Software Development (CS2500)

Lecture 12: JavaDoc and Coding Conventions

M.R.C. van Dongen

October 18, 2013

Outline

- The JavaDoc documentation mechanism, and
- Some important Java coding conventions.
- All assignment should include proper JavaDoc comments.
- Code in assigments should comply with the coding conventions.
- This includes the code for Assignment 3.

Generating Documentation with JavaDoc

- JavaDoc is a tool for creating html documentation.
- The documentation is generated from java comments.
- Comments are formatted as *doc comments*.
- The documentation is generated by the javadoc program.

Unix Session

```
$ javadoc LuvelyClass.java
```

Doc Comments

- Doc comments start with `/**` and end in `*/`.
- The `/**` and `*/` should be on a line of their own.
- Additional lines should start with `*`.
- The comments may contain `html` tags.

Java

```
/**
 * This is an <strong>example</strong>.
 */
```

Doc Comments

- Doc comments are subdivided into *descriptions* and *tags*.

  Description: Provide overview of functionality of the code.

  Tag: Tags specify/address specific information.

  - Includes information about author, version, ...

Descriptions

- First line of each doc is a description of the API.
- It is automatically included in documentation.
- It should be *one single* sentence:
  - It should concise description of the documented API.
  - It should *not* explain the implementation.
- Additional comments may explain the API in further detail.

Example

Java

```
public class Sheep {
    /**
     * Constructor for anonymous sheep.
     */
    public Sheep( ) { ... }

    /**
     * Constructor for sheep with a given name.
     *
     * The name should be unique.
     */
    public Sheep( String name ) { ... }
}
```

Example

Don't Try This at Home

```
public class Sheep {
    /**
     * Constructor.
     */
    public Sheep( ) { ... }

    /**
     * Constructor.
     */
    public Sheep( String name ) { ... }
}
```

Tags

- *Tags* are used to specify content and markup.
- Tags are case-sensitive and should start with @.

Java

```
/**
 * Basic print method.
 *
 * @author Java Joe.
 * @param bar The thing to be printed.
 */
public void printStuff( int bar ) {
    ⋮
}
```

Kinds of Tags

- *Block tags* are of the form @⟨tag name⟩.
  - Should be placed in tag section following main description.
- *Inline tags* are of the form {@⟨tag name⟩ ⟨more⟩}.
  - They may occur anywhere.

Java

```
/**
 * Friendly class.
 *
 * More information {@link #hello here}.
 */
public class Hello {
    /**
     * Friedly method that prints "hello world."
     */
    public static void hello( ) {
        System.out.prinln( "hello world." );
    }
}
```

Some Existing Tags

@author: describes the author(s).

@param: describes a specific parameter.

@version: describes the version.

@return: describes the return value.

Java

```
/**
 * Compute the length of a given list.
 *
 * @param list The given list.
 * @param <T>  The type of the elements in the list.
 * @return     The length of the list.
 */
public int length( List<T> list ) ...
```

Hyperlinks

Java

```
/**
 * {@link  ⟨package⟩.⟨class⟩#⟨member⟩  ⟨text⟩}
 */
```

Proper Order

Java

```
/**
 * @param       ...
 * @return      ...
 * @exception   ...
 * @author      ...
 * @version     ...  (Not needed for CS2500.)
 * @see         ...  (Not needed for CS2500.)
 * @since       ...  (Not needed for CS2500.)
 * @serial      ...  (Not needed for CS2500.)
 * @deprecated  ...  (Not needed for CS2500.)
 */
```

Coding Conventions

Why Bother

- 80% of the lifetime cost of software goes to maintenance.
- Others will have to read your code.
- Improves readability of your lovely code.
- Shipped code should be well packaged and clean.

Coding Conventions

Why Bother

- 80% of the lifetime cost of software goes to maintenance.
- Others will have to read your code.
- Improves readability of your lovely code.
- Shipped code should be well packaged and clean.

Coding Conventions

Why Bother

- 80% of the lifetime cost of software goes to maintenance.
- Others will have to read your code.
- Improves READABILITY *of* your $\mathcal{LUVELY}$ **code.**
- Shipped code should be well packaged and clean.

Coding Conventions

Why Bother

- 80% of the lifetime cost of software goes to maintenance.
- Others will have to read your code.
- Improves readability of your lovely code.
- Shipped code should be well packaged and clean.

Most Importantly

- Follow the company conventions, and
- The conventions of the person whose code you're modifying.

