vertical sidebar with a blue and grey background, containing a small icon of a clipboard with a checkmark. The code itself is displayed on a light yellow background. It consists of a public method named 'getBackgroundColor()' that returns a 'Color' object. The method body contains a comment '// TODO: figure out what this should be' and a return statement 'return Color.WHITE;'.

```
public Color getBackgroundColor() {  
    // TODO: figure out what this should be  
    return Color.WHITE;  
}
```

Bad: Requirements not program logic

Wish it were BASIC

```
for( Object item : items ) {  
    if( item != null ) {  
        bar(item);  
    } // end if item is null  
} // end for
```

Wish it were BASIC

```
for( Object item : items ) {  
    if( item != null ) {  
        bar(item);  
    } // end if item is null  
} // end for
```

Tools

OK

```
public class Foo {  
    private int bar = 12; // NOPMD - used by native method  
}
```

Better

```
public class Foo {  
    @SuppressWarnings( "unused" ) // used by native method  
    private int bar = 12;  
}
```

Summary

```
// multiply by two
sleepTimeRange = sleepTimeRange * 2;
// ensure no more than max
sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
```



```
// exponential backoff
sleepTimeRange = sleepTimeRange * 2;
sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
```

```
sleepTimeRange = exponentialBackoff( sleepTimeRange );
```

Explanation

```
private int exponentialBackoff( int sleepTimeRange ) {  
    sleepTimeRange = sleepTimeRange * 2;  
    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );  
    return sleepTimeRange;  
}
```

```
private int exponentialBackoff( int sleepTimeRange ) {  
    // multiply by two  
    sleepTimeRange = sleepTimeRange << 1;  
    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );  
    return sleepTimeRange;  
}
```

```
private int exponentialBackoff( int sleepTimeRange ) {  
    /*  
    * Multiply by two using left shift;  
    * increases performance under NicheCorp's java compiler 7.2  
    */  
    sleepTimeRange = sleepTimeRange << 1;  
  
    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );  
    return sleepTimeRange;  
}
```

API: JavaDoc

```
/**
 * Returns the new range for sleep time between retries, as calculated
 * using exponential backoff. The next attempt should be retried after a
 * random wait time bounded by the return value.
 *
 * @param sleepTimeRange the existing range for sleep time
 *
 * @return the new range for sleep time
 */
public int exponentialBackoff( int sleepTimeRange ) {
```



```
/**
 * Returns the new range for sleep time between retries, as calculated
 * using exponential backoff. The next attempt should be retried after a
 * random wait time bounded by the return value.
 * <p>
 * This method is thread-safe.
 *
 * @param sleepTimeRange the existing range for sleep time
 *
 * @return the new range for sleep time
 *
 * @throws IllegalArgumentException if the passed value is negative
 */
public int exponentialBackoff( int sleepTimeRange ) {
    if( sleepTimeRange < 0 ) {
        throw new IllegalArgumentException("sleepTimeRange must be non-negative");
    }
}
```

Miscellaneous

- Avoid net emptiness
- Avoid @author - version control's job
- Use @link

Comment Presentation


```
//  
// I'm a summary comment about the next section  
//  
  
// I'm an explanation comment about the next line or block  
blah("blah");  
YADA.yada();  
  
//  
// next section  
//
```

```
//::// I'm a summary comment about the next section

/*
 * I'm a long explanation comment about the next line
 * or block
 */
blah("blah");
YADA.yada();

//::// next section
```

Tips for Eclipse users

Spelling

spelling

- General
 - Editors
 - Text Editors
 - Spelling**

Spelling

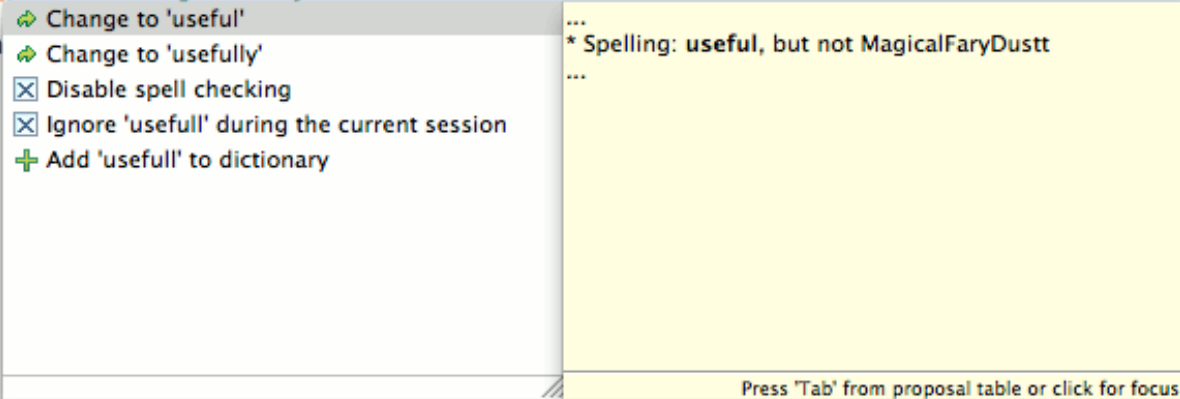
Enable spell checking

Options

- Ignore words with digits
- Ignore mixed case words
- Ignore sentence capitalization
- Ignore upper case words
- Ignore internet addresses
- Ignore non-letters at word boundaries
- Ignore single letters
- Ignore Java string literals
- Ignore '&' in Java properties files

Spelling

```
/*  
 * Spelling: usefull, but not MagicalFaryDustt  
 */  
public class Spellin  
{  
}
```

