

Documentation and Style

February 14, 2005

1

Documentation: conventions & style

- Documentation is essential for reuse and maintenance...
...undocumented code is almost useless,
(...undocumented assignments remain un-graded.)
- What to document?
- How to write documentation?
- Tool support?
- Semi-formal documentation?
 - Doug Lea's, *Draft Java Coding Standard*,
 - JavaSoft's, *Code Conventions for the Java Programming Language*
 - Pike & Kernigan, *The Practice of Programming*, Ch 2

February 14, 2005

2

Structure and documentation

- Packages
 - A new java package for each self-contained project or group of related functionality
 - An index.html file in each directory briefly outlining package purpose and structure

February 14, 2005

3

Structure and documentation

- Program Files
 - Place each class in a separate file. This applies even to non-public classes
 - except in the case of one-shot usages where the non-public class cannot conceivably be used outside of its context.
 - Begin each file with:
 1. file name & related information including copyright
 2. history table listing dates, authors, and summaries of changes.
 3. if the file is principal entry point for a package, briefly describe rationale for the package.
 - Immediately follow with:
 - package name
 - import list.

February 14, 2005

4

Structure and documentation

• Program Files

- Example:

```
/*
File: Example.java
Date      Author    Changes
Sep 1   95 Doug Lea   Created
Sep 13  95 Doug Lea   Added new doc conventions
*/

package demo;
import java.util.NoSuchElementException;
```

February 14, 2005

5

Documenting classes and interfaces

• Classes and Interfaces

- Write all `/** ... */` comments using javadoc conventions.
- Preface each class with a `/** ... */` comment describing purpose of class, guaranteed invariants, usage instructions, & examples.
- include any reminders or disclaimers about required or desired improvements.

- Use HTML format,

```
<pre> _ </pre>      -- preformatted
<b> _ </b>        -- bold
<em> _ </em>     -- emphasis
<code>_ </code>  -- fixed width font
<p> _ </p>       -- paragraph break
```

- with added tags:

```
@author author-name
@version version number of class
@see string
@see URL
@see classname#methodname
```

February 14, 2005

6

Documenting classes and interfaces

• Classes and Interfaces

Example:

```
/**
 * A class representing a window on the screen.
 * For example:
 * <pre>
 *     Window win = new Window(parent);
 *     win.show();
 * </pre>
 *
 * @see      awt.BaseWindow
 * @see      awt.Button
 * @version  1.2 31 Jan 1995
 * @author   Bozo the Clown
 **/
class Window extends BaseWindow {
    ...
```

February 14, 2005

7

Documenting classes and interfaces

• Class Variables

- Use javadoc conventions to describe nature, purpose, constraints, and usage of instances variables and static variables.

- Use HTML format with added tags:

```
@see string
@see URL
@see classname#methodname
```

Example:

```
/**
 * The current number of elements.
 * must be non-negative, and less than or equal to capacity.
 **/
protected int count_;
```

February 14, 2005

8

Documenting classes and interfaces

• Methods

- Use javadoc conventions to describe nature, purpose, preconditions, effects, algorithmic notes, usage instructions, reminders, etc. Use
- HTML format, with added tags:

```
@param paramName description
@return description of return value
@exception exceptionName description
@see string
@see URL
@see classname#methodname
```

February 14, 2005

9

Documenting classes and interfaces

• Methods

- Be as precise as reasonably possible in documenting effects. Here are some conventions and practices for semi-formal specs.

```
@return condition: (condition)
```

- describes postconditions and effects true upon return of a method.

```
@exception exceptionName IF (condition)
```

- indicates the conditions under which each exception can be thrown. Include conditions under which uncommon unchecked (undeclared) exceptions can be thrown.

```
@param paramname WHERE (condition)
```

- indicates restrictions on argument values. Alternatively, if so implemented, list restrictions alongside the resulting exceptions, for example `IllegalArgumentException`. In particular, indicate whether reference arguments are allowed to be null.

February 14, 2005

10

Documenting classes and interfaces

• Methods

```
WHEN (condition)
```

- indicates that actions use guarded waits until the condition holds.

```
RELY (condition)
```

- describes assumptions about execution context. In particular, relying on other actions in other threads to terminate or provide notifications.

```
GENERATE T
```

- to describe new entities constructed in the course of the method.

```
ATOMIC
```

- indicates whether actions are guaranteed to be uninterfered with by actions in other threads (normally as implemented via synchronized methods or blocks).

```
PREV(obj)
```

- refers to the state of an object at the onset of a method.