Files

- Files should consist of sections.
  - Sections should be separated by blank lines and comments.
- Files longer than 2000 lines should be avoided.
- `Javadoc` comments should be used to document the classes/interfaces, attributes, and methods.
- The `import` statements always go to the top of the file.
- If you need more than one import, use the * notation:

Coding Convention

```
import java.util.*;
```

Don't Try This at Home

```
import java.util.TreeMap;
import java.util.Random;
```

Class and Interfaces: Structure

Add Implementation Comments where Needed

1. `package` statement.
2. `import` statements.
3. `JavaDoc` class-related comments.
4. Class variables in decreasing order of visibility:
   - First `public`, then `protected`, and then `private`.
5. Instance variables in decreasing order of visibility.
6. Constructors.
7. Methods.

Classes and Interfaces

Coding Convention

```
class Example { // Opening brace here.
⋮

} // Closing brace here.
```

Classes and Interfaces

Coding Convention

```
class Example { // Opening brace here.
⋮

} // Closing brace here.
```

Don't Try This at Home

```
class Example
{ // Opening brace here.
⋮

} // Closing brace here.
```

Indentation

- Use four spaces as the unit for indentation.
- Use eight spaces if that improves readability.
- Avoid lines that are longer than 75 characters.
- Use the following rules for wrapping lines if they're too long:
  - Break after a comma;
  - Break before an operator;
  - Prefer higher-level breaks to lower-level breaks; and
  - Align text with broken expression on previous line.
- Compound statements (blocks):
  - The enclosed statements should be indented one more level.
  - Opening brace: at end of line that begins the block.
  - Closing brace: indented at same level as line at start of block.

Breaking Method Calls

Coding Convention

```
call1( longExpr1, longExpr2, longExpr3,
       longExpr2, longExpr3 );

int var = call2( longExpr1,
                 call3( longExpr2, longExpr3,
                        longExpr2, longExpr3 ) );
```

Don't Try This at Home

```
int var = call2( longExpr1, call3( longExpr2,
                 longExpr3, longExpr2, longExpr3 ) );
int var = call2( longExpr1, call3( longExpr2,
                                   longExpr3,
                                   longExpr2,
                                   longExpr3 ) );
```

Breaking Arithmetic Expressions

Coding Convention

```
longVariable = longExpr1 + (longExpr2 - longExpr3)
               / longExpr5;
```

Don't Try This at Home

```
longVariable = longExpr1 + (longExpr2
                            - longExpr3) / longExpr4;
```

Further Examples

Coding Convention

```
if ((condition1 && condition2)
        || (condition3 && condition4)) {
    // Stuff
}
```

Don't Try This at Home

```
if ((condition1 && condition2)
    || (condition3 && condition4)) {
    // Stuff
}
```

Further Examples

Don't Try This at Home

```
var = condition ? thisStuff : thatStuff;
```

Coding Convention

```
var1 = (condition) ? thisStuff : thatStuff;
var2 = (condition) ? thisStuff
                   : thatStuff; // Clearer!
var3 = (condition)
       ? thisStuff
       : thatStuff; // Also impossible to miss!
```

Comments

Don't Try This at Home

```
public int answer( ) {
    /* Temporarily commented out for testing.
    /*
     * This gives you the answer.
     */
    */
    return 42;
}
```

Java

```
public int answer( ) {
    /* Temporarily commented out for testing.
    //
    // This gives you the answer.
    //
    */
    return 42;
```

Comments

Don't Try This at Home

```
public int answer( ) {
    /* Temporarily commented out for testing.
    /*
     * This gives you the answer.
     */
    */
    return 42;
}
```

Java

```
public int answer( ) {
    /* Temporarily commented out for testing.
    //
    // This gives you the answer.
    //
    */
    return 42;
```

Declarations

- Ideally, there should be one declaration per line.

Coding Convention

```
int one; // Comment about purpose of one.
int two; // Comment about purpose of two.
```

Don't Try This at Home

```
int one, many[]; // Valid Java, but not for CS2500.
```

- Minimise the scope of your variable [Bloch 2008, Item 29].

One Statement Per Line

- No more than one statement per line.

Don't Try This at Home

```
thisVar++; thatVar--;
```

- Don't use the comma operator.

Don't Try This at Home

```
thisVar++, thatVar--;
```

The `return` Statement

- Avoid parentheses for `return` statements (unless this is clearer).