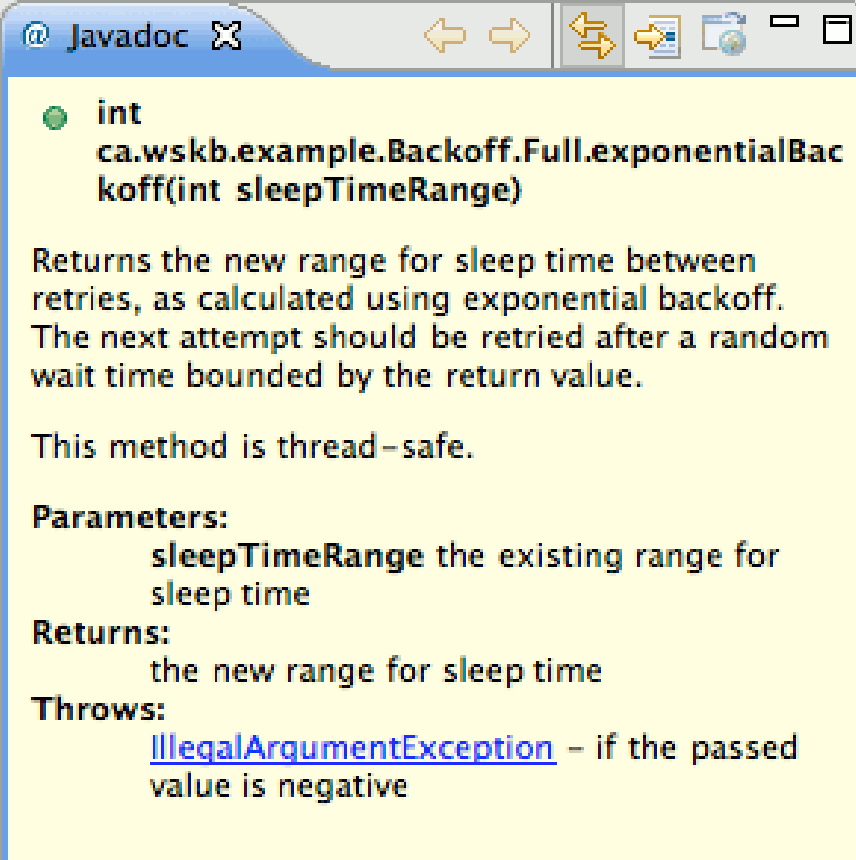


A context menu is displayed over the code editor, showing spelling correction options for the word 'usefull'. The menu items are:

- Change to 'useful'
- Change to 'usefully'
- Disable spell checking
- Ignore 'usefull' during the current session
- + Add 'usefull' to dictionary

The menu also includes a yellow sticky note area with the text: "... * Spelling: useful, but not MagicalFaryDustt ...". At the bottom of the menu, it says "Press 'Tab' from proposal table or click for focus".

JavaDoc view



The screenshot shows a JavaDoc window titled "@ Javadoc". The content is displayed on a yellow background. It features a green bullet point next to the word "int", followed by the package and class name "ca.wskb.example.Backoff.Full.exponentialBackoff" and the method signature "koff(int sleepTimeRange)". Below this, there is a paragraph of text describing the method's purpose: "Returns the new range for sleep time between retries, as calculated using exponential backoff. The next attempt should be retried after a random wait time bounded by the return value." This is followed by a note: "This method is thread-safe." Then, there are sections for "Parameters:", "Returns:", and "Throws:". The "Parameters:" section lists "sleepTimeRange" as the existing range for sleep time. The "Returns:" section states it returns "the new range for sleep time". The "Throws:" section lists "[IllegalArgumentExpection](#)" (note the typo in the image) as an exception thrown if the passed value is negative.

int
ca.wskb.example.Backoff.Full.exponentialBackoff(int sleepTimeRange)

Returns the new range for sleep time between retries, as calculated using exponential backoff. The next attempt should be retried after a random wait time bounded by the return value.

This method is thread-safe.

Parameters:
sleepTimeRange the existing range for sleep time

Returns:
the new range for sleep time

Throws:
[IllegalArgumentExpection](#) - if the passed value is negative

JavaDoc validation

The screenshot displays the 'Javadoc' settings dialog in an IDE. The left-hand navigation pane shows a tree view with 'Javadoc' selected under the 'Compiler' category. The main panel, titled 'Javadoc', contains the following settings:

- Process Javadoc comments
- Severity level for problems in Javadoc comments:
 - Malformed Javadoc comments: Warning
 - Only consider members as visible as: Private
- Validate tag arguments (@param, @throws, @exception, @see, @link)
 - Report non visible references
 - Report deprecated references
- Missing tag descriptions: Validate @return tags
- Missing Javadoc tags: Warning
 - Only consider members as visible as: Private
 - Ignore in overriding and implementing methods
- Missing Javadoc comments: Ignore
 - Only consider members as visible as: Public
 - Ignore in overriding and implementing methods

JavaDoc validation

```
/**
 * This class is useful. See {@link #primary()}.
 */
public class JavaDocValidation {

    /**
     * This method does something useful
     * @param input the input
     * @param configVal the configuration value
     */
    public String primary( String input ) throws IOException {
```

I am always right,
except when I am not

Comments are communication

- Read books — I did:
 - Code Complete, Second Edition
 - Effective Java, Second Edition
- Hear from speakers
- Talk to colleagues
- Listen to customers

Good Comments

- “Good” is subjective and context-sensitive
- Expose yourself to ideas - no obligation
- Know the rationale behind your practices
- Be able to explain them

Good Comments

EJUG – Good Code Lightning Talks

March 23, 2011

Warren Blanchet – wskb.ca

[CC-BY-3.0](https://creativecommons.org/licenses/by/3.0/)