February 14, 2005

11

Documenting classes and interfaces

• Methods

```
OUT(message)
```

- describes messages (including notifications such as `notifyAll`) that are sent to other objects as required aspects of functionality, or referred to in describing the effects of other methods.

```
foreach (int i in lo .. hi) predicate
```

- means that predicate holds for each value of `i`.

```
foreach (Object x in e) predicate
```

- means that the predicate holds for each element of a collection or enumeration.

```
foreach (Type x) predicate
```

- means that the predicate holds for each instance of `Type`.

```
-->
```

- means 'implies'.

February 14, 2005

12

Documenting classes and interfaces

• Methods

unique

- means that the value is different than any other. For example, a unique instance variable that always refers to an object that is not referenced by any other object.

fixed

- means that the value is never changed after it is initialized.

EQUIVALENT to { code segment }

- documents convenience or specialized methods that can be defined in terms of a few operations using other methods.

February 14, 2005

13

Documenting classes and interfaces

• Methods

- Example:

```
/**
 * Insert element at front of the sequence
 *
 * @param element the element to add
 * @return condition:
 * <PRE>
 * size() == PREV(this).size()+1 &&
 * at(0).equals(element) &&
 * foreach (int i in 1..size()-1)
 *     at(i).equals(PREV(this).at(i-1))
 * </PRE>
 */
public void addFirst(Object element);
```

February 14, 2005

14

Documenting decls stmts & exps

• Local declarations, statements, and expressions

- Use /* ... */ comments to describe algorithmic details, notes, and related documentation that spans more than a few code statements.

- Example:

```
/*
 * Strategy:
 * 1. Find the node
 * 2. Clone it
 * 3. Ask inserter to add clone
 * 4. If successful, delete node
 */
```

- Use Running // comments to clarify non-obvious code.

February 14, 2005

15

Documenting decls stmts & exps

• Local declarations, statements, and expressions

- Do not comment obvious code; instead try to make code obvious!
- Example:

```
int index = -1; // -1 serves as flag meaning the index isn't valid
```

Or, often better:

```
static final int INVALID= -1;
int index = INVALID;
```

- Use any consistent set of choices for code layout, including:
 - Number of spaces to indent.
 - Left-brace ("{") placement at end of line
 - Maximum line length.
 - Spill-over indentation for breaking up long lines.
 - Declare all class variables in one place (at the top of the class).

February 14, 2005

16

Naming conventions

packages

- lowercase.
- recommended domain-based conventions described in the Java Language Specification, page 107 as prefixes.

files

- same base name as the public class they define.

classes

- CapitalizedWithInternalWordsAlsoCapitalized

Exception class

- ClassNameEndsWithException.

Interface

- When necessary to distinguish from similarly named classes: InterfaceNameEndsWithIfc.

Class

- When necessary to distinguish from similarly named interfaces:
 - ClassNameEndsWithImpl OR
 - ClassNameEndsWithObject

February 14, 2005

17

Naming conventions

constants (finals)

- UPPER_CASE_WITH_UNDERSCORES

private or protected

- firstWordLowerCaseButInternalWordsCapitalized OR
- trailingUnderscore_, OR
- thisVar (i.e. prefix with this), OR
- myVar (i.e. prefix with my), OR
- fVar (i.e. prefix with f)

static private or protected

- firstWordLowerCaseButInternalWordsCapitalized OR
- twoTrailingUnderscores__

local variables

- firstWordLowerCaseButInternalWordsCapitalized OR
- lower_case_with_underscores

February 14, 2005

18

Naming conventions

methods

- firstWordLowerCaseButInternalWordsCapitalized()

factory method for objects of type X

- newX

converter method that returns objects of type X

- toX

method that reports an attribute x of type X

- X x() or X getX().

method that changes an attribute x of type X

- void x(X value) or void setX(X value).

February 14, 2005

19

Javadoc

◆ Example

```
/*
 * @(#)Blah.java      1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 */
package java.blah;
import java.blah.blahdy.BlahBlah;
```

February 14, 2005

20

Javadoc

```
/**
 * Class description goes here.
 *
 * @version    1.82 18 Mar 1999
 * @author    Firstname Lastname
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** classVar1 documentation comment */
    public static int classVar1;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;
```

February 14, 2005

21

Javadoc

```
/** instanceVar1 documentation comment */
public Object instanceVar1;

/** instanceVar2 documentation comment */
protected int instanceVar2;

/** instanceVar3 documentation comment */
private Object[] instanceVar3;

/**
 * ...constructor Blah documentation comment...
 */
public Blah() {
    // ...implementation goes here...
}
```

February 14, 2005

22

Javadoc

```
/**
 * ...method doSomething documentation comment...
 */
public void doSomething() {
    // ...implementation goes here...
}

/**
 * ...method doSomethingElse documentation comment...
 * @param someParam description
 */
public void doSomethingElse(Object someParam) {
    // ...implementation goes here...
}
}
```

February 14, 2005

23