



# Coding Standards

Based on notes by Philip  
Johnson, University of Hawaii



# Handout

- Take 5 minutes to answer the questions about the code on the handout.



# Motivation

- Understand motivation for coding standards
- Be able to write code that conforms to class coding standards.
- Be able to recognize and repair non-compliant code.
- Be motivated to learn to use automated tools:
- Checkstyle



# Why Coding Standards?

- Improve readability by ensuring a common “look and feel” to code regardless of how many people have worked on it
- Improve understandability by ensuring that basic documentation is always present
- Improve maintainability by improving the predictability of the code



# Principles and conventions

- General Principles
- Formatting conventions
- Naming conventions
- Documentation conventions
- Programming conventions
- Packaging conventions



# General Principles

1. Adhere to the style of the original.
2. Adhere to the Principle of Least Astonishment: simplicity, clarity, completeness, consistency, robustness
3. Do it right the first time.
4. Document any deviations.



# Formatting conventions

- Indent nested code.
  - 2 spaces
  - Open brace at end of line.
  - Close brace appears by itself on a line.
- Break up long lines.
  - 80 characters per line maximum.
- Include whitespace.
- Do not use tabs.

# Naming Conventions

- 9. Use meaningful names.
  - No one character names (except index vars).
- 10. Use familiar names.
  - Learn the application domain, talk to users.
- 11. Question excessively long names.
  - May indicate need to refactor/redesign.
- 12. Join the vowel generation.
  - putSoundFile, not ptSndFl

# Naming Conventions

13. Capitalize only the first letter in acronyms.

- `getNode`, not `getXMLNode`

14. Do not use names that differ only in case.

- `theInstance` and `TheInstance`

18. Capitalize the first letter of each word in a class or interface name.

- ex: `ServerProperties`

19. Use nouns when naming classes.

- `CustomerAccount`, not `MaintainCustomerData`



# Naming Conventions

20. Pluralize class names that group related attributes.

- ex: ServerProperties

21. Use nouns or adjectives when naming interfaces.

- ActionListener, or Runnable

22. Use lowercase first word and capitalize first letter of subsequent words in method names.

- getServerHostName



# Naming Conventions

23. Use verbs when naming methods.

- ex: withdraw, reset, clear,

24. Use conventions for property accessor methods.

- is/get/set

25. Lowercase first word and capitalize remaining words in variable names

- daytimePhone

# Naming Conventions

## 26. Use nouns to name variables

- daytimeAddress, not getDaytimeAddress

## 27. Pluralize names of collection references.

- Customer [] customers;

## 28. Use standard “throwaway” variable names.

- index: i, j, k
- exception: e
- generic type variable: S, T

# Naming Conventions

29. Quantify field variables with 'this' to distinguish them from local variables.

- ex: `this.address = address;`

30. When a parameter assigns a value to a field, use the same name for the parameter and the field.

- ex: `this.address = address;`

31. Use uppercase for words and separate with underscore for naming constants.

- ex: `MAX_VALUE`

# Programming Conventions

71. Make all fields private.

75. Always use block statements in control flow constructs.

- Not: `if (foo) bar( );`
- Instead: `if (foo) { bar( ) };`

# Programming Conventions

79. Use equals, not ==, to test for equality.

80. Always construct objects in a valid state.

82. Use nested constructors to eliminate redundant code.

(Others) Do not use wildcards in import.

Not: `import java.util.*;`

Instead: `import java.util.Set;`

Enforced by Checkstyle.



# Programming Conventions

- 83. Use unchecked, runtime exceptions to report serious unexpected errors that may indicate an error in the program's logic.
- 84. Use checked exceptions to report errors that may occur, however rarely, under normal program operation.
- 85. Use return values to report expected state changes.
- 87. Do not silently absorb a runtime or error exception.



# Documentation Conventions

32. Write comments for those who use and for those who maintain your code.

- Assume familiarity with Java, but not with your application.

33. Keep comments and code in sync.

34. Use the active voice and omit needless words.

- Use forceful, clear, concise language.



# Documentation Conventions

- 35. Use documentation comments to describe the programming interface.
  - Defines a “contract” between a client and a supplier of a service.
- 36. Use standard comments to hide code without removing it.
- 37. Use one-line comments to explain implementation details.
  - Assume the reader knows Java.
  - Do not repeat what the code does.



# Documentation Conventions

38. Describe the programming interface before you write the code.

- This can be your “design phase”.

39. Document public, protected, private, and package private members.



# Documentation Conventions

46. Use a fixed ordering for Javadoc tags.

47. Write in third-person narrative form.

- ex: “Gets...”, “Sets...”, “Allocates...”

# Documentation Conventions

48. Write summary descriptions that stand alone.
- First sentence must summarize behavior.
49. Omit the subject in summary descriptions.
- Not: “This method clears the foo.”
  - Instead: “Clears the foo.”
60. Describe **why** the code is doing what it’s doing, **not what** it does.

# Checkstyle

- Automated support for certain coding style requirements including:
  - variables, parameters, and methods conform to regular expressions.
  - lines do not exceed a specified length.
  - JavaDocs are present and provide parameter and return type information.

# Eclipse Source Formatter

- Can save you time by automatically:
  - indenting correctly
  - formatting braces
  - inserting JavaDoc \*templates\*
- Notes:
- Eclipse format settings are not correct “out of the box”. You must manually configure them.  
**See CITS1220 lab notes**
- Eclipse creates JavaDoc templates, but they are not (typically) useful documentation. You must replace template text even though it will pass Checkstyle!