Coding Convention

```
return myLuvelyComputation( );
...
return (condition ? thisValue : thatValue);
```

The `return` Statement

- Avoid parentheses for `return` statements (unless this is clearer).

Coding Convention

```
return myLuvelyComputation( );
...
return (condition ? thisValue : thatValue);
```

Use a Single Exit Point

- Each block/method/construct should have a single exit point.

Don't Try This at Home

```
public void myLuvelyMethodA( ) {
    if (condition) {
        return; // Valid Java but not for CS2500.
    }
    ...
}

public int myLuvelyMethodB( ) {
    if (condition) {
        return thatValue;
    }
    ...
    return thisValue;
}
```

Use a Single Exit Point

Don't Try This at Home

```
while (thisCondition) {
    if (thatCondition) {
        break;
    }
    ...
}
```

The `if` Statement

Always use Braces for `if` Statements.

Coding Convention

```
if (condition1) {
    ...
}
if (condition2) {
    ...
} else {
    ...
}
if (condition3) {
    ...
} else if (condition4) {
    ...
} ...
```

The `for` Statement

- Use braces for `for` statements with non-empty body.

Coding Convention

```
for ( initialisation; condition; update ) {
    ...
}
```

The `for` Statement

- For `for` statements with empty body add semicolon as follows.

Coding Convention

```
for ( initialisation; condition; update ) ; // empty body
```

- Arguably it is clearer to use a `while` loop:

Java

```
initialisation;
while (condition) {
     update
}
```

The `while` Statement

- Use braces for `while` statements with non-empty body.

Coding Convention

```
while ( condition ) {
    ...
}
```

The `while` Statement

- For `while` statements with empty body add semicolon as follows.

Coding Convention

```
while ( condition ) ; // empty body
```

- Arguably it is *much* clearer if you use the `do-while` statement:

Coding Convention

```
do {
} while ( condition );
```

The do-while Statement

Coding Convention

```
do {
    ...
} while ( condition );
```

White Space

- Adding white space generally improves readability.
- Add a blank line for the following:
  - Between method definitions.
  - Between local variable declarations and statements in a block.
  - Before a block.
  - Between logical sections inside a method to improve readability.

Adding Space

- A keyword followed by a parenthesis:

Coding Convention

```
while␣(condition) {
```

- A parenthesis followed by a brace:

Coding Convention

```
while (condition)␣{
```

- After commas in argument lists.
- Before and after binary operators (except `.`):

Coding Convention

```
var1␣=␣var2␣+␣var3␣*␣var4␣/␣(var5.method( )␣-␣1);
```

- After the semicolons in the `for` statement:

Coding Convention

```
for (start;␣condition;␣update) {
```

Naming Conventions

Classes: Class names should be nouns in mixed case.

First letter in each internal word should be upper case.

Use whole words and avoid non-obvious acronyms.

Interfaces: Interfaces should be mixed case adjectives.

- Many interface names end in '`able`'.

Methods: Method names should be verbs in mixed case.

The first letter should be lower case.

First letter of remaining words should be upper case.

Constants: Class constants should be upper case with words separated with underscores.

Variables: Variables should be short, yet meaningful nouns.

The naming scheme is the same as for methods.

Methods

- Methods should be short.
- If they are longer than 40 lines, shorten them.
  - Introduce sub-methods to carry out sub-tasks.
  - This should improve the readability/understanding/development.

Other Coding Practice

To be announced.

Acknowledgements

- The section about JavaDoc is partially based on [Lewis, and Loftus 2009, Appendix I].
- More information about JavaDoc may be found at
  - http://java.sun.com/j2se/javadoc/writingdoccomments.
- The section about coding conventions is based on [Sun 1997].

Bibliography


Bloch, Joshua [2008]. *Effective Java*. Addison–Wesley. ISBN: 978-0-321-35668-0.


Horstmann, Cay S. [2013]. *Big Java, Early Objects*. International Student Version. Wiley. ISBN: 978-1-118-31877-5.


Lewis, John, and William Loftus [2009]. *Java Software Solutions Foundations of Program Design*. Pearson International. ISBN: 978-0-321-54934-1.


Sun [1997]. *Java Code Conventions*. URL: http://java.sun.com/docs/codeconv.

For Monday

- Study the notes,
- Study [Horstmann 2013, Chapter 5].
- Carry out [Horstmann 2013, Exercises 5.11 and 5.21].
- Carry out [Horstmann 2013, Exercise 5.22 or 5.23] (optional).

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.