

Coding Standards

8/13/00 - [Go Back](#)

These are as simple and short as possible.

Think A Bit Comment - When one encounters complex, long coding standards and very little formal process, what you are seeing is an attempt to improve productivity by imposing a development process via coding standards. This is sure to have only a small effect, because the process is far, far more important than coding standards. However, coding standards are one of the first easy steps to a repeatable process.

Java Class Templates

_Template.txt - Standard new class or interface template
See notes at end.

```
package org.ajug.jsl.jester.cases;

/**
 * This class.... (describe responsibilities fully here)
 *
 * @author Jack Harich
 */
public class ClassName {

//----- Internal Fields -----
//----- Initialization -----
//----- Superclass Overrides -----
//----- Something Implementation -----
//----- Properties -----
//----- Events -----
//----- Public Methods -----
//----- Protected Methods -----
//----- Standard -----
private static void print(String text) {
    System.out.println("ClassName" + text);
}

} // End class

===== Interface =====
package org.ajug.jsl.jester.cases;
```

```

/**
 * This interface ...
 *
 * @author Jack Harich
 */
public interface Delegator {

//----- Public Methods -----

} // End interface

```

```

//----- Notes -----
The source code dividers are in order of importance. Always use
dividers to make your code more organized and readable.

```

Make your code no wider than the dividers. Delete the dividers not needed, such as Properties and Events. Change Something Implementation to InterfaceName Implementation.

The dividers are 67 character long lines designed to allow two files to be side by side on a 1280 pixel wide screen with slider overlap, if using a 12 point font. The standard editor font size is 10 point, but this is too small on a 19" screen. Perhaps 10 point will be okay on the new cheap 21" monitors.

We are now encouraging protected instead of private methods.

Java Coding Standards

_Standards.txt - All classes should follow these.
Last updated 8/13/00. We are keeping these small.

```

=====
>>>> MINIMAL CODING STANDARDS <<<<<

```

Code must be very understandable. THE MOST IMPORTANT IMPLEMENTATION GOAL IS UNDERSTANDABILITY, BY BOTH THE USER AND DEVELOPER.

Comment all classes and public methods fully.
Add models or links when necessary.
Use comments liberally. Err on the side of more rather than less.
Use white space to delineate intentional blocks.

All code must satisfy our architectural requirements.
A *current* high level model must exist for *all* code, so "Model Before You Code".

Start all classes with the template, keep organized.
Emulate JavaSoft code for curly brace and indent style.
Indent 4 spaces.
One outer class or interface per file.
Keep normal inner classes at the end of the class in one place, under the comment:

```
//===== Inner Classes =====
```

A good class does one thing and does it well.
Strive for low coupling and high cohesion.
Keep it simple. Be conservative.

Use very careful when naming. This makes code self documenting.
Start class names with an upper case letter.
Start method and variable names with a lower case letter.
No magic numbers or cryptic variable names besides i, j, k.
Use zero based throughout at code level. However, a visual
representation may show one based, such as door number.
Thus all indexed collections are zero based.

Use constructors with arguments only when appropriate.
All class fields should be protected except for constants that
need to be made public.
Use getters and setters in the Bean Spec style.
Use events in the Bean Spec style, except no source object.
Recursive methods must be marked as RECURSIVE.
Minimize inheritance. Use composition/delegation instead.
Keep classes under 200 LOC plus comments, with deliberate exceptions.

IF - Always use braces unless on same line with if and short statement.

List custom imports before java imports.
Alphabetize imports but group for clarity.
All lowercase package names.
Explicitly import classes unless many from same package.

Good code should be so well written and follow these
standards so closely that you cannot tell who wrote
it, except for the author's name. Be humble...

```
=====
>>>> Class Naming Conventions - Suffixes <<<<
```

Note the suffix is the class role. These are examples.
Each group or project will need its own standards here.

-- Well known patterns -----

View, Model, Controller - Per standard MVC Pattern. The
controller is generally a business object.

Proxy - Per standard Proxy Pattern

Factory - Creates instances on demand

-- Other -----

Mgr - Manages a collection or subsystem

Def - Defines a parameter used in another class, which has
multiple such definitions. Often parameter driven.

Accessor - Carries data for accessing something

Pool - Provides a collection of instances ready for work

Set - Contains a set of like items, ie a RowSet contains rows

Lib - Library with all static fields and methods

Row, Map - Collection of name/value pairs

Event, Listener - Per Java beans spec

Source - An event source. Emits callbacks

Supplier - Supplies something on demand

Std - Standard interface implementation

Test - The unit test for a class

Services - Fascade for a subsystem, ie DatastoreServices

=====

>>>> Method Naming Conventions <<<<<

Follow the Java Bean spec for getters, setters, boolean properties, events, listeners.

----- Method Prefixes -----

-- Simple accessors

getPropertyName() - This should return the same value every time, unless its been changed by a set() or the class has done some "significant" internal change. Calling get() should not cause lots of work or state change, to allow class inspection.

createXXX() - Use this when a different, and probably new, instance is returned every time. For example a Factory will have a createXXX(), and Services classes often have them.

loadXXX() - Use this when neither getXXX() or createXXX() is appropriate. (not sure what's a good standard here)

-- Other

put - Use this when previous values may be overwritten.

add - Use this when previous value is irrelevant or not allowed.

put, remove - Return the previous value or null if none.

removeAll - Empties the collection