Coding Standards

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii
Honolulu HI 96822

Objectives

Understand motivation for coding standards

Be able to write code that conforms to JOSSE coding standards.

Be able to recognize and repair non-compliant code.

Be able to apply automated tools:
- Checkstyle ant task
- JRefactory pretty printer

Why coding standards?

Coding standards:
- improve readability by ensuring a "common look and feel" to the 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.

Elements of Java Style

Consolidates the best practices of the Java community gained from the past five years of language use (and experience with other languages before that).

The "authoritative" source for coding style in this class.

Additional coding standards available at the JOSSE web site.

Coding standard principles and conventions

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

The next sections overview important rules from Elements of Java Style. However, these slides overview the rules, they do not contain enough information to implement them! You must still study the book.

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

5. Indent nested code.
   - 2 spaces
   - Open brace at end of line.
   - Close brace appears by itself on a line.

6. Break up long lines.
   - 100 characters per line maximum.

7. Include whitespace.

8. 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.
 •getXmlNode, not getXMLNode

14. Do not use names that differ only in case.
 •theInstance and TheInstance

15. Use reversed internet domain names as package prefix.
 •"edu.hawaii." for this class

16. Use a single lowercased word as the root name for each package.
 •not edu.hawaii.serverkernel;

Naming conventions

17. <binary compatibility constraint on new package names>
 •Not appropriate for pre-release development.
 •Omit for JOSSE.

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. Follow JavaBeans 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

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

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.
- Private documentation comments not currently enforced by Checkstyle settings in Ant.

40. Provide a summary description and overview of each package.
- package.html

Documentation conventions

41. Provide a summary description for your application.
- overview.html

42. Use a consistent format and organization for all JavaDoc comments.
- JRefactory pretty printer helps with this!

43. <Wrap keywords with <code> tags.>
- Optional for JOSSE

Documentation conventions

44. Wrap code with <pre> tags.

45. Consider using @link tags.
- Optional for JOSSE.

46. Use a fixed ordering for JavaDoc tags.
- Use default JRefactory pretty printer order.

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.

Programming conventions

71. Make all fields private.

74. Encapsulate enumerations as classes.
- Google "typesafe enum" for a good pattern.

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.
- See Dr. Hacker (1/27/2002)

82. Use nested constructors to eliminate redundant code.

(JOSSE) 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 codes to report expected state changes.

87. Do not silently absorb a runtime or error exception.

More on programming conventions

See "Effective Java", by Joshua Bloch, for a spectacular treatment of best practices in Java programming.

"I sure wish I had this book 10 years ago. Some might think that I don't need any Java books, but I need this one."

- James Gosling

(Inventor of Java)

Packaging conventions

104. Place types that are commonly used, changed, and released together into the same package.

Observe JOSSE build packaging conventions.